Entregando serviços seguros

7COMm
Serviços e Soluções em TI

Segurança de uma aplicação

1. Autenticação
2. Autorização
3. Auditoria
4. Comunicação segura interprocessos

Autenticação

- Verifica a identidade da aplicação ou do usuário
- Feita normalmente através de *user_id/password* ou *API key* e *secrets*

Autorização

- Verifica se o usuário tem permissão para executar a operação especificada
- Normalmente são utilizadas uma combinação de papeis e listas de controle de acessos (ACLs)
- Cada usuário pode ter um ou mais papeis que adicionam permissão para um determinado recurso

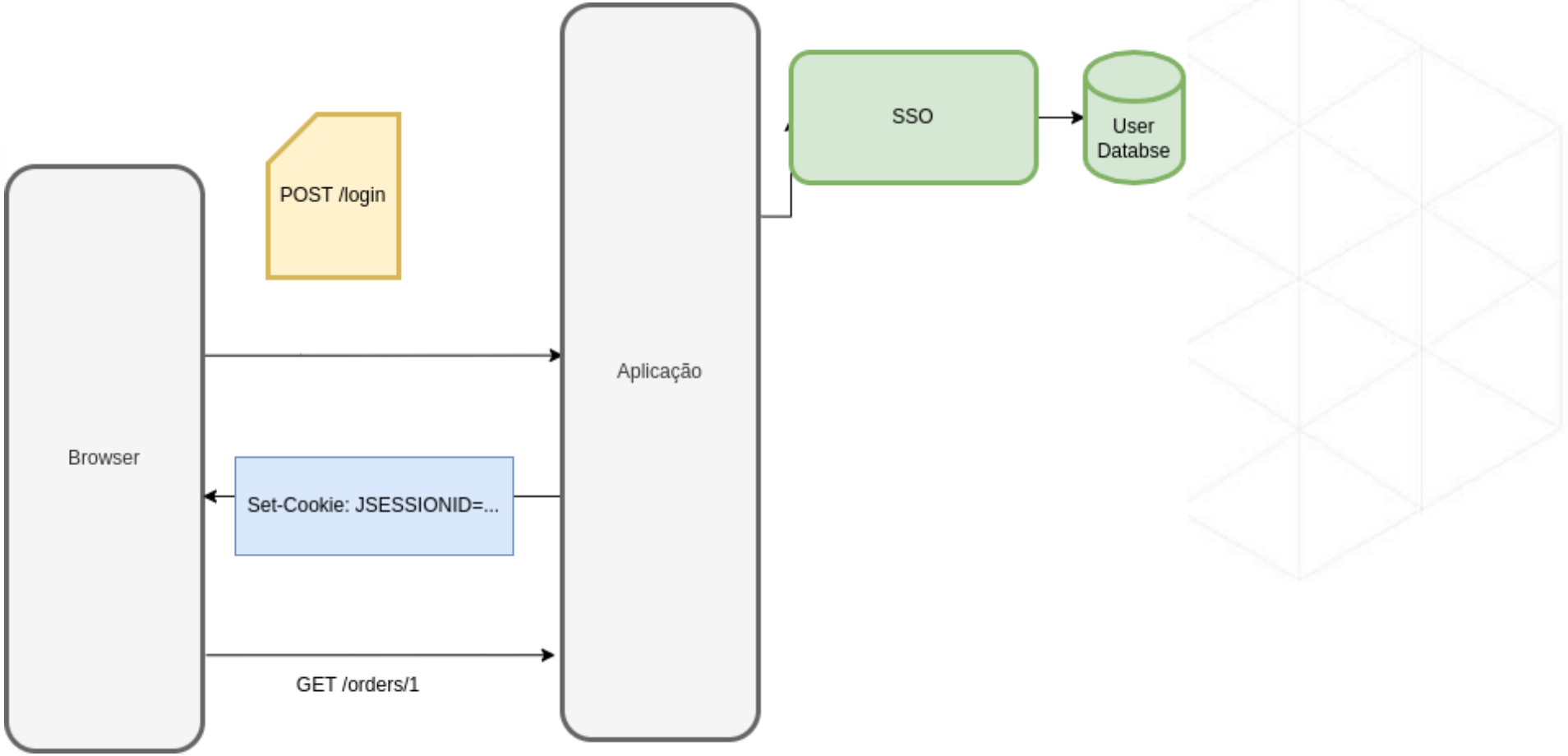
Auditoria

- A capacidade de monitorar as operações que um usuário realiza
- Ajuda a detectar pontos de segurança
- Ajuda no suporte ao usuário
- Reforça *compliance*

Comunicação segura

- Toda a comunicação entre serviços deve ser segura
- TLS
- Autenticação

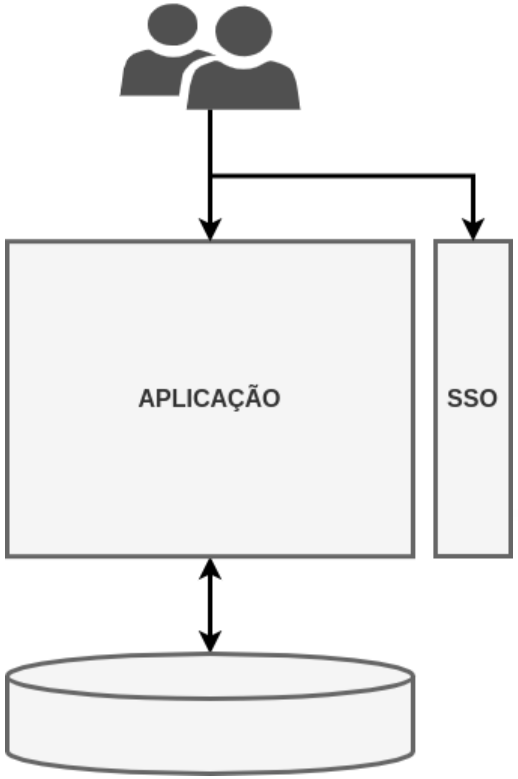
Segurança em monolito



Segurança em uma arquitetura de microsserviços

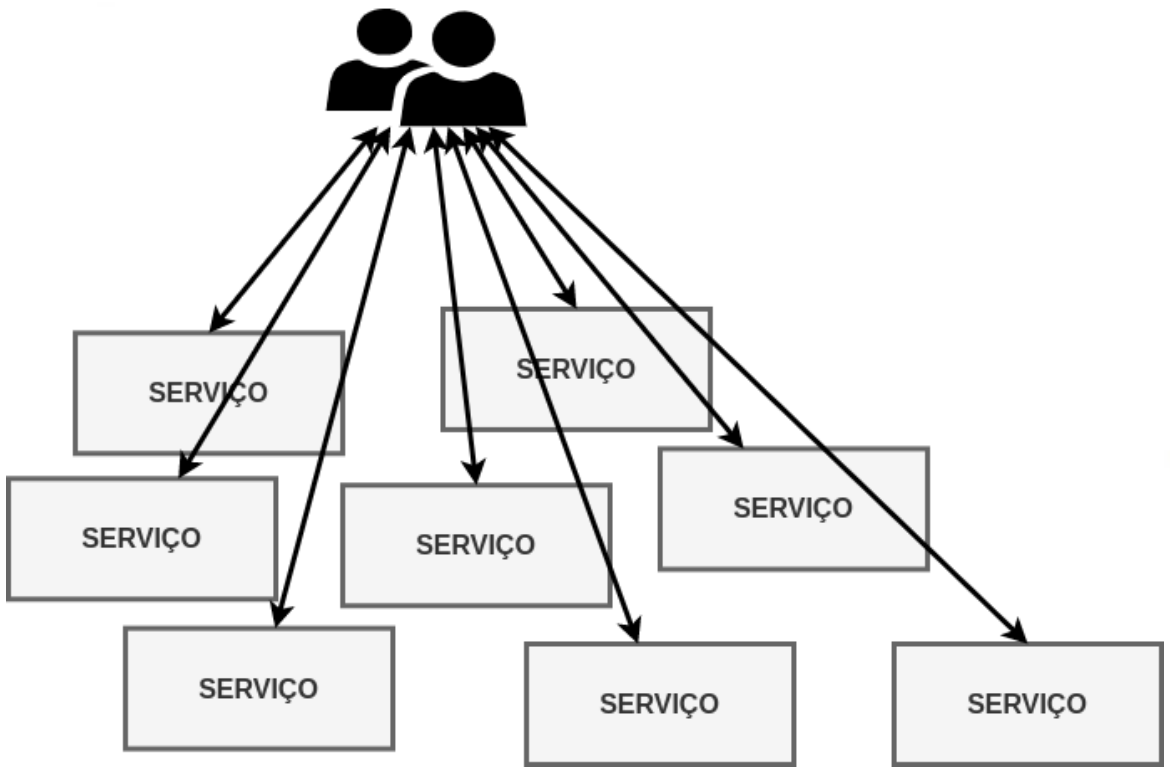
- Não devem seguir exemplos de uma arquitetura monolita
- Não utilizam sessão
- Servidores não compartilham sessão
- Sessão centralizada quebra o baixo acoplamento

