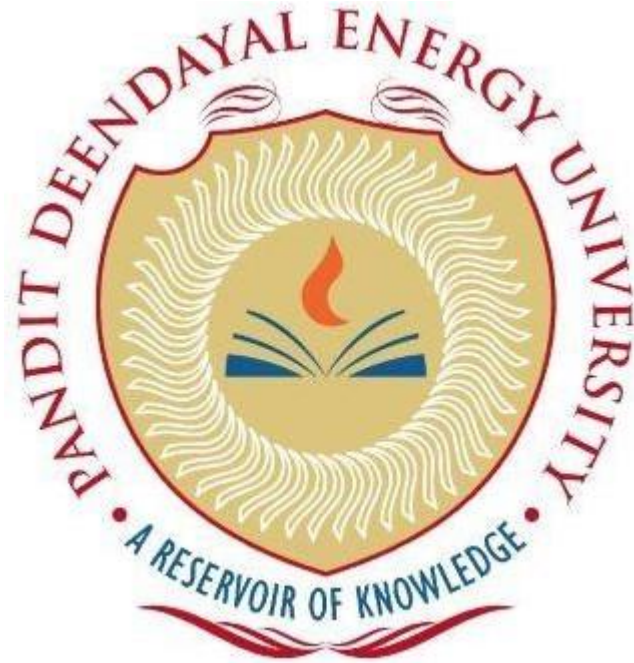


PANDIT DEENDAYAL ENERGY UNIVERSITY
SCHOOL OF TECHNOLOGY



Course: Cyber Security Lab

Course Code: 23CP310P

B.Tech. (Computer Engineering)

Semester 6(G9)

Submitted by:

Mihir Chaudhary

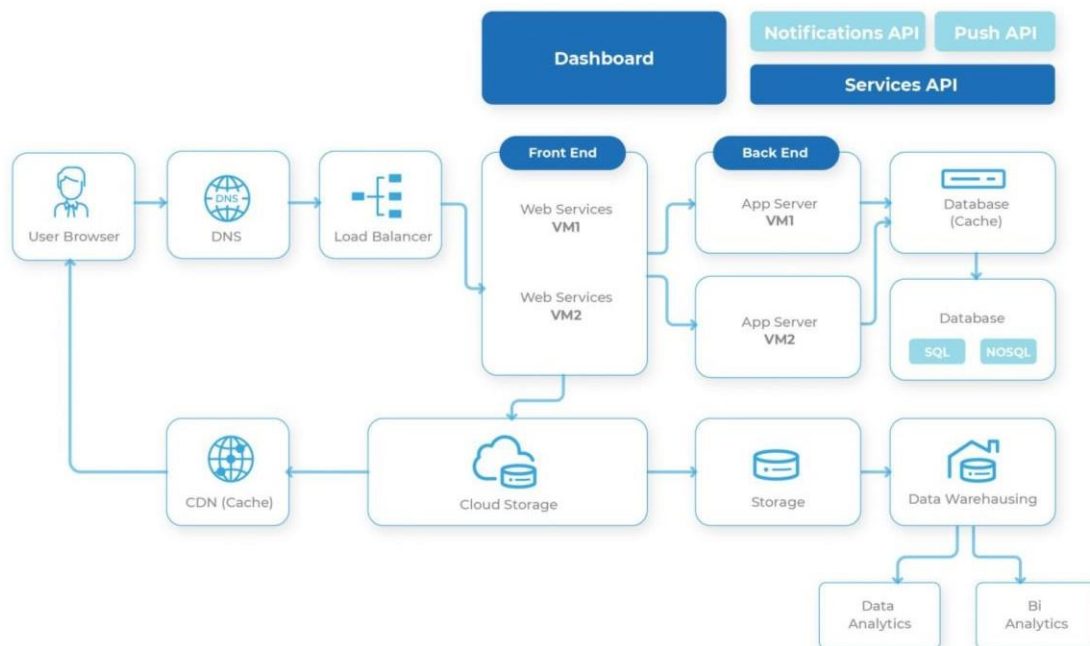
22BCP327

1. Architecture of Web Services and the Role of Servers

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They typically use standardized protocols like HTTP, XML, SOAP, WSDL, and REST to facilitate communication between different systems.

