APRESENTAÇÃO
DE APOIO

WEB SERVICES

Cássio Trindade – Aula 01

PUCRS online uol edtech

Professores

CÁSSIO TRINDADE
Professor Convidado

Profissional da área de TI, trabalhando há mais de uma década com a formação de profissionais, dando aulas no Instituto Federal do Rio Grande do Sul, na Faculdade Dom Bosco, Universidade Luterana do Brasil (ULBRA), Pontifícia Universidade Católica do RS (PUCRS) e na TargetTrust. Atualmente, atuando como Arquiteto de Software na PUCRS, sendo responsável pela condução e elaboração de mais de 90 projetos diretamente com alunos do curso de Engenharia de Software, trabalhando com as mais variadas tecnologias. Mais de 30 anos de experiência nas áreas de desenvolvimento de software, aplicativos para celulares e sistemas corporativos para internet desde projetos de e-commerce para o Sonae Portugal e site de classificados digitais do Grupo RBS a dezenas de aplicativos mobiles.

MIGUEL GOMES XAVIER
Professor PUCRS

Possui mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul e está cursando doutorado em Ciência da Computação na mesma instituição, atuando principalmente nas areas de alto desempenho, sistemas distribuídos, virtualização e cloud computing. Atualmente participa de projetos de pesquisa em cooperação com diferentes universidades envolvendo gerência de recursos em arquiteturas de alto desempenho. Tem participado de projetos de análise de dados (BigData), realizando contribuições científicas em prol do avanço da área na industria e na academia.

Ementa da disciplina

Estudo sobre conceitos de arquitetura monolítica. Revisão dos conceitos sobre SOAP, REST, GraphQL e descritores de serviços. Estudo sobre soluções serveless. Construção de soluções com framework REST e framework GraphQL.

PUCRS online

Desenvolvimento Full Stack

Web Services

Miguel Xavier

miguel.xavier@pucrs.br

Áreas de Interesse e Pesquisa

- **Formação** e **capacitação**
  - Mestrado e Doutorado (PPGCC/PUCRS)

- Projetos de **pesquisa, desenvolvimento e inovação**
  - Cloud Computing, Internet of Things (IoT), Big data e HPC, Virtualização e Redes de Computadores

PUCRS ESCOLA POLITÉCNICA | CePES | Ages AGÊNCIA EXPERIMENTAL DE ENGENHARIA DE SOFTWARE | PUCRS Ideia LabLAD

https://www.linkedin.com/in/miguel-xavier/

Agenda

- Introdução
- A Necessidade de Integrar Sistemas
- Arquiteturas Distribuídas
- Web Services
  - Tipos de Web Services
- Application Programming Interface (API)
  - SOAP
  - REST
  - GraphQL
  - gRPC
- Use Cases
  - Microservices
  - Serverless

Introdução

Introdução

- A evolução da internet tem se manifestado de muitas formas: as características de tráfego, a interconexão entre topologias, o relacionamento entre negócios e componentes autônomos, a facilidade do acesso à informação, entre outros.

- É importante compreender o por quê (e como) dessa evolução estar acontecendo, e como isso pode afetar drasticamente a forma que os usuários navegam, a interação das empresas que lidam com aplicações web e com os desenvolvedores dessas aplicações (MA; LUI; MISRA, 2015).

Introdução

- A forma mais comum utilizada para criar esses serviços é através do desenvolvimento de aplicações web, o qual o usuário é capaz de interagir através de entrada de dados (clicks, textos em formulários, arquivos) e esperam receber uma saída de dados ou um feedback dessa aplicação (MULLOY, 2012).

- Até o presente momento, essas aplicações foram desenvolvidas utilizando tecnologias como REST e SOAP. O problema de utilizar essas tecnologias é que às vezes exige um nível abstração baixo e sem controle na resposta das requisições. Enquanto com GraphQL a aplicação pega somente o que é necessário para ela. Isso reduz a carga de dados enviados do servidor para o navegador.

Introdução

- Até o presente momento, essas aplicações foram desenvolvidas utilizando tecnologias como REST e SOAP.

- O problema de utilizar essas tecnologias é que às vezes exige um nível abstração baixo e sem controle na resposta das requisições. Enquanto com GraphQL a aplicação pega somente o que é necessário para ela. Isso reduz a carga de dados enviados do servidor para o navegador.

Web Services

Fundamentos

Definição

- Ian Sommerville, Engenharia de Software, 8ed.:
  - “Um Web Service é um serviço, ou seja, um componente de software independente e fracamente acoplado que engloba funcionalidade discreta que pode ser distribuída e acessada por meio de uma aplicação, através de protocolos padrão.”

Definição

- W3C:
  - “Um Web Service é um sistema de software cujo propósito é suportar de maneira interoperável a interação máquina-máquina sobre uma rede de comunicação. Ele possui uma interface descrita em um formato processável por máquinas. Outros sistemas interagem com ele de acordo com a interface através de mensagens, tipicamente sobre um protocolo padrão da internet via serialização em conjunto com outros padrões web relacionados.”

