UNIT – 2

SOA & WEB SERVICES

Service-Oriented Architecture (SOA) is an architectural style that uses web services to build software applications. Web services are software systems that enable machine-to-machine interaction over a network, and are the preferred way to implement SOA.

Here are some ways web services and SOA work together:

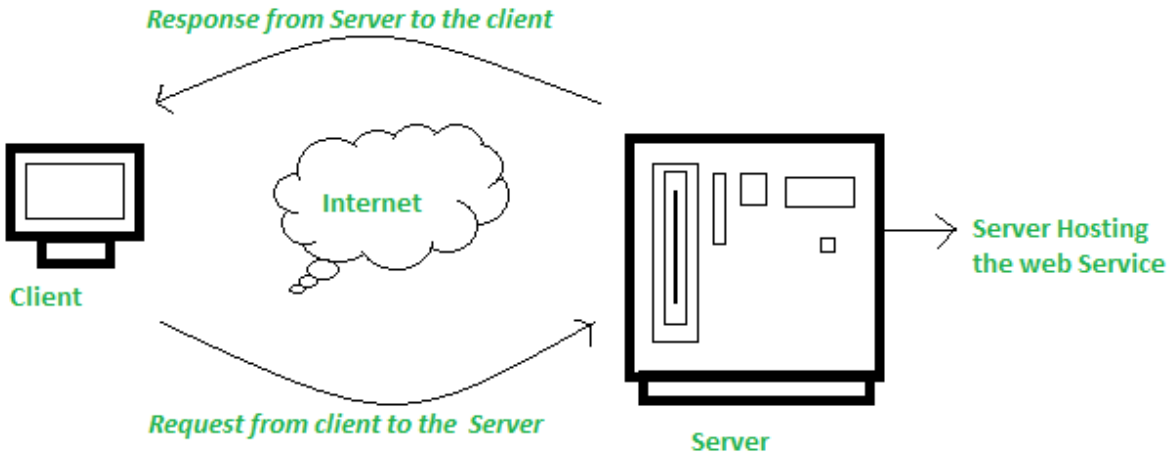
- Loose coupling: SOA is designed to promote loose coupling between software components, so they can be reused. Web services allow clients to be coupled to services, so the server can be integrated outside of the client application programs.
- Dynamic components: Services can be incorporated dynamically during run time.
- Standardized communication: Web services provide a standardized way for components within an SOA to communicate over a network, using protocols like HTTP and XML. This interoperability allows organizations to integrate different systems, promoting reusability, flexibility, and scalability.

Web Services

Web services are applications or software that use standardized web protocols to communicate and exchange data over the internet. They can be either a service offered by one electronic device to another, or a server that listens for requests on a computer device.

(OR)

Web Services are a type of internet software that use standardized messaging protocols and are made available from an application service provider's web server for a client or other web-based programs to use. These services are sometimes referred to as web application services.



Web services can be used for a variety of purposes, including:

- **Interoperability**: Web services provide a standard way for software applications running on different platforms and frameworks to interact with each other.
- **A2A communication**: Web services can be used for application-to-application communication.
- **Streamlining connectivity**: Web services can help streamline connectivity.
- **Minimizing development time**: Web services can help minimize development time.
- **Efficient technology distribution**: Web services can help distribute technology efficiently.

What is an example of a web service?

The World Wide Web is filled with all types of web service examples. Amazon is a company that offers a variety of web-based services. Users can browse Amazon's catalog of items through its search engine. Anytime someone searches for a particular product on Amazon's website, they use Amazon's web services.

What are the types of web services?

There are a few central types of web services: XML-RPC, UDDI, SOAP, and REST: XML-RPC (Remote Procedure Call) is the most basic XML protocol to exchange data between a wide variety of devices on a network.

What are the 3 roles of web services?

The architecture of web service consists of three roles:

Service Provider, Service Requester, and Service Registry.

When to use web services?

Web services let different organizations or applications from multiple sources communicate without the need to share sensitive data or IT infrastructure. Instead, all information moves through a programmatic interface across a network.

What is difference between web service and web application?

A Web Application is meant for humans to read, while a Web Service is meant for computers to read. Web Application is a complete Application with a Graphical User Interface (GUI), however, web services do not

necessarily have a user interface since it is used as a component in an application.

Web Services Architecture

Service Discovery: This part of the architecture is responsible for centralizing services into a common registry and providing easy publish/search functionality. UDDI handles service discovery.