Autenticação no monolito



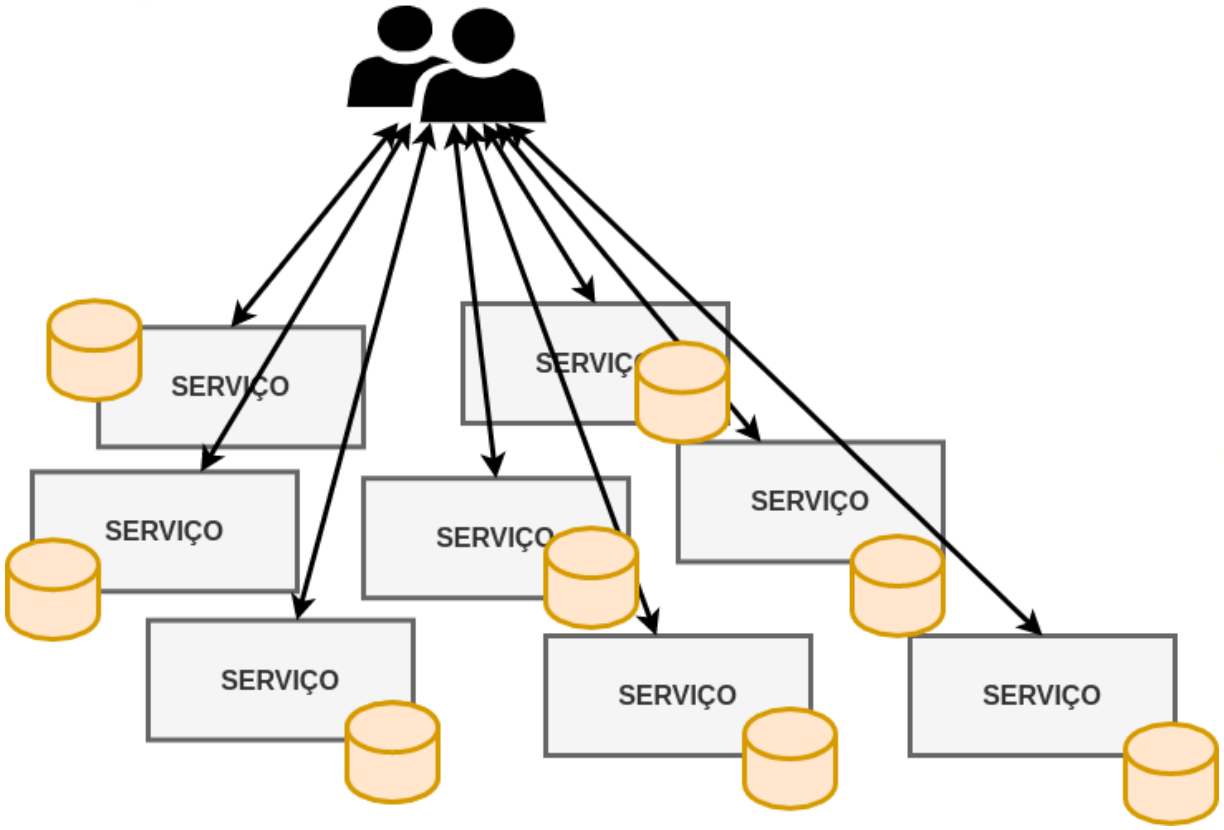
- Base de dados única
- Sessão de usuário centralizada

Autenticação em microsserviços



- Bancos descentralizados?

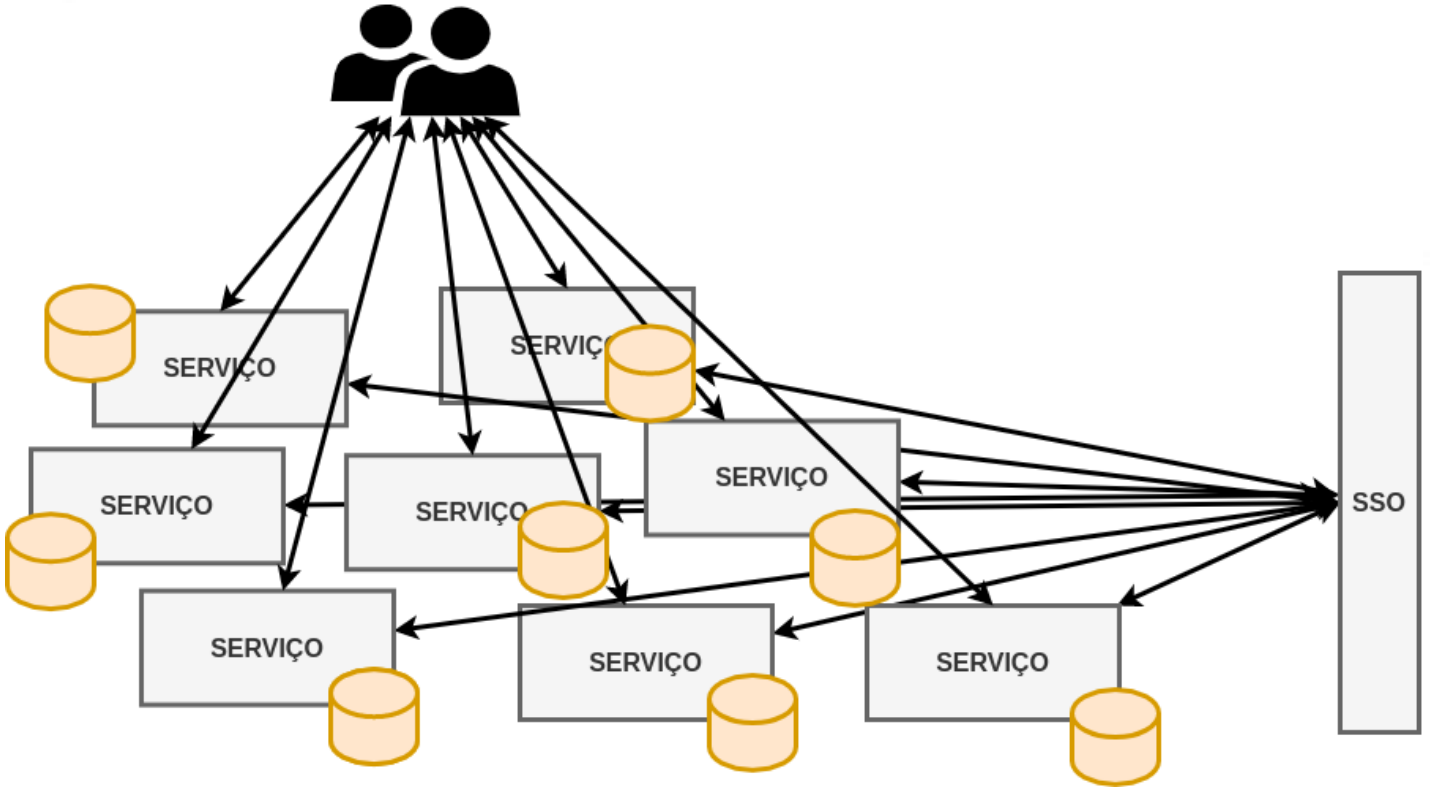
Autenticação em microsserviços



- Single Responsability
- Dificuldade de manutenção

- SSO?

