Webservices Technologies

Lecture handouts

Feb 2020

Definition of Web Service

A **Web service** is an interface that describes a collection of operations that are **network-accessible** through standardized **XML** messaging.

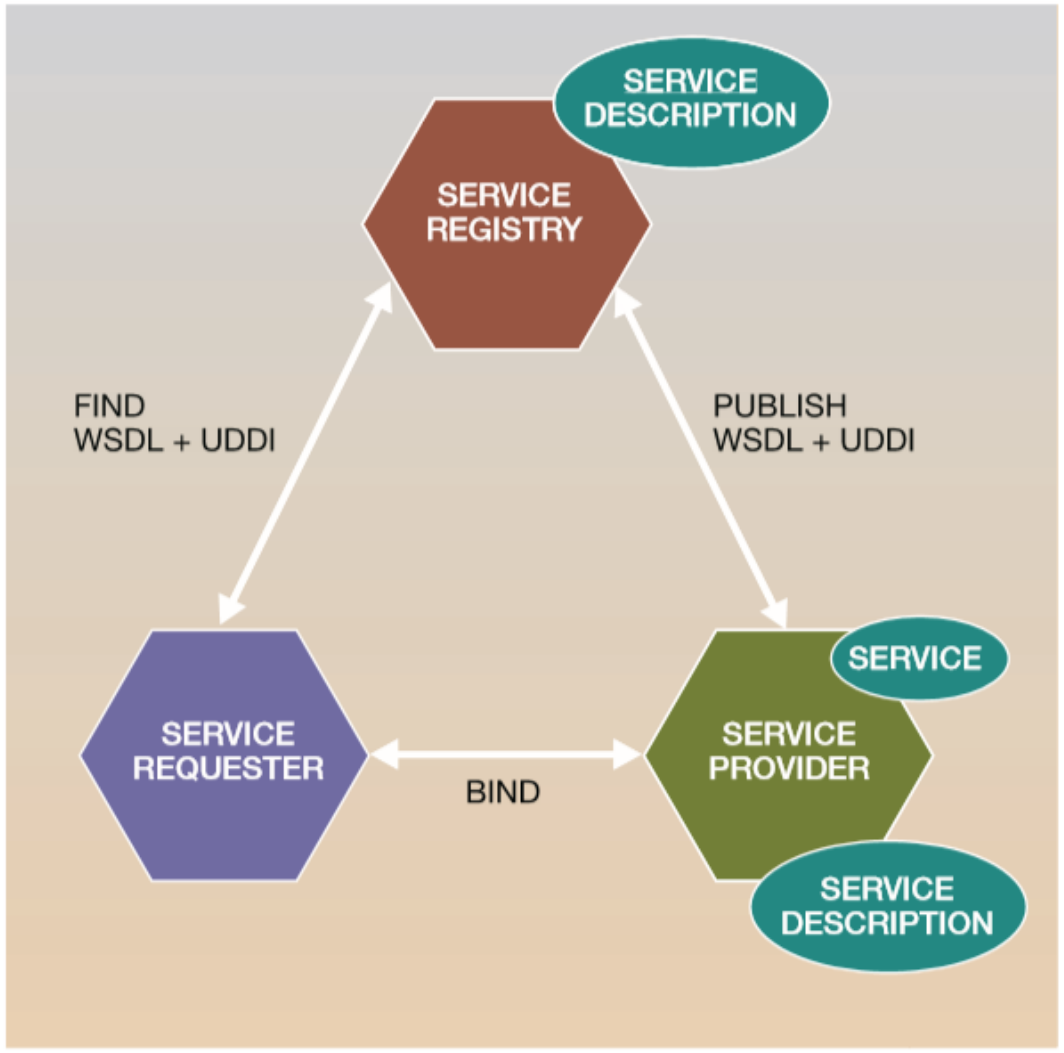
Web services built on existing and emerging standards such as HyperTextTransferProtocol (HTTP), Extensible Markup Language (XML), SimpleObjectAccessProtocol(SOAP), WebServices Description Language (WSDL), and the Universal Description, Discovery, and Integration(UDDI)project.

A Web service performs a specific task or set of tasks.
Web service is describes dusingastandard,formalXMLnotation,calledit service description,that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. Web service descriptions are expressed in WSDL.