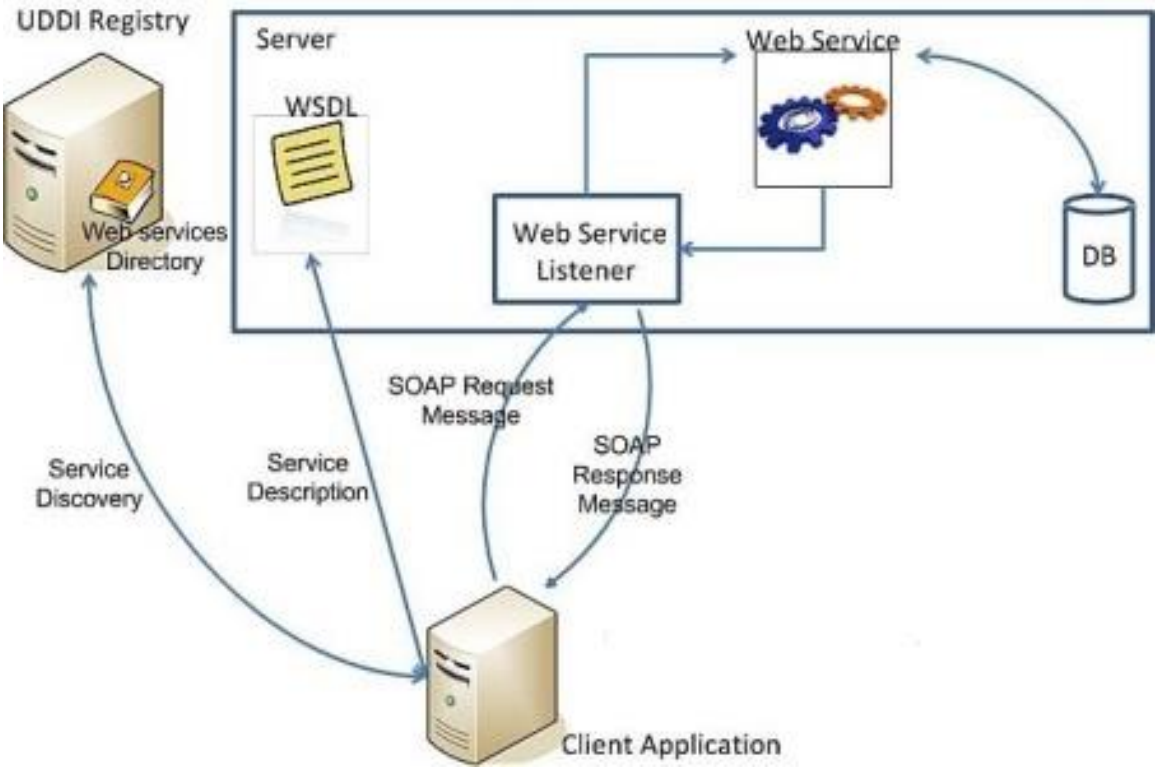
Service Description: One of the most interesting features of Web Services is that they are self-describing. This means that, once a Web Service is located, it will let us know what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).

Service Invocation: Invoking a Web Service involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format request messages to the server, and how the server should format its response messages.

Service Transport: Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP (Hyper Text Transfer Protocol) – the

protocol used to access conventional web pages on the Internet. We could also use other protocols, but HTTP is currently the most used one.

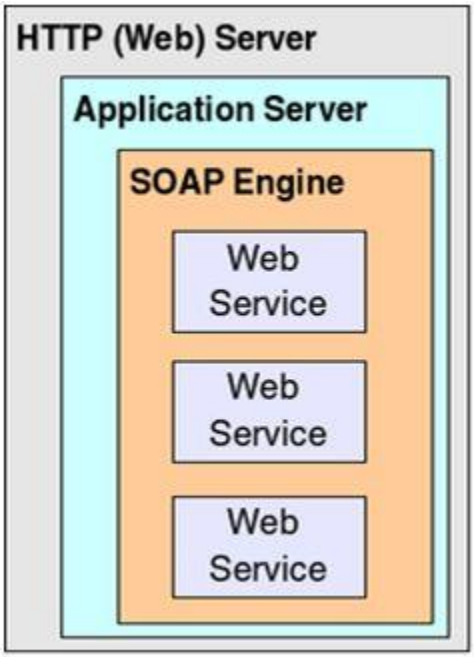
How Web Services work?



1. The Service Provider generates the WSDL describing the application or service and registers the WDSL in UDDI directory or Service Registry.
2. The Service Requestor or client application which is in need of web service contacts the UDDI and discovers the web service.
3. The client based on the web service description specified in the WSDL sends a request for a particular service to the web service application listener in SOAP message format.
4. The web service parses the SOAP message request and invokes a particular operation on the application to process that particular request. The result is packed in an appropriate SOAP response message format and sent to the client.