Autenticação em microsserviços



- Alto acoplamento

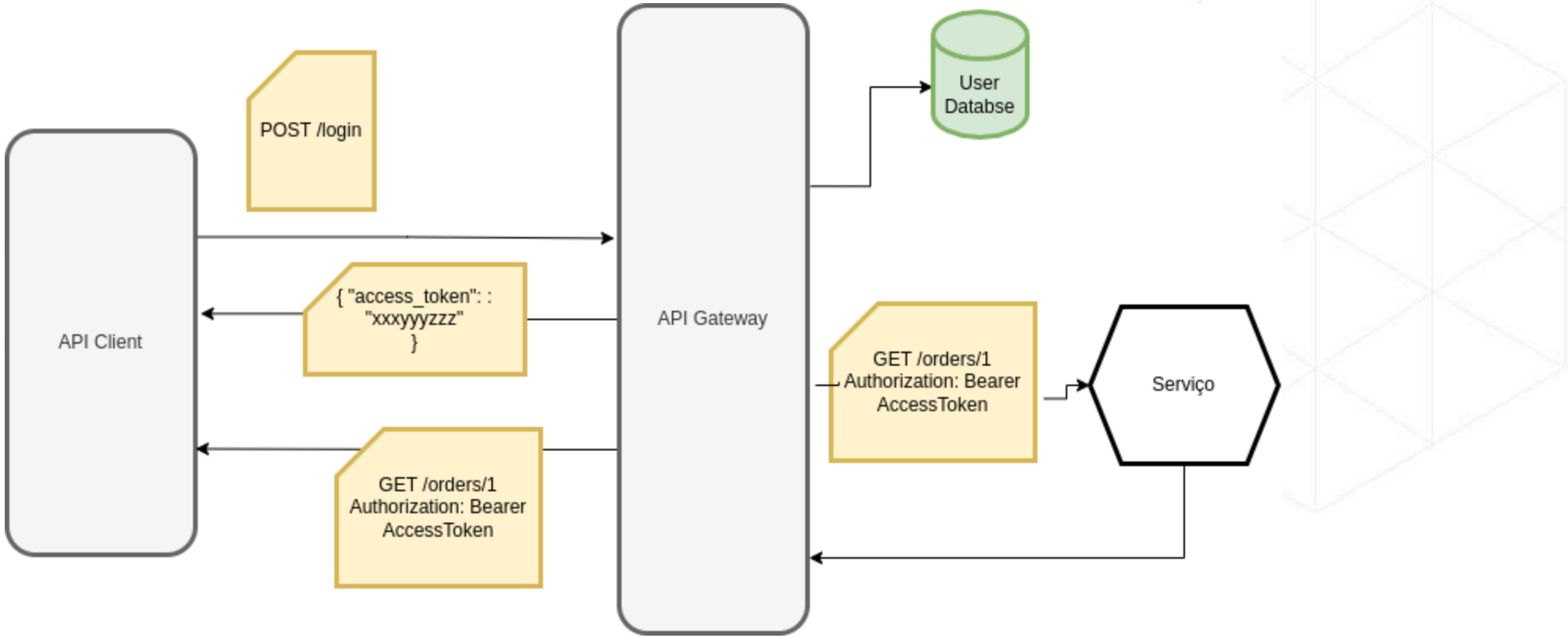
Autenticação no API Gateway

- Autenticação em cada serviço pode não ser ideal, pois permite requisições não autenticadas na rede interna
- Aumenta a complexidade da aplicação como um todo
- Uma melhor alternativa é implementar autenticação no próprio API Gateway

Autenticação no API Gateway

- Evita-se complexidade no ecossistema de autenticação
- Somente um lugar para lidar com a segurança da aplicação
- Somente ele deve se preocupar com a heterogeneidade das autenticações

Autenticação no API Gateway



Autorização

- Não necessariamente deve ser implementada no API Gateway
- Aumenta o acoplamento entre o API gateway e os serviços
- Requerem profundo conhecimento do domínio dos serviços
- Como comunicar a identidade do usuário aos outros serviços?
- http://microservices.io/patterns/security/access-token.html

Tokens de Autorização

- Uma vez autenticado, pode-se gerar tokens de autorização
- Estes podem ser validados nos próprios serviços
- Cada token pode conter os papéis de cada usuário

Passando identidade e papeis em um JWT

- Token com informações sobre o usuário
- Maneira segura de compartilhar tokens, identidade de usuários e papéis
- Pode possuir uma data de expiração
- Assinado com uma chave, o que assegura que terceiros não possam modificar este token
- Uma desvantagem é que este token é irrevogável
- Jwt.io

Header

```
base64enc({
    "alg":  "HS256",
    "typ":  "JWT"
})
```

Payload

```
base64enc({
      "iss": "toptal.com",
      "exp": 1426420800,
      "company": "Toptal",
      "awesome": true
})
```

Signature

```
HMACSHA256(
  base64enc(header)
  + '.' +,
  base64enc(payload)
  , secretKey)
```

toptal

JWT - Header

- Consiste normalmente de duas informações
  - Tipo de token
  - Algoritmo de assinatura

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

JWT - Payload

- Contém as *claims*
- Dados do usuário

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

JWT - Assinatura

- Aplica-se o algoritmo do header + secret conhecido somente pelo serviço nos dois primeiros campos do token

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

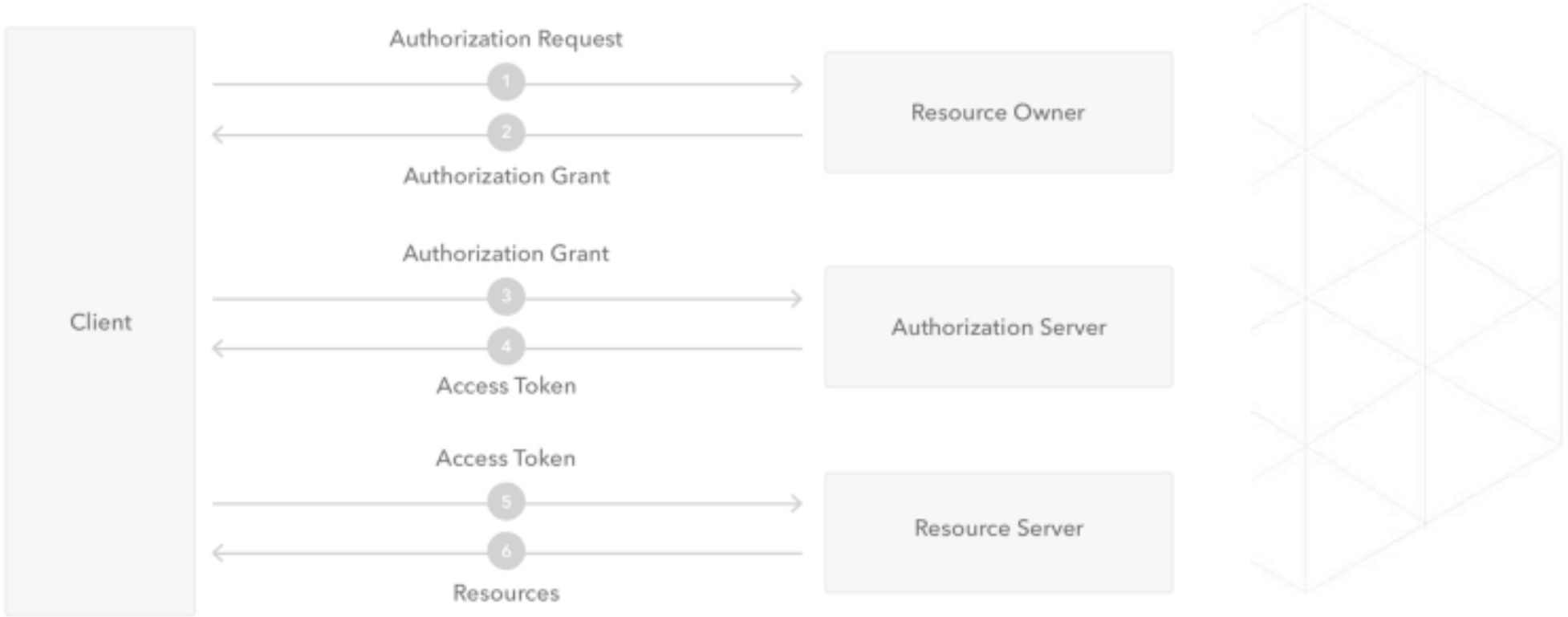
JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

OAuth 2

- Protocolo por procuração
- Permite acesso ***limitado*** aos dados de usuários
- Autorização baseada em *Roles*

OAuth 2


Client
Authorization Request
1
2
Authorization Grant
Resource Owner
Authorization Grant
3
4
Access Token
Authorization Server
Access Token
5
6
Resources
Resource Server

Testes em uma arquitetura de microsserviços

7COMm
Serviços e Soluções em TI

Testes, Visão geral

- https://en.wikipedia.org/wiki/Test_case
- Verificar se o comportamento de uma funcionalidade está correto
- Pode ter o escopo tanto de uma aplicação quanto de uma única classe
- Uma coleção de testes forma uma suíte de testes

Testes tradicionais

- Feitos no final do processo de desenvolvimento
- Realizado manualmente
- Muito ineficiente

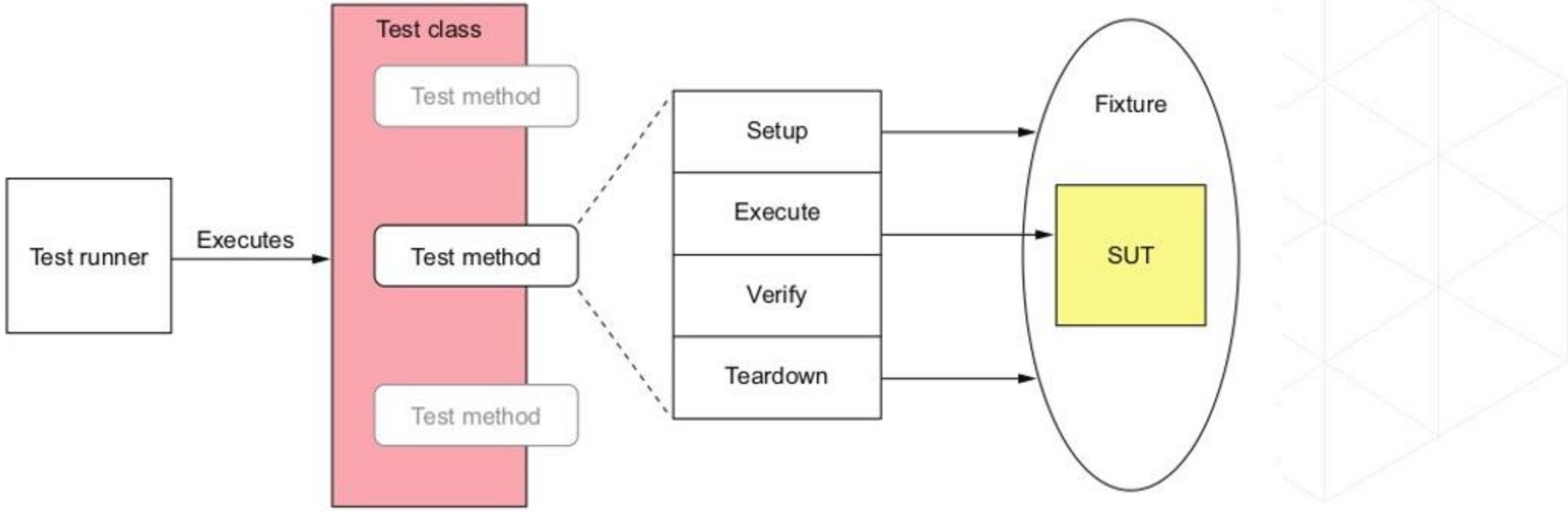
Testes automatizados

- Normalmente escrito por um framework de testes
- Pode ser realizado através de outras aplicações
- Fundamental para CI/CD

