Microservices Architecture Workshop

Arquitetura de Referência e Plano de Modernização - Migração de ESB para MSA

Claudio Acquaviva

"All problems in Computer Science can be solved by another level of indirection, except for the problem of too many layers of indirection".

David J. Wheeler, Computer Scientist

Inventor of the "Closed Subroutine", 1927-2004.

As informações são públicas.

3 principais tarefas

- Qualificar conteúdo.
- Estruturar conteúdo.
- Aplicar conteúdo.

Contexto:

“Migração” de
Infraestrutura baseada em
ESB – Enterprise Service Bus
para
MSA – Microservice Architecture
(em Cloud Computing)

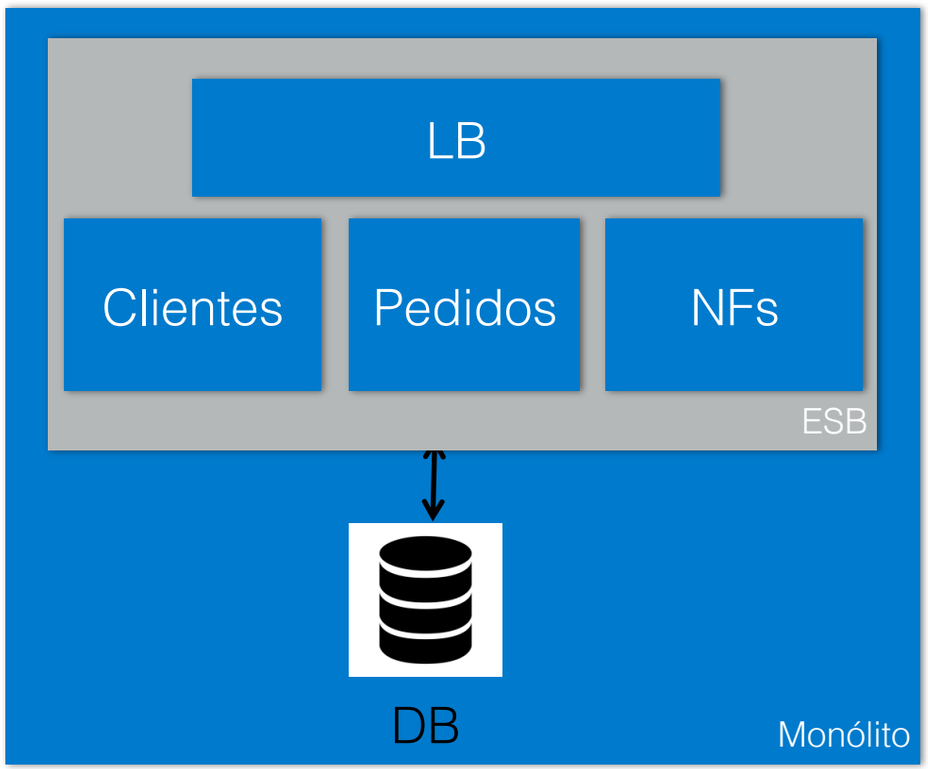
Service Orientation – SOA e MSA

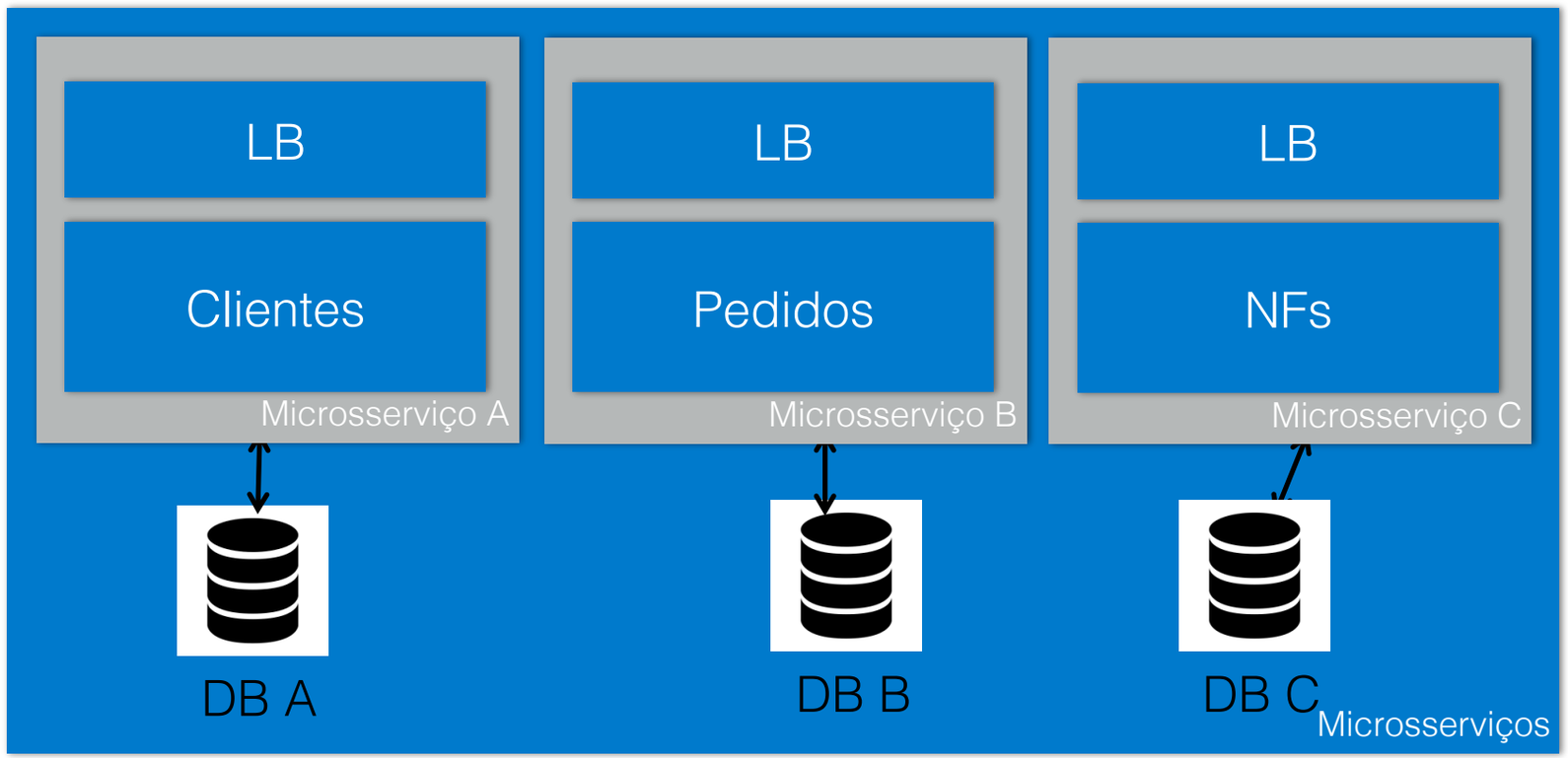
Em resumo, a Orientação a Serviços é benéfica. A sua implementação em ESB não é uma boa solução.

Em 2005, Anne Thomas Manes, VP do Gartner escreveu o famoso paper “SOA is Dead; Long Live Services” (http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html).

MSA por Adrian Cockcroft, VP da AWS - Amazon Web Services: “Service-oriented architecture composed of loosely coupled elements that have bounded contexts”.

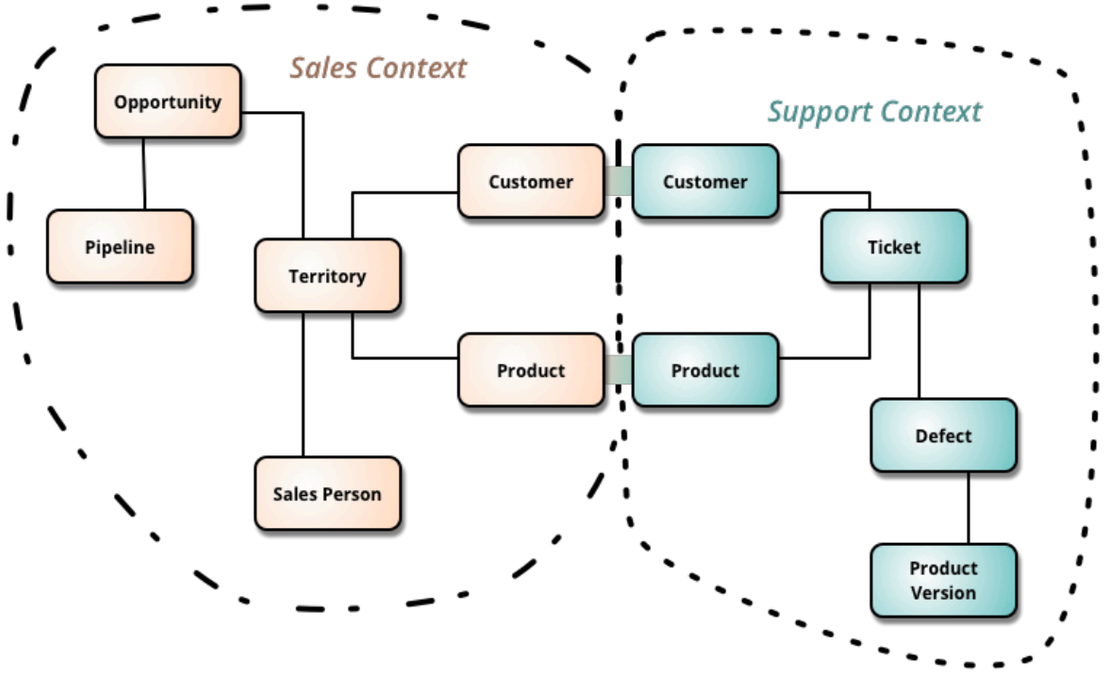
Mesmos princípios, implementações distintas. Monólitos -> Microsserviços.

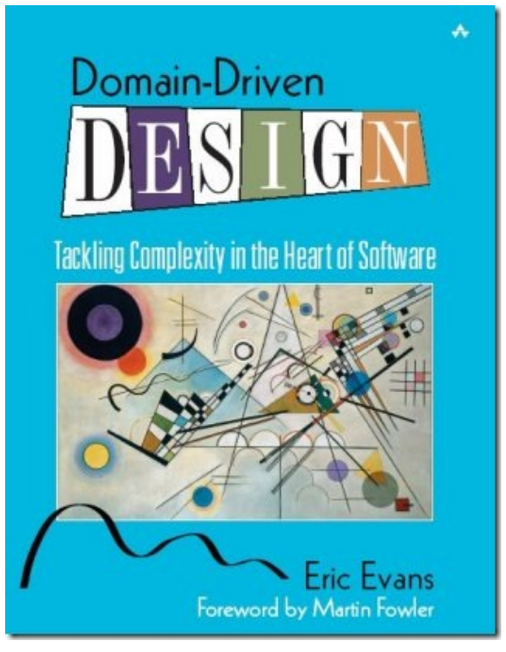




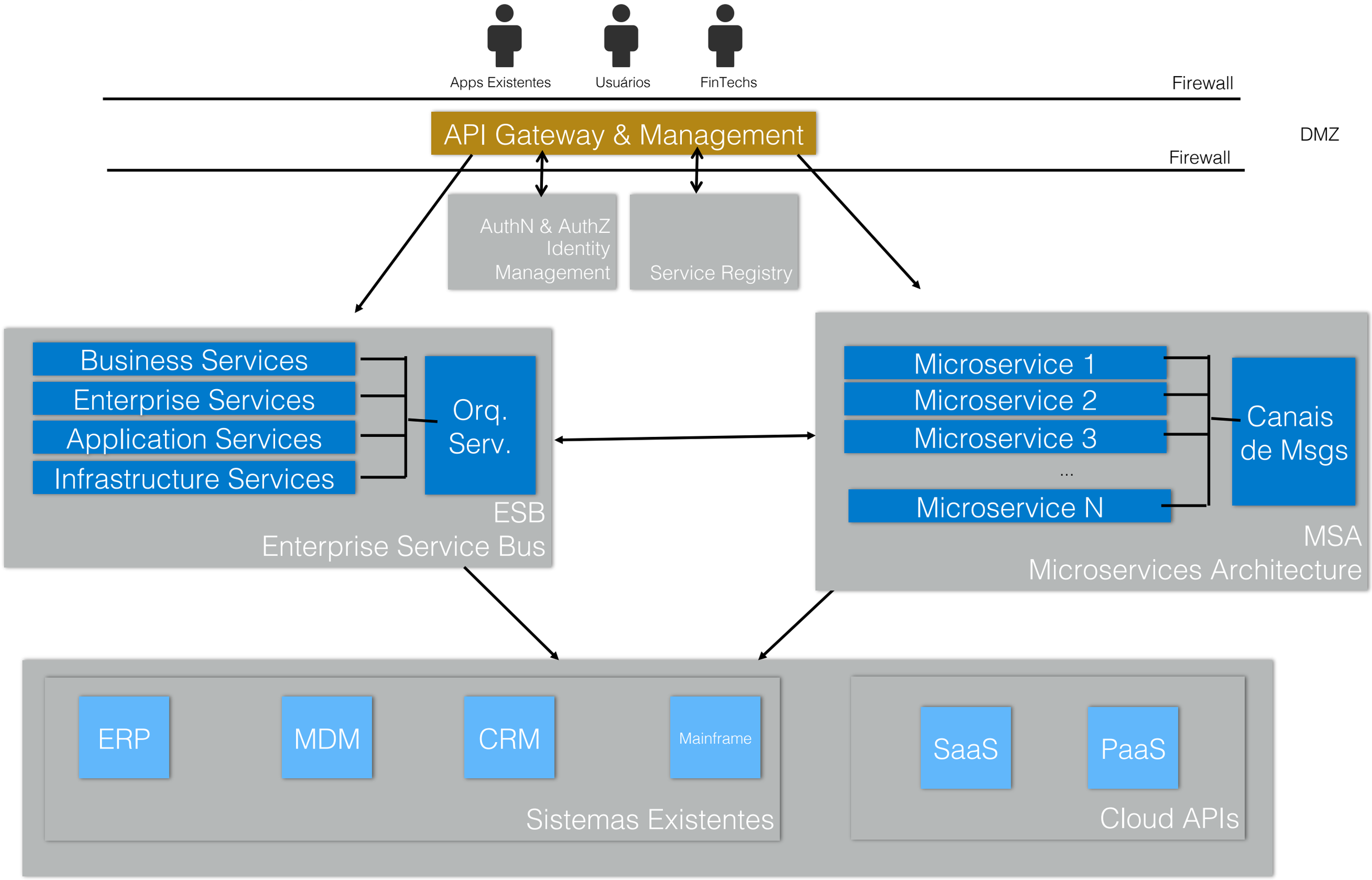
DDD – “Domain-Driven Design”

- Não há como construir um modelo de domínio unificado para todos os sistemas.
- Divisão de um sistema complexo em “Bounded Contexts”.
- Cada “Context” possui o seu modelo unificado e define a relação entre os outros contextos.
- Um contexto é um implementado por um Microsserviços (ou um conjunto de Microsserviços).