Definição

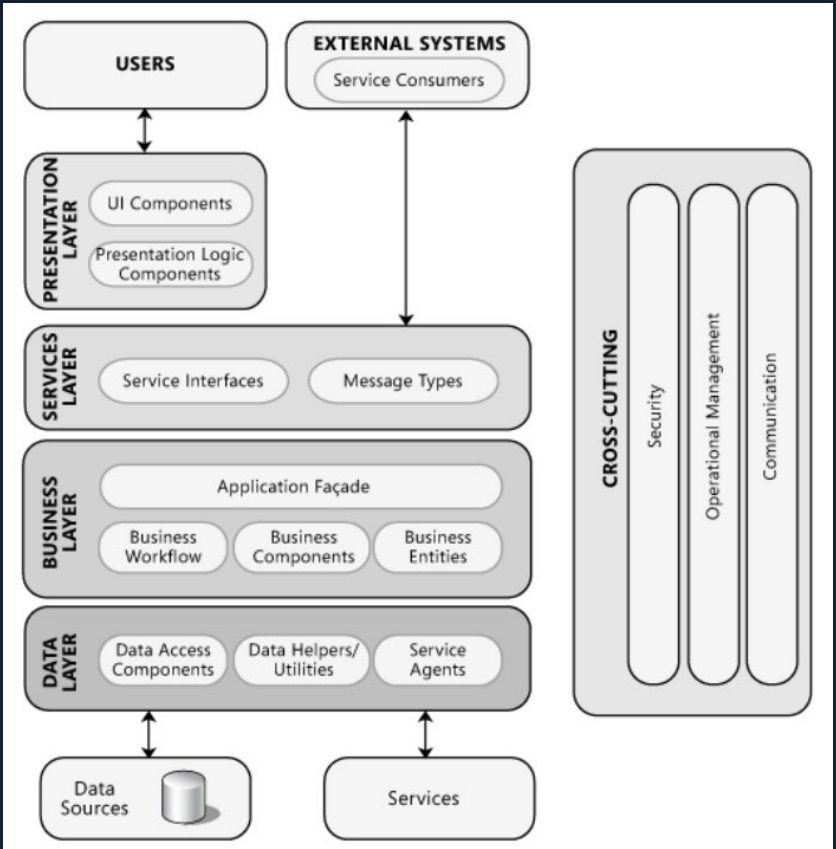
- Alonso et al, Web Services – concepts, architectures and applications:
  - “A way to expose the functionality of an information system and make it available through standards Web technologies.”

Características

- Objetos remotos
- Residem em um servidor Web e têm um endereço URL
- Trabalham sobre o modelo de requisição/resposta
- Utilizam protocolos que facilitam a comunicação entre sistemas
  - Independente do sistema operacional e da linguagem de programação (web services interoperáveis)
- São objetos para soluções fracamente acopladas

Arquiteturas Web

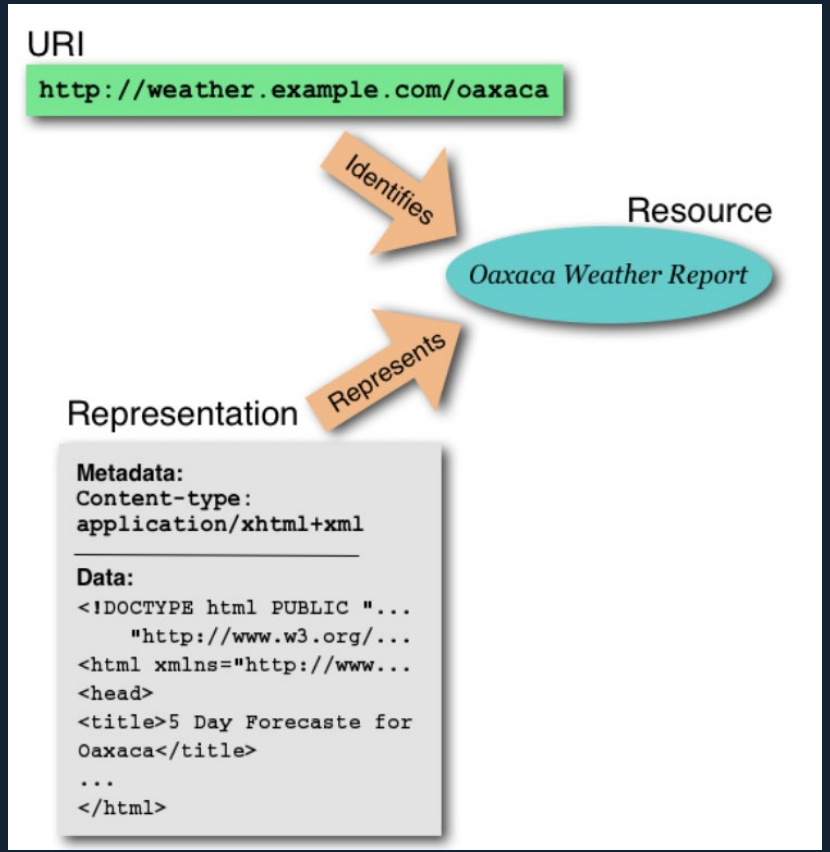
Arquitetura



Web

- De acordo com o W3C:
  - “A World Wide Web (WWW) é um espaço de informações no qual os itens de interesse, referidos como recursos, são identificados por identificadores globais chamados Uniforme Resource Identifiers (URI)”.

Web


URI
http://weather.example.com/oaxaca
Identifies
Resource
Oaxaca Weather Report
Represents
Representation
Metadata:
Content-type:
application/xhtml+xml
Data:
<!DOCTYPE html PUBLIC "...
"http://www.w3.org/...
<html xmlns="http://www...
<head>
<title>5 Day Forecaste for
Oaxaca</title>
...
</html>

Protocolo HTTP

Comunicar-se com servidores e aplicativos web se dá através do protocolo Hypertext Transfer Protocol

- Protocolo de nível de aplicação
- Protocolo textual
- Protocolo baseado em mensagens de requisição/resposta no modelo cliente/servidor
- Protocolo sem manutenção de estado
- Versões (em uso):
- 1.1 (RFCs 7230, 7231, 7232, 7233, 7234, 7235)
- 2 (RFC 7540)

Protocolo HTTP

Um URL Uniform Resource Locator é utilizado para identificar elementos em um servidor web

- Ex.: http://java.sun.com/index.html
- Formato geral:

"http:" "//" host [":" port] [path ["?" query]]

Protocolo HTTP

- Uma requisição HTTP consiste de:
  - Uma linha inicial
  - Um ou mais campos de cabeçalho
  - Uma linha em branco
  - Possivelmente um corpo da mensagem
- Uma resposta HTTP consiste de:
  - Uma linha de status com seu código (ver RFC, Wikipédia) e mensagem associada
  - Um ou mais campos de cabeçalho
  - Uma linha em branco
  - Possivelmente um corpo da mensagem

Protocolo HTTP

- Alguns métodos (também chamados de verbos):

GET	Solicita um recurso ao servidor
POST	Fornece a entrada para um comando do lado do servidor e devolve o resultado
PUT	Envia um recurso ao servidor
DELETE	Exclui um recurso do servidor
TRACE	Rastreia a comunicação com o servidor

Protocolo HTTP

HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Protocolo HTTP

- São dois os comandos mais utilizados para fornecer entrada de dados aos programas no lado servidor:
  - GET
  - POST

Protocolo HTTP

- GET:
  - Método mais simples
  - Quantidade de dados muito limitada
    - Limite implementado nos navegadores
  - Dados acrescentados à URL após um caractere “?”, no formato “campo=valor”, separados pelo caractere “&”
    - Recebe o nome de query-string
  - Ex.: http://www.aqui.com/prog?id=1&ano=2008

Protocolo HTTP

- Requisição:
  - GET /index.html HTTP/1.1
  - Host: www.example.com

Protocolo HTTP

- POST:
  - Método mais robusto
  - Quantidade de dados não é limitada como no GET
  - Dados (query-string) enviados no corpo da requisição do protocolo
  - Permite efeitos colaterais na execução no lado do servidor

Protocolo HTTP

- Requisição:

```
POST /index.html HTTP/1.0
Accept: text/html
If-modified-since: Sat, 29 Oct 1999 19:43:31 GMT
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

Nome=NomePessoa&Idade=20&Curso=Computacao
```

Protocolo HTTP

- Resposta:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

WEB SERVICES

SOAP + XML

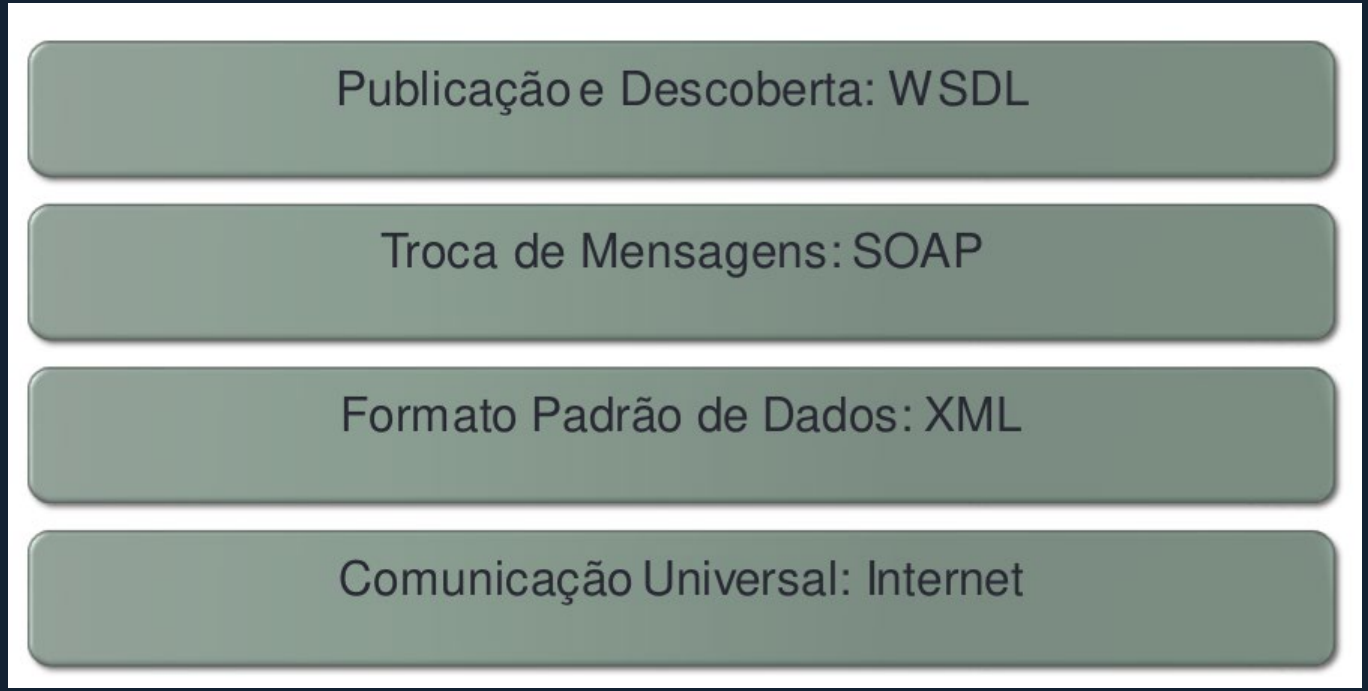
Arquitetura Básica

- Web Services (tipo SOAP+XML) provêm meios de objetos interagirem utilizando a Internet como meio de transmissão
- Baseado em diversos padrões:
  - Extensible Markup Language (XML)
  - SOAP
  - Web Services Description Language (WSDL)
  - Hypertext Transfer Protocol (HTTP)
  - Etc