Testes automatizados

- http://xunitpatterns.com/Four%20Phase%20Test.html
- *Setup*, inicializa os acessórios de testes
- *Exercise*, invoca o método na classe que está sendo testada
- *Verify*, *assertions* sobre o resultado da etapa anterior
- *Teardown*, limpa os acessórios de testes

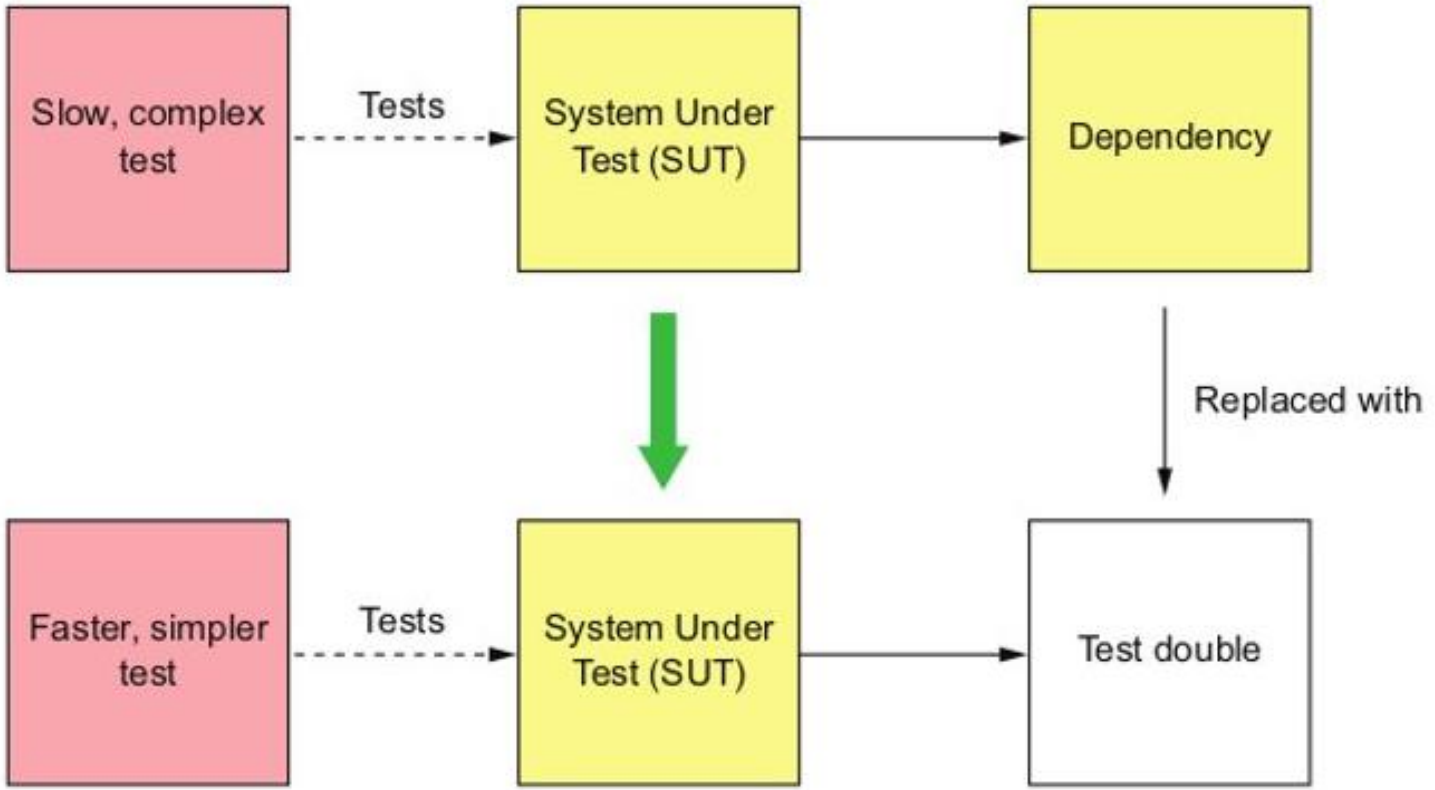
Testes automatizados



Mocks e Stubs

- Durante um teste automatizado, dependências podem ser necessárias
  - Banco de Dados
  - Apis
  - Etc
- Estas dependências devem ser carregadas de maneira que não corram riscos de funcionamento
  - Falha no banco, por exemplo
  - Serviço fora do ar
  - Custos de infra
  - ...

Mocks e Stubs



Mocks e Stubs

- Para resolver essa dependência pode-se utilizar um *TestDouble*
- https://martinfowler.com/bliki/TestDouble.html
- Um *Stub* é algo que **retorna valor** para o recurso que está sendo testado
- Um *mock* é algo que verifica se a **dependência é chamada corretamente**
- Um *mock* pode ser um tipo de *stub*

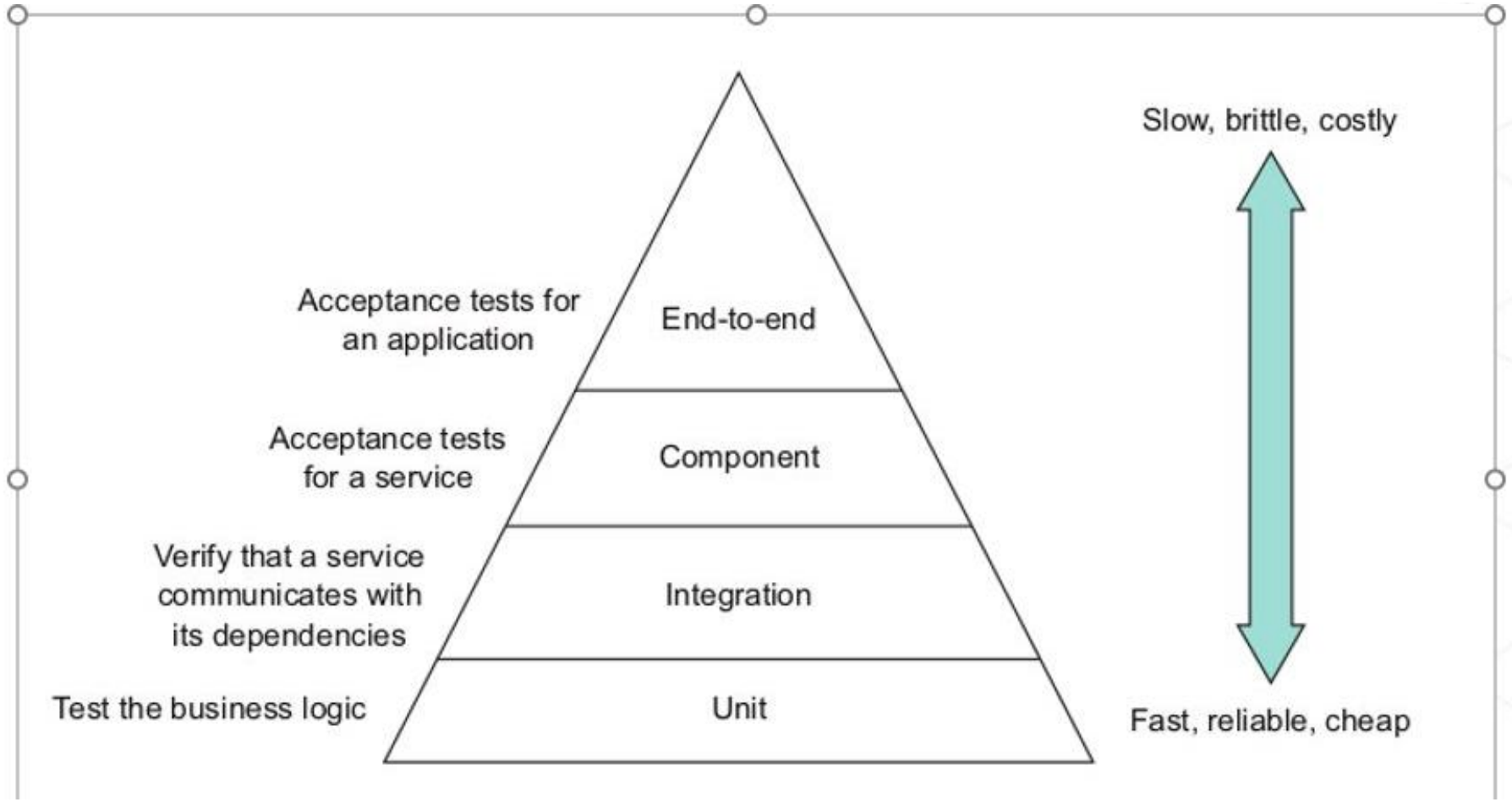
Tipos de Testes

- Testes Unitários
- Testes de Integração
- Testes de Componentes e aplicação
- *End-to-end*
- A grande diferença do tipo de testes é o escopo.