Key Components of Web Services Architecture:

1. **Client:** The application or system that initiates a request to the web service.
2. **Server:** The system that hosts the web service and processes client requests.
3. **Protocols:** Standards like HTTP, HTTPS, SOAP, REST, etc., used for communication.
4. **Data Formats:** XML, JSON, or other formats used to structure the data being exchanged.
5. **Service Description:** WSDL (Web Services Description Language) for SOAP-based services or OpenAPI for RESTful services, which describe how to interact with the service.



Role of Servers:

- **Hosting:** Servers host the web service, making it accessible over the internet or an intranet.
- **Request Handling:** Servers receive client requests, process them, and return appropriate responses.
- **Scalability:** Servers can be scaled horizontally (adding more servers) or vertically (increasing server resources) to handle increased load.
- **Security:** Servers implement security measures like SSL/TLS encryption, authentication, and authorization to protect the service.

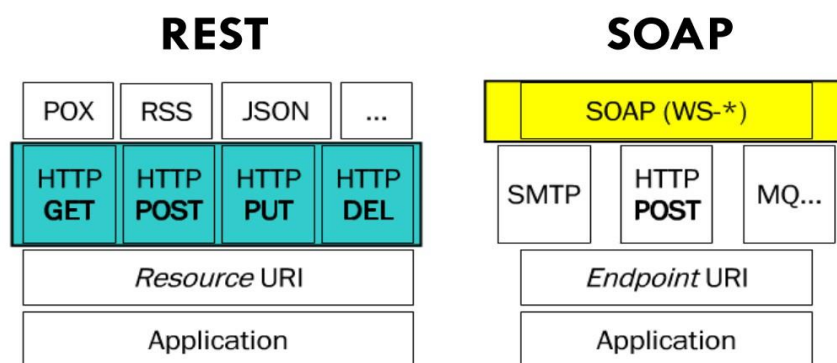
2.RESTful vs. SOAP-Based Services

	SOAP	vs	REST
Approach	Function driven		Data driven
Statefulness	Stateless by default but a SOAP API can be made stateful		Stateless in nature, no server side sessions
Meaning	Simple Object Access Protocol		Representational State Transfer
Performance	Requires more power, resources and bandwidth		Requires fewer resources
Design	Standard protocol with predefined rules to follow		Architectural style with loose recommendation and guidelines
Caching	API calls are not cached		API calls are cached
Security	WS-Security with SSL support. Provides an in built ACID compliance		Supports SSL and HTTPS
Messaging Format	Only XML		XML, JSON, plain text YAML, HTML, and others
Nature	Heavy weight		Light weight
Transfer Protocols	SMTP, HTTP, UDP and others		Only HTTP
Advantages	Standardization, security, extensibility		High Performance, Scalability, Flexibility and browser friendliness
Recommended for	Financial services, enterprise level apps, payment gateways, high security apps, telecommunication services		Public APIs for web services, social networks and mobile services
Disadvantages	More complex, poor performance, less flexibility		Unsuitable for distributed environments, less security

RESTful and SOAP-based services are two prominent web service architectures. REST (Representational State Transfer) is known for its simplicity, statelessness, and use of standard HTTP methods. It is widely adopted for web APIs due to its lightweight nature and ease of integration with web technologies. RESTful services use JSON or XML for data exchange and can be easily scaled.