5. The client parses the SOAP response message and retrieves the result or error messages if any.

Server-side Components of Web Services Application

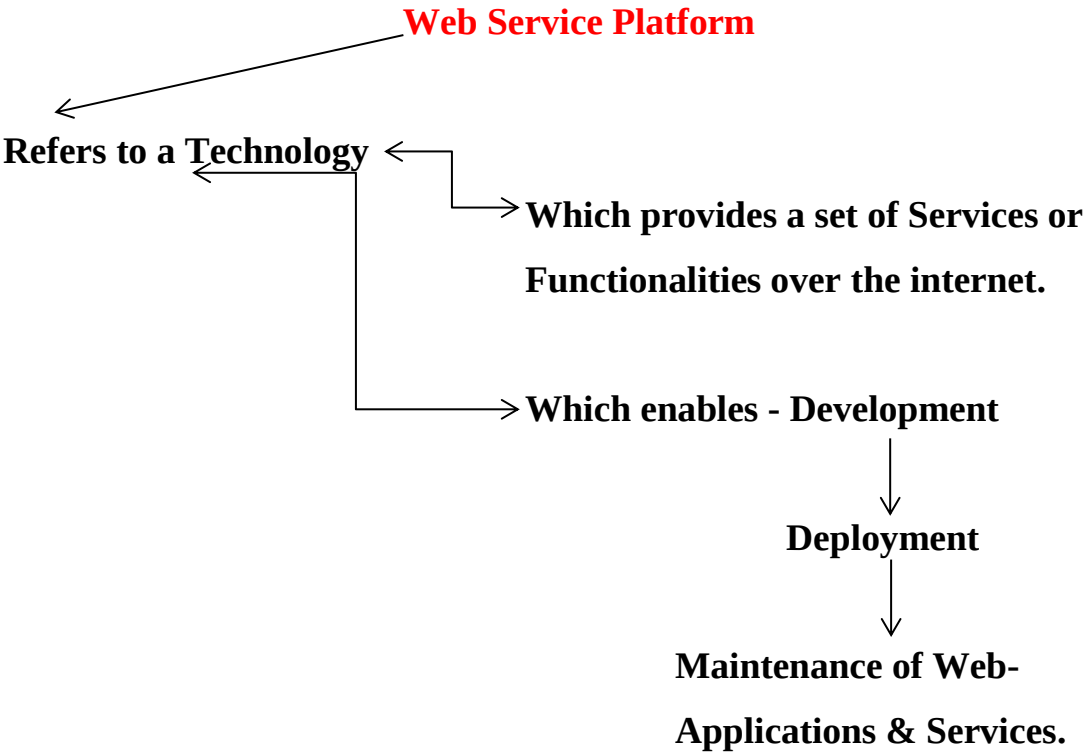


Web service: This is the software or component that exposes a set of operations. For example, if we are implementing our Web service in Java, our service will be a Java class (and the operations will be implemented as Java methods). Clients will invoke these operations by sending SOAP messages.

SOAP Engine: Web service implementation does not know anything about interpreting SOAP requests and creating SOAP responses. To do this, we need a SOAP engine. This is a piece of software that handles SOAP requests and responses. Apache Axis is an example of SOAP engine. The functionality of the SOAP engine is usually limited to manipulating SOAP.

Application Server: To actually function as a server that can receive requests from different clients, the SOAP engine usually runs within an Application Server. This is a piece of software that provides a 'living space' for applications that must be accessed by different clients. The SOAP engine runs as an application inside the application server. A good example is the Apache Tomcat server – a Java Servlet and JSP container.

HTTP Server: Many application servers already include some HTTP functionality, so we can have Web services up and running by installing a SOAP engine and an application server. However, when an application server lacks HTTP functionality, we also need an HTTP Server. This is more commonly called a 'Web server'. It is a piece of software that knows how to handle HTTP messages. A good example is the Apache HTTP Server, one of the most popular web servers in the Internet.



(OR)

A Web Service Platform is a comprehensive environment or framework that provides the necessary tools, services, and infrastructure for developing, deploying, and managing web services. It typically includes both software and hardware components that support the creation and interaction of web services.

Here are some examples of web service platforms:

File servers

Allow clients to store and access information on a company's server, such as YouTube and Google Drive.

Google Maps

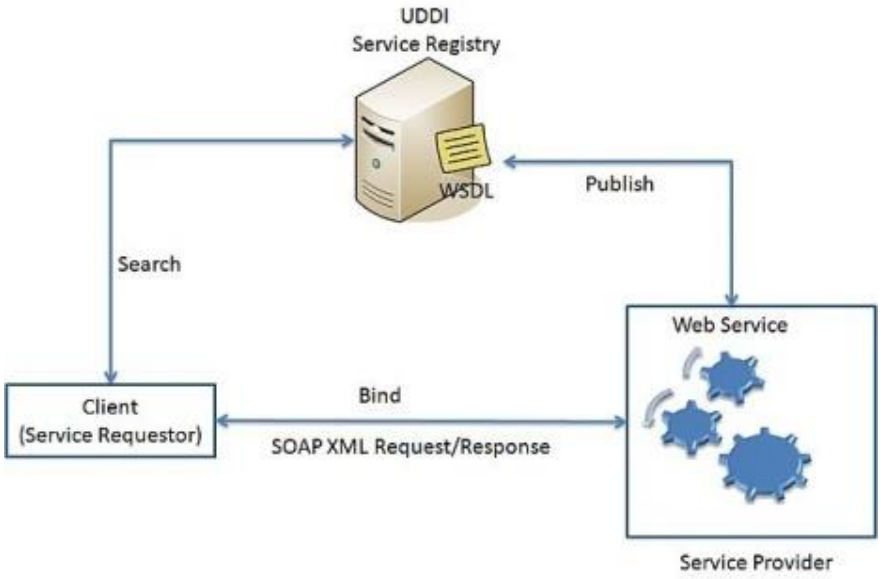
A major service that websites can use to access and display map data. For example, if you use Google Maps on your mobile phone to find a location, your phone's application is the client and Google Maps is the server that returns the data.

Web Services Platform Architecture

XML along with HTTP forms the basis of web services. XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions. The HTTP protocol is the most used Internet protocol.

Web services platform consists of the following components:

- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)
- SOAP (Simple Object Access Protocol)



UDDI

UDDI (Universal Description, Discovery and Integration) is a platform-independent, XML based registry service where companies can register and search for Web services.