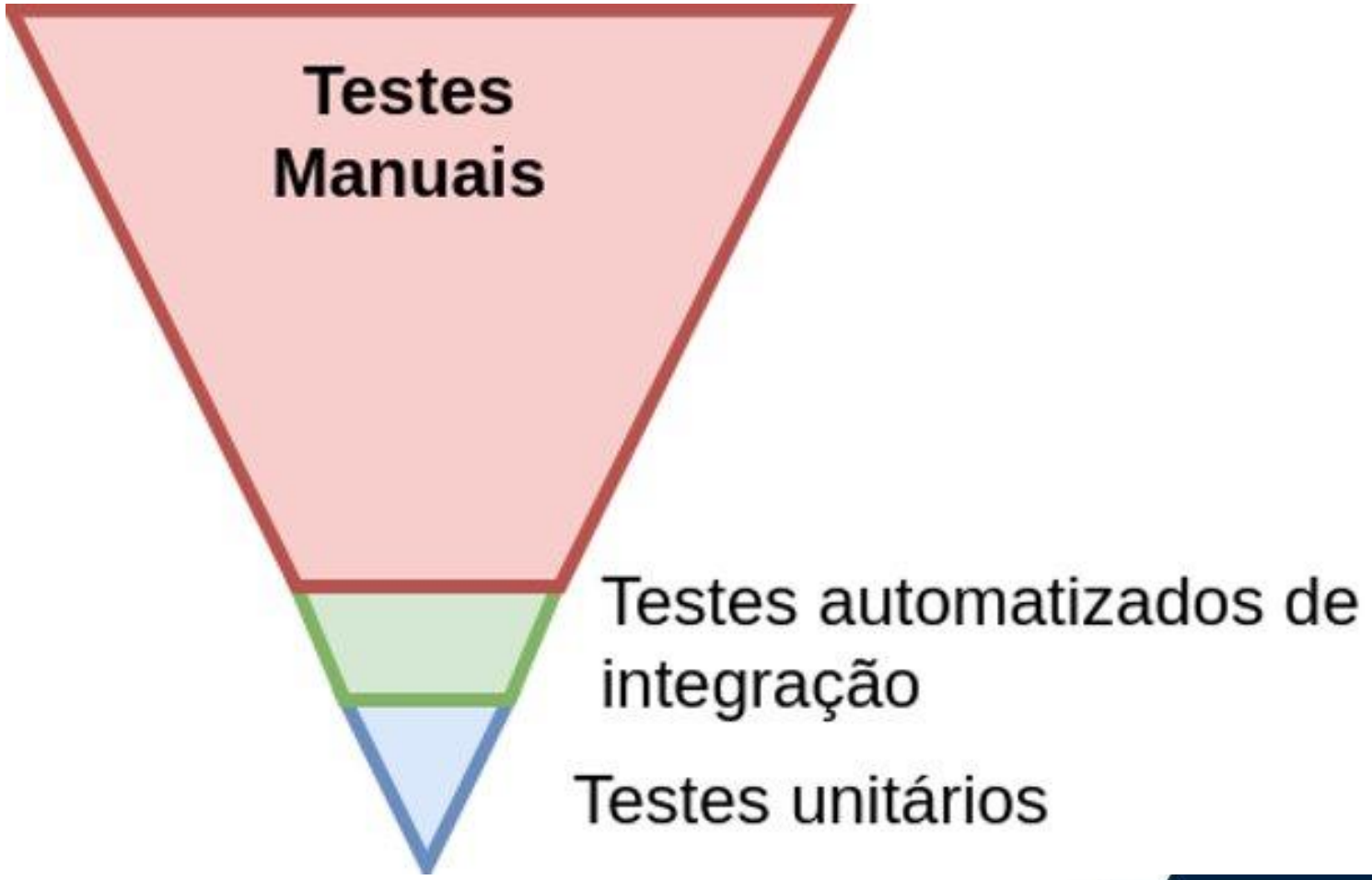
A pirâmide de testes

- https://martinfowler.com/bliki/TestPyramid.html
- Todos os diferentes tipos de testes são necessários para termos a confiança no funcionamento da aplicação
- Deve-se tomar cuidado com o esforço dispendido em cada tipo, conforme o nível de testes vai aumentando
- Testes unitários são muito mais fáceis de ajustar pelo próprio desenvolvedor
- Testes *end-to-end* tendem a ser devagar e difíceis de ajustar, devido sua complexidade