Arquitetura Básica

- Utilizam um modelo de chamada remota de procedimentos
- Provedores de serviços projetam e implementam serviços e os especificam em uma linguagem chamada WSDL
- Provedores de serviço publicam informações sobre esses serviços em um serviço de registro
- Os solicitantes de serviços, que desejam fazer uso de um serviço, buscam o registro para descobrir a especificação do serviço e para localizar o provedor do serviço
- O solicitante do serviço pode então vincular sua aplicação a um serviço específico e se comunicar com ele através de um protocolo como o SOAP
- Interoperável sobre diferentes protocolos de transporte

Arquitetura Básica



Arquitetura Básica

- Exemplos:
  - http://www.gxchart.com/service/webchart.asmx
  - http://fc.isima.fr/~lacomme/ORWebServices/index.php?idx=1
  - http://www.thomas-bayer.com/soap/csv-xml-converter-webservice.htm

Arquitetura Básica

- Comunicação:
  - Protocolo HTTP para envio e recebimento de dados (é um dos mais utilizados)
    - GET dados enviados via query string na URL
    - POST dados enviados no corpo da mensagem
    - Sem manutenção de estado
    - Recursos identificados por URL (“Uniform Resource Locator”)

Arquitetura Básica

- Dados:
  - Informações serializadas em XML
- Troca de Mensagens:
  - Mensagens para objetos remotos via protocolo SOAP
  - Envelopes SOAP encapsulam dados XML
    - Nome do método
    - Parâmetros do método
    - Valores de retorno

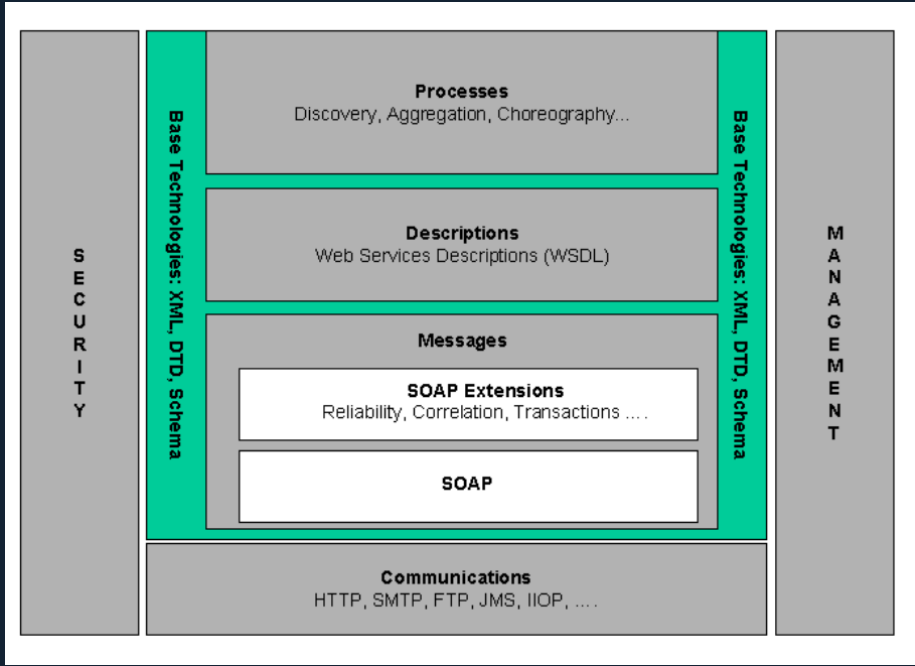
Arquitetura Básica

- Descrição:
  - Arquivo WSDL descreve as mensagens e tipos de retorno do web service
  - É um documento XML

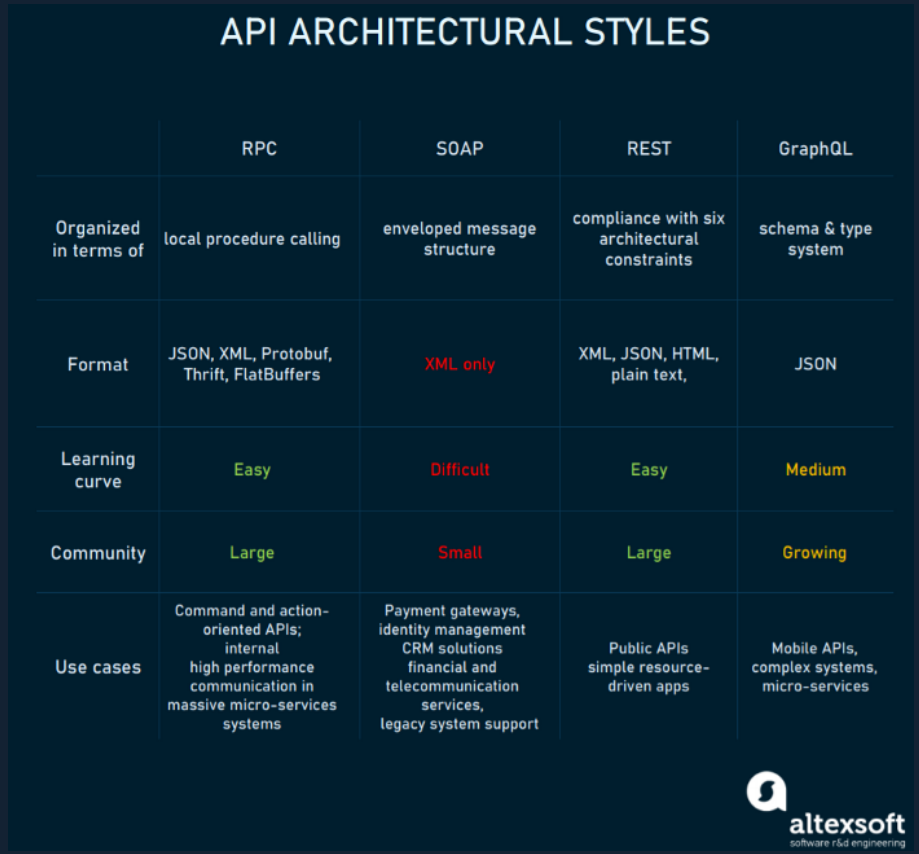
Arquitetura Básica

- A arquitetura para disponibilizar web services pode ser bastante complexa
  - Grande conjunto de protocolos e camadas de serviços adicionais WS-*
- Referência:
- http://www.w3.org/TR/ws-arch/