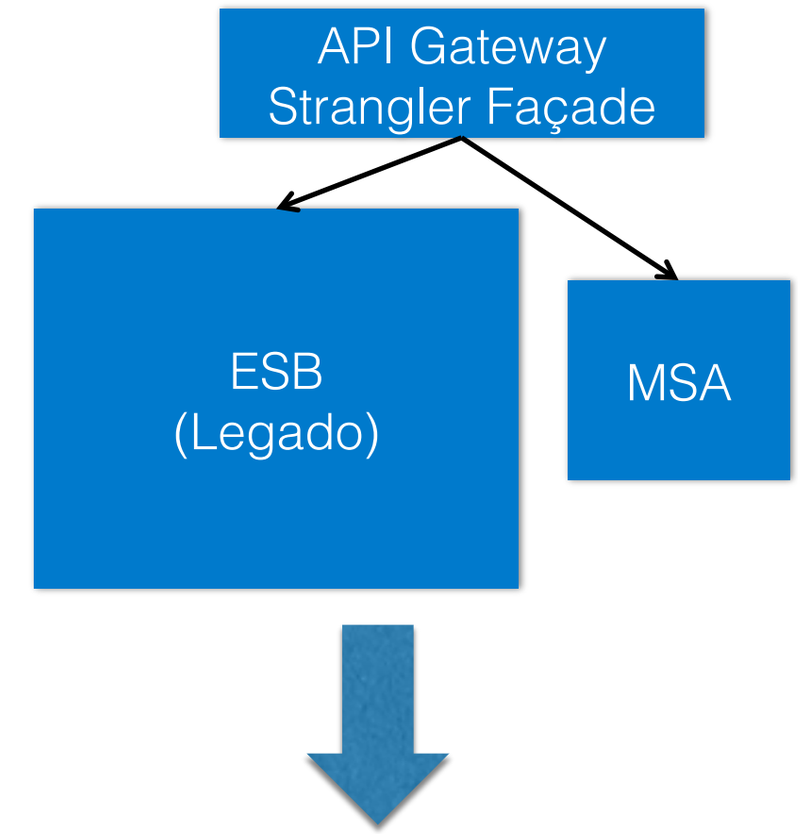




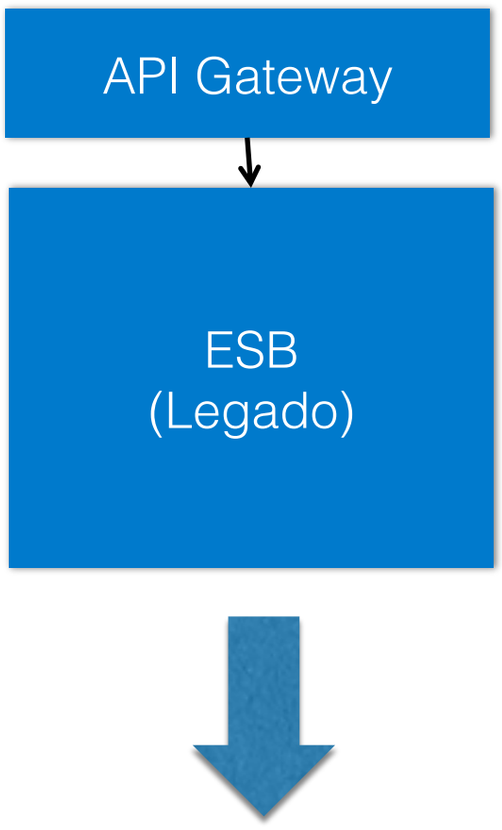
Arquitetura de Referência – ESB e MSA



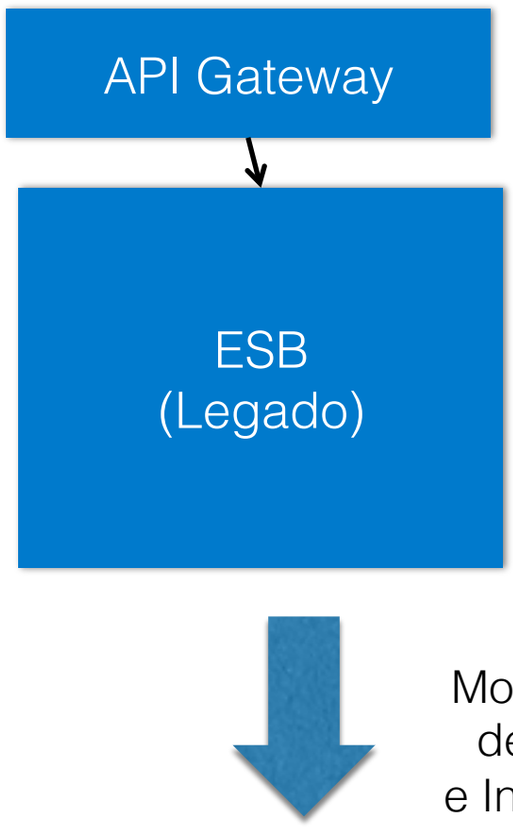
Estratégias para Migração



Strangler

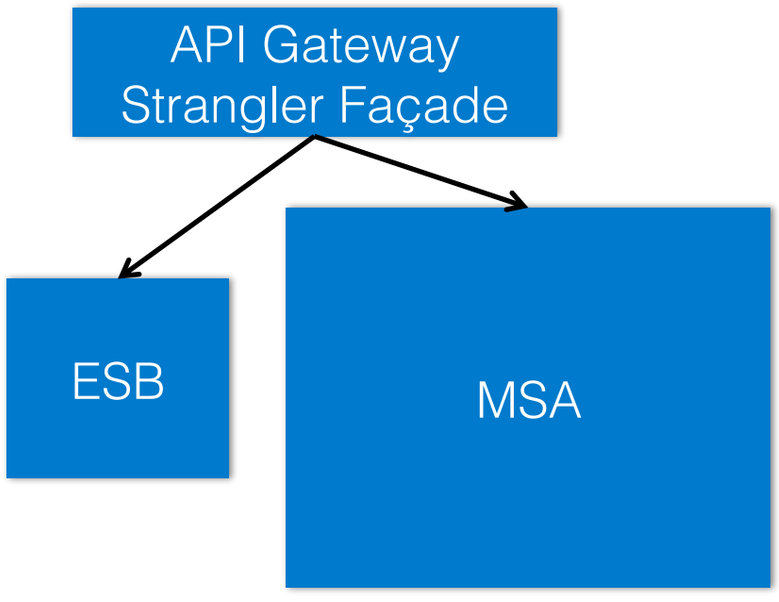


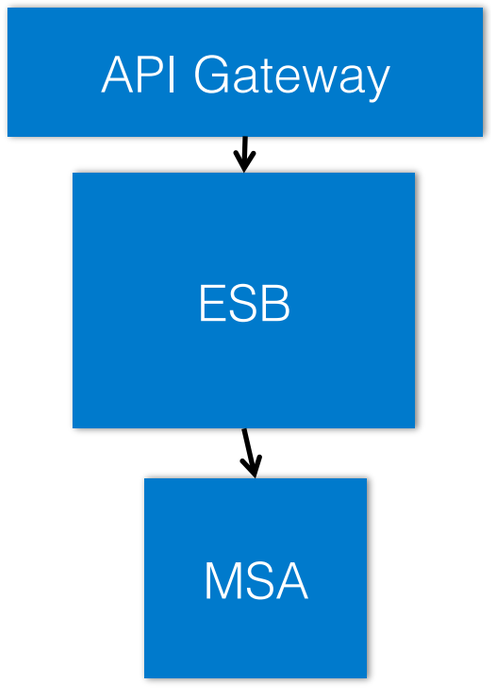
Spliting Services

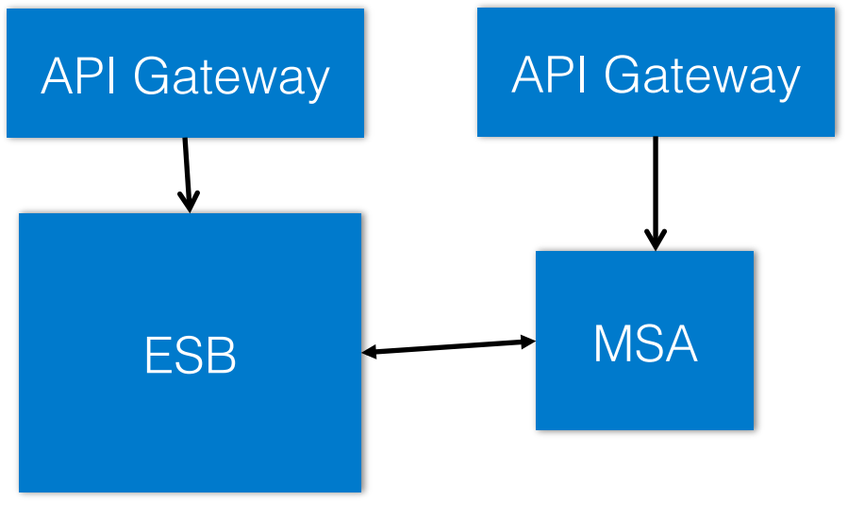


Extracting Services

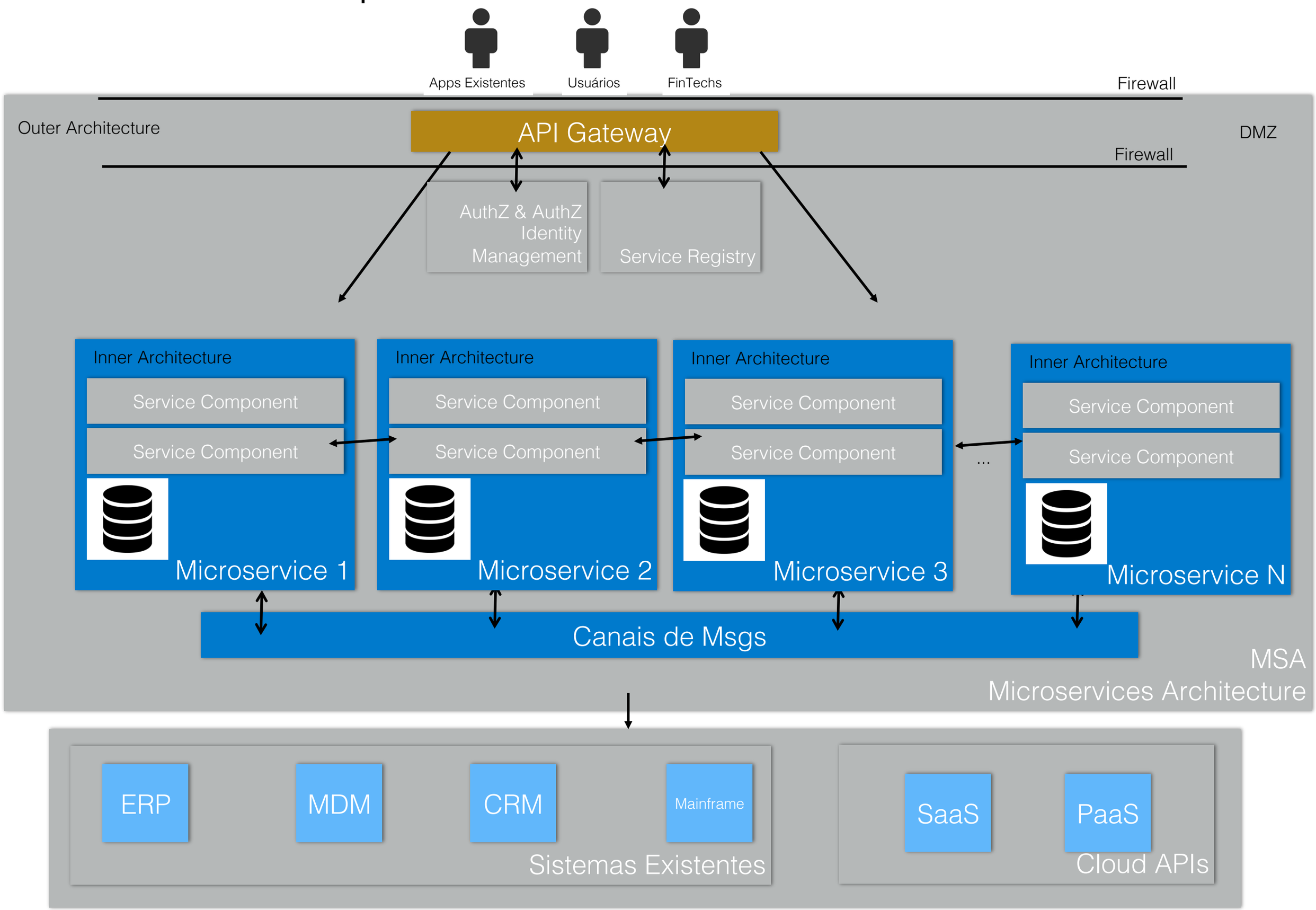
Modernização
de Serviços
e Infraestrutura





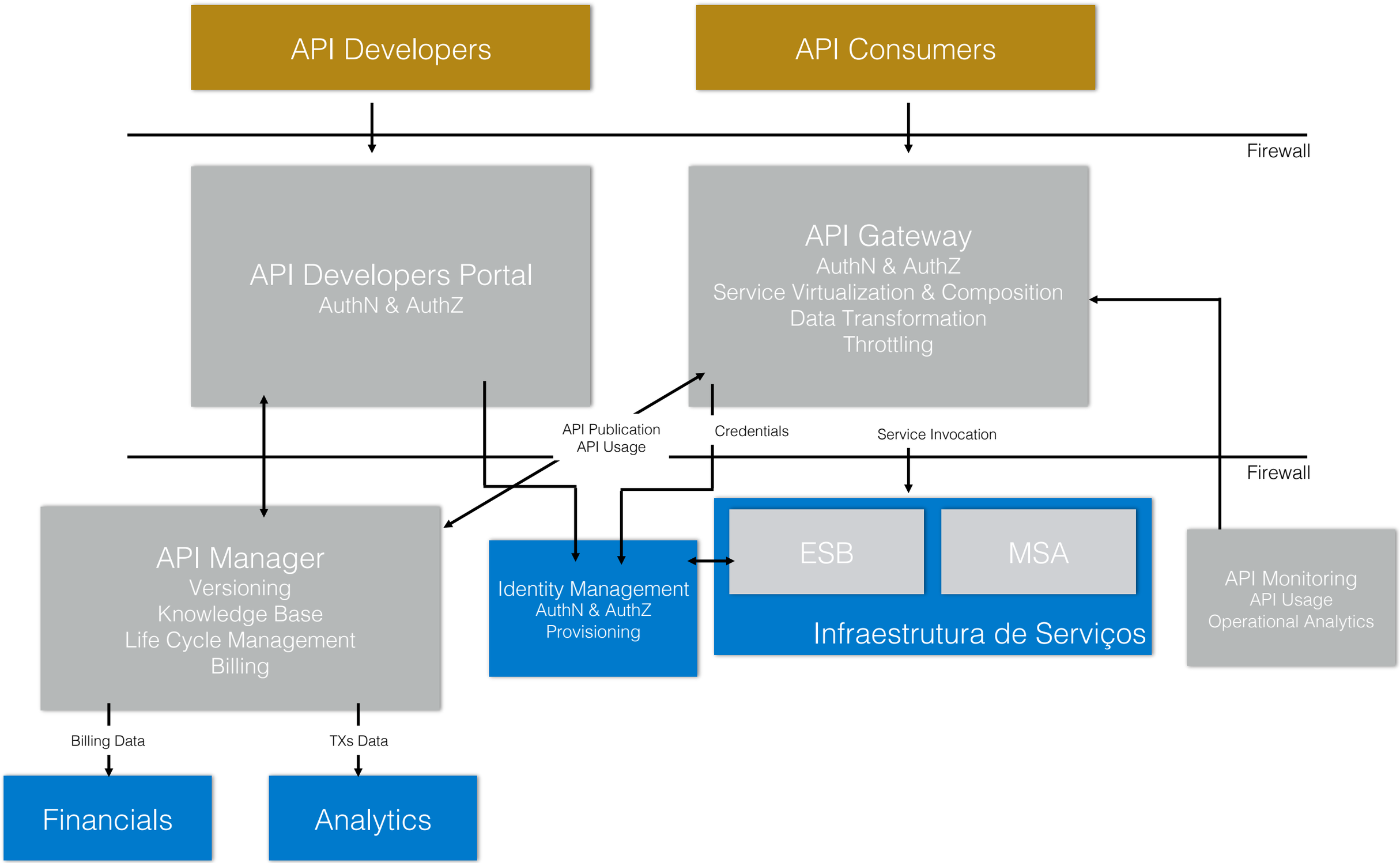


Arquitetura de Referência - MSA

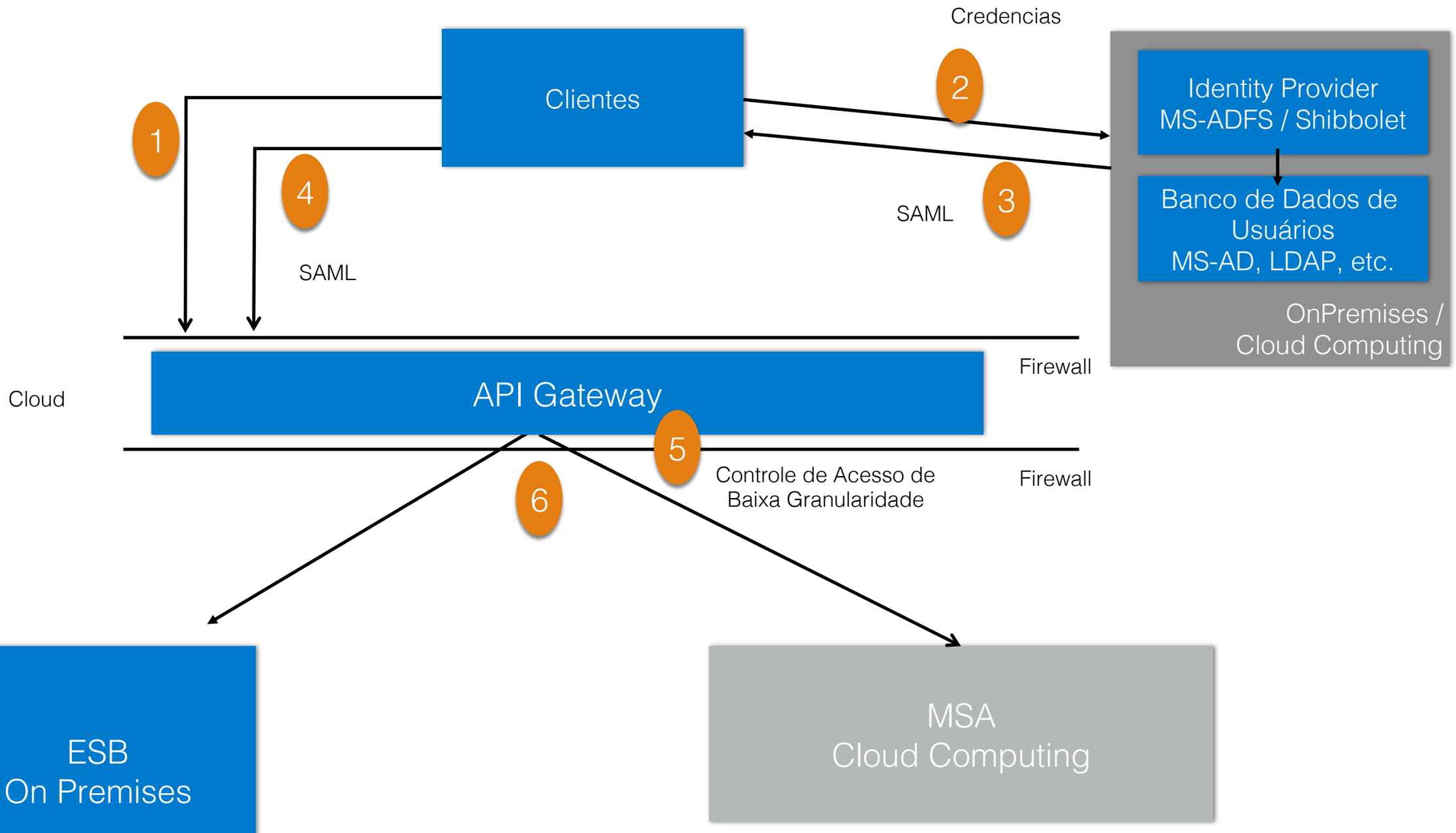


A Guidance Framework for Architecting Portable Cloud and Multicloud Applications, Gartner
Eric Knipp, Traverse Clayton, Richard Watson, Gartner, 16/12/16

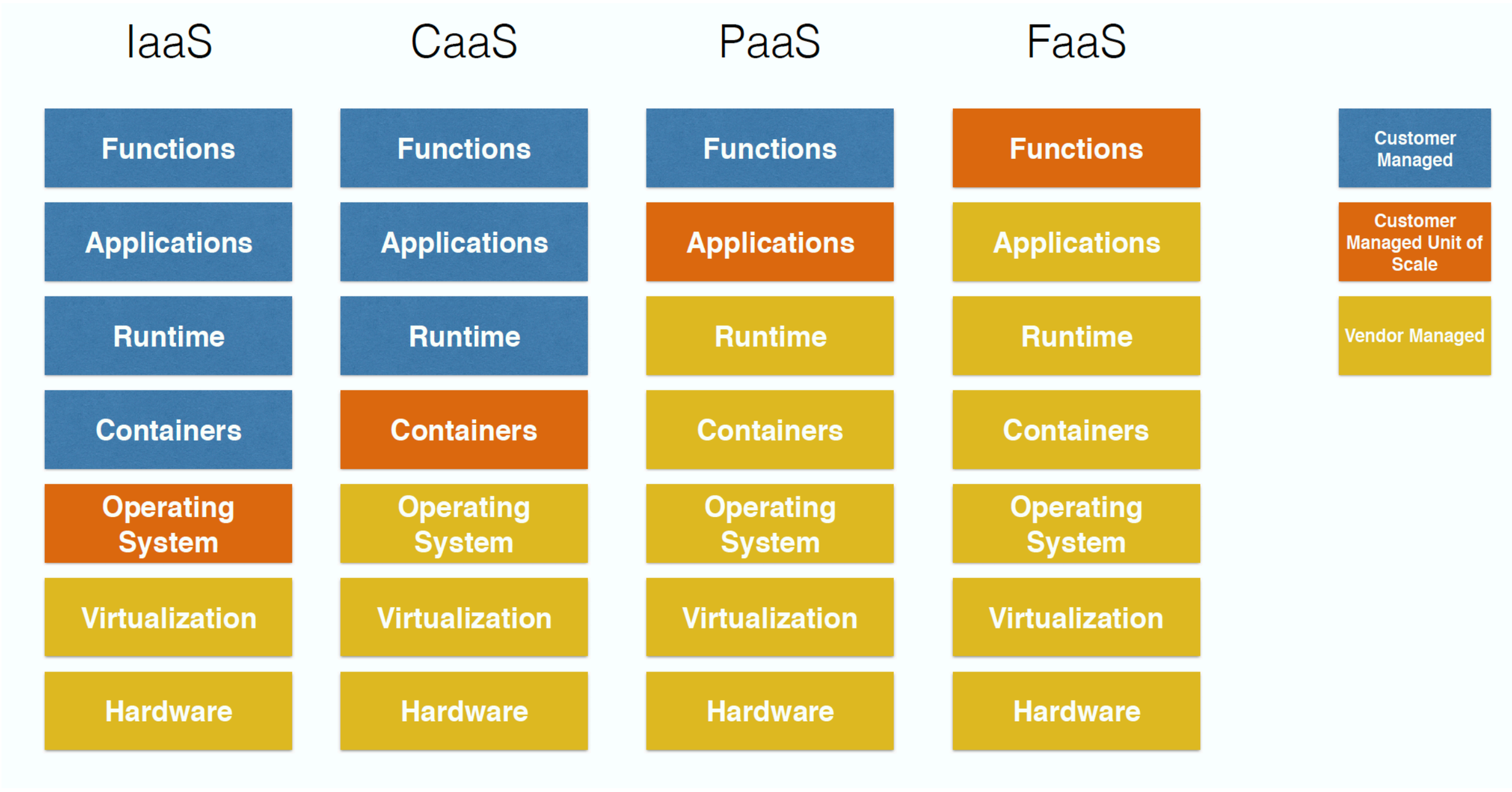
API Management - Arquitetura de Referência