- UDDI is a directory for storing information about web services
- UDDI communicates via SOAP
- UDDI is a directory of web service interfaces described by WSDL

WSDL

WSDL (Web Services Description Language) is an XML-based language for locating and describing Web services. WSDL definition describes how to access a web service and what operations it will perform along with the message format and protocol details for the web service. WSDL is a W3C standard.

SOAP

SOAP (Simple Object Access Protocol) is an XML-based communication protocol for exchanging structured information between applications over HTTP, SMTP or any other protocol. In other words, SOAP is a protocol for accessing a Web Service.

SOAP Message Structure

A SOAP message is an ordinary XML document containing the following elements:

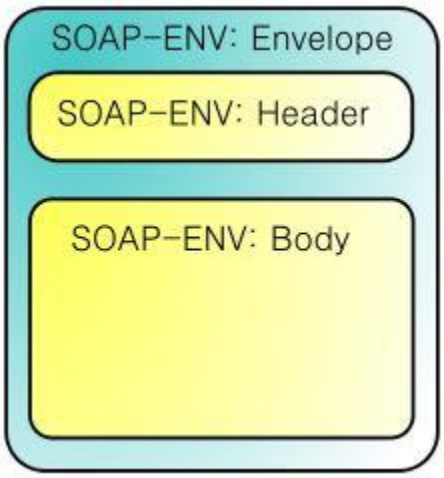
- An Envelope (required) element that identifies the XML document as a SOAP message
- An optional Header element that contains header information
- A Body (required) element that contains call and response information
- An optional Fault element containing errors and status information

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```



Example SOAP Request and Response Message

Below is a sample SOAP Request message for a Stock Quote service whose WSDL is available at http://www.webservicex.net/stockquote.asmx?WSDL. The message is sent as a HTTP Post Request whose content is nothing but a SOAP Request message. The soap request message contains a soap envelope and a soap body with a stock quote request for stock symbol "GOOG" (Google).

```
POST /stockquote.asmx HTTP/1.1
Content-type: text/xml;charset="utf-8"
Soapaction: "http://www.webserviceX.NET/GetQuote"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
*; q=.2, */*; q=.2
User-Agent: JAX-WS RI 2.1.6 in JDK 6
Host: localhost
Connection: keep-alive
Content-Length: 194

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <GetQuote xmlns="http://www.webserviceX.NET/">
            <symbol>GOOG</symbol>
        </GetQuote>
    </S:Body>
</S:Envelope>
```

Below is a sample SOAP Response message for the above request. Apart from the HTTP response headers the content is a SOAP response message with a soap envelope and a soap body. The result is enclosed within “GetQuoteResponse” tag. The last traded price for stock symbol “GOOG” (Google) is in “last” tag put at $594.34 on 5/29/2012 04:00PM.

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Length: 975
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 30 May 2012
09:14:05 GMT

```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <GetQuoteResponse xmlns="http://www.webserviceX.NET/">
            <GetQuoteResult>
                <StockQuotes>
                    <Stock>
                        <Symbol>GOOG</Symbol>
                        <Last>594.34</Last>
                        <Date>5/29/2012</Date>
                        <Time>4:00pm</Time>
                        <Change>0.00</Change>
                        <Open>N/A</Open>
                        <High>N/A</High>
                        <Low>N/A</Low>
                        <Volume>100</Volume>
                        <MktCap>193.8B</MktCap>
                        <PreviousClose>594.34</PreviousClose>
                        <PercentageChange>0.00%</PercentageChange>
                        <AnnRange>473.02 - 670.25</AnnRange>
                        <Earns>32.998</Earns>
                        <P-E>18.01</P-E>
                        <Name>Google Inc.</Name>
                    </Stock>
                </StockQuotes>
            </GetQuoteResult>
        </GetQuoteResponse>
    </soap:Body>
</soap:Envelope>
```

Choice of Web Service Platform for Development & Deployment

→ Picking the right tool for a job

EX:-

If you choose a platform that well known (Microsoft, AWS, Azure) & widely used. ⟶ It might have lots of resources & support available. This can make it easier for developers to build services.

If you pick a less popular platform ⟶ you might struggle to find help when you run into problems.

Here are some considerations:-

1. **Compatibility & Interoperability:-**
   Choosing a platform with strong support for widely accepted standards like SOAP/ REST. (How well Different systems can work together without any issues).
   In context of web services platforms, " it's like asking whether they speak the same language". (Choosing a language that many system understand).
2. **Tooling & Development Environment :-**
   Each web service provider has its own tools, so developer need to be familiar with these tools.
   Selecting a platform tools that align with the development team's skill set & preferences is crucial. (Which helps them do their work smoothly & Confidence).