Arquitetura Básica



Arquiteturas de API



API ARCHITECTURAL STYLES

	RPC	SOAP	REST	GraphQL
Organized in terms of	local procedure calling	enveloped message structure	compliance with six architectural constraints	schema & type system
Format	JSON, XML, Protobuf, Thrift, FlatBuffers	XML only	XML, JSON, HTML, plain text,	JSON
Learning curve	Easy	Difficult	Easy	Medium
Community	Large	Small	Large	Growing
Use cases	Command and action-oriented APIs; internal high performance communication in massive micro-services systems	Payment gateways, identity management CRM solutions financial and telecommunication services, legacy system support	Public APIs simple resource-driven apps	Mobile APIs, complex systems, micro-services

altexsoft
software r&d engineering

Arquitetura Básica

Business Domain Specific extensions	Various	Business Domain
Distributed Management	WSDM, WS-Manageability	Management
Provisioning	WS-Provisioning	
Security	WS-Security	Security
Security Policy	WS-SecurityPolicy	
Secure Conversation	WS-SecureConversation	
Trusted Message	WS-Trust	
Federated Identity	WS-Federation	
Portal and Presentation	WSRP	Portal and Presentation
Asynchronous Services	ASAP	Transactions and Business Process
Transaction	WS-Transactions, WS-Coordination, WS-CAF	
Orchestration	BPEL4WS, WS-CDL	
Events and Notification	WS-Eventing, WS-Notification	Messaging
Multiple message Sessions	WS-Enumeration, WS-Transfer	
Routing/Addressing	WS-Addressing, WS-MessageDelivery	
Reliable Messaging	WS-ReliableMessaging, WS-Reliability	
Message Packaging	SOAP, MTOM	
Publication and Discovery	UDDI, WSIL	Metadata
Policy	WS-Policy, WS-PolicyAssertions	
Base Service and Message Description	WSDL	
Metadata Retrieval	WS-MetadataExchange	

WEB SERVICES

REST + JSON

REST

- Representational State Transfer é um estilo arquitetural
  - https://www.ics.uci.edu/~fielding/pubs/dissertation/top.hm
- Web services baseados em REST são usualmente chamados de RESTfull

REST

- Características:
  - Serviços sem estado
  - Baseados no protocolo HTTP/HTTPS
  - Dados e funcionalidades são considerados recursos acessados via URIs
  - Infraestrutura mais leve que SOAP+XML

REST

- Exemplos:
  - http://predic8.com/rest-demo.htm
  - http://jsonplaceholder.typicode.com/
  - https://randomuser.me/
  - http://petstore.swagger.io/

REST

- Arquitetura baseada em quatro princípios:
  - Identificação dos recursos através de URIs – Uniform Resource Indentifiers
  - Interface uniforme de acesso aos recursos
    - Operações de criação, leitura, alteração e remoção
    - Implementadas via HTTP
  - Mensagens autodescritivas
  - Iteração com manutenção de estado através de hiperlinks

REST

- Questões para o desenvolvedor:
  - Definir quais são os “recursos” expostos
  - Definir o formato das URIs para os recursos
  - Decidir quais verbos do HTTP serão realmente utilizados
  - Estabelecer a real semântica da aplicação de cada verbo sobre um recurso

REST

- Exemplo: URI de coleção
  - http://exemplo/recursos/
  - GET: lista as URIs e outros detalhes dos elementos da coleção
  - PUT: substitui a coleção por uma outra
  - PATCH: não é muito utilizado
  - POST: adiciona um novo elemento na coleção, retornando a URI para o novo elemento
  - DELETE: remove a coleção inteira

REST

- Exemplo: URI de elemento
  - http://exemplo/recursos/123
  - GET: obtém a representação de um elemento específico da coleção
  - PUT: substitui um membro específico da coleção ou, se ele não existe, cria um novo
  - PATCH: atualiza (podendo ser só uma parte) do membro específico da coleção
  - POST: geralmente não utilizado; trata o elemento da coleção como
  - uma própria coleção, adicionando um novo elemento nele
  - DELETE: remove o elemento da coleção

REST

- Serialização de informação em vários formatos:
  - XML
  - JSON – JavaScript Object Notation
  - Texto
  - etc

JSON

- JSON = JavaScript Object Notation
- Formato textual para serialização de dados
- Documentação: http://json.org/

```
{
  "productName": "Computer Monitor",
  "price": "229.00",
  "specifications": {
     "size": 22,
     "type": "LCD",
     "colors": ["black", "red", "white"]
  }
}
```

REST

- JSON é capaz de representar:
  - Tipos primitivos
    - Strings, números, booleanos, null
  - Tipos estruturados
    - Objetos
      - Coleção não-ordenada de zero ou mais pares chave/valor
    - Arranjos
      - Coleção ordenada de zero ou mais valores