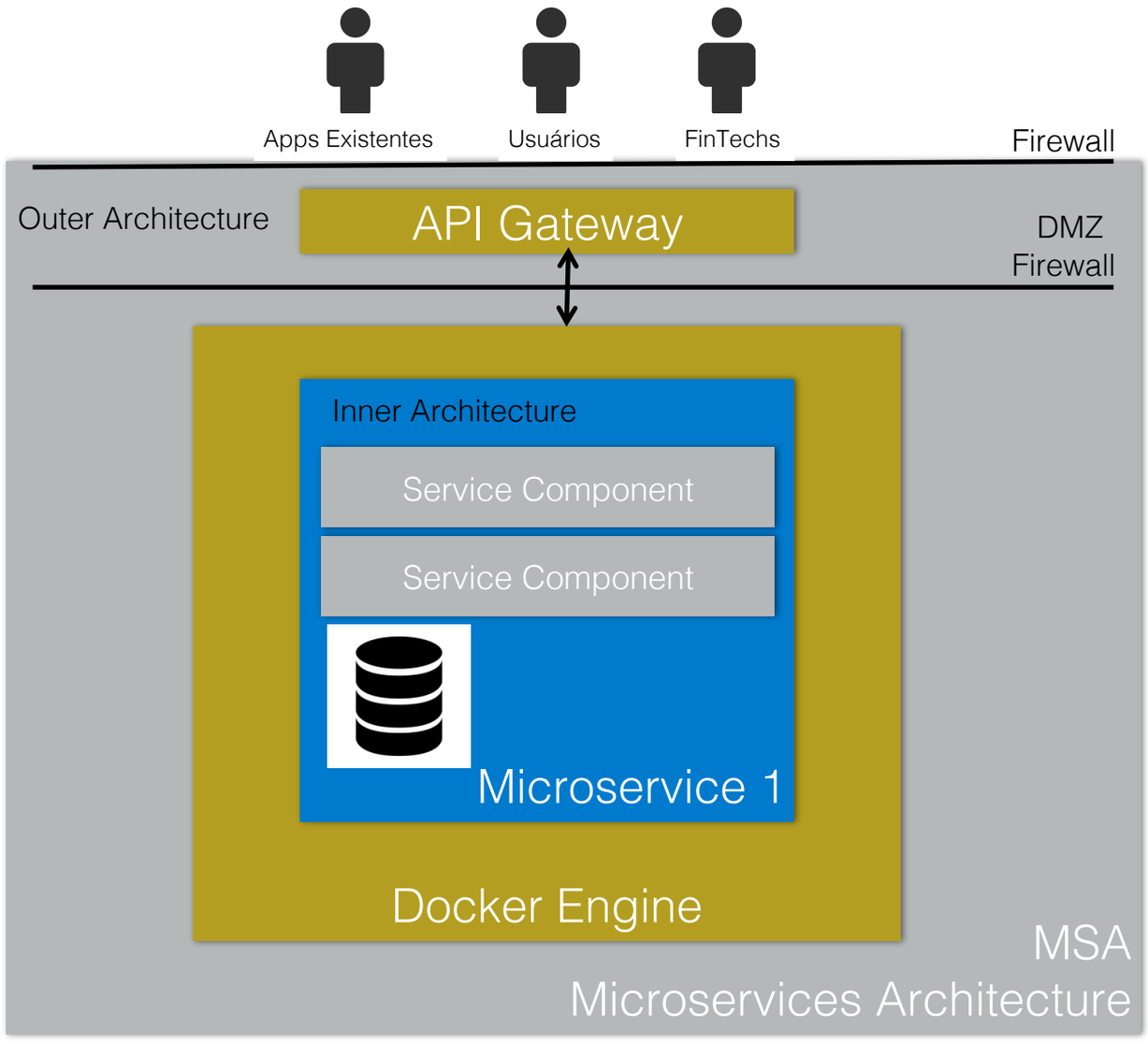
Federação – ESB e MSA



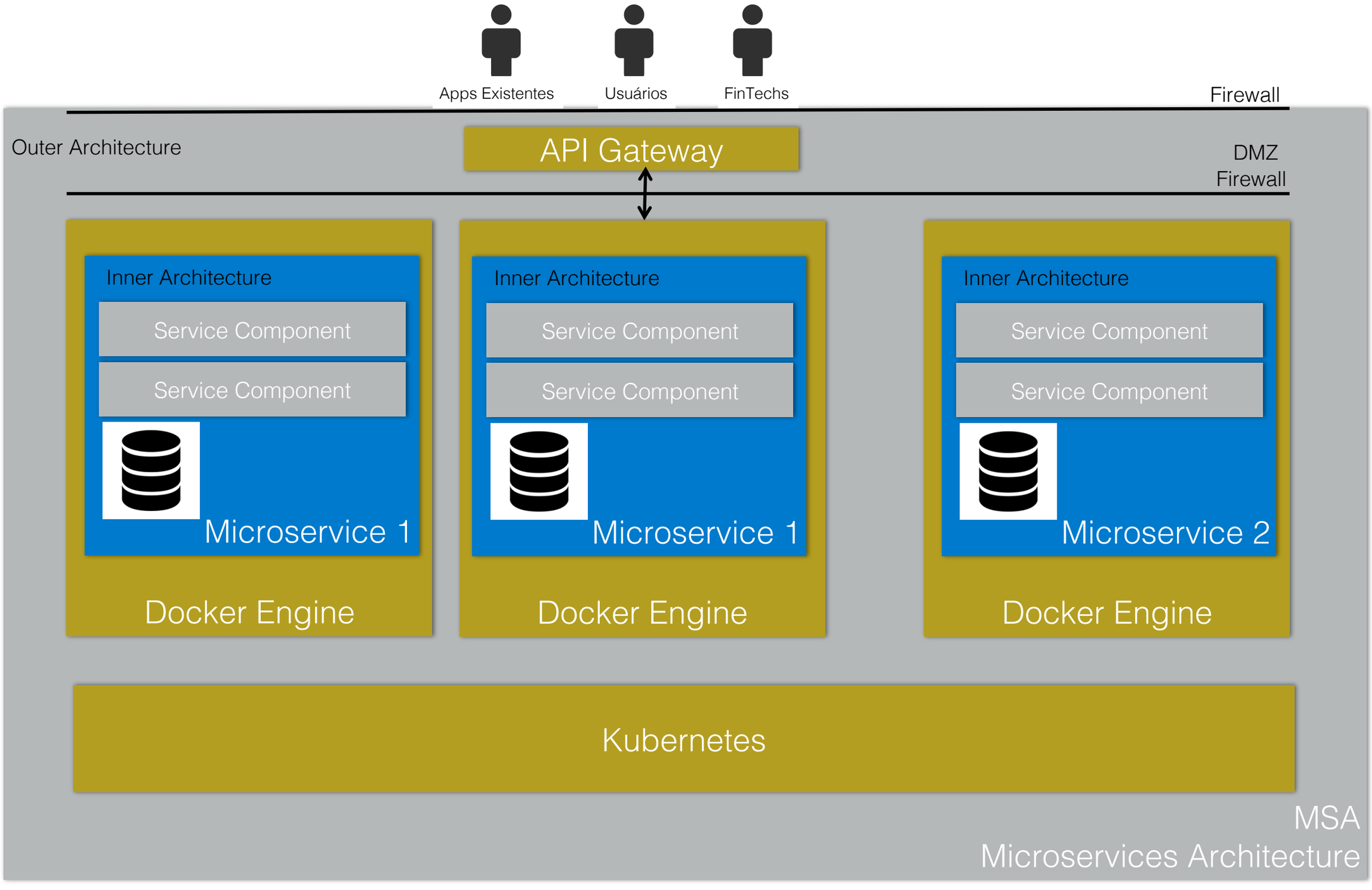
Compute Services



CaaS – Container as a Service - Docker



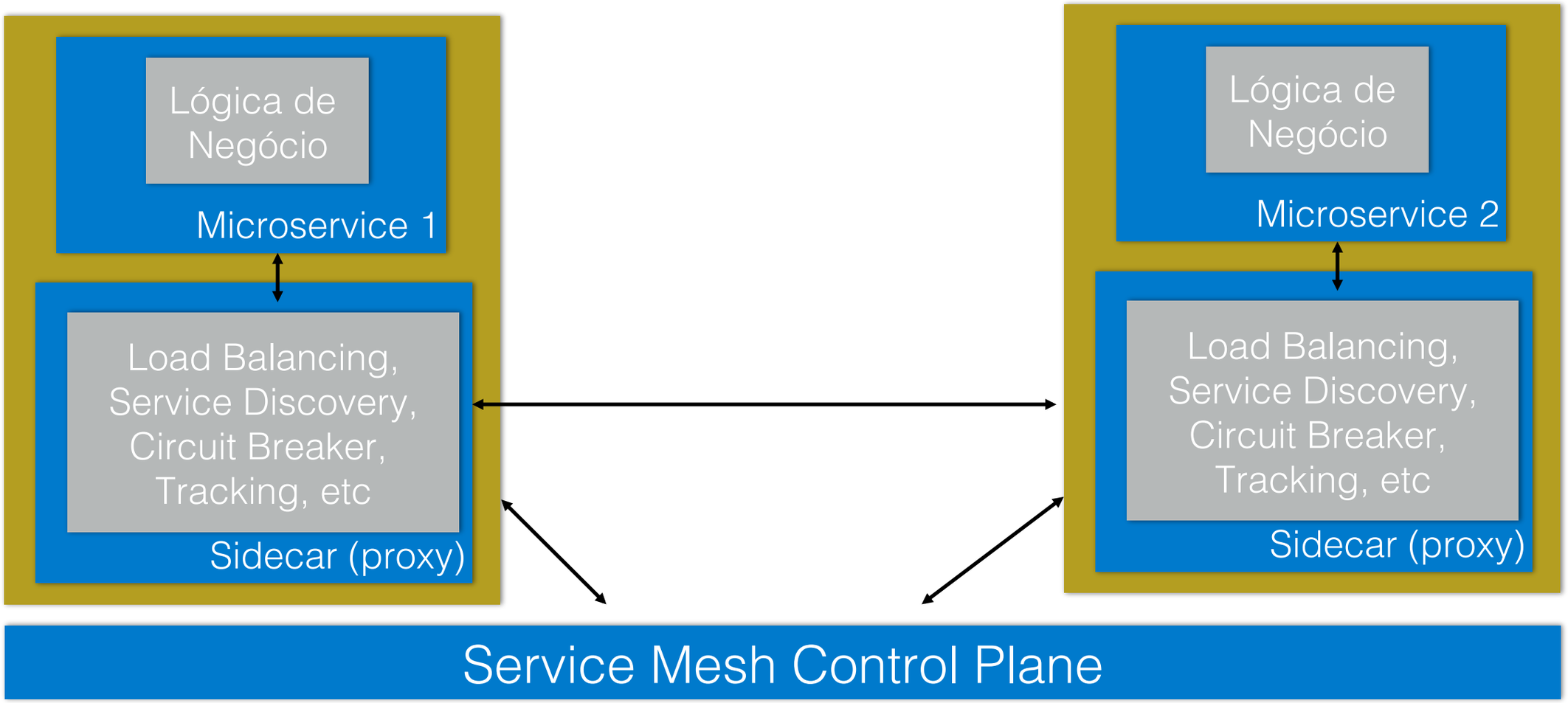
CaaS – Container as a Service - Kubernetes



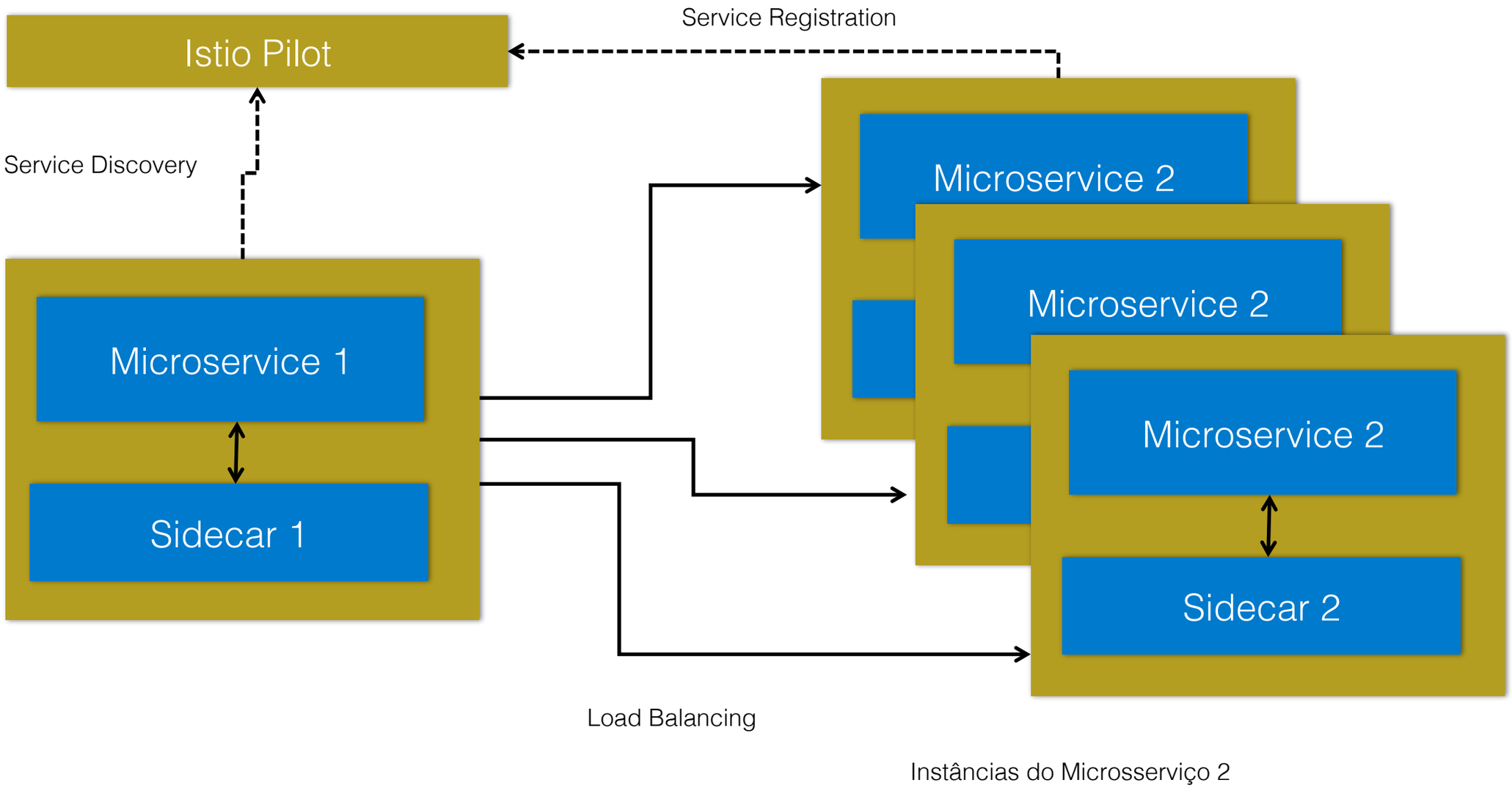
Service Mesh Pattern



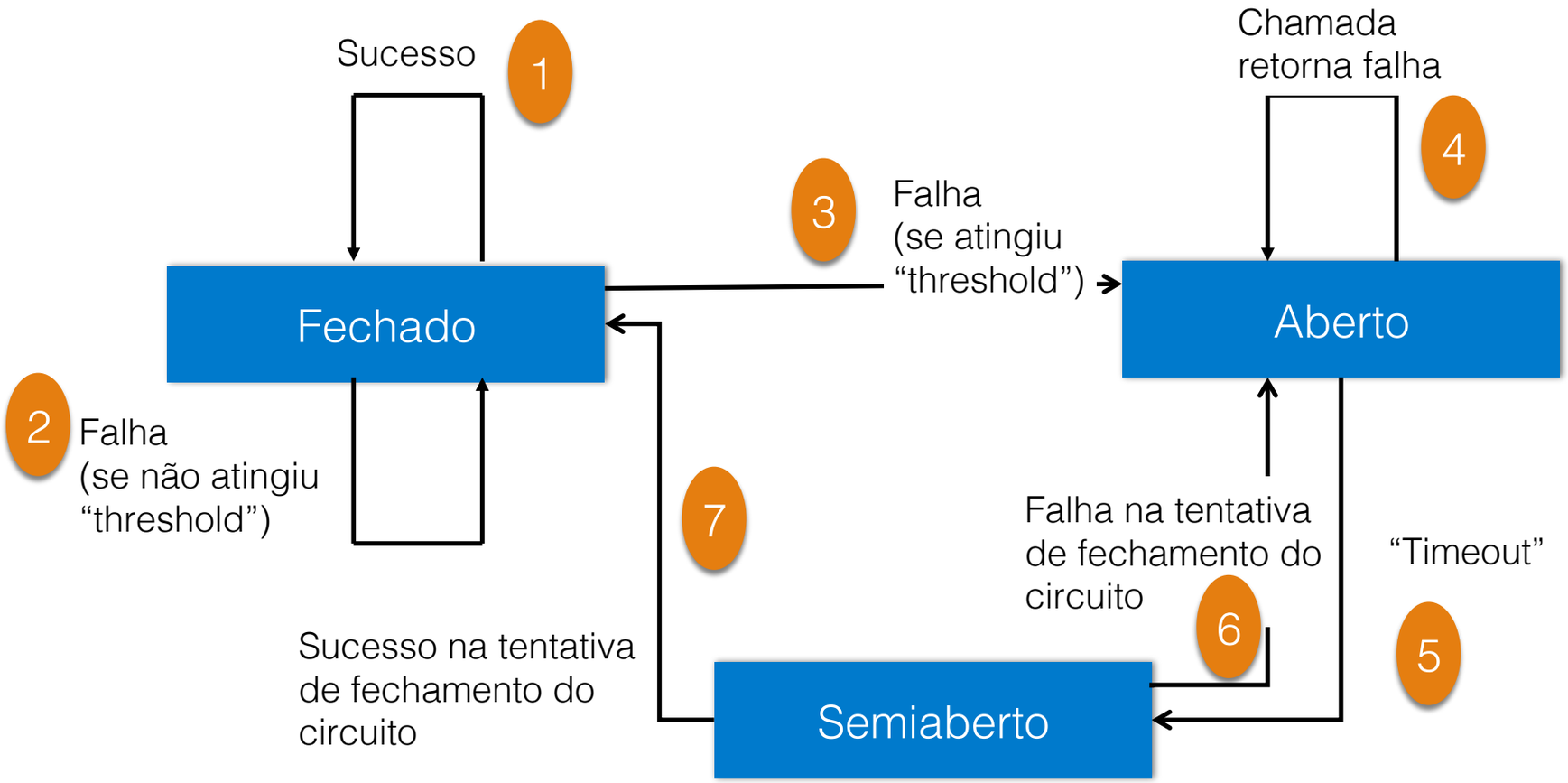
- Service Mesh é uma camada de infraestrutura dedicada para tratar de necessidades comuns dos Microsserviços. Implementado como uma coleção de Sidecars (ou “proxies”) padronizados, que isolam estas questões da Lógica de Negócio.
- O Service Mesh Control Plane é responsável pelo gerenciamento dos Sidecars.
- Algumas implementações de Service Mesh estão disponíveis como o Istio (http://www.istio.io) e o Linkerd (http://www.linkerd.io).
- O Envoy (https://www.envoyproxy.io) é uma dos “frameworks” usados para a implementação dos Sidecars.



Service Mesh – Load Balancing & Service Discovery

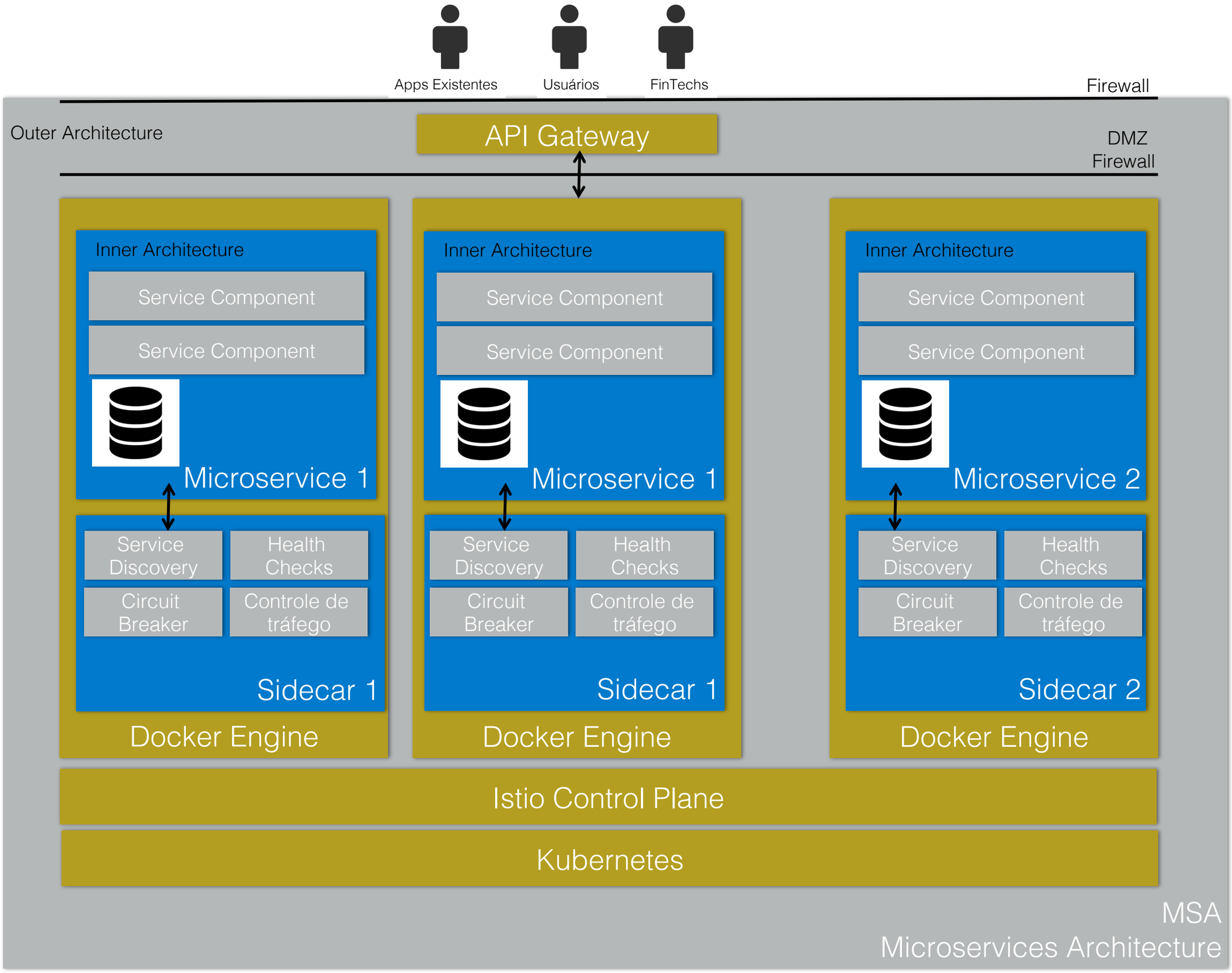


Service Mesh – Circuit Breaker



1. Regime normal de operação do “Circuit Breaker”. O Microsserviço invocado não apresenta problemas.
2. A invocação de um Microsserviço apresentou uma falha (erro na invocação, “timeout”, etc). “Threshold” não foi atingido. O serviço invocador recebe um código de resposta pertinente à situação.
3. A invocação de um Microsserviço apresentou falha e o limite de aceite de falhas foi ultrapassado. O “Circuit Breaker” muda o seu estado para “Aberto”.
4. Qualquer tentativa de nova invocação para este Microsserviço receberá um erro pertinente. A invocação não será feita.
5. A partir de um “timeout” pré-definido, o “Circuit Breaker” muda de estado para “Semiaberto” para tentar a reconexão com o Microsserviço com problemas.
6. Caso a tentativa de reconexão não tenha sucesso, o “Circuit Break” volta ao estado “Aberto” e espera pelo próximo “timeout”.
7. Caso a tentativa de reconexão tenha ocorrido com sucesso o “Circuit Breaker” é fechado e volta ao estado normal de operação. Invocações ao Microsserviço voltam a ser realizadas.