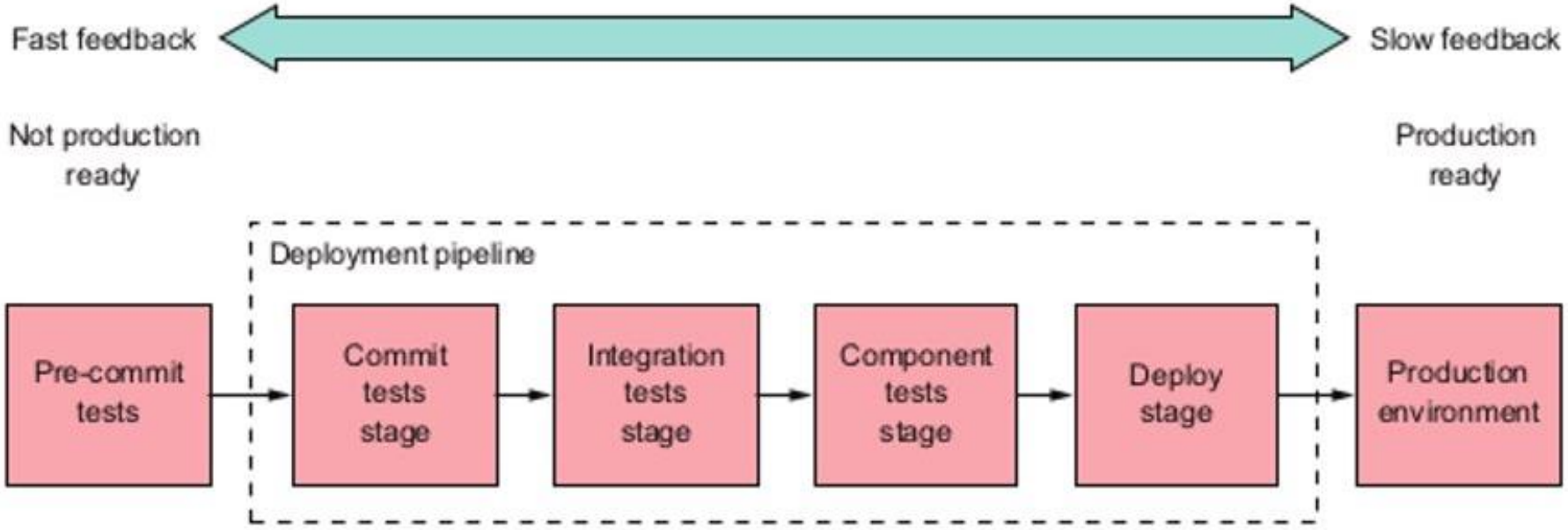
A pirâmide de testes



A pirâmide de testes tradicional



Pipeline de deploy



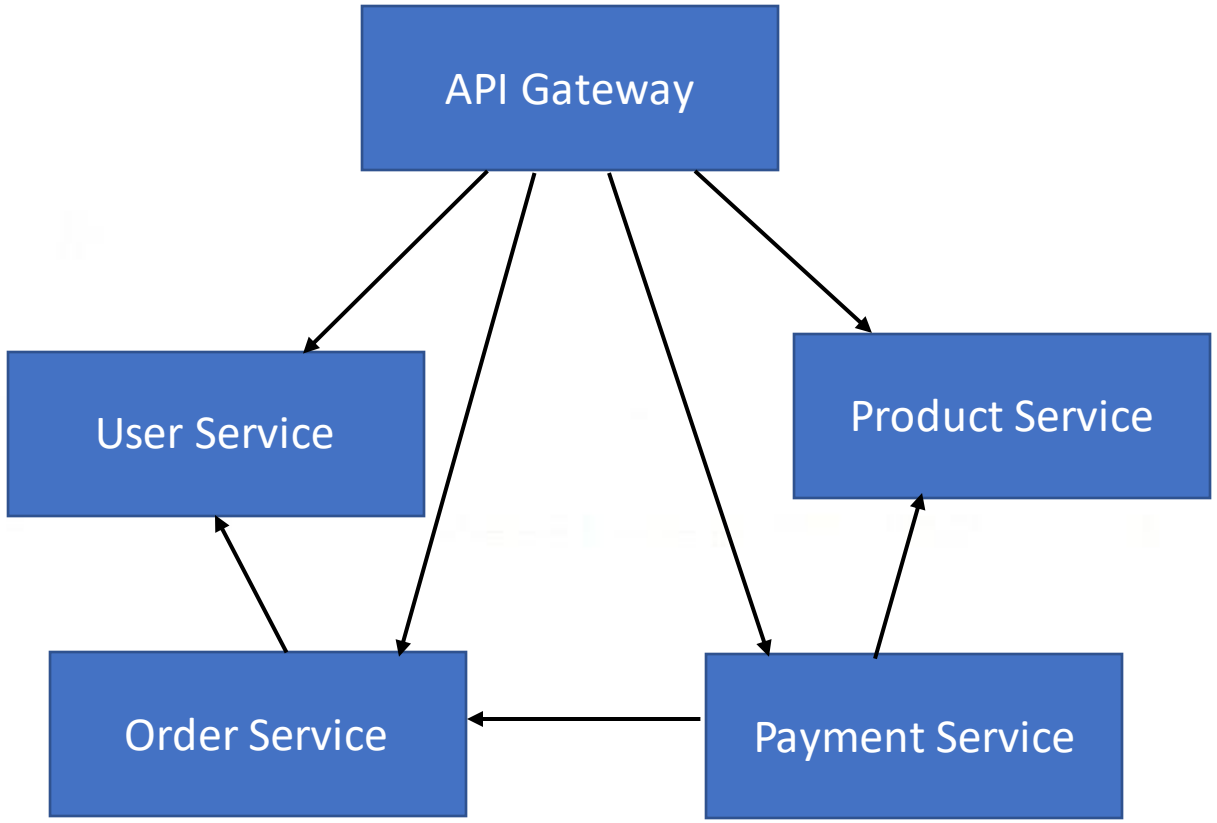
Testes em microsserviços

7COMm
Serviços e Soluções em TI

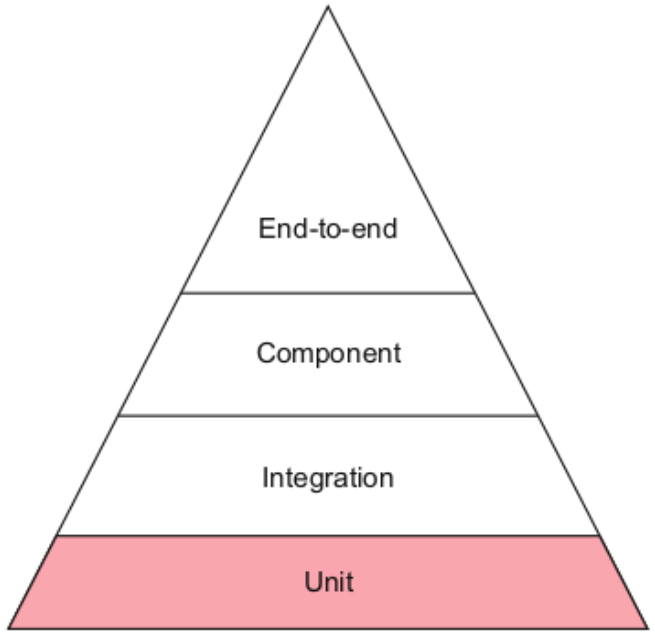
Testes em microsserviços

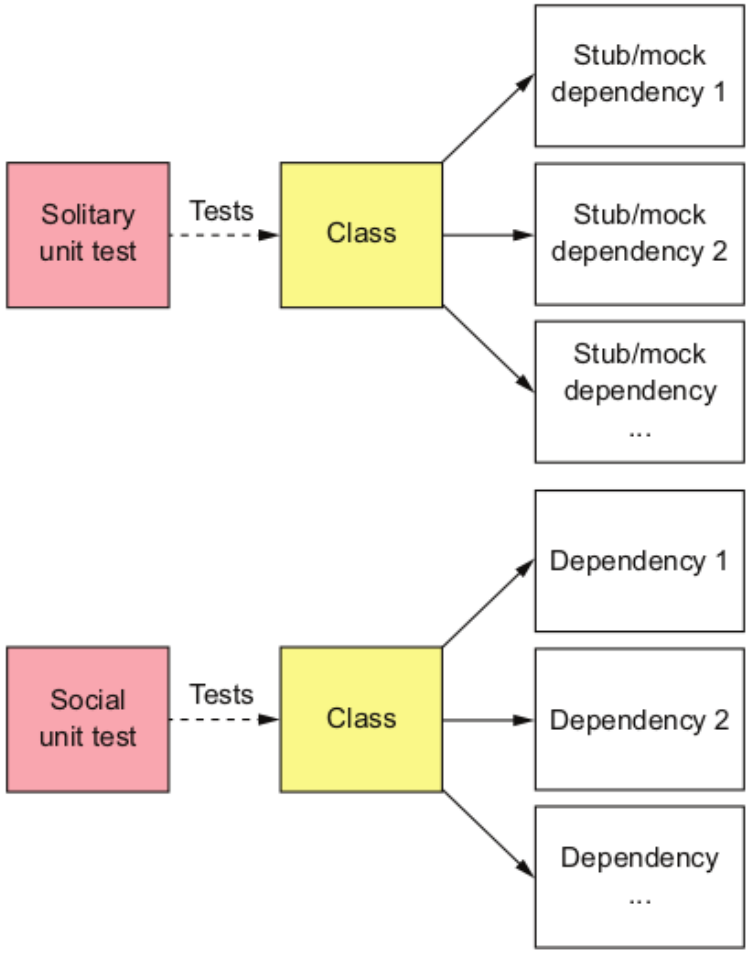
- Por definição, microsserviços são sistemas distribuídos
- Comunicação entre processos é essencial
- É essencial a escrita de testes de aceitação pelos times de desenvolvimento dos seus respectivos serviços
- Qual a melhor maneira de testar dois sistemas que interagem entre si? Subir os dois e um invocar o outro?

Testes em microsserviços



Testes Unitários





Testes Unitários

- Verifica o comportamento das menores unidades de implementação
- Devem funcionar isoladamente, pois devem ser rápidos
- Não devem utilizar nenhum processo ou sistema externo
- Garante que novas implementações não quebrem código já testado
- Documentação viva
- Quanto mais bem desenvolvida a aplicação, mais fácil de implementar os testes

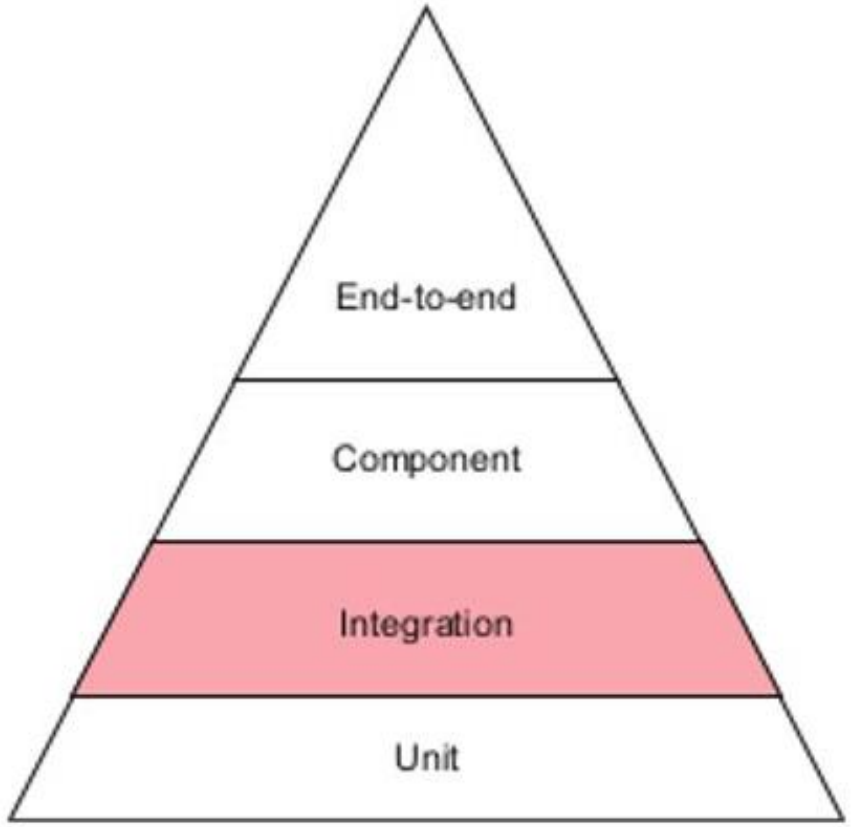
Testes Unitários

- Feitos pelos próprios desenvolvedores
- Devem ser rápidos e testar uma funcionalidade por vez
- Base para o funcionamento do TDD

Testes Unitários - Tipos

- Solitários
  - Testa a classe somente
  - Services, Controllers, Inbound/Outbound gateways
- Sociável
  - Testa as classes e suas dependências
    - Entities, VOs