3. **Security Features:- (Level of Security)**

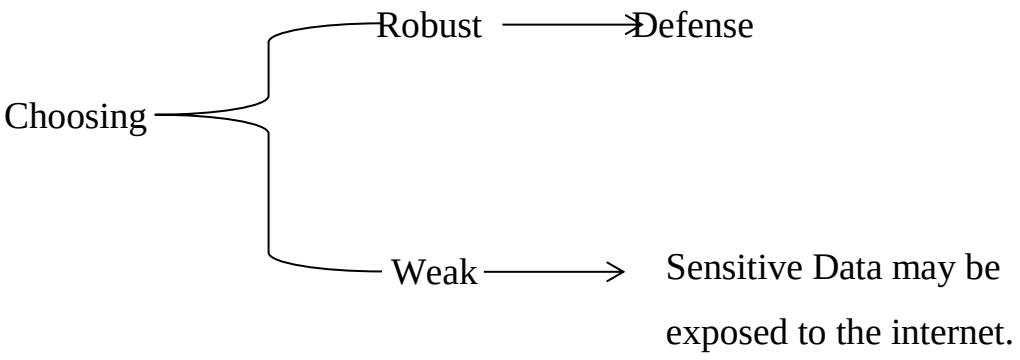
Different Platforms may have varying levels of built-in security features & mechanisms.

Platform A: Well known for its Robust security (Protected from Unauthorized access) measures. It offers features like audit (Check for digital-System, N/w safety) & a built-in firewall.

Platform B: Less Established & doesn't have as many built-in security features (Basic security measures like password protection & some limited firewall options).

- Choosing a Platform A with its advance security features your website is better protected against potential threats & attacks. (Give an extra layer of defense).
- Choosing a Platform B, you might need to invest more time & resources in implementing additional security measures.

  (This could involve installing third-party security plugins & setting up manual security checks. Being more vigilant (Risk of Dangers) about potential vulnerabilities).



4. **Integration Capabilities:-**

   How well a system (or) platform can work together with other systems , software (or) technologies.

   EX:- Like a different piece of puzzles fit together smoothly.

   If a system has good I-C , it can easily connect & collaborate with other systems.

5. **Vendor Support & Community;-**

   A strong community & available resources can be invaluable for trouble shooting, learning & best practices.

   A group of people & available resources i.e, (Helpful materials) can provide support, advice & knowledge when you encounter problems.

   EX:- Learning how to use a new s/w program. If there's a strong community around that s/w. (You can ask question when you get stuck, & experienced users will often provide guidance (or) share solutions).

6. **Cost & Licensing:-**

   Different platforms may have different pricing models, ranging from open-source options to commercial licenses.

7. **Long term- viability & support:-**

   It consider the track record & long term viability of the web service platforms.

   It is important to choose a platform that is actively maintained with roadmap for future development & updates.

SERVICE CONTRACT

A service contract, also known as a service agreement, is a legal agreement between two parties that outlines the terms and conditions of a service to be provided by one party to the other. The service provider is compensated for the service they provide. Service contracts are often used in industries like education, healthcare, construction, and IT.

(OR)

It is a legal binding agreement between a service provider & a client that outlines the terms & conditions of the services being offered.

Service contracts can include details such as:

- Scope and style: The scope, style, and timeline of the service, including any deadlines for completion
- Work to be performed: What services are to be performed for which objects, and under which conditions
- Compensation: Payment terms and agreements
- Changes: What process needs to take place if changes need to be made
- Dispute resolution: Protocols for resolving disputes

Service contracts can be used in many different situations, such as:

- Between an employer and an individual for a limited time period or scope
- Between a contractor and a homeowner
- Between a business and a freelance web designer

EX:-

When you hire someone to do a job for you

It says exactly what they'll do

How long it'll take,

How much you'll pay what everyone's supposed to do,

& what happens if things don't go as planned.

SERVICE LEVEL DATA MODEL

A Service Level Data Model is a structured framework used to define, measure, and manage the performance of services based on **predefined criteria** and agreements. It integrates various components to ensure that services meet the agreed-upon standards and performance levels as stipulated in Service Level Agreements (SLAs). This model is crucial for maintaining service quality, optimizing performance, and ensuring customer satisfaction in a service-oriented environment.

- **Predefined criteria** in a Service Level Data Model are specific, measurable standards established beforehand to evaluate the performance and quality of services. These criteria are often outlined in Service Level Agreements (SLAs) and are used to set expectations for service delivery.

Ex: E-commerce Website

Service: Online Shopping Platform

Predefined Criteria:

Page Load Time:

Criteria: Web pages should load within 2 seconds.

Metric: Average time taken for a web page to fully load.

Order Processing Time:

Criteria: Orders should be processed within 1 hour of placement.

Metric: Time taken from order placement to processing completion.

Cart Abandonment Rate:

Criteria: Maintain a cart abandonment rate of less than 60%.

Metric: Percentage of shopping carts that are abandoned before purchase.

Transaction Success Rate:

Criteria: Ensure 98% of transactions are successfully completed.

Metric: Percentage of transactions completed without errors.

Definition

Service Level Data Model: A comprehensive schema that represents and organizes data related to the performance and quality of services, encompassing metrics, agreements, and **operational details**. It serves as a foundation for monitoring, analyzing, and improving service delivery against predefined service level objectives.

- **Operational details** in a Service Level Data Model refer to the specific elements and aspects involved in the day-to-day management, execution, and monitoring of services. These details ensure that the services operate smoothly, meet performance criteria, and adhere to predefined standards.

Example of Operational Details in a **Email Hosting Service**

Service: Email Hosting

Operational Details:

1. Service Components:

Endpoints:

- **Incoming Mail Server:** imap.emailservice.com
- **Outgoing Mail Server:** smtp.emailservice.com

Interfaces:

- **Webmail Interface:** https://webmail.emailservice.com
- **Email Client Configuration:** IMAP/SMTP settings for email clients.