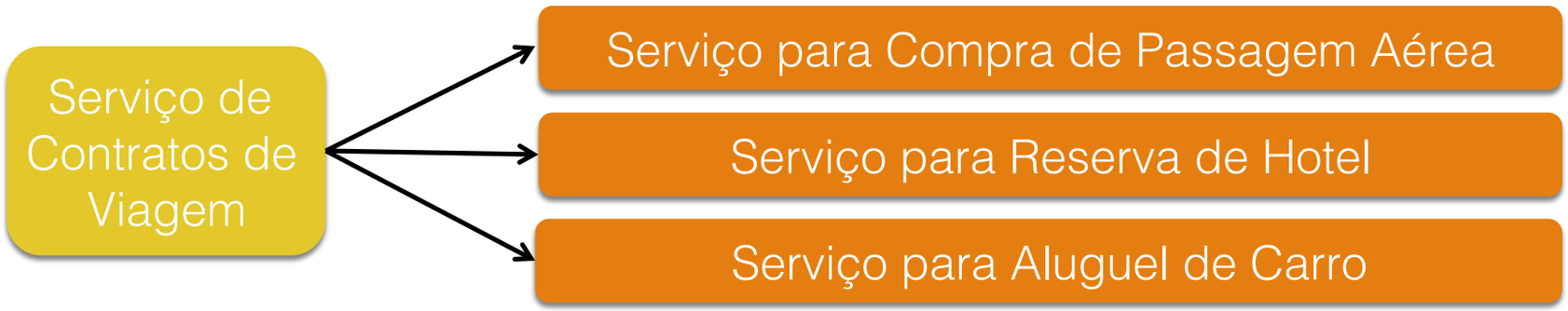
CaaS – Container as a Service - Istio



Modelos de Comunicação

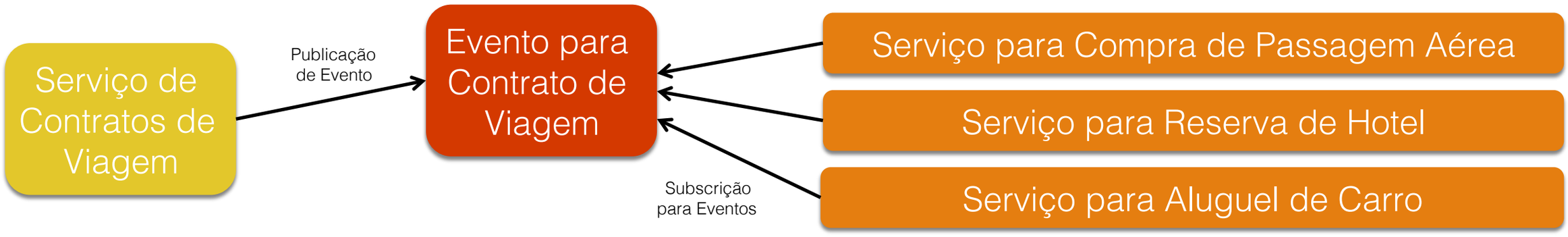
Orquestração

. Alto Acoplamento
. Há ordem de execução
. Atomicidade (?). Implementação de alta complexidade, se (realmente) desejável.



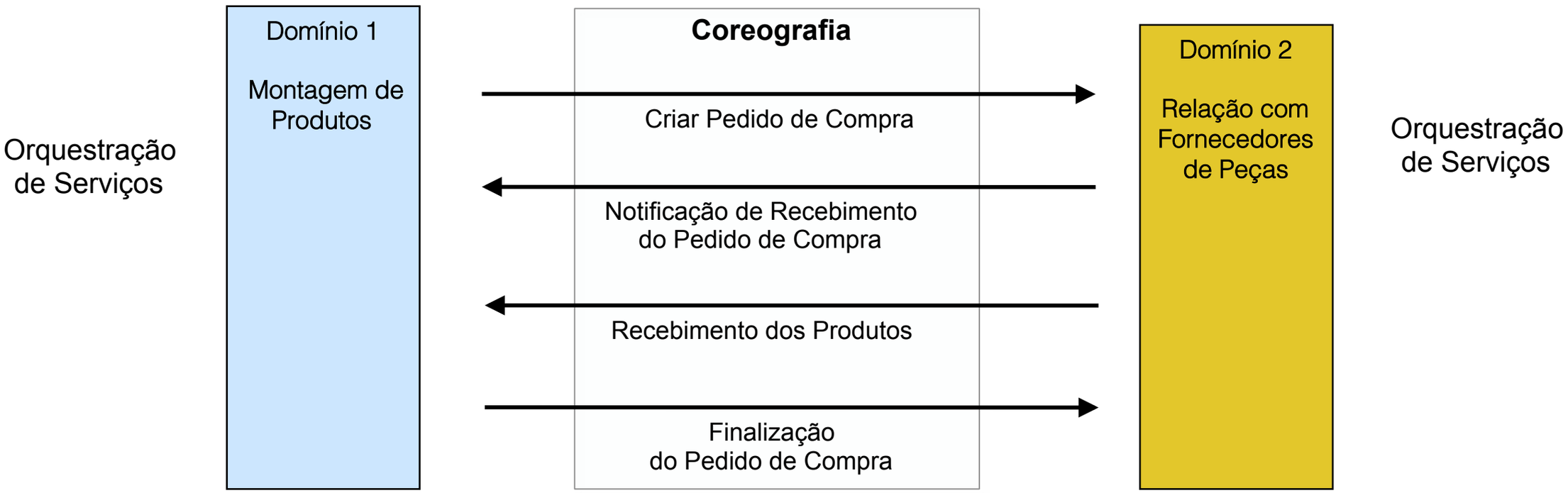
Coreografia

. Baixo Acoplamento
. Não há ordem de execução
. Não há atomicidade

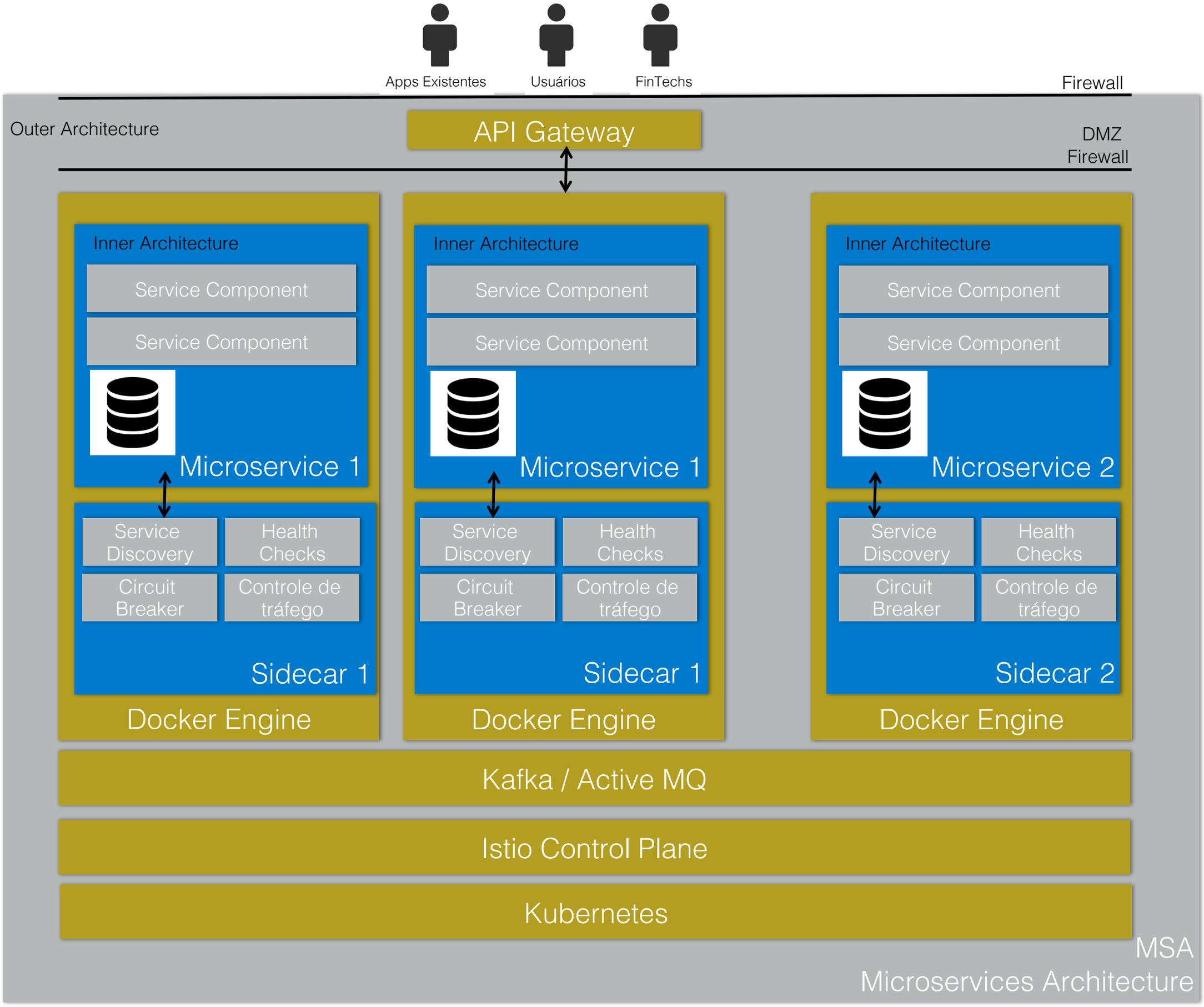


Modelos de Comunicação – Coreografia e Orquestração

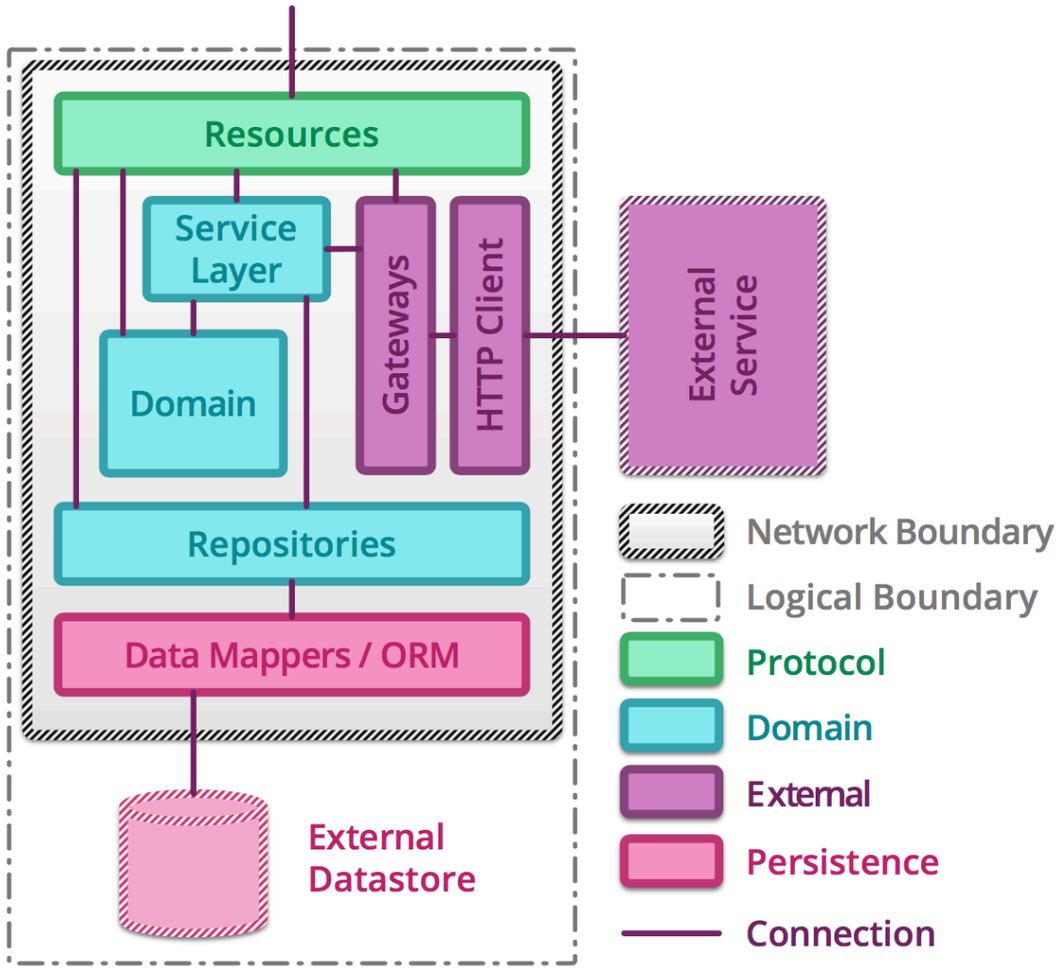
- Em um nível de abstração mais alto, podem ser aplicados os dois modelos:
  - Coreografia entre Microsserviços de Domínios Distintos.
  - Orquestração ou Coordenação entre Microsserviços de um mesmo Domínio.



CaaS – Container as a Service – Kafka & Active MQ

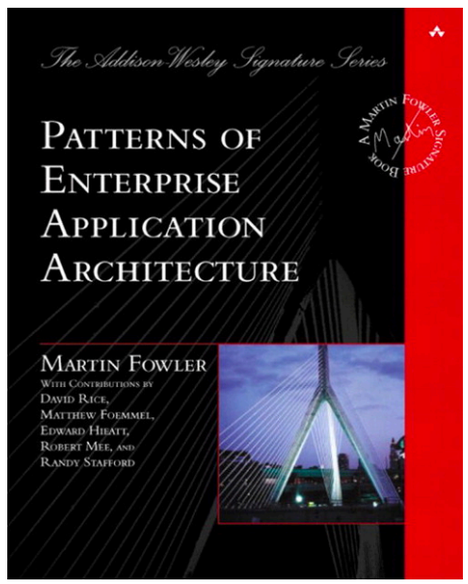


Anatomia de um Microsserviço



Os conceitos descritos em “Patterns of Enterprise Application Architecture” (PofEAA) aplicados em Microsserviços:

- Resources: mapeadores entre o protocolo de aplicação exposto e as mensagens para os objetos que representam o domínio.
- Domain: representa o Domínio de Negócio.
- Repositories: coleção de entidades do Domínio frequentemente persistida.
- Service Layer: coordena as múltiplas atividades do domínio.
- Gateways e HTTP Client: resolve questões pertinentes à comunicação com outros Microsserviços.
- Data Mappers / ORM: executa a persistência de objetos usados pelo repositório.



https://martinfowler.com/articles/microservice-testing/
https://martinfowler.com/eaaCatalog/

Fatores de um Microsserviço

I. Codebase: One codebase tracked in revision control, many deploys
Um “codebase” rastreado no controle de revisão, vários “deploys”.

II. Dependencies: Explicitly declare and isolate dependencies
Explicitamente declare e isole as dependências

III. Config: Store config in the environment
Armazene a configuração no ambiente

IV. Backing services: Treat backing services as attached resources
Trate serviços de retaguarda como recursos anexados

V. Build, release, run: Strictly separate build and run stages
Estágios de “build” e “run” estritamente separados

VI. Processes: Execute the app as one or more stateless processes
Execute a app como um ou mais processos “stateless”

VII. Port binding: Export services via port binding
Exporte serviços via “port binding”

VIII. Concurrency: Scale out via the process model
“Scale out” via modelo de processo

IX. Disposability: Maximize robustness with fast startup and graceful shutdown
Maximize a robustez com “startup” rápidos e “shutdown gracioso”

X. Dev/prod parity: Keep development, staging, and production as similar as possible
Mantenha desenvolvimento, “staging” e produção o mais similares quanto possível

XI. Logs: Treat logs as event streams
Lide com logs como cadeias de eventos

XII. Admin processes: Run admin/management tasks as one-off processes
Execute tarefas e administração/gerenciamento como processos únicos (a não serem repetidos)

https://www.12factor.net

Princípios de um Microsserviço

I. Responsive: System responds in a timely manner

O sistema responde em tempo hábil

II. Resilient: System stays responsive in the face of failure

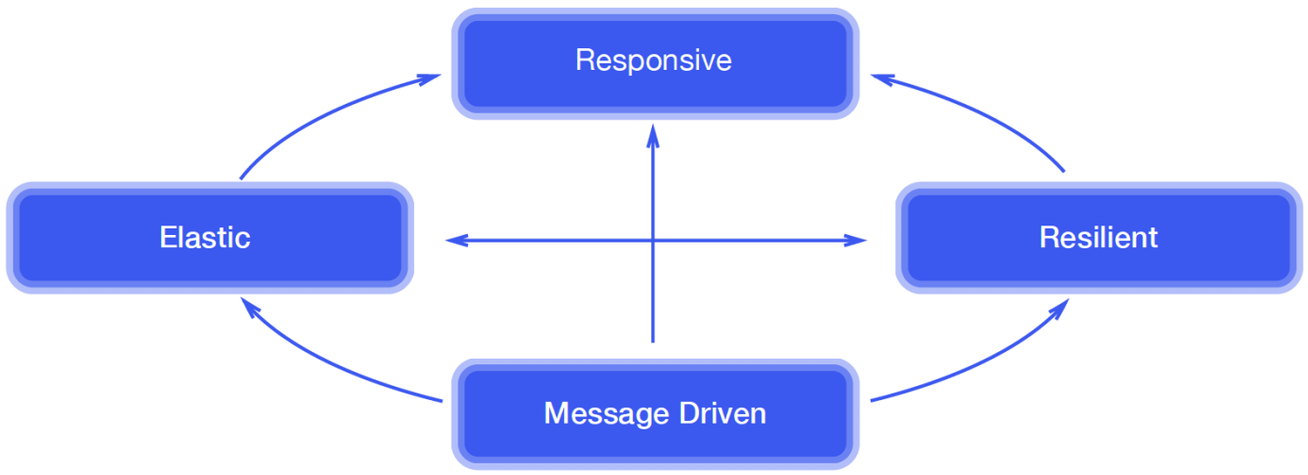
O sistema continua responsivo na ocorrência de falha

III. Elastic: The system stays responsive under varying workload

O sistema continua responsivo sob variação de carga de trabalho

IV. Message Driven: Rely on asynchronous message-passing

Baseia-se em passagem assíncrona de mensagens.

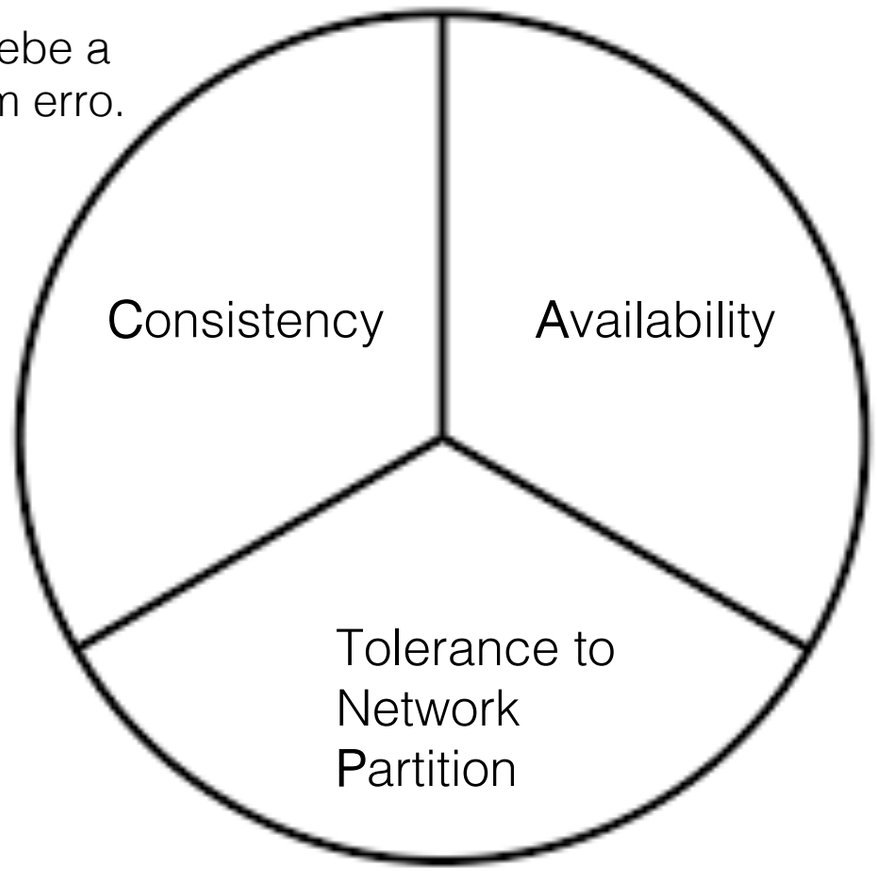


1 Conceito, 4 Princípios, 12 Fatores

http://www.reactivemanifesto.org/

The CAP Theorem

Toda leitura de dados recebe a escrita mais recente ou um erro.



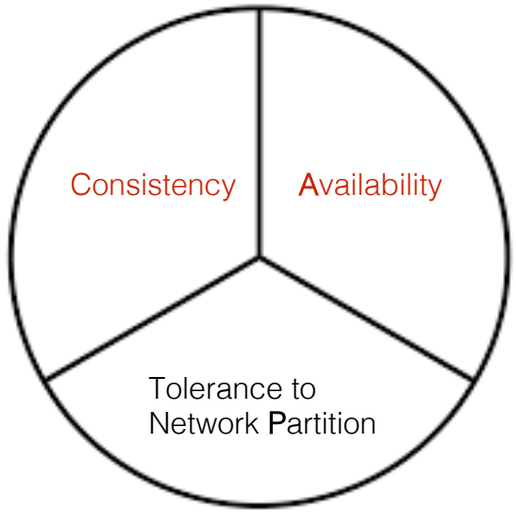
Toda requisição recebe um resposta (sem erro) sem a garantia que contém a escrita mais recente.