WEB SERVICES

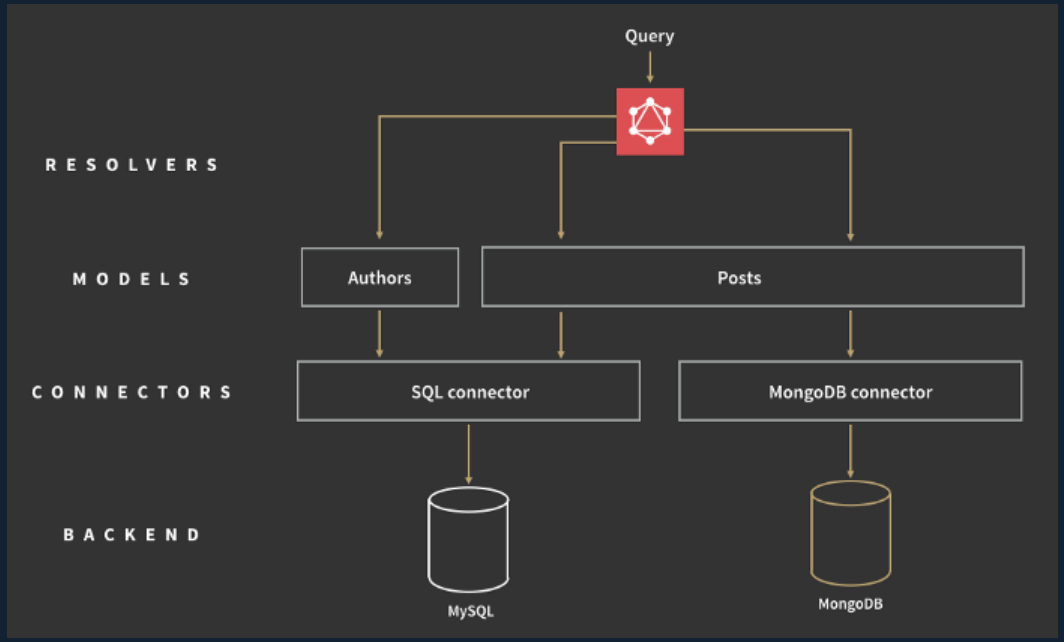
Evolução com GraphQL

GraphQL

- O GraphQL começa com a construção de um esquema, que é uma descrição de todas as consultas que você pode fazer em uma API do GraphQL e todos os tipos que eles retornam.

- O cliente pode validar sua consulta para garantir que o servidor possa responder a ela.
- Uma operação do GraphQL é interpretada em todo o esquema e resolvida com dados para o aplicativo de front-end. Enviando uma consulta massiva ao servidor, a API retorna uma resposta JSON com exatamente a forma dos dados que solicitamos.

GraphQL


Query
RESOLVERS
MODELS
Authors
Posts
CONNECTORS
SQL connector
MongoDB connector
BACKEND
MySQL
MongoDB

GraphQL

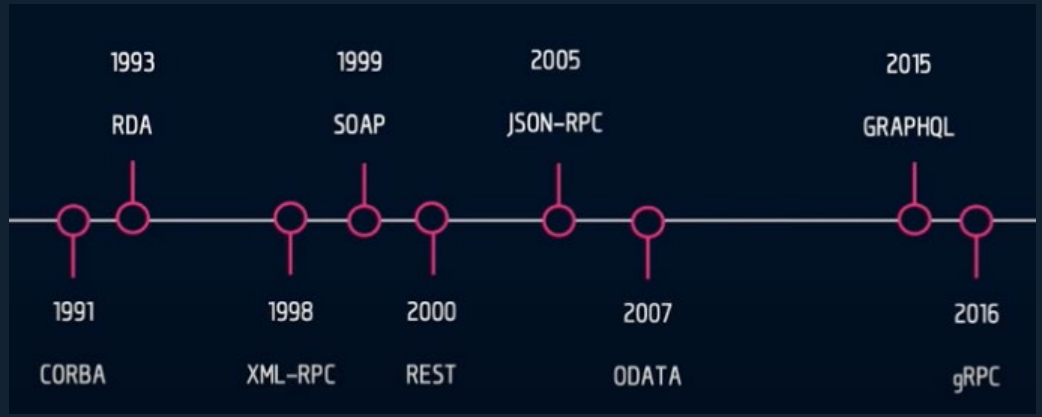
- O GraphQL começa com a construção de um esquema, que é uma descrição de todas as consultas que você pode fazer em uma API do GraphQL e todos os tipos que eles retornam.

- O cliente pode validar sua consulta para garantir que o servidor possa responder a ela.
- Uma operação do GraphQL é interpretada em todo o esquema e resolvida com dados para o aplicativo de front-end. Enviando uma consulta massiva ao servidor, a API retorna uma resposta JSON com exatamente a forma dos dados que solicitamos.

Casos de uso do GraphQL

- API mobile:
  - Nesse caso, o desempenho da rede e a otimização da carga útil da mensagem única são importantes. Assim, o GraphQL oferece um carregamento de dados mais eficiente para dispositivos móveis.
- Sistemas complexos e microsserviços:
  - O GraphQL é capaz de ocultar a complexidade da integração de vários sistemas por trás de sua API. Agregando dados de vários lugares, ele os mescla em um esquema global. Isso é particularmente relevante para infraestruturas legadas ou APIs de terceiros que se expandiram ao longo do tempo.

Linha do tempo



Tecnologias de Suporte

- Descrição e descoberta de serviços:
  - WADL (Web Application Description Language)
    - https://wadl.java.net/
    - Especificação de linguagem para descrição de serviços REST
    - Criada inicialmente para implementação JavaEE
  - RAML (RESTful API Modeling Language)
    - http://raml.org/
    - Especificação de linguagem para descrição de serviços REST
  - Swagger
    - http://swagger.io/
    - Especificação de linguagem para descrição de serviços REST
    - Formato de arquivo JSON (http://json.org/) e YAML (http://yaml.org/)
  - OpenAPI
    - https://www.openapis.org/

WEB SERVICES

Qual padrão eu devo usar?

Qual padrão eu devo usar?