Characteristics of Web Service

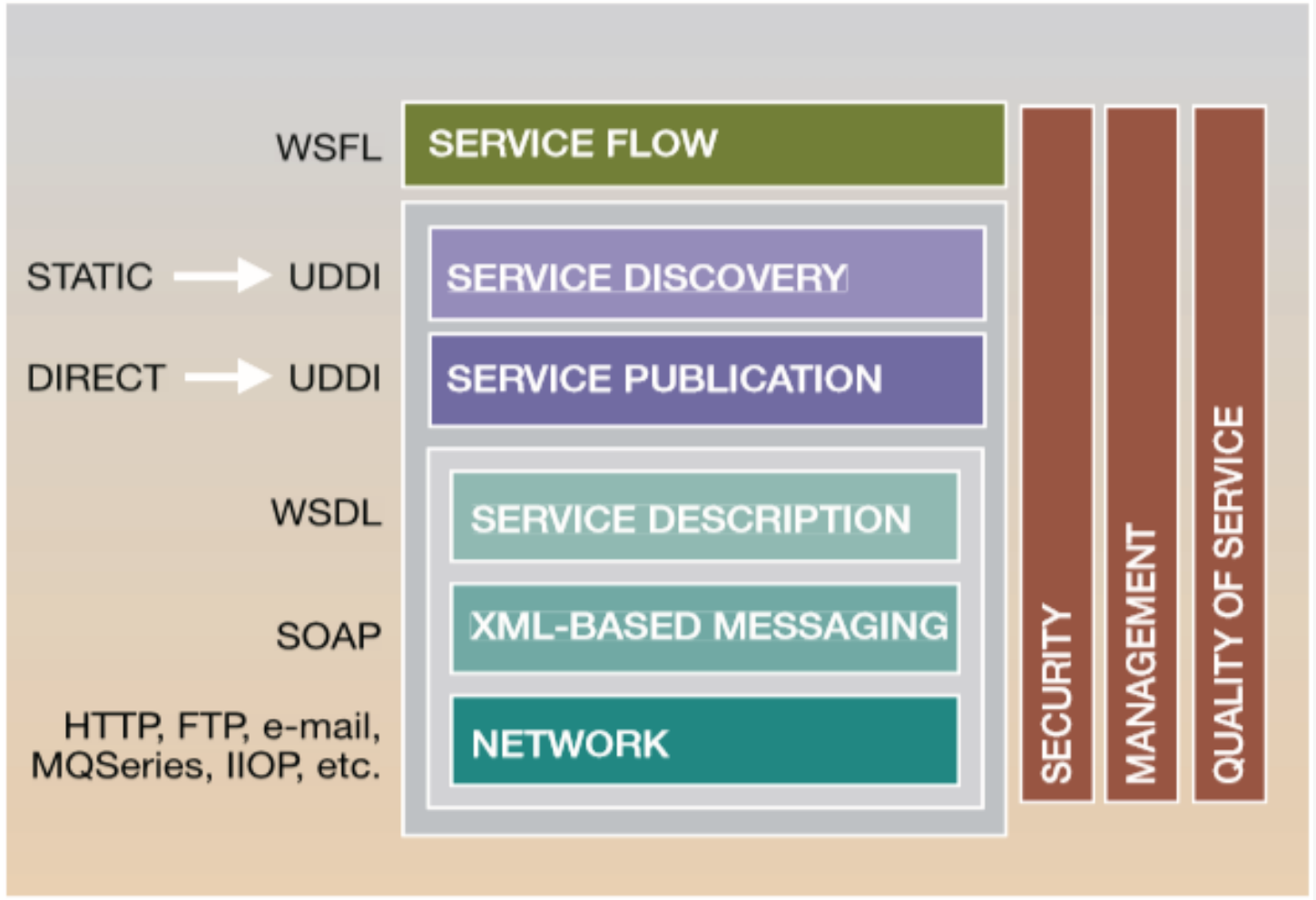
- Available over the internet or private (intranet) networks
- Uses a standardized xml messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common xml grammar
- Is discoverable via a simple find mechanism

Web services actors, objects, and operation



Gottschalk, Karl, et al. "Introduction to web services architecture." *IBM systems Journal* 41.2 (2002): 170-177.

Web services programming stack



Gottschalk, Karl, et al. "Introduction to web services architecture." *IBM systems Journal* 41.2 (2002): 170-177.