O Sistema continua a operar mesmo se houver uma falha de comunicação (componentes não se comunicam)

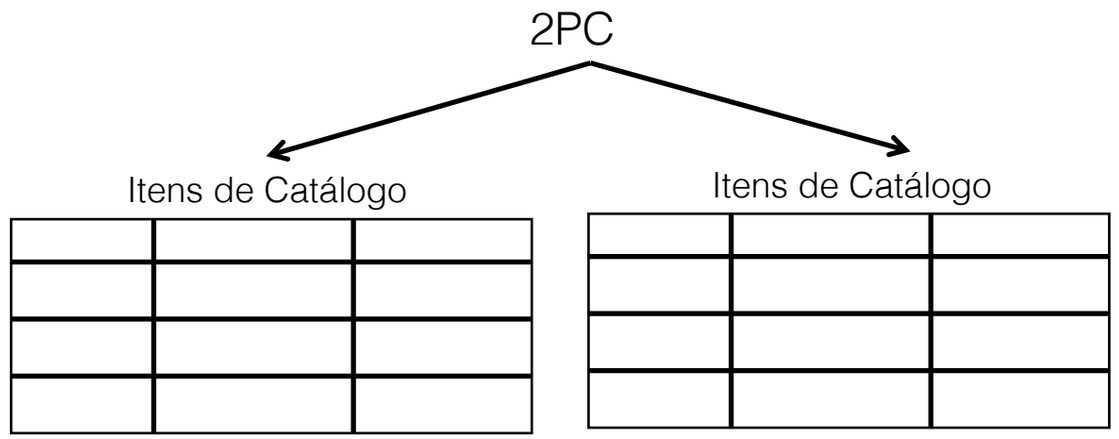
- Consistência, Disponibilidade e Tolerância à Erros de Comunicação são características desejadas em um Sistema Distribuído.
- “Theorem: You can have at most two of these properties for any shared-data system.”

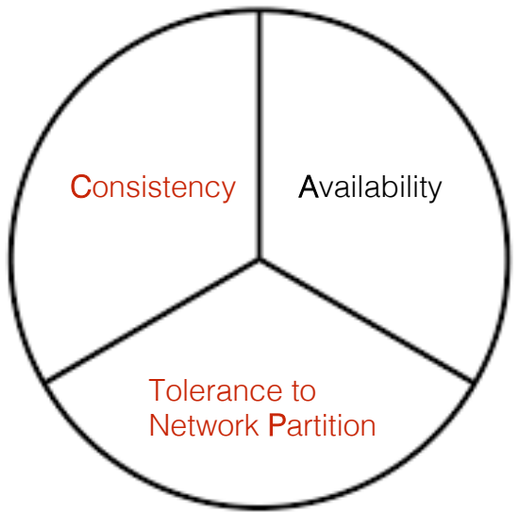
“Towards Robust Distributed Systems”, Dr. Eric A. Brewer, Symposium on Principles of Distributed Computing - PODC Keynote, July 19, 2000

The CAP Theorem

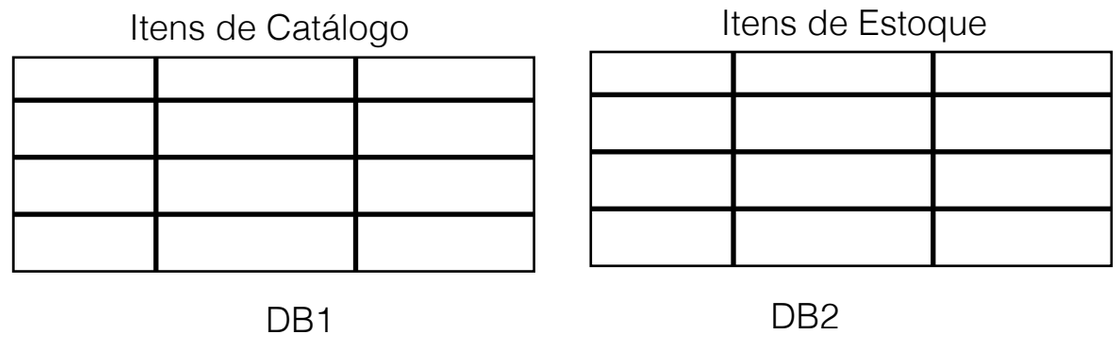


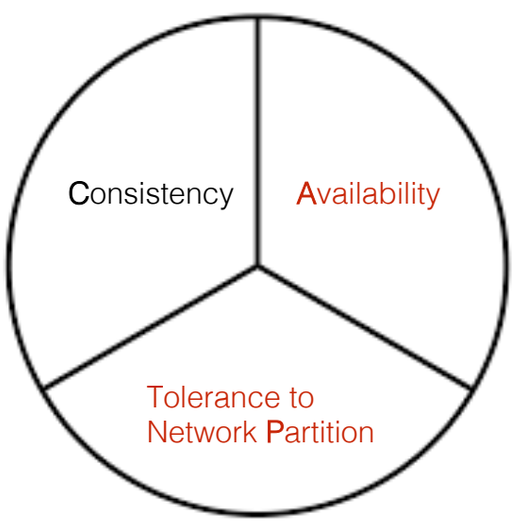
CA
Consistente e Disponível
se não houver “Partition”.
– p.e. RDBMS “single site database” ou cópias de bancos de dados com 2PC commit.



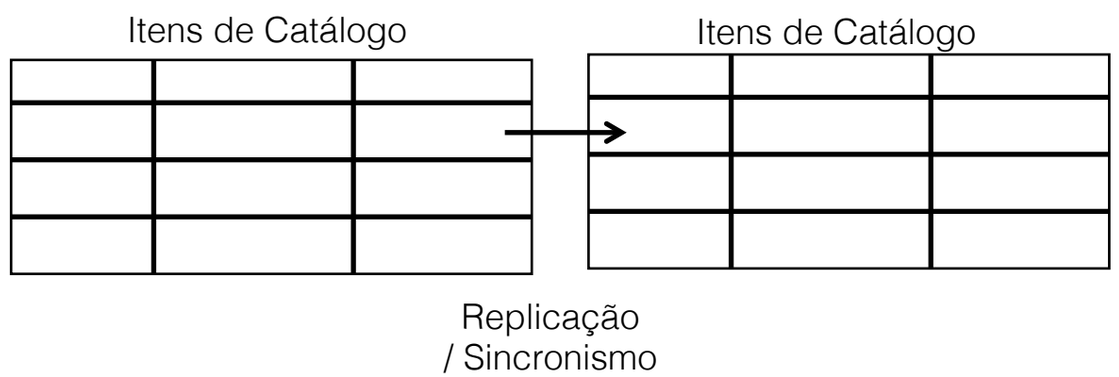


CP
Sempre consistente mesmo se houver “Partition”. Talvez não disponível.
– p.e. “distributed database”, DNS, MongoDB, BigTable, BerkeleyDB.





AP
Sempre disponível mesmo se houver “Partition. Talvez não seja consistente.
– p.e. “Web cache”, CouchBase, Cassandra, DynamoDB.



The CAP Theorem

ACID

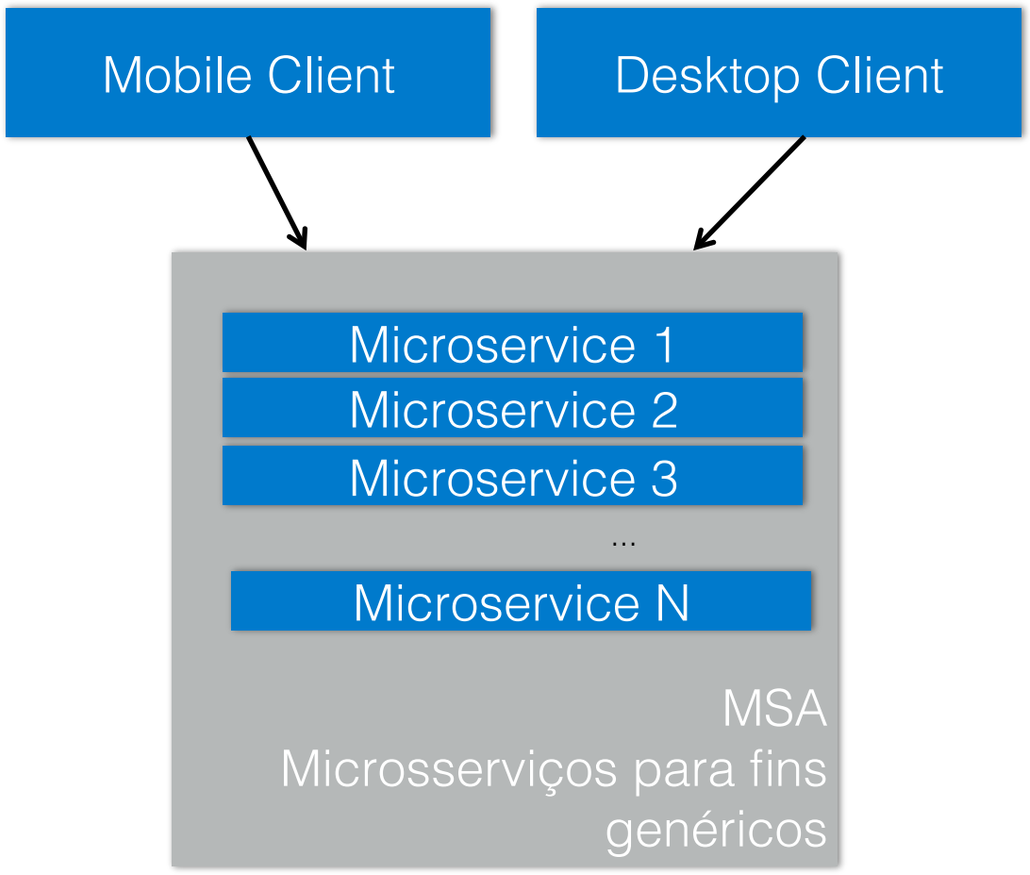
- Atomicity: Todos os dados e comandos em uma transação são executados com sucesso ou são totalmente cancelados.
- Consistency: Todos os dados confirmados (“commited”) devem estar consistentes com as regras de dados (“constraints”, “cascades”, “triggers”)
- Isolation: outras operações concorrentes não podem acessar os dados sendo alterados em um transação que ainda não foi finalizada (“locks”)
- Durability: uma transação finalizada terá os seus dados disponíveis em caso de falha de sistema

BASE

- BAsic availability: Sistema disponível mas não necessariamente com todos os seus componentes.
- Soft-state: O estado de um sistema pode variar.
- Eventually consistent: O sistema ficará consistente com o passar do tempo. (replicação de dados otimista – “optimistic replication”)

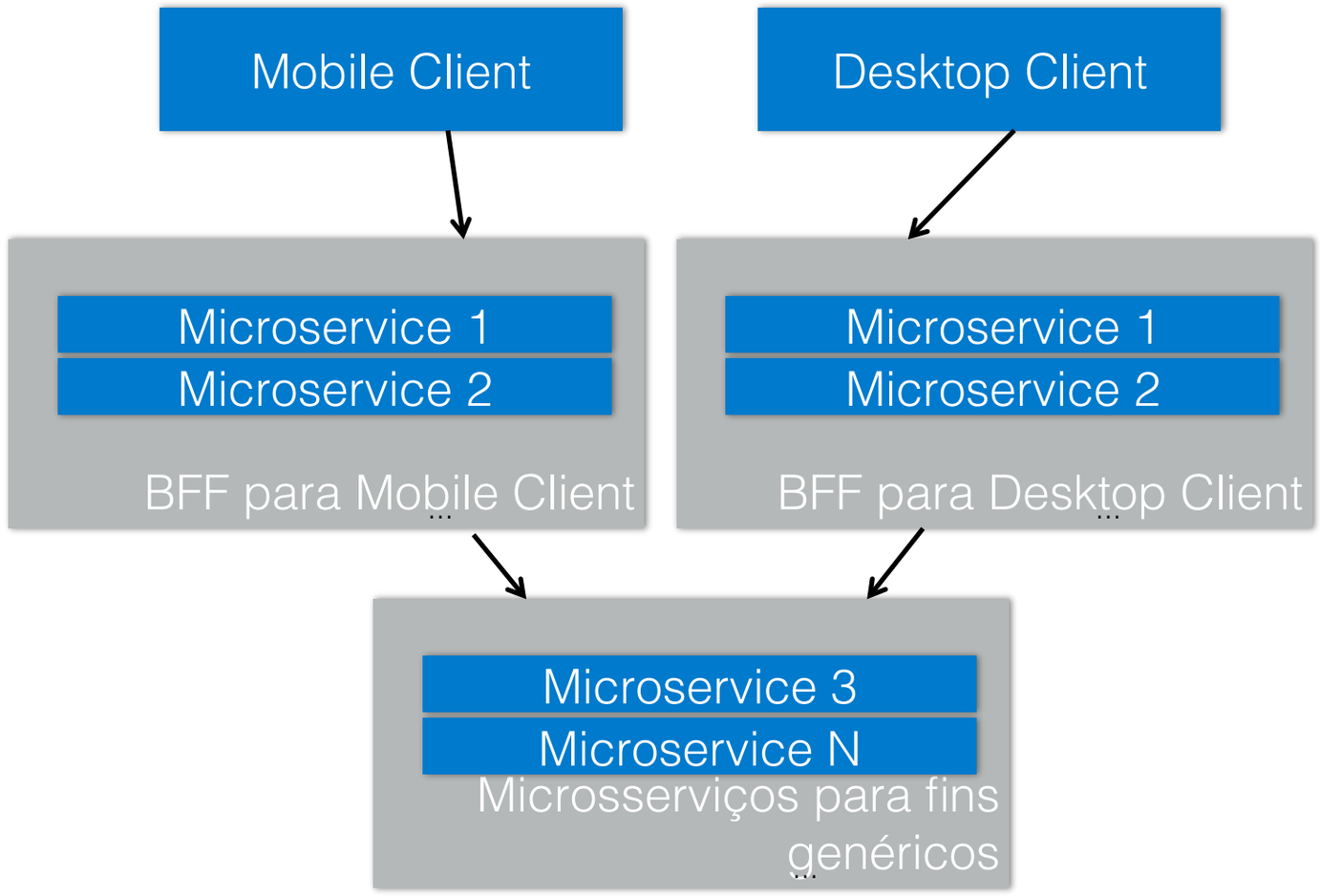
- O modelo ACID promove consistência em detrimento da disponibilidade, diferentemente de BASE.
- BASE inclui um nível de “falta de consistência”. Exemplo: tempo de propagação de operações de uma transação bancária.

Backends to Frontends – BFF Pattern



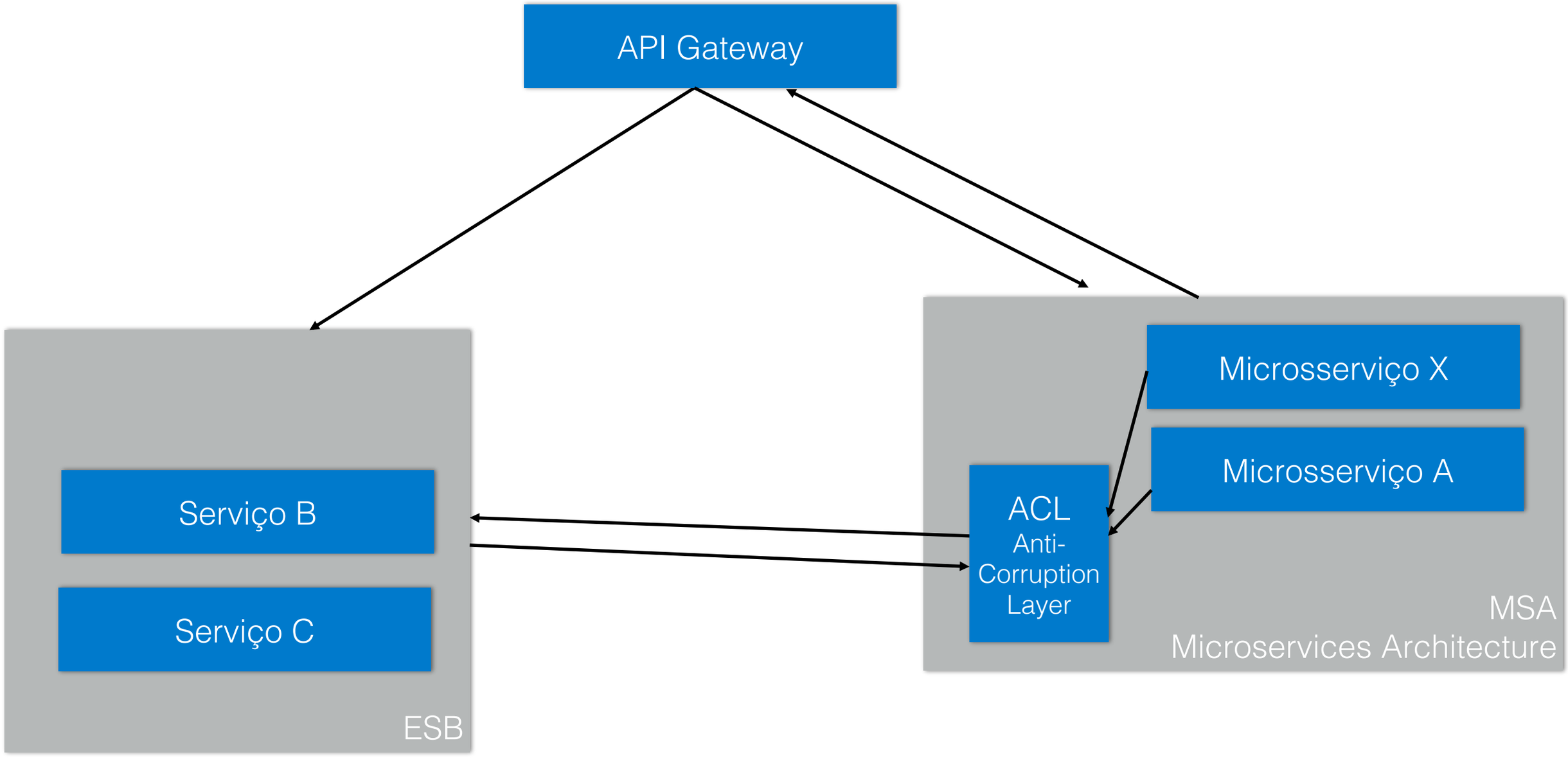
- Diferenças entre clientes Móveis e Desktop (tamanho de tela, usabilidade, recursos computacionais, bateria, etc).
- Serviços para fins genéricos podem se transformar em gargalos.
- Times de desenvolvimento distintos manipulando um mesmo conjunto de microsserviços pode levar a inconsistências.
- Redes diferentes

https://samnewman.io/patterns/architectural/bff/

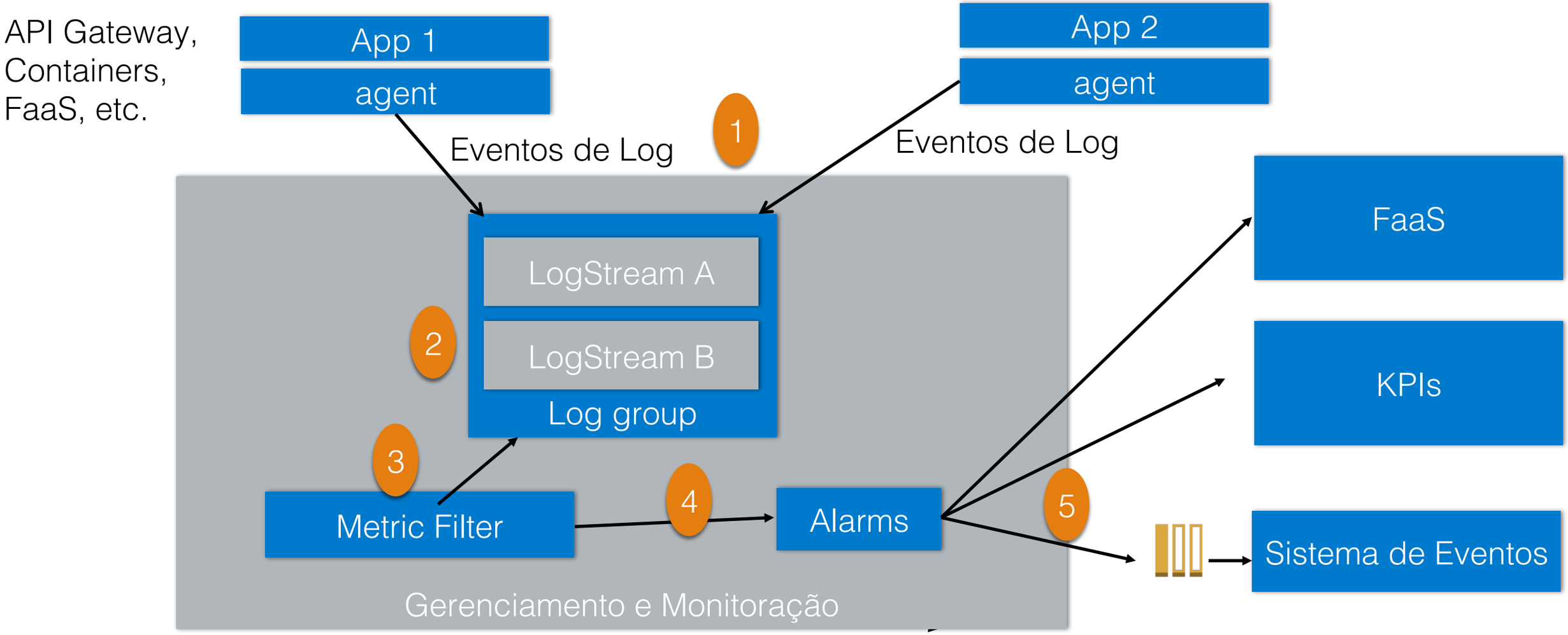


- Os microsserviços específicos para cada tipo de cliente são isolados em seus próprios conjuntos.
- Esses conjuntos ainda compartilham o conjunto de microsserviços em comum.