2. Performance Metrics:

Email Delivery Time:

- **Criteria:** Emails should be delivered within 5 minutes.
- **Metric:** Average time from sending an email to delivery to the recipient's inbox.

3. Incident Management:

Incident Detection:

- **Method:** Automated alerts for issues like email delivery failures or server downtime.

Incident Response:

- **Procedure:** Response within 1 hour for critical issues, with escalation to senior support if unresolved within 4 hours.

4. Service Configuration:

Settings:

- **Maximum email size limit:** 25MB

- **Spam filter settings:** Moderate (filtering emails with known spam characteristics)

5. Customer Interaction:

Support Channels:

- **Email support:** support@emailservice.com
- **Phone support:** 1-800-EMAIL-SUP

Service Level Agreements (SLAs):

- **Support response time:** Within 4 hours for non-critical issues.

Core Components

1. **Service Definitions**:

   Service: The individual units or offerings that are monitored and managed. Each service is defined by its attributes such as name, description, and endpoint details.
2. **Service Level Agreements (SLAs)**:

   SLA: A formal agreement that specifies the expected performance standards and metrics for a service. SLAs detail the targets for service availability, response times, resolution times, and other performance indicators.
3. **Metrics**:

   Performance Metrics: Quantitative measures used to evaluate the performance of a service. Examples include response time, throughput, error rates, and availability.

Operational Metrics: Measures related to the operational health of a service, such as resource utilization and system uptime.

Business Metrics: Metrics that link service performance to business outcomes, such as customer satisfaction and transaction volume.

4. **Data Collection**:

   Monitoring Tools: Systems and tools used to collect data on service performance. This includes logs, monitoring software, and performance dashboards.

5. **Reporting and Analysis**:

   Reports: Documents or dashboards that summarize service performance data, compare it against SLA targets, and highlight trends or issues.

   Analysis: The process of evaluating performance data to identify trends, assess compliance with SLAs, and uncover areas for improvement.

6. **Incident Management**:

   Incident: Any disruption or issue affecting service performance. The model tracks incidents, their impact, and resolution status.

Certainly! Let's walk through an example of a Service Level Data Model within a fictional IT service management context. This example will illustrate how different components fit together to monitor and manage service performance.

Scenario

Imagine a company, **TechSolutions**, provides cloud-based storage services. The company wants to ensure their service meets certain performance standards as agreed with their customers.

Components and Example Data

1. Service Entity

- **Service ID**: S001
- **Service Name**: Cloud Storage
- **Service Description**: Provides scalable cloud storage for businesses.
- **Service Endpoint**: https://api.techsolutions.com/cloudstorage

2. SLA Entity

- **SLA ID**: SLA001
- **Service ID**: S001
- **SLA Name**: Premium Storage SLA
- **SLA Conditions**:
  - **Uptime**: 99.9% monthly
  - **Response Time**: < 200ms
  - **Resolution Time**: < 1 hour for critical issues
- **Start Date**: 2024-01-01
- **End Date**: 2024-12-31

3. Metric Entity

- **Metric ID**: M001

- **Service ID**: S001
- **Metric Name**: Average Response Time
- **Metric Value**: 185ms
- **Measurement Unit**: milliseconds
- **Threshold**: 200ms
- **Metric ID**: M002
- **Service ID**: S001
- **Metric Name**: Uptime
- **Metric Value**: 99.95%
- **Measurement Unit**: percentage
- **Threshold**: 99.9%