Webservice Architecture

Web Service Roles

❑Service Provider

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

❑Service Requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

❑Service Registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

Web Service Protocol Stack

has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.
Service Description

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).
As web services evolve, additional layers may be added and additional technologies may be added to each layer.
The next chapter explains the components of web services.

How Does a Web Service Work?

- A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −
- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.
- You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.
- You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

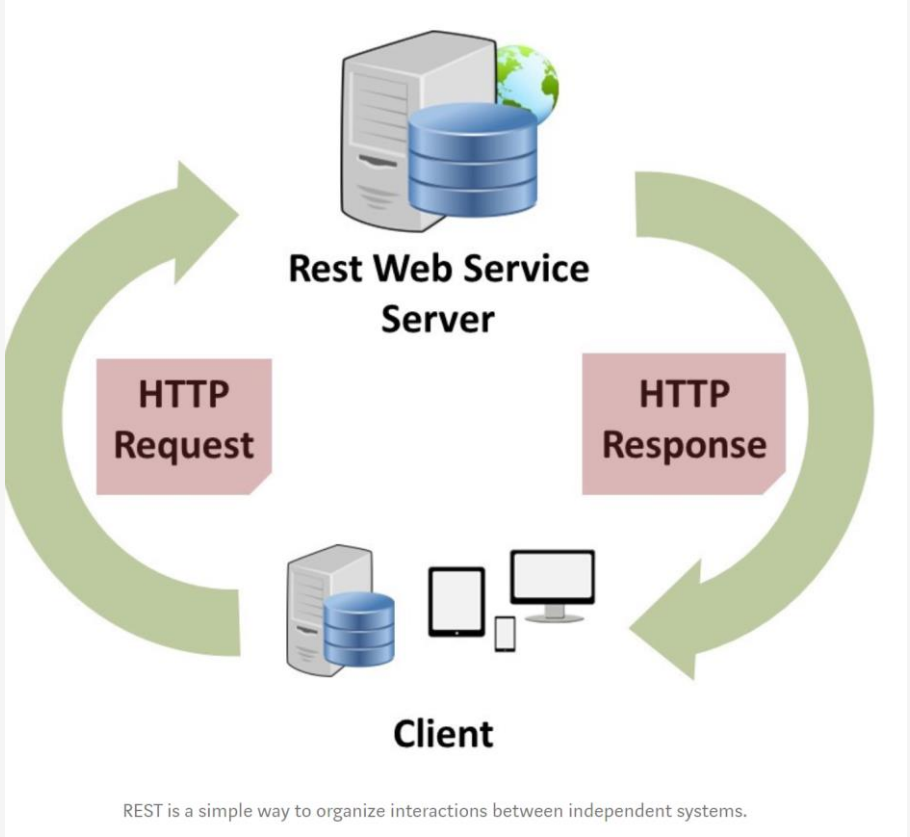
The steps to perform this operation are as follows −

Example

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.
- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

Introduction to RESTful Services

REST is ***an architecture style*** for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.



Source: https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc

Benefits of REST

RESTful as lightweight Web Services: The RESTful architecture was a reaction to the more heavy-weight SOAP-based standards. In REST web services, the emphasis is on simple point-to-point communication over HTTP using plain XML. In addition, RESTful permits many different data formats whereas SOAP only permits XML.

Scalability: As RESTful forbids conversational state, which means we can scale very wide by adding additional server nodes behind a load balancer.

The simplicity of RESTful: The RESTful architecture is much simpler to develop than SOAP. One of the main reasons for REST popularity is the simplicity and ease of use, as it does an extension of native Web technologies such as HTTP.

Expose APIs as HTTP Services: When developers need the universal presence with minimum efforts, given the fact that RESTful APIs are exposed as HTTP Services, which is virtually present on almost all the platforms.

RESTful architecture is closer in design to the Web: RESTful is the architectural style of the web itself, so the developer with knowledge in web architecture will naturally develop in the RESTful architecture.

Architectural Constraints

1. Interface / Uniform Contract: Once a developer becomes familiar with one of your API, he should be able to follow the similar approach for other APIs.
2. Client-Server: Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.
3. Stateless: No client context shall be stored on the server between requests. The client is responsible for managing the state of the application.

The server will not store anything about the latest HTTP request the client made. It will treat every request as new. No session, no history.
If the client application needs to be a stateful application for the end-user, where user logs in once and do other authorized operations after that, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.