ACL – Anti-Corruption Layer



LogStream

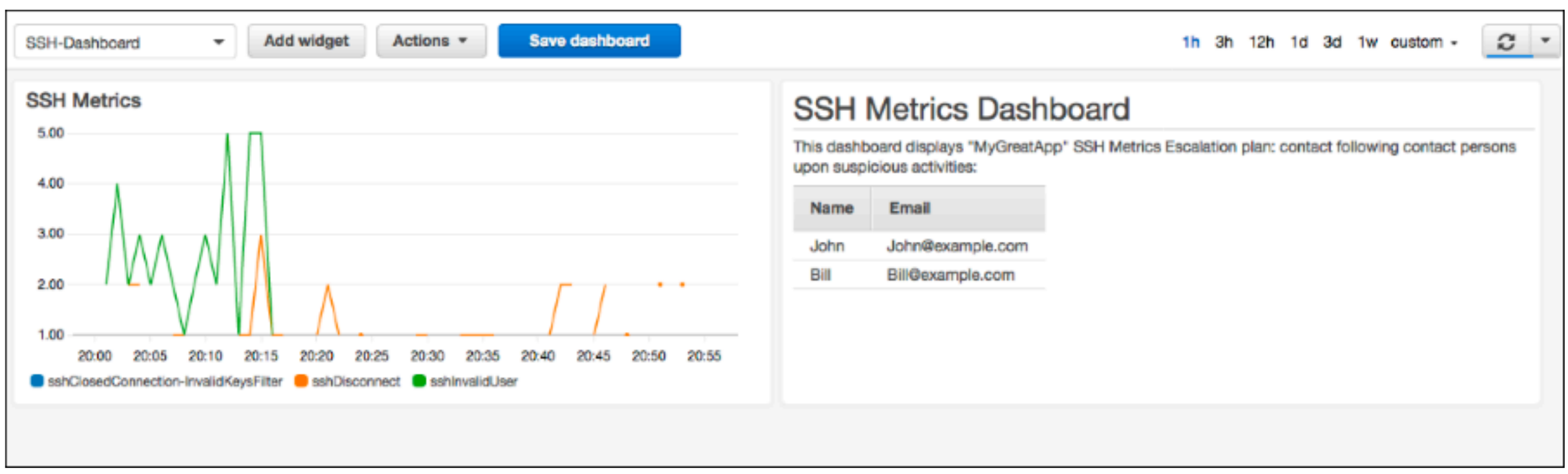


1. Cada aplicação possui um agente instalado. Os agentes enviam suas informações de log para uma instância de “Log stream”.
2. Os “Log streams” são agregados em grupos (p.e. um grupo de logs para instâncias de aplicações).
3. Filtros de métricas são aplicados a um grupo de logs para pesquisas específicas. Quando a métrica encontra resultados, incrementa um contador.
4. Quando o contador atinge um “threshold”, um alarme é disparado.
5. Os alarmes notificam aplicações.

LogStream

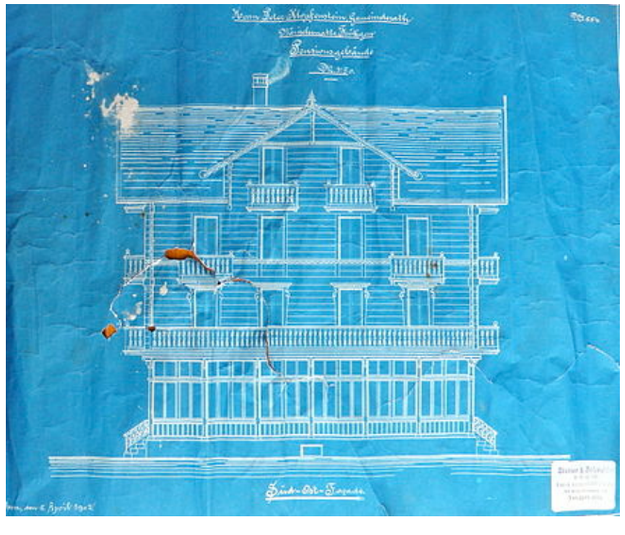
```
InvalidUser:
    Type: AWS::Logs::MetricFilter
    Properties:
      LogGroupName:
        Ref: WebServerLogGroup
      FilterPattern: "[Mon, day, timestamp, ip, id, status = Invalid, ...]"
      MetricTransformations:
      - MetricValue: '1'
        MetricNamespace: SSH
        MetricName: sshInvalidUser
```

Exemplo de Metric Filter



“Blueprint” para Microsserviços

- “Blueprint”: técnica desenvolvida na metade do século XIX onde cientistas descobriram que citrato de ferro de amônio e ferrocianeto de potássio criavam uma solução fotosensível que poderia ser usada para reproduzir documentos. A reação química produzia um composto chamado “ferrocianeto ferroso azul”.
- “Blueprint” de Tecnologia
  - Documento com objetivos futuros com seus passos específicos para implementação.
  - Determina tecnologias específicas a serem usadas, perfis de profissionais necessários, etc.
  - Define macrocronograma de implementação.
- “Blueprint” para Microsserviços
  - Arquitetura para Refactoring de Serviços.
  - “Template” para Desenvolvimento de Microsserviços.
  - “Visão principal” de cada Microsserviço para permitir:
    - Definição de Esteiras de Migração: Softwares de Sistema (OS, Container, Compute Service, DB, etc), pacotes, software customizados e desenvolvidos e comunicação.
    - Análise de Portfolio de Serviços e decisão técnica de Refactoring.
    - Definição do Pipeline de Migração.



The Zachman Framework for Enterprise Architecture

Colunas

1. What (Data) - dados de negócio, informações técnicas, etc. Auxilia na identificação do domínio.
2. How (Function) – como o processo funciona. Eventuais conexões com outros processos.
3. Where (Network) – as operações e transações são realizadas aonde?
4. Who (People) – quais são as “personas” usuárias do processo.
5. When (Time) – quando os processos e transações são executados? Há agendamento de execução?
6. Why (Motivation) – qual a proposta de valor?

Generic Classification Structure of Design Artifacts

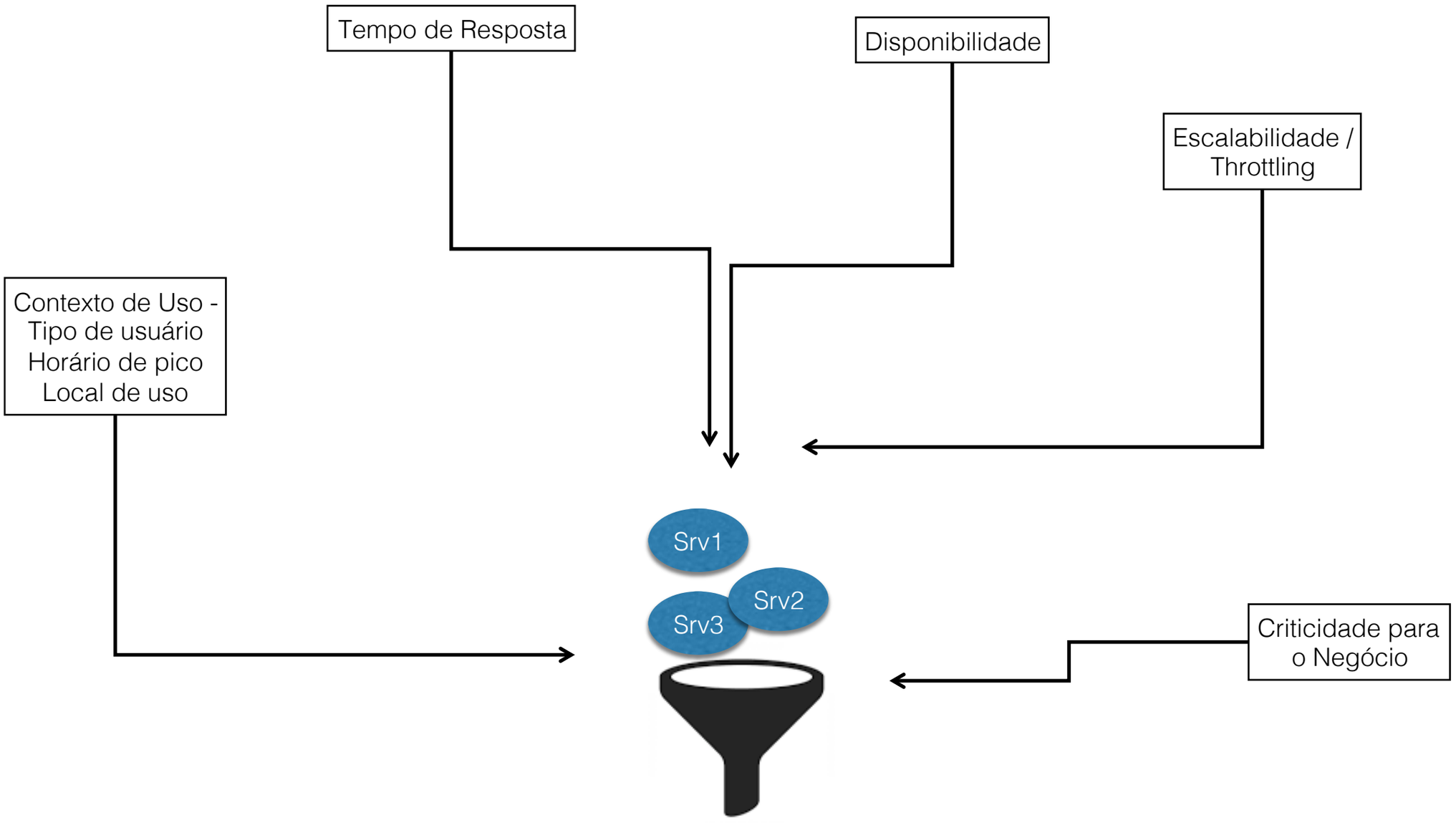
	What	How	Where	Who	When	Why	
Planner							Scope
Owner							Concepts
Designer							Logic
Builder							Physics
Imple-menter							Technology
Operator			THE ENTERPRISE				Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

Linhas

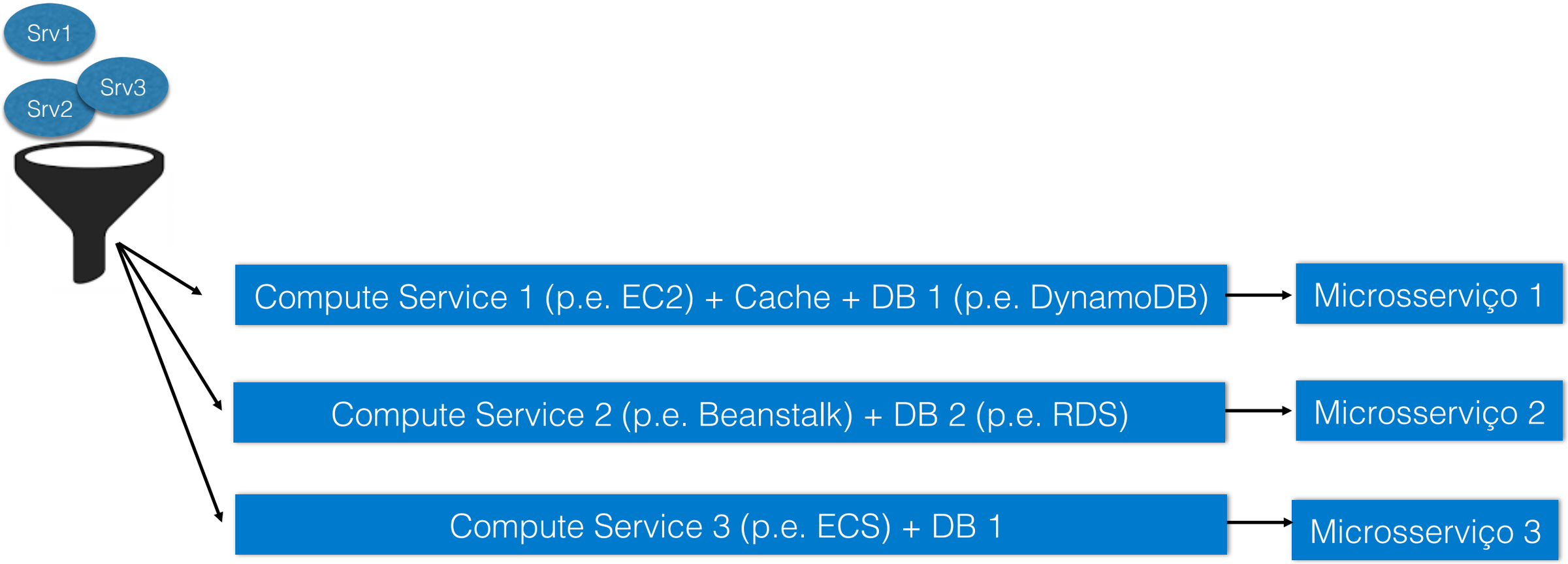
1. Planner – Arquiteto responsável pela definição de de objetivos e desenho de alto nível.
2. Owner – Analista de Negócio responsável pela compreensão do modelo e processo de negócio
3. Designer – Arquiteto responsável pelo desenho lógico e funcional do sistema
4. Builder – Arquiteto responsável pelo desenho técnico e arquitetura física do microsserviço.
5. Implementer – Implementador do processo ou serviço.
6. Operator – fornece a visão funcional a partir da perspectiva dos usuários (funcionário, parceiro, cliente, etc).

The Zachman Framework for Enterprise Architecture
https://www.zachman.com/resources/ea-articles-reference/327-the-framework-for-enterprise-architecture-background-description-and-utility-by-john-a-zachman

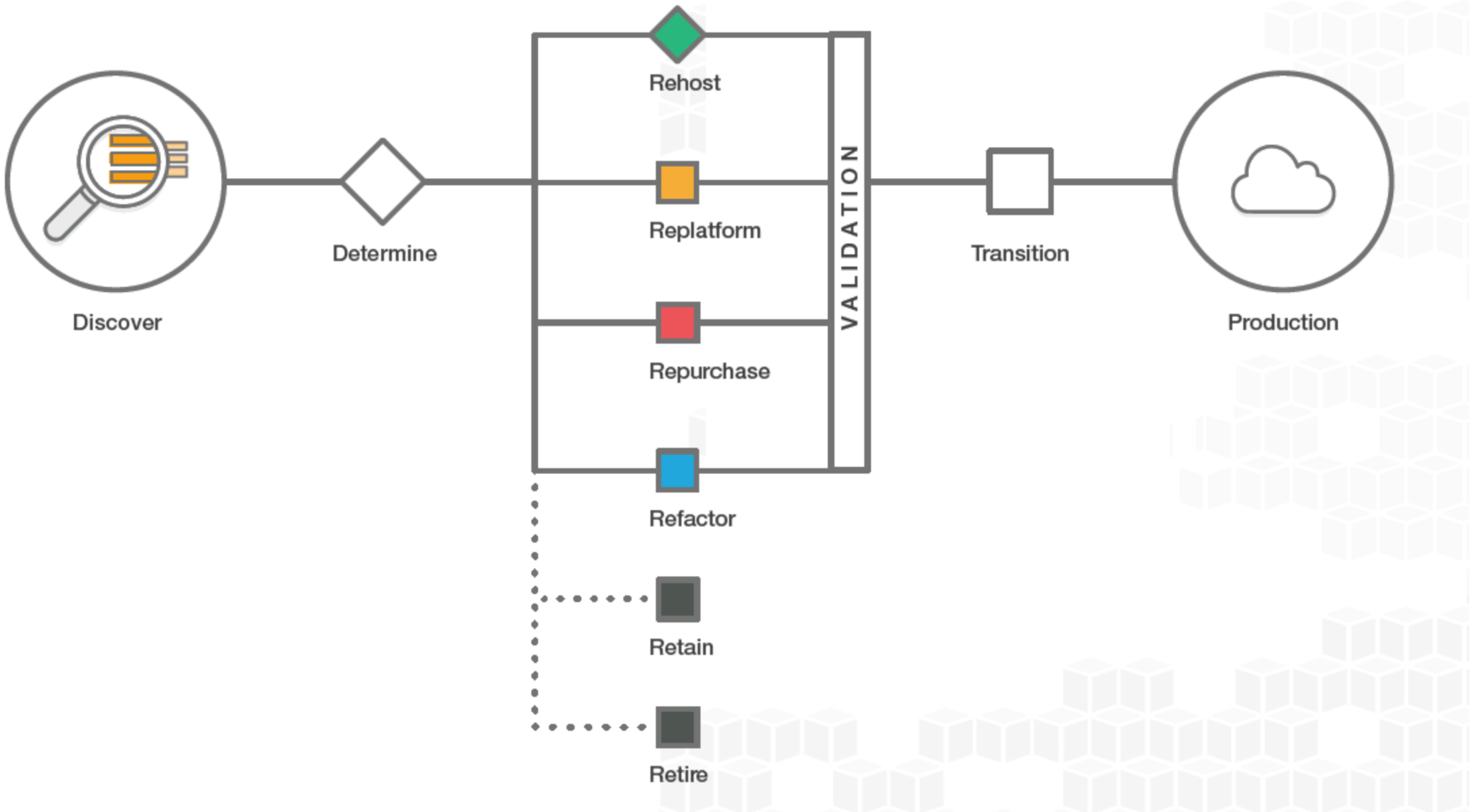
Processo de Qualificação de Ativo (Serviço / Aplicação)



Esteiras de Refactoring / Desenvolvimento

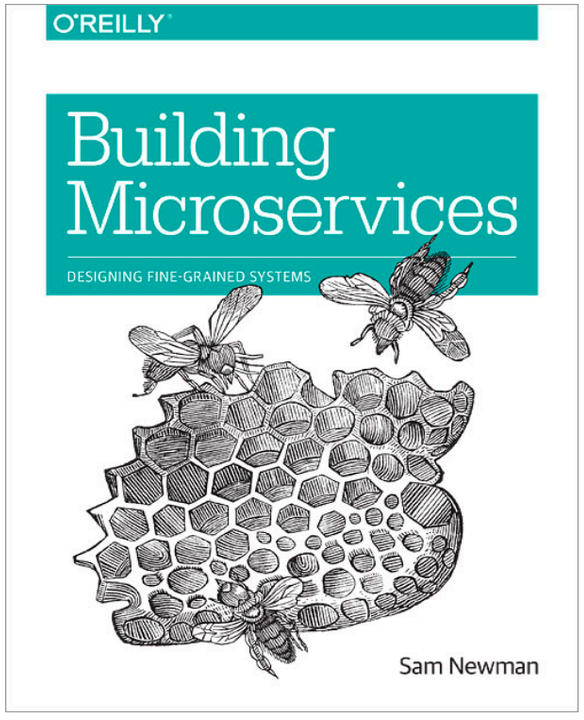


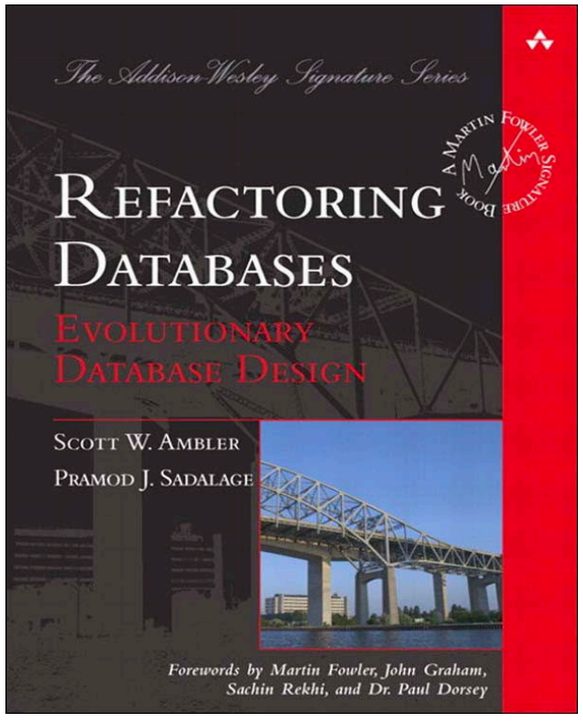
Estratégias de Migração

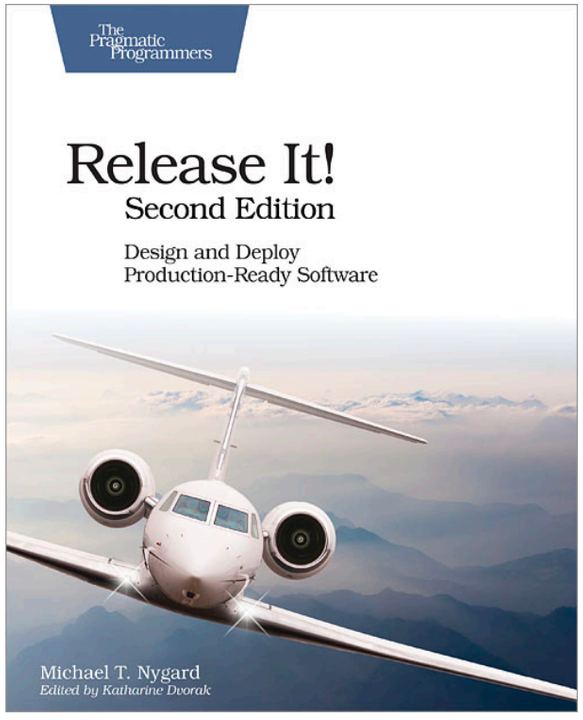


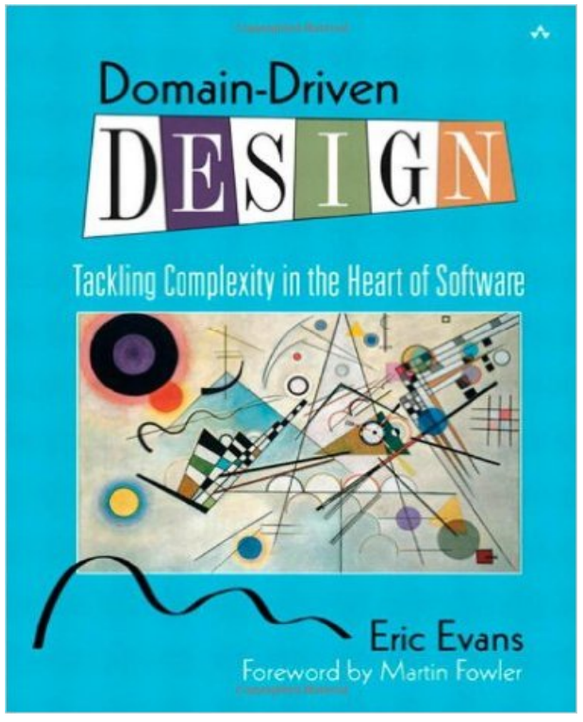
"Migrating to AWS: Best Practices and Strategies" - Stephen Orban
https://d1.awsstatic.com/Migration/migrating-to-aws-ebook.pdf