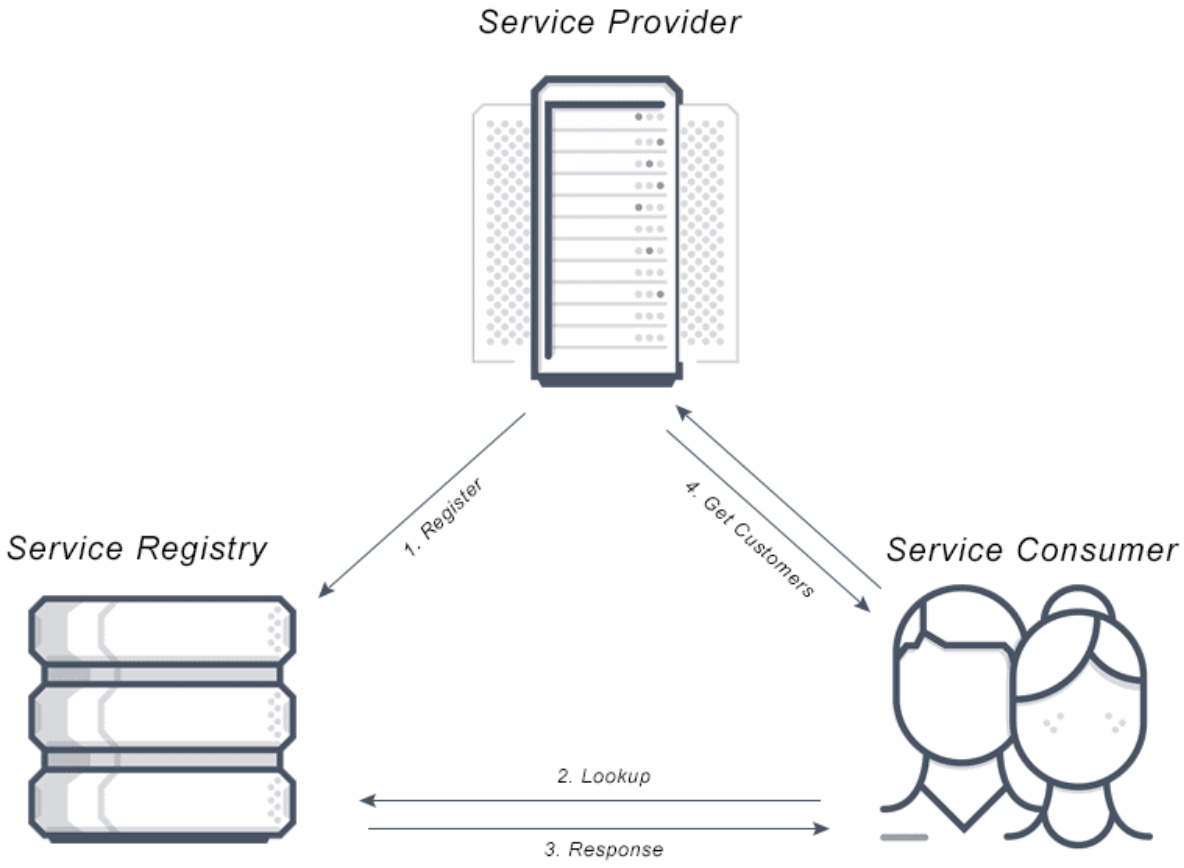
4. Incident Entity

- **Incident ID**: I001
- **Service ID**: S001
- **Incident Description**: Service outage for 30 minutes
- **Reported Time**: 2024-08-15 14:30
- **Resolution Time**: 2024-08-15 15:00
- **Impact Level**: Critical
- **Incident ID**: I002
- **Service ID**: S001
- **Incident Description**: Slow response time due to high traffic
- **Reported Time**: 2024-08-16 10:00
- **Resolution Time**: 2024-08-16 11:00
- **Impact Level**: Major

SERVICE DISCOVERY

Service discovery is the process of automatically detecting devices and services on a network. Service discovery protocol (SDP) is a networking standard that accomplishes detection of networks by identifying resources. Traditionally, service discovery helps reduce configuration efforts by users who are presented with compatible resources, such as a bluetooth-enabled printer or server.

More recently, the concept has been extended to network or distributed container resources as ‘services’, which are discovered and accessed.



Service Discovery has the ability to locate a network automatically making it so that there is no need for a long configuration set up process. Service discovery

works by devices connecting through a common language on the network allowing devices or services to connect without any manual intervention. (i.e Kubernetes service discovery, AWS service discovery)

There are two types of service discovery: Server-side and Client-side. Server-side service discovery allows clients applications to find services through a router or a load balancer. Client-side service discovery allows clients applications to find services by looking through or querying a service registry, in which service instances and endpoints are all within the service registry.

How does Service Discovery Work?

There are three components to Service Discovery: the service provider, the service consumer and the service registry.

1) The Service Provider registers itself with the service registry when it enters the system and de-registers itself when it leaves the system.

2) The Service Consumer gets the location of a provider from the service registry, and then connects it to the service provider.

3) The Service Registry is a database that contains the network locations of service instances. The service registry needs to be highly available and up to date so clients can go through network locations obtained from the service registry. A service registry consists of a cluster of servers that use a replication protocol to maintain consistency.

Example: Smart Home Device Control

Scenario:

You have a smart home app (service consumer) that controls various smart devices in your home, like smart lights (service provider).

Process:

1. **Device Registration:**
    - When you set up your smart light bulb, it registers itself with a service discovery protocol (like mDNS or SSDP). It provides information such as its unique identifier, IP address, and the types of commands it can respond to (e.g., turn on/off, change color).
2. **Service Discovery:**
    - When you open the smart home app, it automatically scans the local network for registered smart devices.
    - The app retrieves a list of available devices, including the smart light bulb.
3. **Service Consumption:**
    - You select the smart light bulb from the app interface and choose to change its color or brightness.
    - The app sends a command to the bulb using the information obtained during the discovery process.
4. **Response:**
    - The smart light bulb receives the command and adjusts its settings accordingly. It may also send a confirmation back to the app.

Summary:

In this simple example, the smart home app discovers and interacts with a smart light bulb without needing to know its exact network address beforehand. This seamless interaction enhances user experience and simplifies device management in smart homes.

SERVICE LEVEL INTEGRATION PROCESS

Service Level Integration (SLI) refers to the process of ensuring that various services work together effectively, meeting defined service level agreements (SLAs) and performance metrics. Here's a general outline of the SLI process:

1. Define Services and Interfaces

- Identify the services that need to be integrated.
- Define the interfaces, APIs, and protocols that these services will use to communicate.

2. Establish Service Level Agreements (SLAs)

- Define clear SLAs for each service, including performance metrics such as response times, availability, and reliability.
- Ensure that all stakeholders agree on these SLAs.

3. Design Integration Architecture

- Create an architecture that outlines how the services will interact, including data flows and dependencies.
- Consider using middleware or service orchestration tools if necessary.

4. Implement Integration

- Develop the necessary code and configuration to enable communication between services.
- Ensure that all services can handle the data formats and protocols defined in the earlier steps.