4. Cache: Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.

Source: https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc

Architectural Constraints

5. Layered System

layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting. By restricting knowledge of the system to a single layer, we place a bound on the overall system complexity and promote substrate independence

6. Code-On-Demand (optional)

REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. Allowing features to be downloaded after deployment improves system extensibility. However, it also reduces visibility, and thus is only an optional constraint within REST.

REST Verbs

Methods or verbs commonly used

1. GET – Provides a read only access to a resource.
2. POST – Used to create a new resource.
3. DELETE – Used to remove a resource.
4. PUT – Used to update a existing resource or create a new resource.

Sr.No.	URI	HTTP Method	POST body	Result
1	/UserService /users	GET	empty	Show list of all the users.
2	/UserService /addUser	POST	JSON String	Add details of new user.
3	/UserService /getUser/:id	GET	empty	Show details of a user.

Source: https://www.tutorialspoint.com/restful/index.htm

Uniform Interface

Resource identification in requests

Individual resources are identified in requests, for example using URIs in RESTful Web services. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server could send data from its database as HTML, XML or as JSON—none of which are the server's internal representation.

Resource manipulation through representations

When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.

Self-descriptive messages

Each message includes enough information to describe how to process the message. For example, which parser to invoke can be specified by a media type.[3]

Hypermedia as the engine of application state (HATEOAS)

Having accessed an initial URI for the REST application—analogous to a human Web user accessing the home page of a website—a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs. As access proceeds, the server responds with text that includes hyperlinks to other actions that are currently available. There is no need for the client to be hard-coded with information regarding the structure or dynamics of the application.

HTTP Response Codes

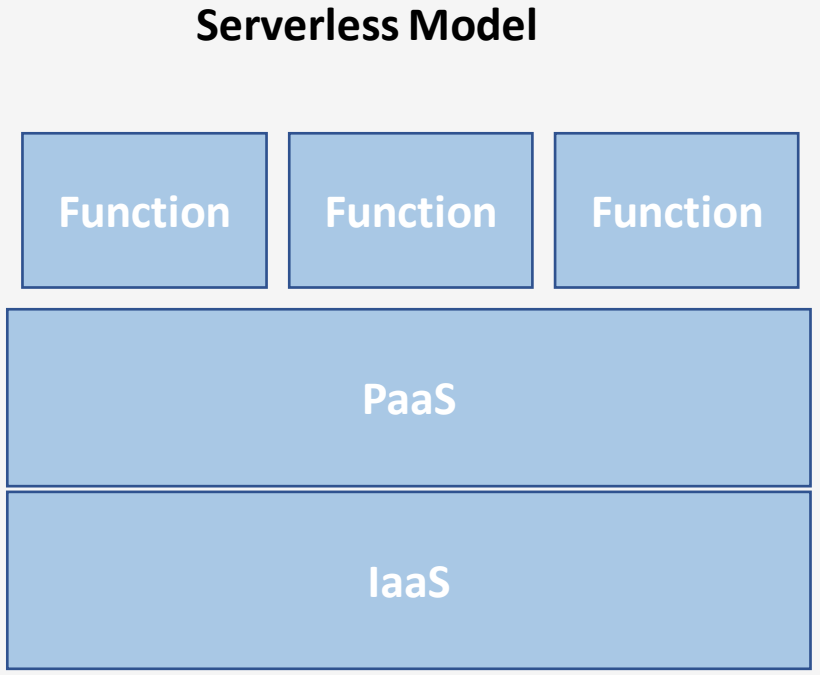
CATEGORY	DESCRIPTION
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