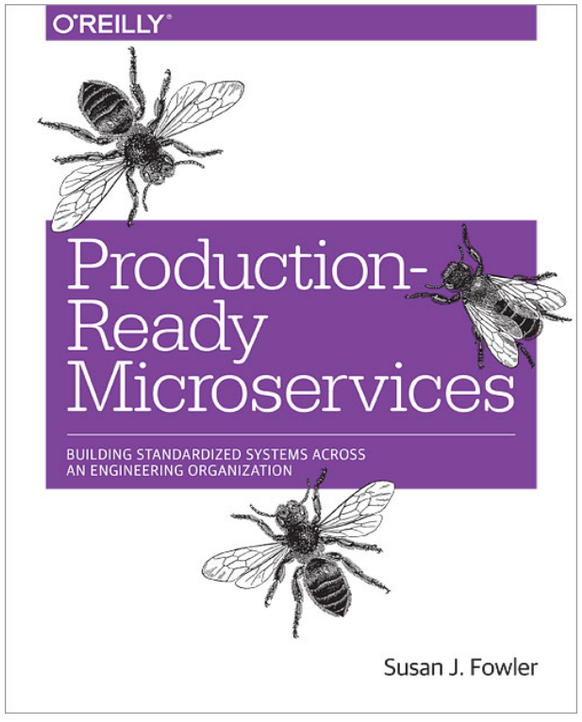
Referências

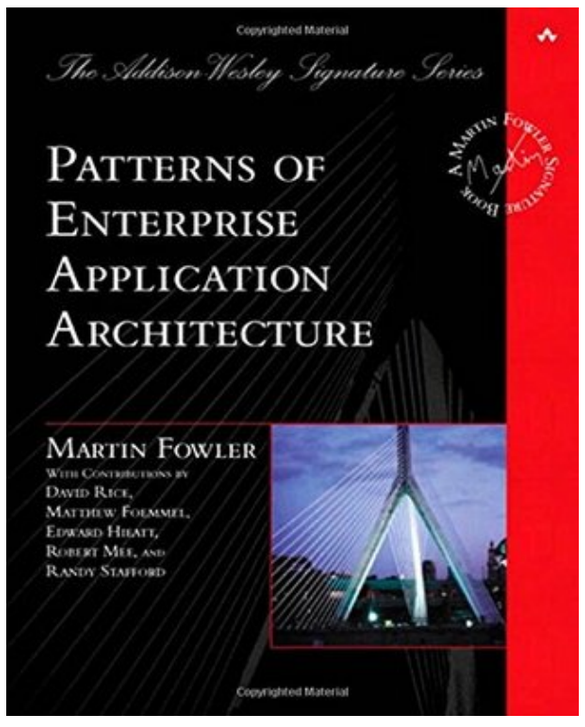












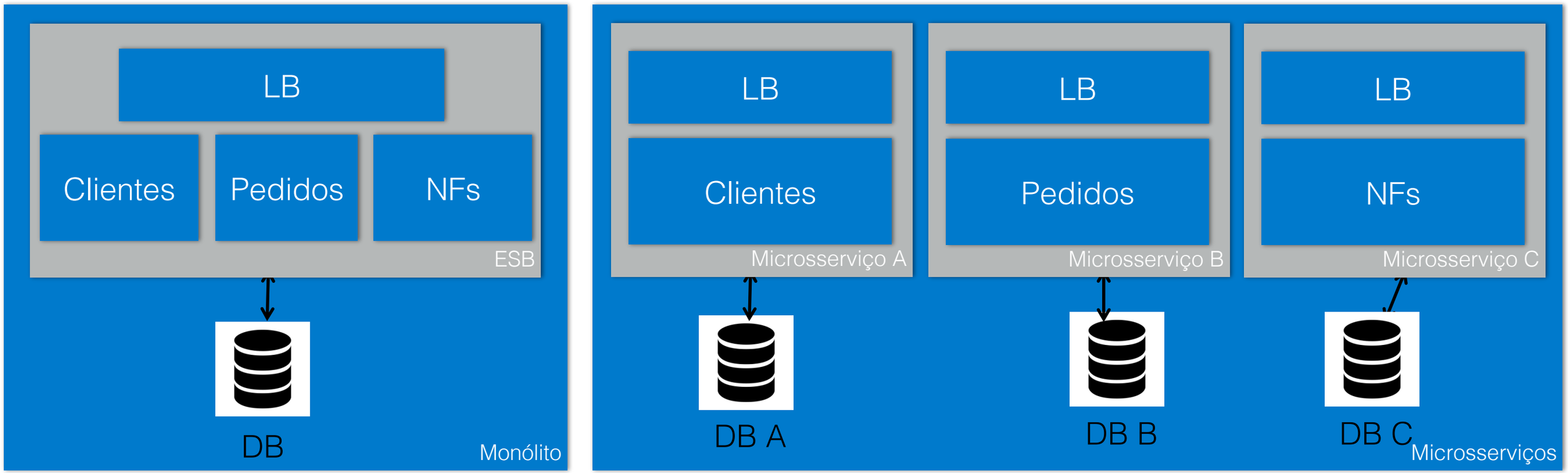
Microservices Architecture Workshop

Arquitetura de Referência e Plano de Modernização - Migração de ESB para MSA

Claudio Acquaviva

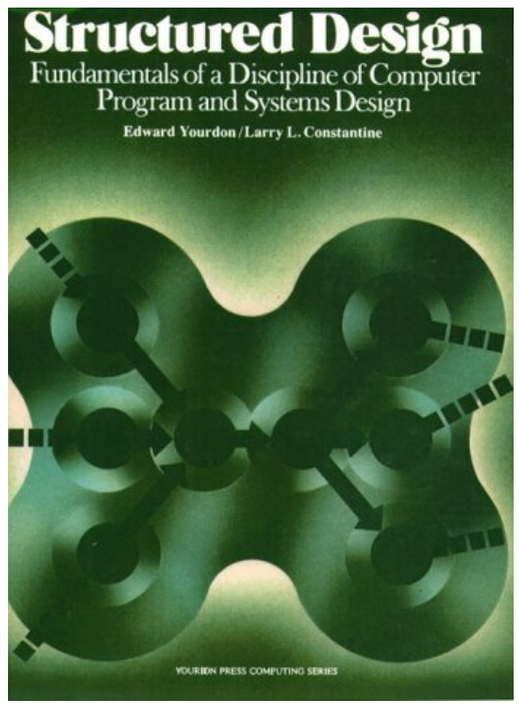
Slides extras

Monólitos e Microsserviços



Acoplamento e Coesão – Coupling and Cohesion

- Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Larry Constantine, Ed Yourdon, 1979
    - “Low coupling is a sign of a well structured computer system.” (Baixo acoplamento é um sinal de um sistema de computador bem estruturado).
    - “High cohesion tend to be preferable because it is associated with several desirable traits of software including robustness, reliability, reusability, and understandability.” (Alta coesão tende a ser preferível porque está associada com vários traços de software desejáveis incluindo robutez, confiabilidade, reusabilidade e compreensão)
    - “...Clearly, cohesion and coupling are interrelated. The greater the cohesion of individual modules in the system, the lower the coupling between modules will be...”. (Claramente, coesão e acoplamente estão inter-relacionados. Quanto maior a coesão dos módulos individuais em um sistema, menor será o acoplamente entre os módulos.



Service Orientation – SOA e ESB

- Pontos positivos: “Service Orientation” (Orientação a Serviços) – conceitos de Acoplamente e Coesão para níveis altos de abstração.
- Pontos negativos
  - O processo para a definição de Serviços Corporativos exigia uma envolvimento, de difícil implementação, de várias áreas da Companhia.
  - Baixo “Time-to-Market” pela complexidade no desenvolvimento e evolução de serviços com tecnologias baseadas em Web Service (XML, SOAP, WSDL, UDDI).
  - Por conta dos itens acima, os Serviços definidos ficaram restritos a questões de baixo nível de abstração voltadas a integração básica de sistemas. A não implementação de Serviços Corporativos levou a uma situação de entregas de baixo valor ao negócio.
  - O uso de um ESB apresentou baixa escalabilidade da infraestrutura.
  - Custos altos de desenvolvimento, manutenção e operação.
  - Custos altos de licenciamento de softwares On-Premises.

Em resumo, a Orientação a Serviços é muito benéfica. A sua implementação em ESB não é uma boa solução.

Em 2005, Anne Thomas Manes, VP do Gartner escreveu o famoso paper “SOA is Dead; Long Live Services” (http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html).

Outer Architecture & Inner Architecture

- “Outer Architecture”
  - API Gateway
  - Autenticação de Usuários
  - Autorização de Usuários com políticas de Baixa Granularidade
  - Federação de Identidades entre MSA e ESB
  - Ambiente para Execução de Microsserviços
  - “Service Registration” e “Service Discovery”
  - Balanceamento de Carga entre instâncias de um Microsserviço
  - Comunicação entre Microsserviços: troca de mensagens e processamento de eventos
  - Gestão de instâncias dos Microsserviços (elasticidade, “health-check”)
  - Monitoração do uso dos Microsserviços.
- “Inner Architecture”:
  - Responsável pela coleção de componentes e serviços definidos em um Domínio de Microsserviços como Lógica de Negócio, Persistência de Dados, Autorização de Usuários com Políticas de Alta Granularidade, etc
  - Usualmente, é o nível considerado para os processos de DevOps
  - Padronização para o desenvolvimento dos Microsserviços

API Management

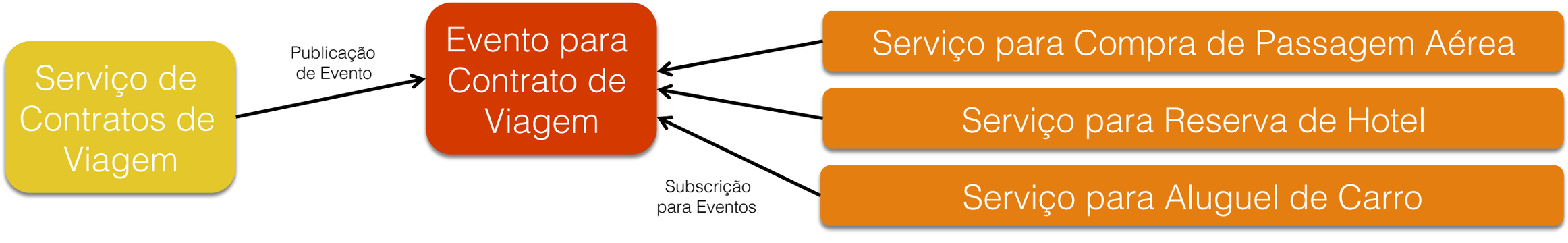
- Definição de API para o uso dos Microsserviços.
    - Padrões: OpenAPI (“Swagger” - http://www.openapis.org) ou RAML (“RESTful API Modeling Language” - http://www.raml.org)
- Segurança de acesso às APIs:
    - Autenticação de Usuários
    - Autorização (Controle de Acesso) com Políticas de Baixa Granularidade
    - OAuth2: implementação de “Access Token” (“by-reference token”).
    - OpenID Connect: implementação de “Id Token” (“by-value token”). Padrão JWT (“JSON Web Token”).
- Federação de Identidades entre ESB e MSA
- Mediação e roteamento das requisições de clientes para MSA ou ESB.

Federação - ESB e MSA

1. O cliente tenta o acesso a alguma API.
2. Como o cliente não possui nenhum “token” SAML, ele é direcionado ao Identity Provider (IdP) definido. O usuário fornece as suas credencias.
3. O IdP autentica o usuário. Em casa de sucesso, gera um “token” SAML e o repassa ao cliente.
4. O cliente faz a invocação novamente ao API Gateway, desta vez com o “token” SAML.
5. O API Gateway checa as políticas de Autorização para permitir ou não o acesso ao recurso necessário ao cliente. O API Gateway emite um JWT a partir do “token” SAML. O JWT será repassado entre os Microsserviços para implementação do Modelo de “Identity Propagation”.
6. O API Gateway faz o roteamento ao Serviço responsável repassando o JWT.

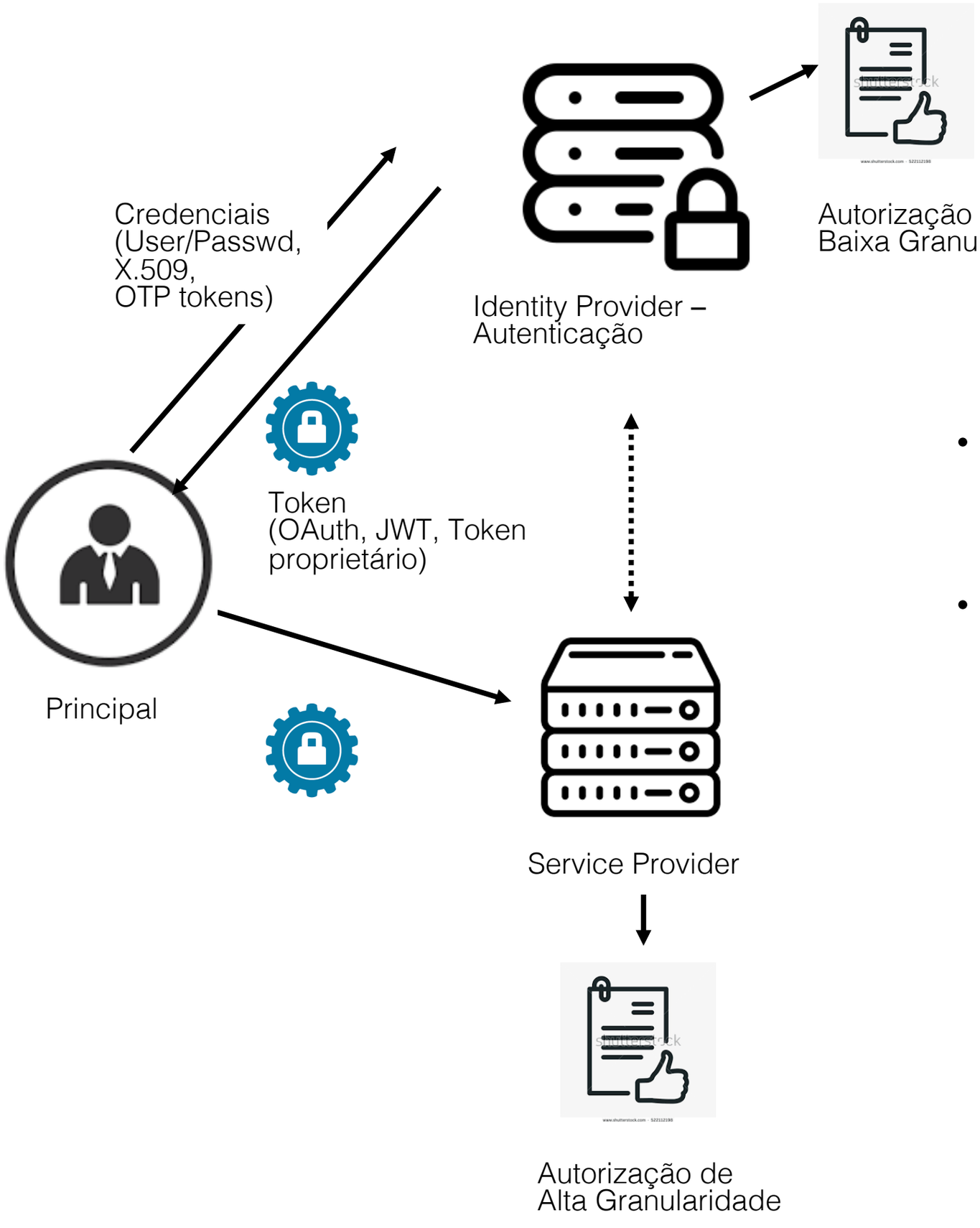
Modelos de Comunicação – Coreografia

- Em MSAs, é fortemente aconselhável o modelo de Comunicação de Serviços baseado em Coreografia. A Coreografia é um modelo de Comunicação Assíncrono com mensagens Um-A-Muitos.
- Neste modelo, não há serviço principal ou um coordenador. A arquitetura seria esta:



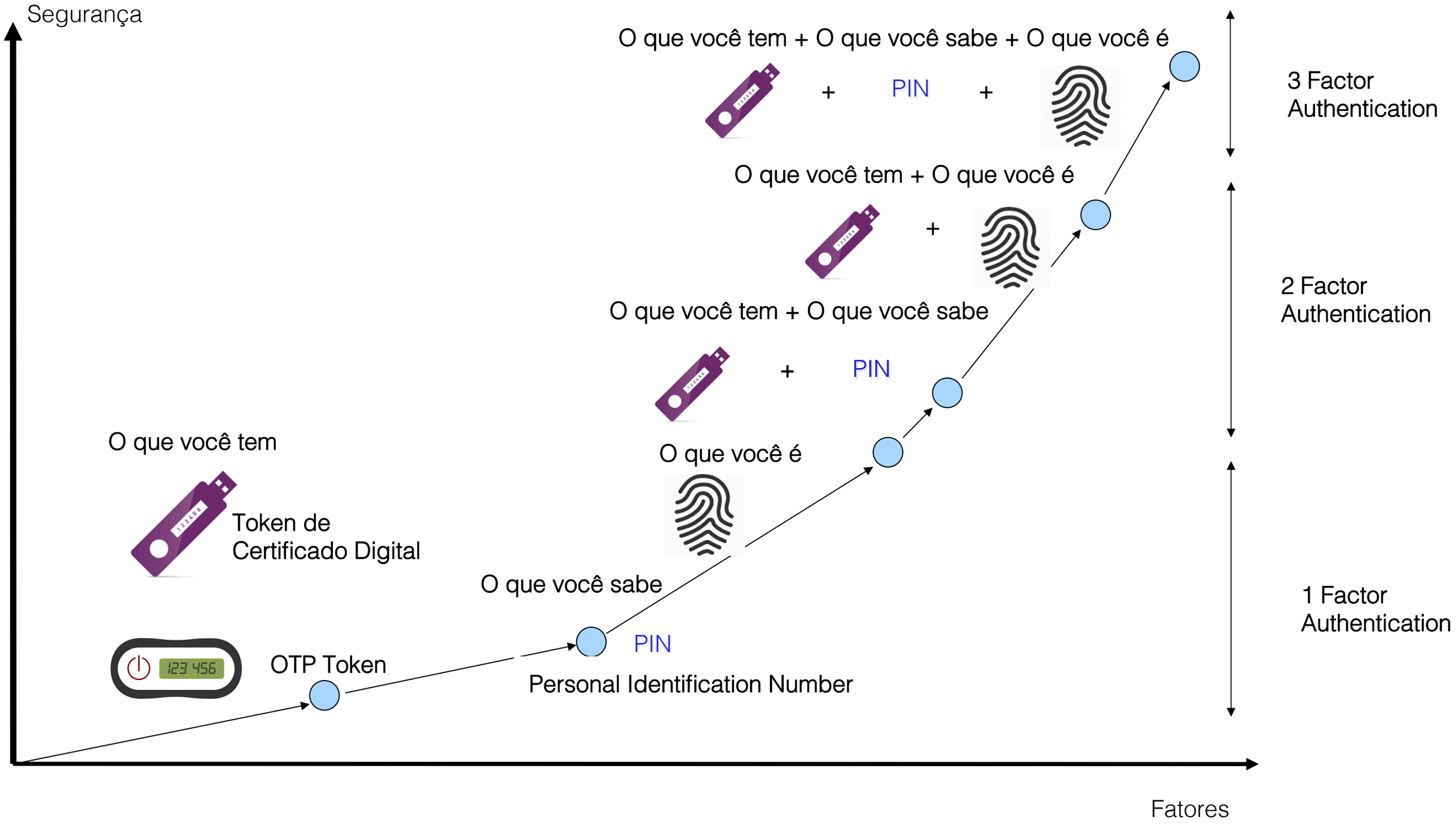
  - O "Serviço de Contratos de Viagens" publica, assincronamente, um "Evento para Contrato de Viagem".
  - Os demais serviços subscrevem-se a esses Eventos e reagem independemente, cada um a sua maneira.
- Um ponto de atenção é a rastrebilidade dos Serviços durante a execução do mesmo. Uma opção é a inclusão de um Serviço específico de Monitoração dos Serviços em execução.
- A Arquitetura baseada em Coreografia apresenta um cenário de nível de Acoplamento muito mais baixo do que aquele baseado em Orquestração. Por este motivo deve ser a opção preferencial.