5. Testing and Validation

- Perform integration testing to ensure that services work together as expected.
- Validate that SLAs are being met under various load conditions.

6. Monitoring and Performance Management

- Implement monitoring tools to track the performance of integrated services against the defined SLAs.
- Set up alerts for any SLA breaches or performance issues.

7. Continuous Improvement

- Regularly review performance data to identify areas for improvement.
- Update SLAs and integration strategies as needed based on changing requirements or service performance.

8. Documentation and Training

- Document the integration process, architecture, and any specific configurations.
- Provide training for relevant personnel to ensure they understand how the integrated services operate.

This structured process helps ensure that services not only integrate smoothly but also meet the expected performance standards, providing a reliable and efficient user experience.

Example: E-Commerce Platform Integration

Scenario:

An e-commerce platform needs to integrate several services, including user authentication, product inventory, payment processing, and order management.

1. Define Services and Interfaces

- **Services Identified:**
  - User Authentication Service
  - Product Inventory Service
  - Payment Processing Service
  - Order Management Service
- **Interfaces Defined:**
  - RESTful APIs for each service, with clear endpoints for each function (e.g., /login, /products, /checkout).

2. Establish Service Level Agreements (SLAs)

- **SLAs Defined:**
  - User Authentication: 99.9% uptime, response time under 200 ms.
  - Product Inventory: 99% accuracy in product availability, response time under 300 ms.
  - Payment Processing: 99.5% transaction success rate, response time under 500 ms.

- Order Management: 99% order processing success rate, response time under 300 ms.

3. Design Integration Architecture

- **Architecture Created:**
  - Use of a microservices architecture where each service operates independently but communicates through a centralized API Gateway.
  - Data flow diagram created to illustrate how user requests are processed across services.

4. Implement Integration

- **Development:**
  - APIs are developed and tested.
  - Authentication service handles login and token generation.
  - Payment service integrates with third-party payment gateways.
  - Order management service handles order placement and tracking.

5. Testing and Validation

- **Integration Testing:**
  - Simulate user actions like logging in, adding products to a cart, and completing a purchase.
  - Verify that all services communicate correctly and meet SLA performance metrics.

6. Monitoring and Performance Management

- **Monitoring Tools Implemented:**

- Use of monitoring tools like Prometheus and Grafana to track response times, uptime, and error rates for each service.
- Alerts set up to notify the development team of any SLA breaches.

7. Continuous Improvement

- **Review Meetings Held:**
  - Monthly reviews of performance data to discuss any SLA breaches or performance issues.
  - Adjustments made to the architecture or code based on feedback and data analysis.

8. Documentation and Training

- **Documentation Created:**
  - Comprehensive documentation covering API specifications, integration architecture, and troubleshooting guidelines.
- **Training Provided:**
  - Workshops conducted for the development and operations teams to ensure everyone understands the integrated services and how to manage them.

ATOMIC & COMPOSITE SERVICES

In SOA, Services can be categorized into two main types:-

They are:-

1. Atomic
2. Composite

1. **Atomic Service:** It is a standalone, self-contained unit of functionality that performs a specific task (or) operation.
   Ex:- Calculate Total Cost (Service take item prices & quantities as input returns the total cost).
   Characteristic: It doesn't rely on (or) call other services, it handles its task independently.
2. **Composite Service:** It is higher-level Service that composed of multiple atomic (or) other composite services.
   Ex:- Process Order (it might use check inventory & charge customer along with other composite services to handle the entire order process).
   Characteristic: it orchestrates the work of multiple services to accomplish a broader business goal.

ATOMIC	COMPOSITE
Nature of Functionality: 1. Performs a specific standalone task (or) operation.	1. Orchestrates & Coordinates multiple services to achieve a broader business goal.
Independence: 2. Operates Independently & doesn't rely on other services.	2. Relies on the collaboration of multiple services to accomplish its task.

Granularity: (level of detail) 3. Represents a fine-grained single unit of functionality.	3. Represents a coarse-grained service, combining several atomic (or) other composite services to fulfill a complex operation.
Complexity of Task: 4. Handles a relatively simple, well-defined task.	4. Handles more complex business processes by coordinating multiple services.
Reusability & Flexibility: 5. Often highly reusable across different business process (or) applications	5. May be more tailored to specific business processes & may have less general reuse potential.

RETROSPECTIVE ON SOA

Review (or) Past Events (or) Actions

Involves on looking back & evaluating the **Success** & what are the **Challenges Faced**, **Lessons Learned** from implementing this architectural approach.

- It's a reflective process to understand what worked well & what could be improved for future projects.

Success:-

1. Interoperability
2. Reusability
3. Scalability
4. Modularity

Challenges:-

1. Complexity (Designing & Managing SOA requires careful planning)
2. Integration Challenges (with new services expecially in organizations with existing infrastructure)
3. Governance & Standards (Consistent standards to be provided could be a struggle)
4. Change Management (Moving from a large (centralized team) to smaller (specialized teams) with different roles & responsibilities (change the behavior))

Lessons Learned:-

1. Clear Business Alignment (success)
2. Effective Governance (consistency & complexity)
3. Modularity & Loosely Coupling
4. Continious Monitoring & Improvement.