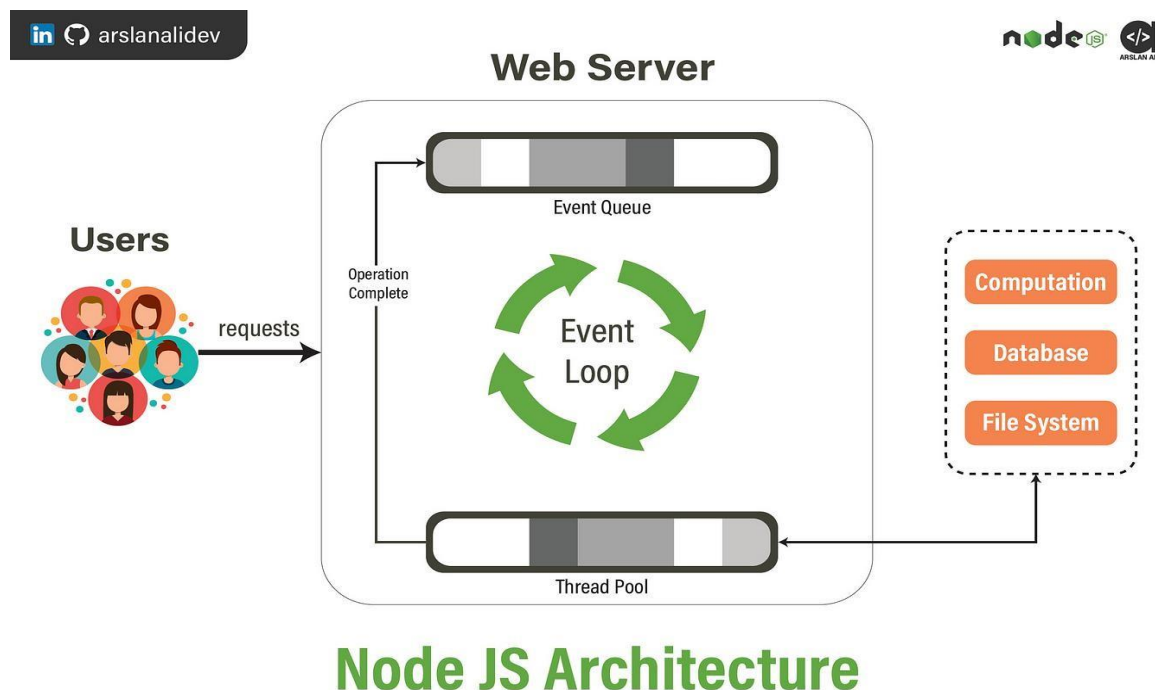
SOAP (Simple Object Access Protocol), on the other hand, is a protocol that relies on XML-based messaging. It provides more rigid standards for security, transactions, and error handling, making it suitable for enterprise-level applications that require high security and reliability. However, SOAP can be more complex and resource-intensive compared to REST.

□ Protocol Layering



2. Implementing a Simple HTTP-Based Web Service

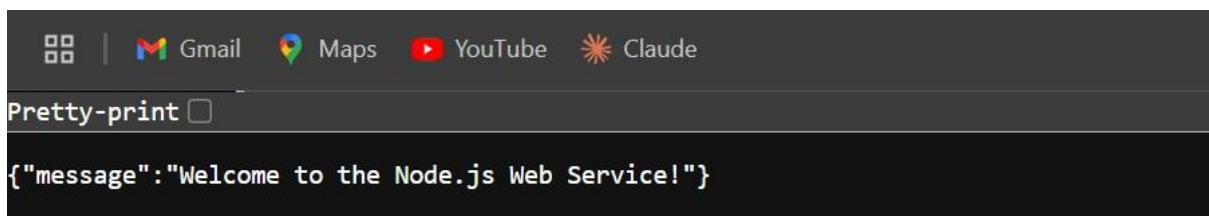
1. Using Node.js (Express)



Code:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 5000;

app.use(bodyParser.json());
app.get('/', (req, res) => {
  res.json({ message: "Welcome to the Node.js Web Service!" });
});
app.get('/greet', (req, res) => {
  const name = req.query.name || 'Guest';
  res.json({ greeting: `Hello, ${name}!` });
});
app.post('/data', (req, res) => {
  res.json({ received: req.body });
});
app.listen(port, () => {
  console.log(`Node.js server running on port ${port}`);
});
```

Output:

```
PS C:\new> node app.js
Node.js server running on port 5000
```

Deploying the Web Service

1. Deploying with Node.js

- Install dependencies: `npm install express`
- Run the app: `node app.js`
- Use a process manager like **PM2**: `pm2 start app.js`
- Deploy to a cloud platform like **Heroku**, **AWS**, or **Google**

Cloud. Example: Deploying to Heroku

1. Install the Heroku CLI.
2. Login: `heroku login`
3. Create a new app: `heroku create`
4. Push your code: `git push heroku main`
5. Open the app: `heroku open`

3.Dockerfile for Node.js

- FROM node:14
- WORKDIR /app
- COPY . .
- RUN npm install
- CMD ["node", "app.js"]