AuthN & AuthZ - Fluxo Principal

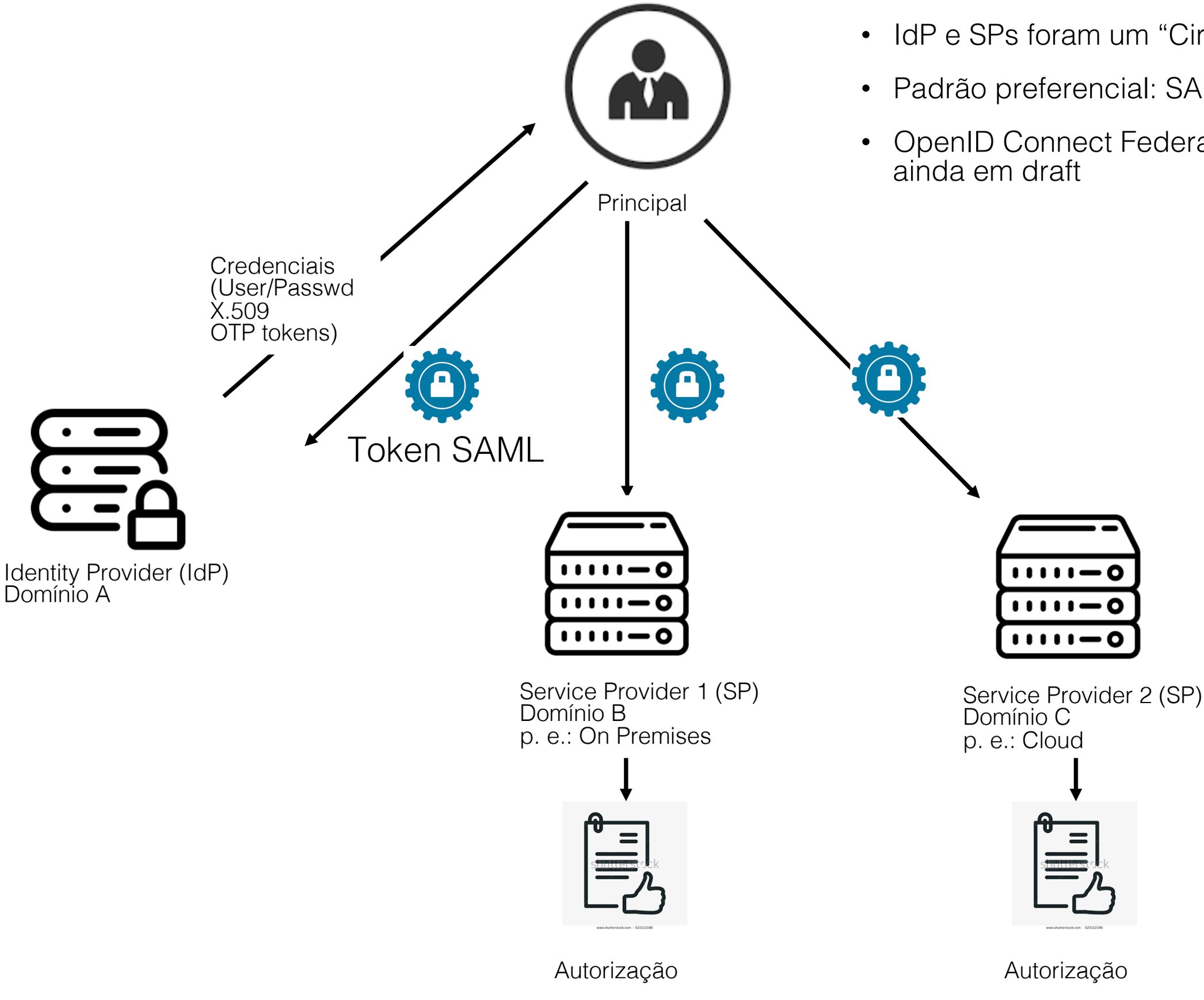


- Externalização (isolamento) dos Processos de AuthN & AuthZ
- Padrões atuais: OAuth, OpenId Connect (OIDC)

Fatores de Autenticação

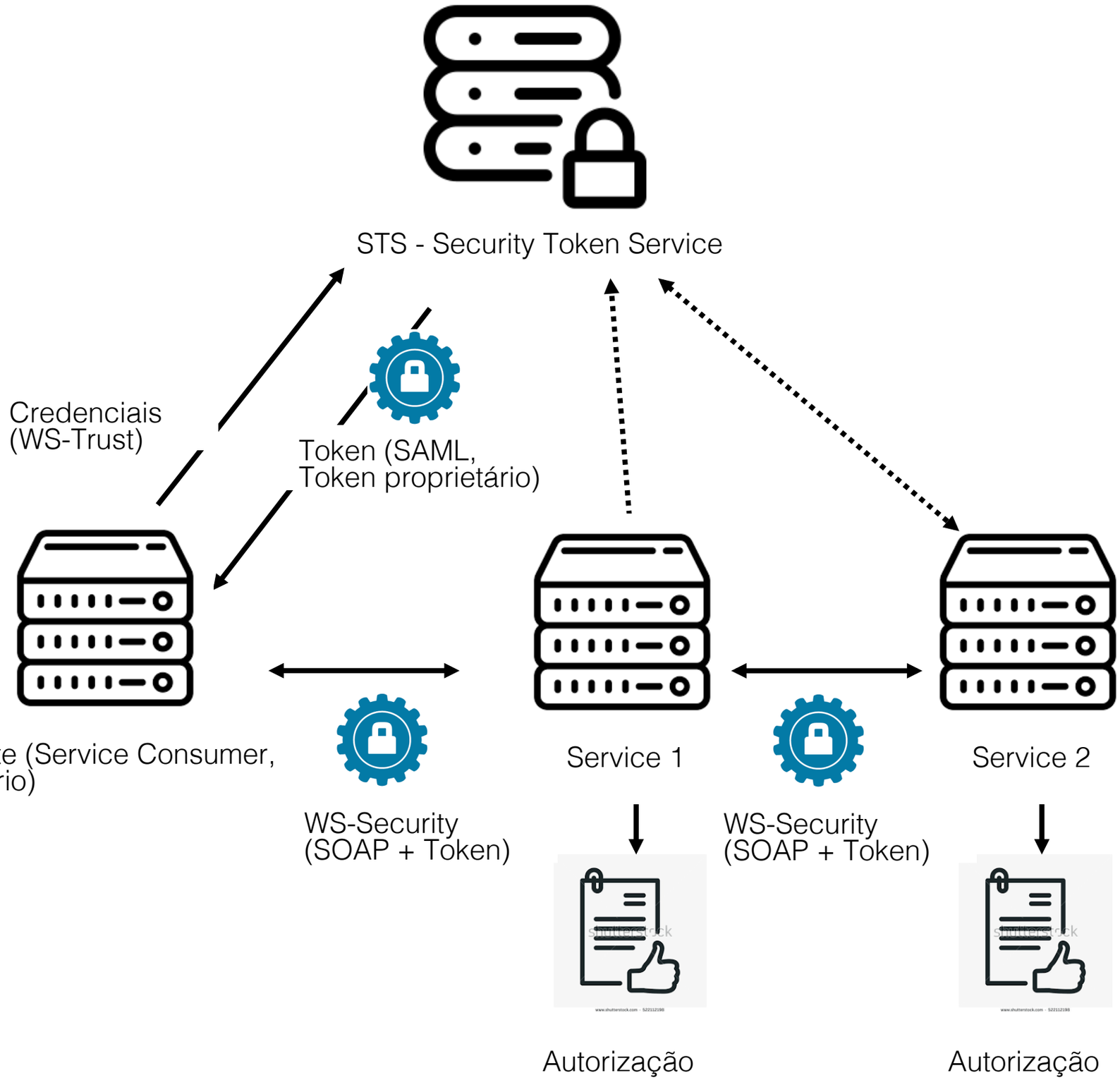


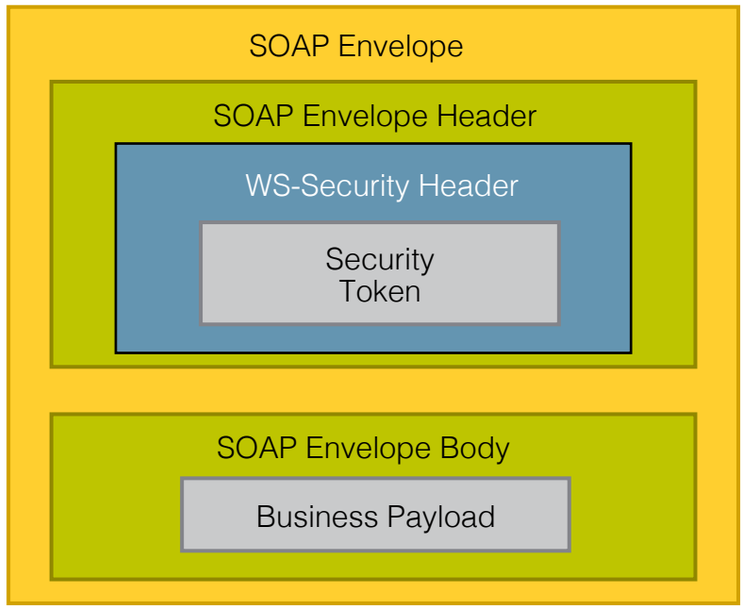
Federação



- IdP e SPs foram um “Circle of Trust”.
- Padrão preferencial: SAML 2.0
- OpenID Connect Federation 1.0 ainda em draft

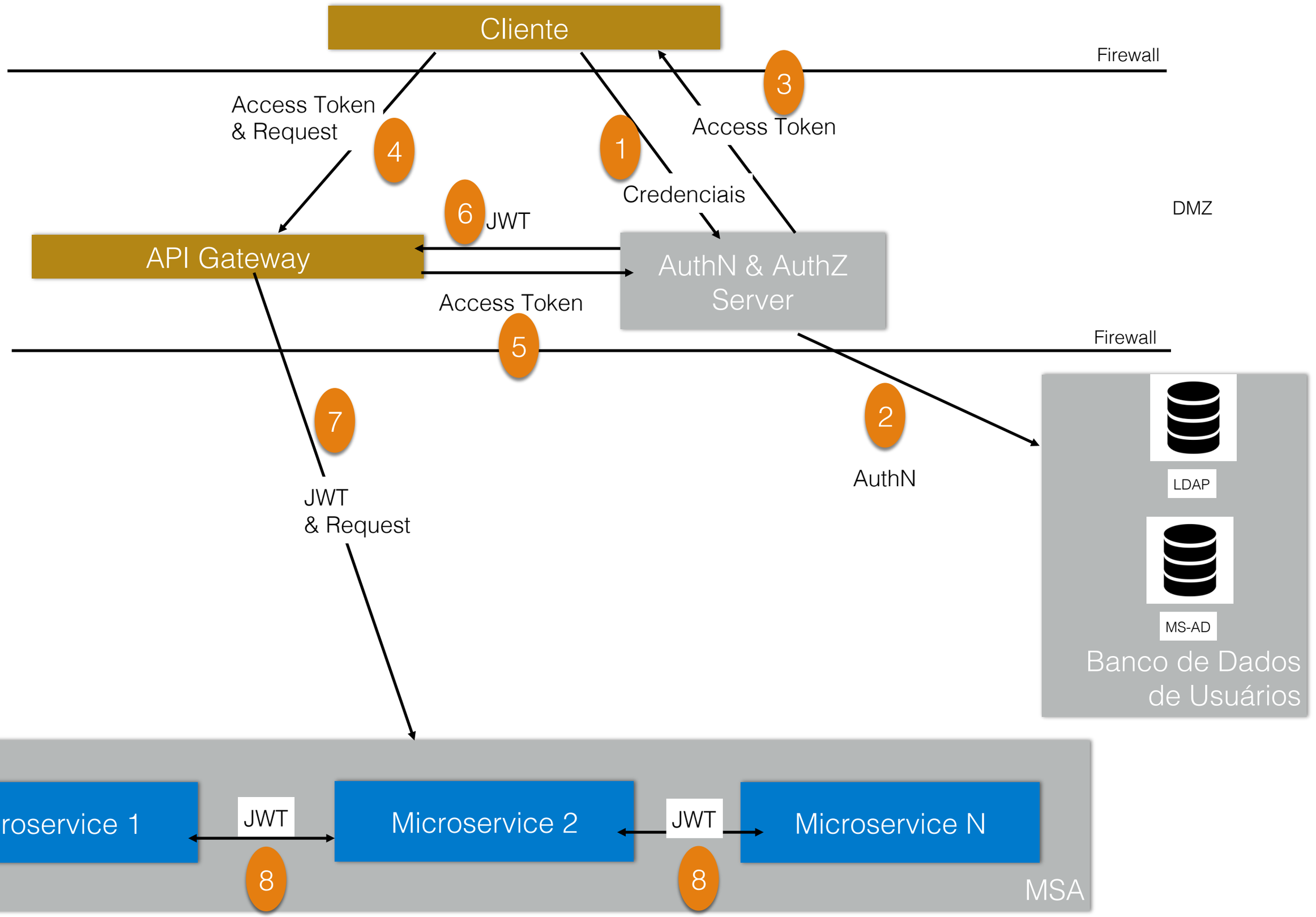
Segurança em Serviços – Fluxo Principal





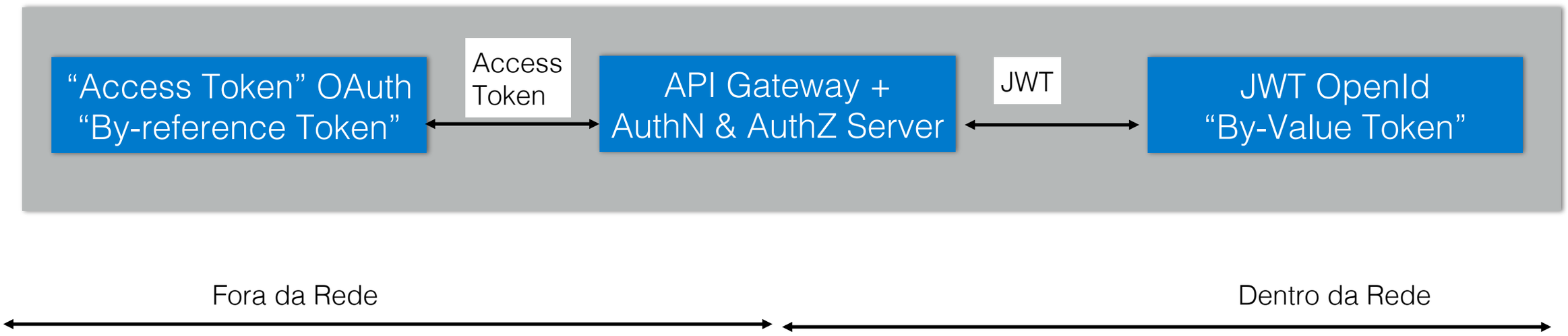
Tipos de “tokens”: UidPwd, Certificados X.509, “Tickets” Kerberos, SAML, “Binary Security Token”.

Arquitetura de Referência de Segurança em MSA



Autenticação em MSA

1. O Cliente (usuários ou aplicação) apresenta suas credenciais (UserId/Passwd, Certificado Digital, etc.) para o Servidor de AuthN e AuthZ.
2. O Servidor autentica o Cliente mediante políticas descritas e bases de dados de usuário sob sua responsabilidade.
3. O Servidor cria e repassa ao Cliente o “Access Token” (“By-reference token”)
4. O Cliente repassa a requisição e o “Access Token” ao API Gateway.
5. O API Gateway envia o “Access Token” para o Servidor de AuthN e AuthZ para a sua tradução para um JWT (“By-value token”).
6. O Servidor de AuthN e AuthZ devolve ao API Gateway o JWT.
7. O API Gateway repassa a requisição ao Microsserviço juntamente com o JWT.
8. O JWT é repassado aos demais Microsserviços nas requisições derivadas.



ESB -> MSA em Cloud Computing: Processo de Migração

Fase 1: Opportunity Evaluation

- Quais os motivadores de negócio para migrar uma aplicação para a MSA em Cloud Computing? Redução de custos (licenciamento, operação)? Maior velocidade de entrega de produtos?

Fase 2: Portfolio Discovery and Planning

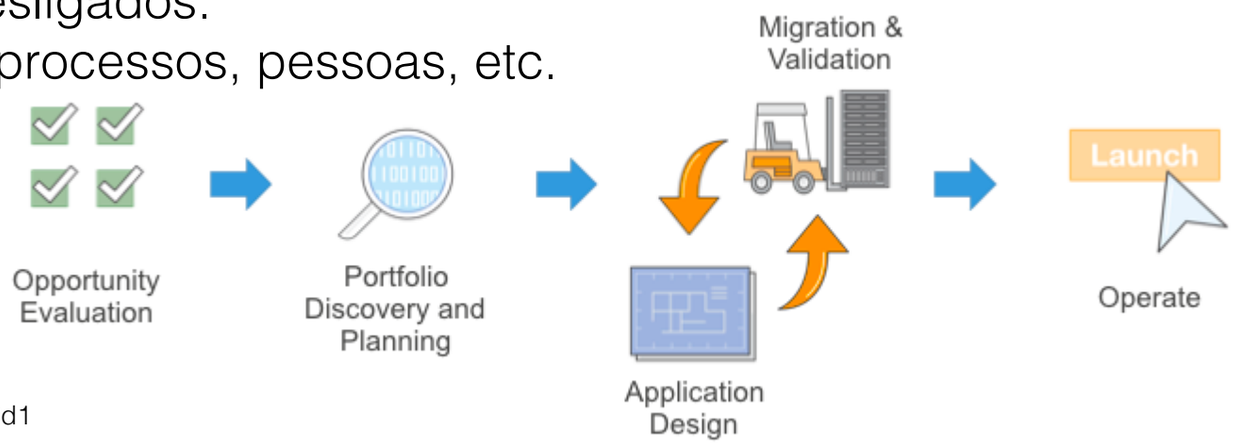
- Quais são as interdependências no seu ambiente de aplicações e sistemas? O que migrará primeiro? Como você migrará? Usando este conhecimento formal obtido através da inspeção dos Configuration Management Databases (CMDBs) por exemplo, é possível a confecção de um plano de migração.
- A complexidade de migração de aplicações existentes varia, dependendo da arquitetura, da forma de contratação, etc. Como exemplo, migração de infraestrutura para Arquitetura Orientada a Serviços (SOA) tem menor complexidade quando comparada com um monólito rodando em mainframes.
- O “lift-and-shift” (“rehosting”) pode parecer mais fácil mas em certos casos deve ser evitado em favor de uma “re-architecture”.

Fases 3 e 4: Designing, Migrating, and Validating Applications

- Fases chamadas de “Fábrica de Migração”.
- O foco desloca-se do nível de portfolio para aplicações individuais.
- Aplica-se, para cada aplicação, uma das 6 estratégias possíveis (Rehosting, Replatforming, Repurchasing, Refactoring, Retire, Retain)

Fase 5: Modern Operating Model

- Depois da migração, os sistemas antigos são desligados.
- Inicia-se o novo modelo de operação, incluindo processos, pessoas, etc.



https://medium.com/aws-enterprise-collection/a-process-for-mass-migrations-to-the-cloud-ee7ff5d7b3d1

ESB -> MSA em Cloud Computing: Processo de Migração

6 estratégias possíveis
Rehosting (“lift-and-shift”)
- Processo “mais simples”
- Pode ser automatizado com ferramentas de Importação/Exportação

Replatforming (“lift-tinker-and-shift”)
- Migração de Banco de Dados On-Premises para DBaaS
- Migração de código Java em WebLogic para equivalente Open Source.

Repurchasing (mover para um produto diferente)
- Migração de CRM para SaaS

Refactoring / Re-architecting (versão nova da aplicação com recursos nativos de Cloud Computing)
- Desenvolvimento de Apps com MSA, FaaS, NoSQL, etc.
- Uso de características não funcionais de Cloud Computing como Elasticidade, Segurança, etc.

Retire (aposentadoria de aplicação)
- Um processo de migração para Cloud Computing é o melhor momento de descartar aplicações obsoletas.

Retain (preservar aplicação)
- A aplicação será mantida e não migrada.

- Para Microservices Architectures o foco está na estratégia de Refactoring/Re-architecting

https://medium.com/aws-enterprise-collection/6-strategies-for-migrating-applications-to-the-cloud-eb4e85c412b4

The Zachman Framework for Enterprise Architecture

Generic Classification Structure of Design Artifacts

	What	How	Where	Who	When	Why	
Planner							Scope
Owner							Concepts
Designer							Logic
Builder							Physics
Imple-menter							Technology
Operator			THE ENTERPRISE				Product
	Material	Process	Geometry	Instructions.	Timing	Objectives	

The Zachman Framework for Enterprise Architecture
https://www.zachman.com/resources/ea-articles-reference/327-the-framework-for-enterprise-architecture-background-description-and-utility-by-john-a-zachman