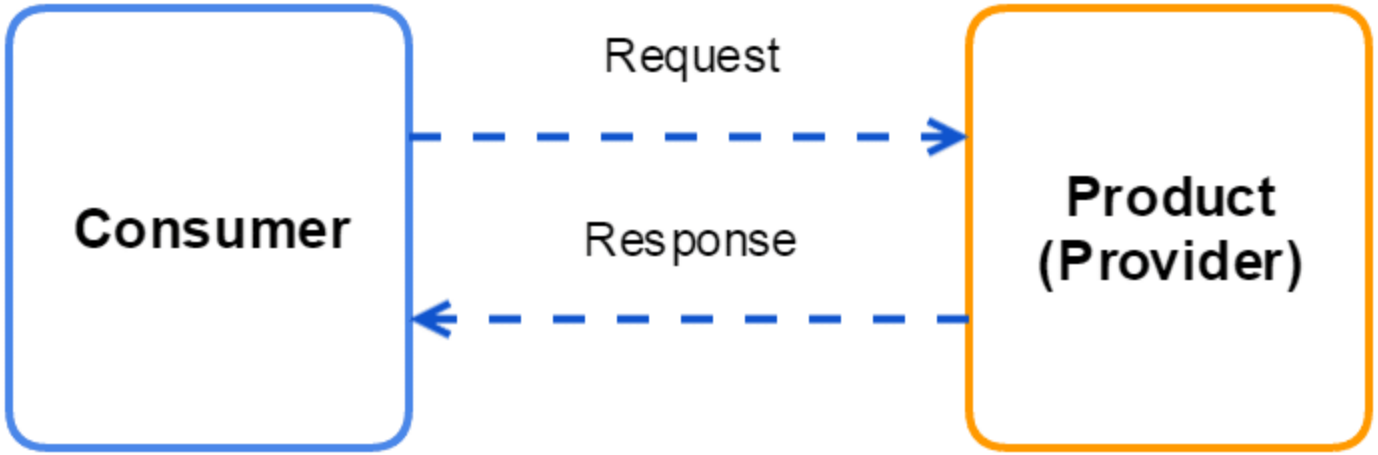
Testes de Integração



Testes de Integração

- Valida se os componentes de um sistema interagem corretamente
  - Bancos de dados
  - *Message Broker*
  - Outros serviços
- Pode-se utilizar mock

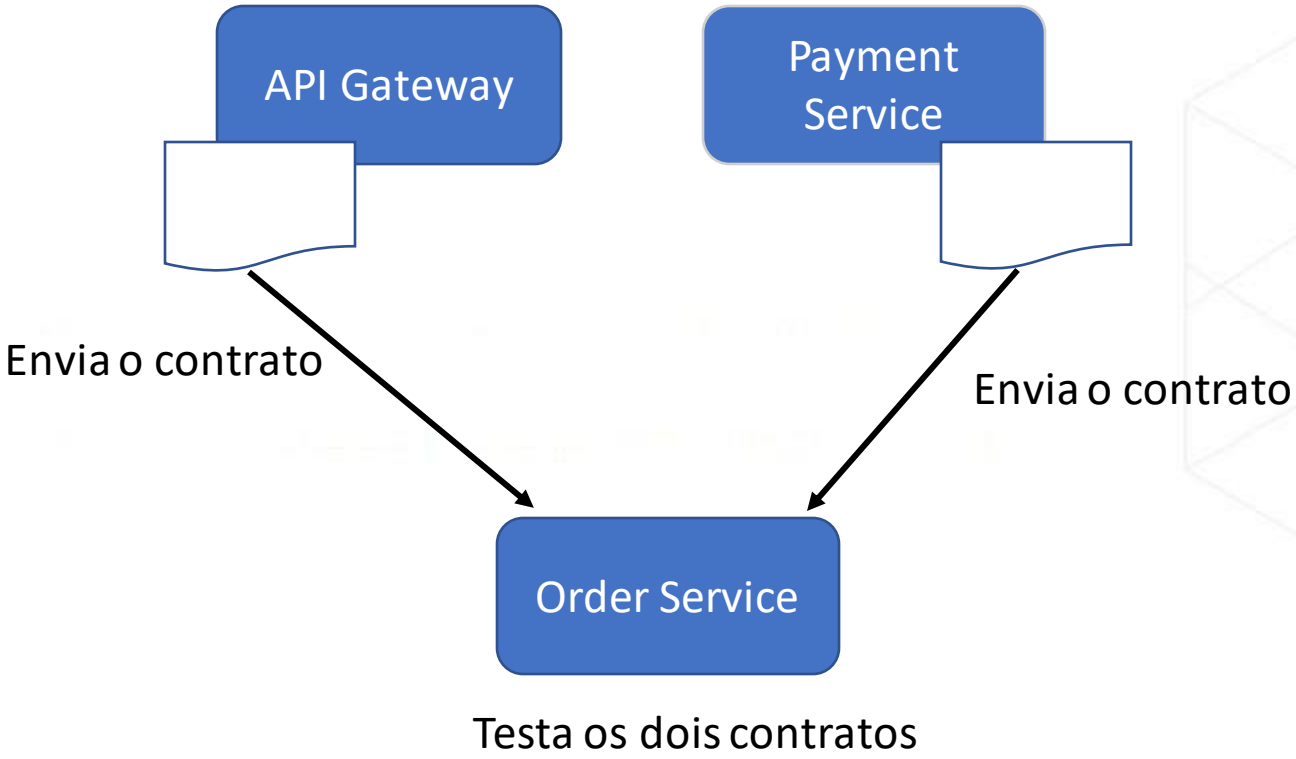
Testes em microsserviços



Consumer Driving Contract

- Estratégia de testes entre dois (ou mais) serviços
- *Clients* (ou *consumer*) escreves os contratos de testes e os enviam ao *provider*
- O *provider* inclui esses contratos em sua suíte de testes
- Caso os testes não passam, entende-se que houve uma *breaking change*
- http://microservices.io/patterns/testing/service-integration-contract-test.html

Consumer Driving Contract



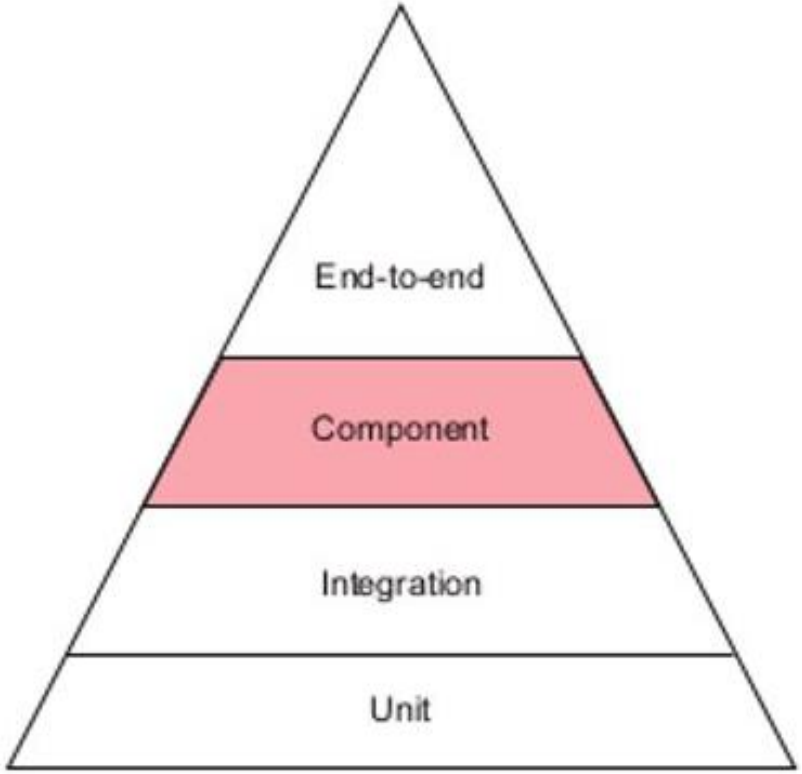
Spring Cloud Contract

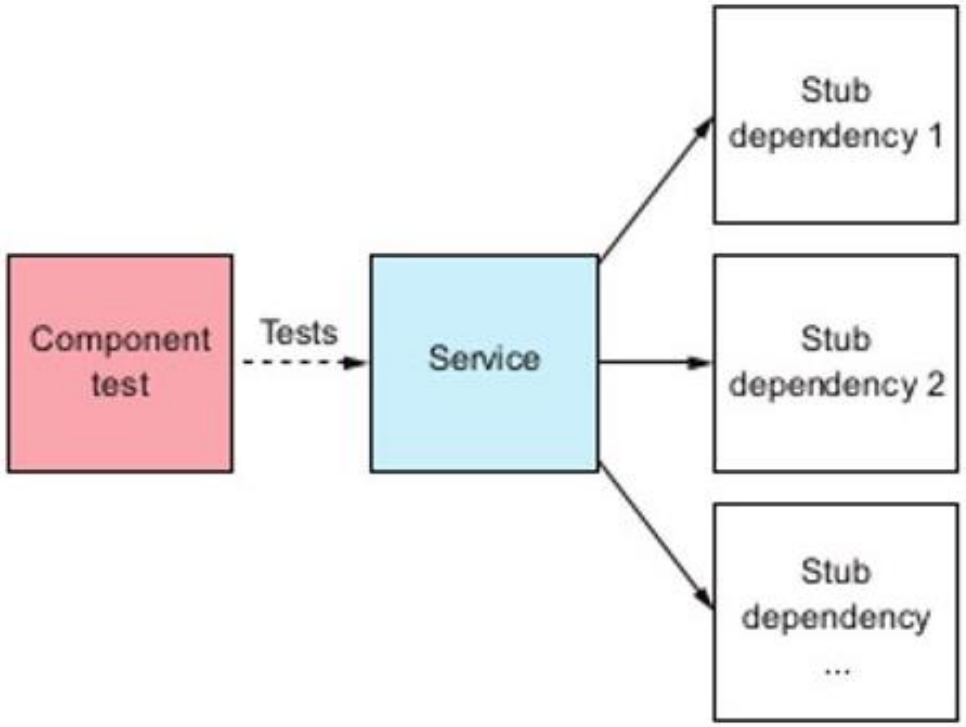
- Framework para implementação de Consumer Driven Contract
- Embarcado com Contract Definition Language escrito em Groovy ou YAML
- Feedback rápido
- Nenhum requisito de infra é necessário