Source: https://restfulapi.net/http-status-codes/

Serverless Architecture

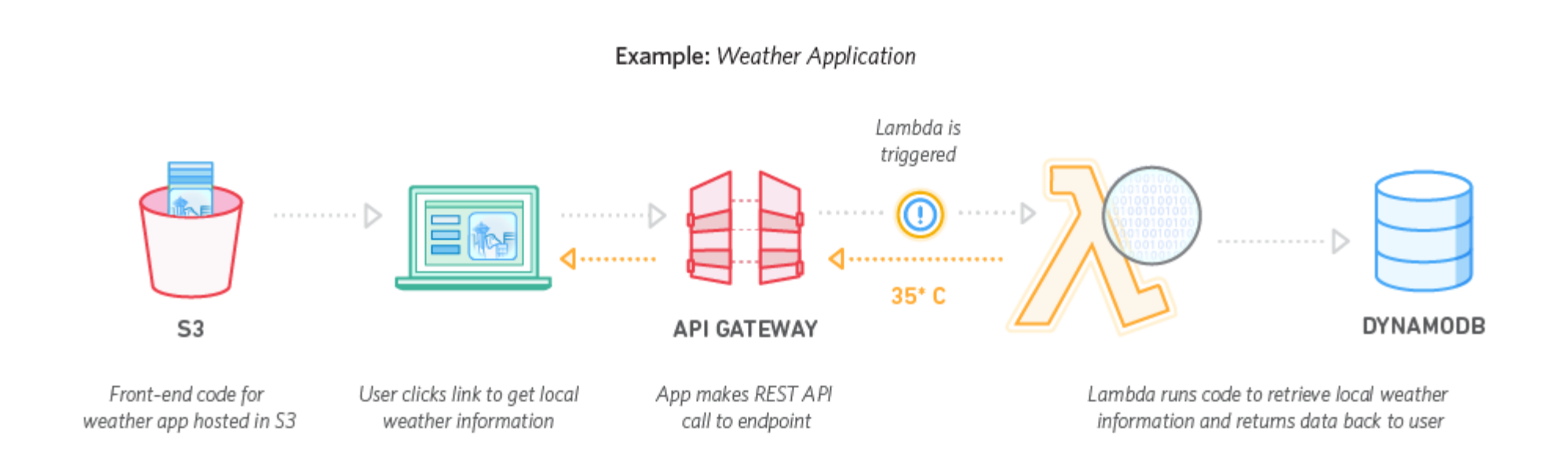
Serverless Computing

What is serverless computing?
Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.



-https://aws.amazon.com/serverless/

.



Example of AWS Serverless Architecture

AWS API Gateway

- Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers. API Gateway creates RESTful APIs that:
- Are HTTP-based.
- Enable stateless client-server communication.
- Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

Source: https://aws.amazon.com/serverless/

NOSQL Database

- NoSQL databases are purpose built for specific data models and have flexible schemas for building modern applications. NoSQL databases are widely recognized for their ease of development, functionality, and performance at scale.

DynamoDB is the **Serverless** NoSQL Database offering by AWS. Being **Serverless** makes it easier to consider **DynamoDB** for **Serverless** Microservices since it goes inline with the patterns and practices when designing **serverless** architectures in AWS

https://medium.com/totalcloudio/aws-dynamodb-for-serverless-microservices-2acbbbff1bca

NO SQL Example

•In a relational database, a book record is often dissembled (or “normalized”) and stored in separate tables, and relationships are defined by primary and foreign key constraints. In this example, the Books table has columns for ISBN, Book Title, and Edition Number, the Authors table has columns for AuthorID and Author Name, and finally the Author-ISBN table has columns for AuthorID and ISBN. The relational model is designed to enable the database to enforce referential integrity between tables in the database, normalized to reduce the redundancy, and generally optimized for storage.

•In a NoSQL database, a book record is usually stored as a JSON document. For each book, the item, ISBN, Book Title, Edition Number, Author Name, and AuthorID are stored as attributes in a single document. In this model, data is optimized for intuitive development and horizontal scalability.

AWS Lambda Pricing

Region: US East (Ohio)

	Price
Requests	$0.20 per 1M requests
Duration	$0.0000166667 for every GB-second

Serverless Advantages

- **No Hardware or Operating System Management:** Developers don’t need to worry about hardware and operating systems, focusing on business logic. No Server Management.

- **High Availability and auto scalability:** If function needs to be run in multiple instances, the vendor's servers will start up, run, and end them as they are needed, often using containers (the functions start up more quickly if they have been run recently)

- **Cost Efficient :** Developers are only charged for the server space they use, reducing cost. As in a 'pay-as-you-go' phone plan, **No costs when functions aren't running.**

- **Faster Development :** Decreased time to market and faster software release.

Sources

1. https://medium.com/systems-architectures/advantages-and-disadvantages-of-serverless-a-technical-perspective-2eb1fa5c69ec
2. https://jaxenter.com/benefits-drawbacks-serverless-computing-135074.html
3. https://www.cloudflare.com/learning/serverless/why-use-serverless/

Severless Disadvantages

- **Difficult to test and and Debug :**It is difficult to replicate the serverless environment in order to see how code will actually perform once deployed. Debugging is more complicated because developers do not have visibility into **backend processes**, and because the application is broken up into separate, **smaller functions.**

- **Vendor lock-in is a risk :** Allowing a vendor to provide all backend services for an application inevitably increases reliance on that vendor. Setting up a serverless architecture with one vendor can make it difficult to switch vendors if necessary, especially since each vendor offers slightly different features and workflows.