- Cada projeto de API tem requisitos e necessidades diferentes.
- Depende da linguagem de programação em uso, o ambiente em que você está desenvolvendo, e os recursos que você tem de sobra, tanto humanos quanto financeiros.
- Conhecendo todas as vantagens de cada estilo de design, os designers de API podem escolher aquele que melhor se encaixa no projeto.

Qual padrão eu devo usar?

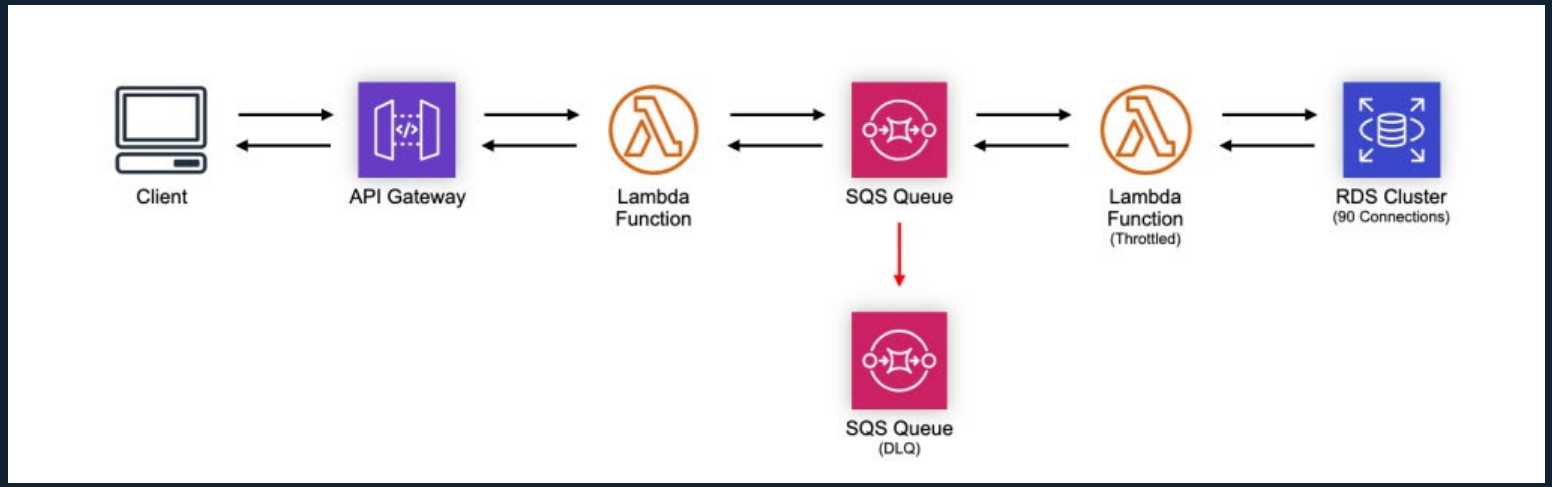
- Com seu acoplamento rígido, o RPC funciona para microsserviços internos, mas não é uma opção para uma API externa forte ou um serviço de API.
- O SOAP é problemático, mas seus ricos recursos de segurança permanecem insubstituíveis para operações de cobrança, sistemas de reservas e pagamentos.
- REST tem a maior abstração e melhor modelagem da API. Mas tende a ser mais pesado no fio e mais tagarela – uma desvantagem se você estiver trabalhando no celular.
- O GraphQL é um grande passo à frente em termos de busca de dados, mas nem todos têm tempo e esforço suficientes para pegar o jeito.

Qual padrão eu devo usar?

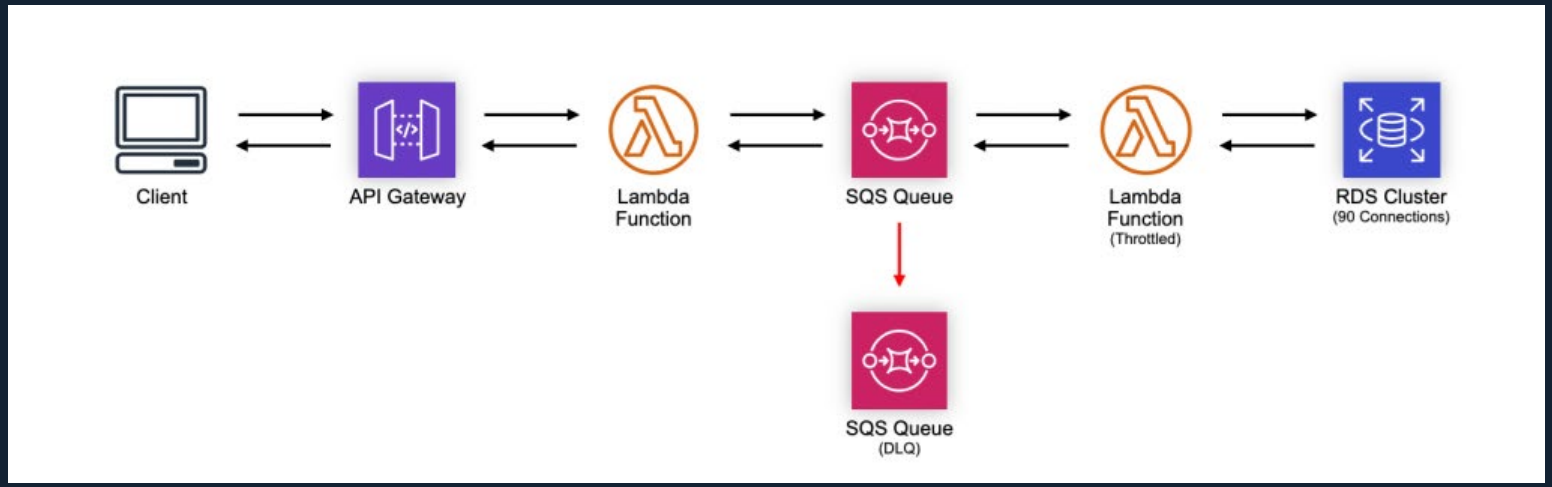
No final das contas, faz sentido experimentar alguns pequenos casos de uso com um estilo específico e ver se ele se encaixa no seu caso de uso e resolve seus problemas.