Spring Cloud Contract

- Assegura que HTTP stubs são exatamente como em produção
- Promove testes de aceitação
- Provê mudanças na publicação dos contratis imediatamente visíveis em ambos os lados da comunicação
- Gera código de boilerplate usado no lado do servidor

Testes de Componentes





Testes de Componentes

- http://microservices.io/patterns/testing/service-component-test.html
- Verifica o funcionamento do serviço isolado, através da utilização de suas APIs
- Troca suas dependências por *stubs*, que simulam seus comportamentos

Testes de Componentes

- Descrevem o comportamento visível externamente da perspectiva dos clients
- São testes derivados dos requisitos ou de *user stories*
- Cada cenário de teste define um teste de aceitação
- Pode-se escrever testes de aceitação utilizando uma DSL (Domain Specific Languages)

Behaviour Driven Development (BDD)

- Refinamento de práticas de TDD mais aplicação de testes de aceitação
- Integra regras de negócios com a implementação
- Facilita a comunicação entre equipes de qualidade e desenvolvedores
- Artefatos de documentação podem ser gerados automaticamente
- Podem ser úteis para testar código-fonte já em produção

Behaviour Driven Development (BDD)

- Baseado em uma *user story* deve-se criar exemplos reais da nova funcionalidade
- Documenta-se esses exemplos de maneira que possam ser automatizados
- Implementa-se o comportamento descrito iniciando com um teste automatizado

BDD - Ferramentas

- Ghenkin
  - Um conjunto de regras de escrita em linguagem natural
  - Especificação de execução sem ambiguidade
  - Testes automatizados
- Cucumber
  - Lê os arquivos escritos através da DSL
  - Valida se o software faz exatamente o que é esperado

Cucumber + Ghenkin

- Um exemplo é chamado cenário
- Um cenário é definido em um arquivo *.feature*

```
Feature: Is it Friday yet?
  Everybody wants to know when it's Friday

  Scenario: Sunday isn't Friday
    Given today is Sunday
    When I ask whether it's Friday yet
    Then I should be told "Nope"
```

Testes end-to-end

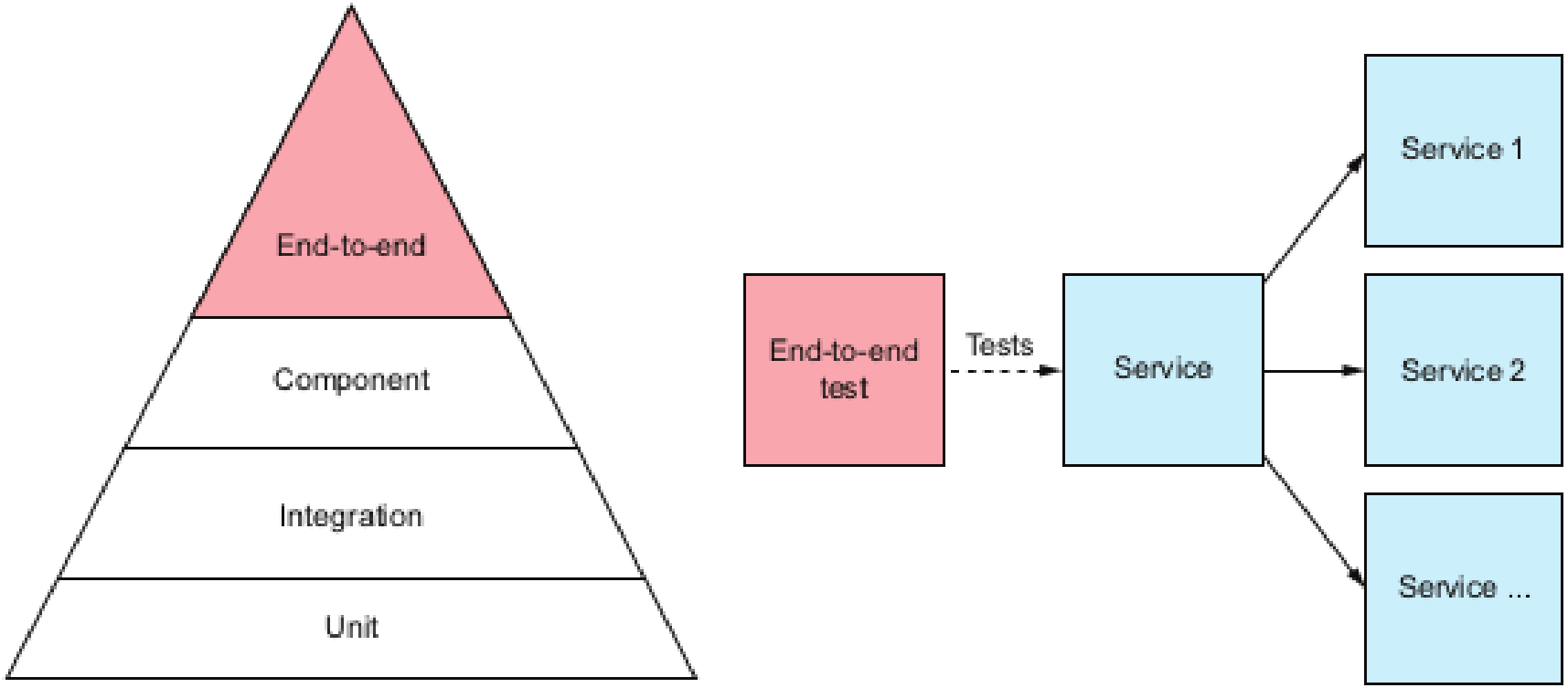


Figure 10.8 End-to-end tests are at the top of the test pyramid. They are slow, brittle, and time consuming to develop. You should minimize the number of end-to-end tests.

Testes end-to-end

- Testes menos performáticos em uma aplicação
- Deve ser feito o *deploy* de múltiplos serviços, além de toda infraestrutura responsável para suporte do serviço em questão
- O ideal é escrever o mínimo destes testes possíveis
- Deve-se pensar o teste *end-to-end* como testes de jornada de usuário, onde vários cenários são testados com somente uma implementação
- Pode-se utilizar BDD para sua implementação também

Testes end-to-end

Feature: Adicionar ao carrinho, revisar e cancelar

Como um consumidor de Order Service
Eu quero adicionar produtos no carrinho, revisar meu pedido e cancelar

Cenário: Carrinho criado, revisado e cancelado
Dado um consumidor valido
Dado um cartão de crédito válido
Dado um produto válido
Quanto eu coloco um produto no carrinho
Então, o carrinho deve ser CRIADO
Então, o valor do pedido deve ser de 10.0
E eu reviso o carrinho adicionando dois outros produtos
Então o valor do pedido deve ser 30.0
E quando eu cancelo o carrinho
O carrinho deve ser CANCELADO

Testes end-to-end

- Cada cenário deve testar o maior número de funcionalidades possíveis
- Pode ser desenvolvido pela equipe de qualidade
- É o mais custoso da pirâmide de testes

Para saber mais

- https://en.wikipedia.org/wiki/Test_case
- http://xunitpatterns.com/Four%20Phase%20Test.html
- https://martinfowler.com/bliki/TestDouble.html
- https://martinfowler.com/bliki/TestPyramid.html
- https://junit.org/
- https://howtodoinjava.com/junit5/junit-5-vs-junit-4/
- http://microservices.io/patterns/testing/service-integration-contract-test.html
- http://microservices.io/patterns/testing/service-component-test.html

Próximos passos

- Microsserviços em produção

OBRIGADO!

Centro
Rua Formosa, 367 - 29º andar Centro, São Paulo - SP, 01049-000

Alphaville
Avenida Ipanema, 165 - Conj. 113/114 Alphaville, São Paulo - SP,06472-002

+55 (11) 3358-7700
contact@7comm.com.br

7COMm
Serviços e Soluções em TI