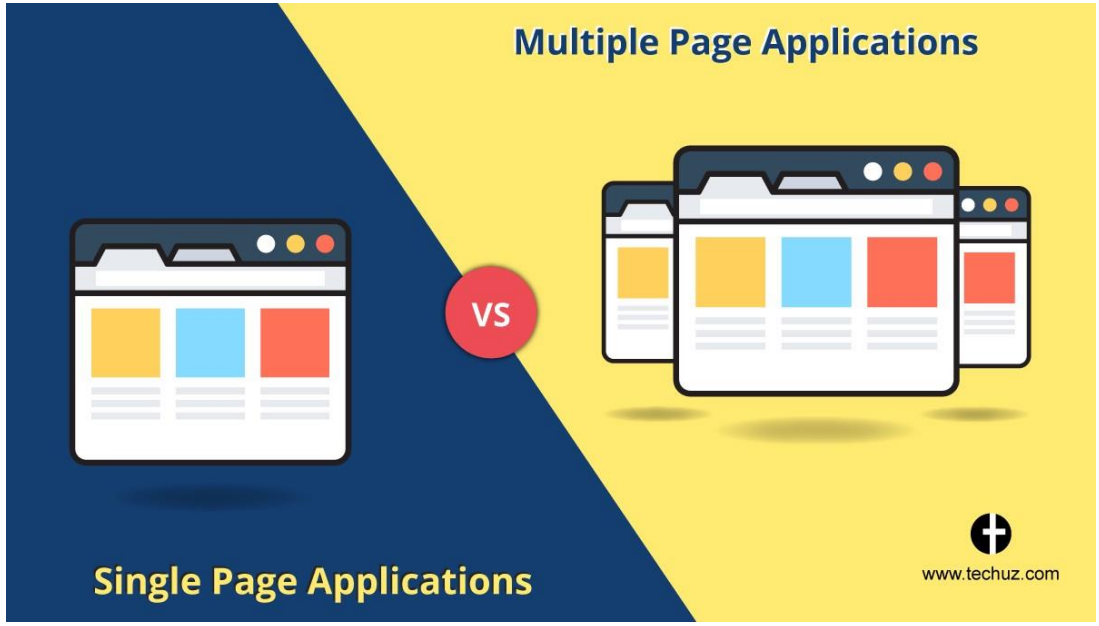
- **Possible Impact on Performance :**Because it's not constantly running, serverless code may need to 'boot up' when it is used. This startup time may degrade performance

Web Applications
Restful Approach

Single Page and Mutlipe Page App

Traditional web application

Perform most of the application logic on the server



Single Page App with APIs

single page applications (SPAs) that perform most of the user interface logic in a web browser, communicating with the web server primarily using web APIs

https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

Possible Advantages of SPA with APIs

Easier design

Easier testing & continuous integration

Better performance

Better scaling

Client – server loose coupling

Source: https://applandeo.com/blog/single-page-applications-rest-better-dynamic-web-pages/

Using SPA Or Dynamic Web App: *Factors to Consider*

Use traditional web applications when:

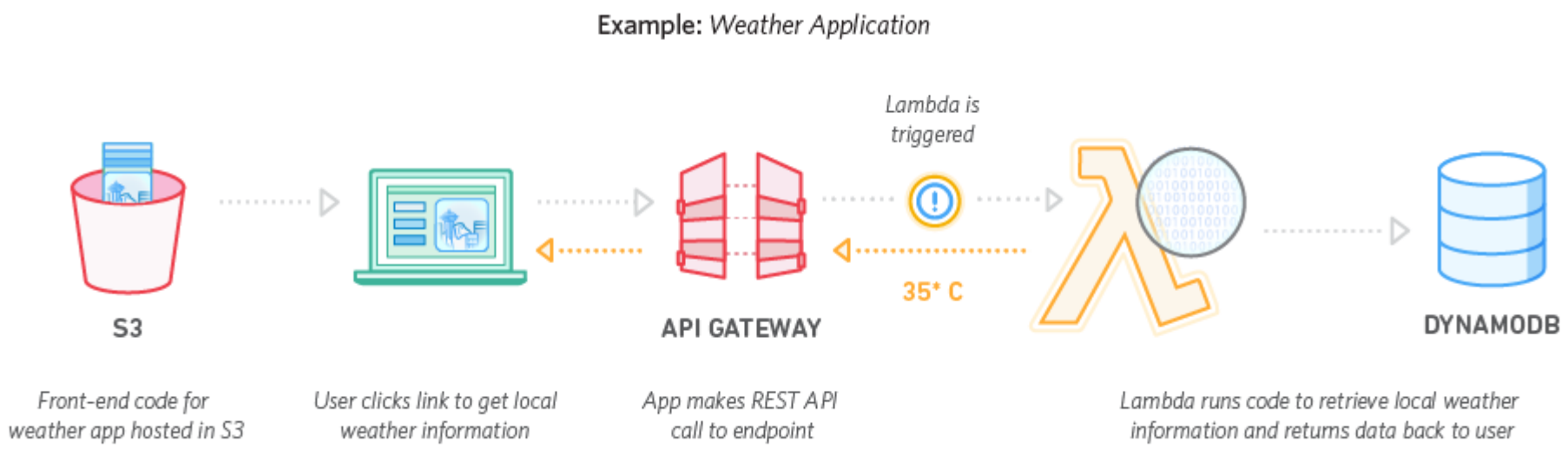
- Your application's client-side requirements are simple or even read-only.
- Your application needs to function in browsers without JavaScript support.
- Your team is unfamiliar with JavaScript or TypeScript development techniques.

Use a SPA when:

- Your application must expose a rich user interface with many features.
- Your team is familiar with JavaScript and/or TypeScript development.
- Your application must already expose an API for other (internal or public) clients.

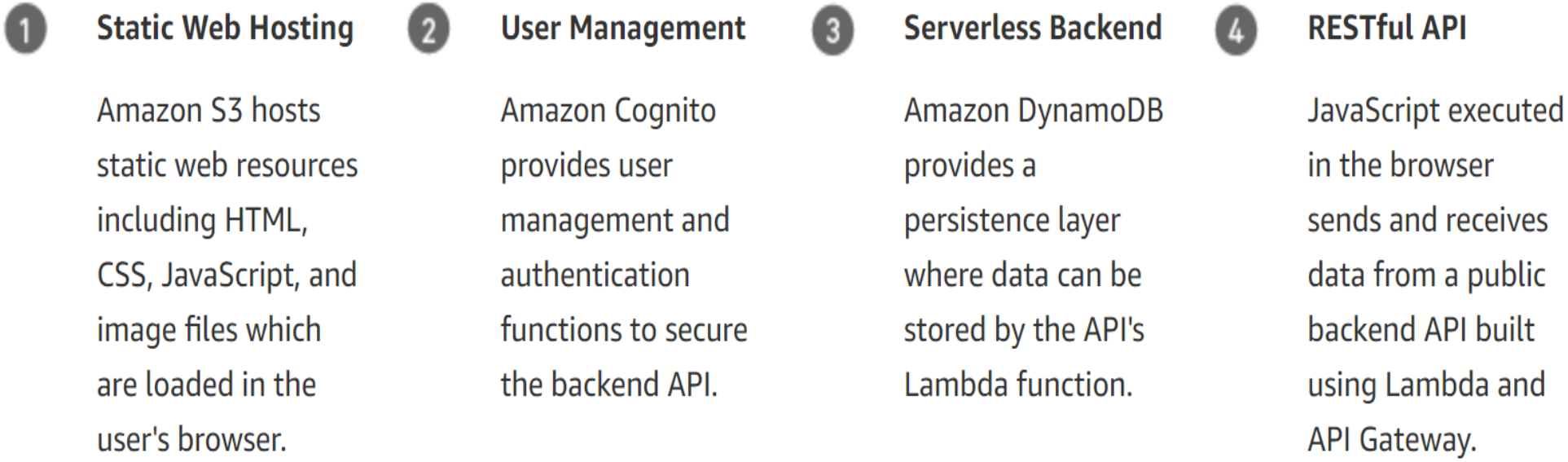
https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps

SPA+ API: Fits with Serverless



https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

Sample App:
Building a Serverless APP with RESTful



Sample App Tutorial

- https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/