Exemplo: Arquitetura Serverless - Básica

Exemplo: Arquitetura Serverless - Webhook



Exemplo: Arquitetura Serverless - Agregação



Importante ! API vs. Web Services

- Serviços da Web e APIs são frequentemente confundidos entre si, o que não é tão surpreendente, pois há alguns pontos em comum distintos.

- A maioria dos serviços da Web fornece uma API, que, com seu conjunto de comandos e funções, é usada para recuperar dados. Aqui está um exemplo: o Twitter oferece uma API que autoriza um desenvolvedor a acessar tweets de um servidor e, em seguida, coleta dados no formato JSON.

PUCRS online

Desenvolvimento Full Stack

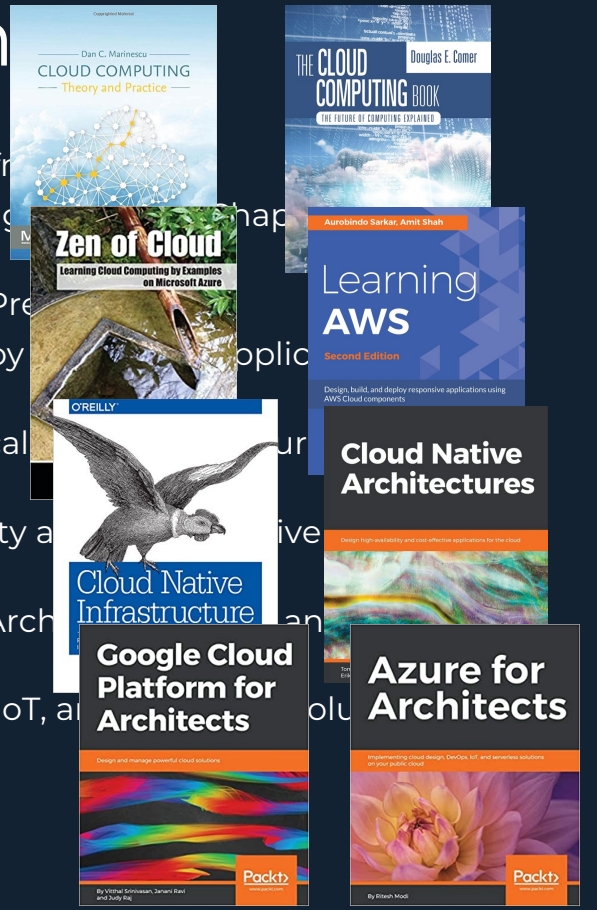
Web Services

Miguel Xavier

miguel.xavier@pucrs.br

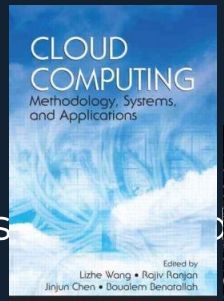
Referências – Cloud Computin

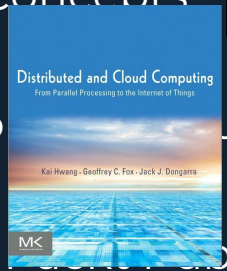
1. MARINESCU, Dan C. Cloud computing: theory and practice. Morgan Kaufr
2. COMER, Douglas E. The Cloud Computing Book: The Future of Computing Hall/CRC, 2021.
3. BAI, Haishi. Zen of Cloud: Learning Cloud Computing by Examples. CRC Pre
4. SARKAR, Aurobindo; SHAH, Amit. Learning AWS: Design, build, and deploy using AWS Cloud components. Packt Publishing Ltd, 2018.
5. GARRISON, Justin; NOVA, Kris. Cloud native infrastructure: Patterns for scal applications in a dynamic environment. O'Reilly, 2017.
6. LASZEWSKI, Tom et al. Cloud Native Architectures: Design high-availability a applications for the cloud. Packt Publishing Ltd, 2018.
7. SRINIVASAN, Vitthal; RAVI, Janani; RAJ, Judy. Google Cloud Platform for Arch powerful cloud solutions. Packt Publishing Ltd, 2018.
8. MODI, Ritesh. Azure for architects: Implementing cloud design, DevOps, IoT, a your public cloud. Packt Publishing Ltd, 2017.

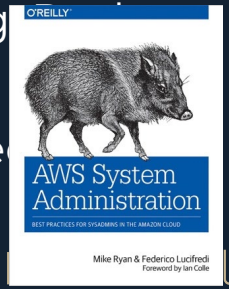


Referências – Cloud Computing

9. BENATALLAH, B. Cloud Computing – Methodology, Sys… Applications. CRC Press. 2011.
10. ERL, T.; PUTTINI, R.; MAHMOOD, Z. Cloud Computing: Concepts, Technology & Architecture. Prentice Hall. 2013.
11. FOX, G; DONGARRA, J.; HWANG, K. Distributed and Clo… Morgan Kaufmann. 2013.
12. JACKSON, K. OpenStack Cloud Computing Cookbook. … Ltd. 2012.
13. SABHARWAL, N; SHANKAR R. S.. Apache CloudStack Computing … Publishing Ltd. 2013.
14. LUCIFREDI, F; RYAN, M. AWS System Administration. O'Reilly Me… 2018








PUCRS online
uol edtech