MIT World Peace University

Cloud Infrastructure and Security
Third Year B. Tech, Semester 6

Deploying Web server in Python on Cyclic (PaaS)

Assignment 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 10

March 28, 2024

Contents

1 Aim

Write a Web Service using Java or Python. Deploy the services using PaaS tools such as Cloud Foundry/ GoogleAppEngine / OpenShift.

2 Objectives

1. To understand the concept of web service.
2. To get familiar with PaaS Service.
3. Learn how to deploy the web service on cloud Foundry/GoogleAppEngine/ OpenShift

3 Theory

3.1 Introduction to Web services

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They are built on standard web technologies such as HTTP, XML, and SOAP, and can be accessed using standard protocols like REST, JSON, and WSDL. Web services provide a platform-independent way of integrating applications and services across different platforms and technologies.

1. **Web Service Architecture:** Web services follow a client-server architecture, where the client sends requests to the server, and the server processes the requests and sends back responses. The communication between the client and server is done using standard protocols like HTTP, SOAP, or REST.
2. **Web Service Standards:** Web services are built on standard web technologies and protocols, such as XML, SOAP, WSDL, and UDDI. These standards define the message format, communication protocols, and service description for web services, ensuring interoperability and compatibility across different platforms and technologies.
3. **Web Service Security:** Web services need to be secure to protect sensitive data and prevent unauthorized access. Security mechanisms like SSL/TLS, OAuth, and WS-Security can be used to secure web services and ensure data confidentiality, integrity, and authentication.
4. **Web Service Deployment:** Web services can be deployed on different platforms, such as on-premises servers, cloud platforms, or PaaS services. Deployment involves configuring the web service, setting up the server environment, and making the service accessible to clients over the network.

3.2 Web Service Protocols

1. **SOAP (Simple Object Access Protocol):** SOAP is a protocol for exchanging structured information in the implementation of web services. It uses XML as the message format and can be accessed using standard protocols like HTTP, SMTP, and TCP.
2. **REST (Representational State Transfer):** REST is an architectural style for designing networked applications. RESTful web services use standard HTTP methods like GET, POST, PUT, and DELETE to perform operations on resources. They typically use JSON or XML as the message format and are lightweight and easy to use.

3. **WSDL (Web Services Description Language):** WSDL is an XML-based language for describing web services and their interfaces. It defines the operations, messages, and bindings of a web service and provides a standard way to communicate with the service.

4. **UDDI (Universal Description, Discovery, and Integration):** UDDI is a standard for publishing and discovering web services. It provides a directory service where businesses can register their web services and clients can search for and access the services.

5. **HTTP (Hypertext Transfer Protocol):** HTTP is the standard protocol for transferring data over the web. It is used by web services to send and receive messages between clients and servers.

3.3 Web Service Standards

1. **XML (Extensible Markup Language):** XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is widely used in web services for representing data in a structured and standardized way.

2. **JSON (JavaScript Object Notation):** JSON is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is commonly used in RESTful web services for data serialization and exchange.

3. **SSL/TLS (Secure Sockets Layer/Transport Layer Security):** SSL/TLS are cryptographic protocols that provide secure communication over a computer network. They are used to encrypt data transmitted between clients and servers, ensuring data confidentiality and integrity.

4. **OAuth (Open Authorization):** OAuth is an open standard for access delegation that allows users to grant third-party applications access to their resources without sharing their credentials. It is commonly used in web services to authenticate and authorize users.

5. **WS-Security (Web Services Security):** WS-Security is a standard for securing web services using message-level security mechanisms. It provides features like encryption, digital signatures, and authentication to ensure the confidentiality, integrity, and authenticity of messages.

6. **HTTP Methods:** HTTP methods like GET, POST, PUT, and DELETE are used in RESTful web services to perform operations on resources. GET is used to retrieve data, POST is used to create data, PUT is used to update data, and DELETE is used to delete data.

7. **HTTP Status Codes:** HTTP status codes like 200 OK, 201 Created, 400 Bad Request, and 404 Not Found are used in web services to indicate the status of a request. They provide information about the success or failure of a request and help clients understand the response from the server.

3.4 Different types of Web services

Web services can be categorized into two main types:

1. **SOAP-based Web Services:** SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in the implementation of web services. SOAP-based web services use XML as the message format and can be accessed using standard protocols like HTTP, SMTP, and TCP.

2. **RESTful Web Services:** REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful web services use standard HTTP methods like GET, POST, PUT, and DELETE to perform operations on resources. They typically use JSON or XML as the message format and are lightweight and easy to use.

3.5 SOAP

1. **Description:** SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in the implementation of web services. It uses XML as the message format and can be accessed using standard protocols like HTTP, SMTP, and TCP.

2. **Advantages:**

   - **Standardized Protocol:** SOAP is a standardized protocol that defines a set of rules for exchanging messages between clients and servers.
   - **Interoperability:** SOAP-based web services are platform-independent and can be accessed from any programming language or platform that supports the protocol.
   - **Security:** SOAP supports security features like encryption, digital signatures, and authentication to ensure data confidentiality and integrity.
   - **Complex Data Types:** SOAP allows for the use of complex data types and structures in messages, making it suitable for enterprise-level applications.

3. **Disadvantages:**

   - **Complexity:** SOAP messages can be complex and verbose, making them less efficient for simple data exchange.
   - **Performance Overhead:** SOAP-based web services can have higher performance overhead due to the XML parsing and processing required.
   - **Limited Browser Support:** SOAP is not well-supported by web browsers, limiting its use in client-side applications.
   - **Versioning:** SOAP services can be difficult to version and maintain over time, leading to compatibility issues between clients and servers.

4. **Example:** A SOAP-based web service for a banking application that allows clients to transfer funds between accounts using XML messages over HTTP.

5. **Use Cases:** SOAP-based web services are commonly used in enterprise applications, financial services, and healthcare systems that require secure and reliable communication between clients and servers.

3.6 REST

1. **Description:** REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful web services use standard HTTP methods like GET, POST, PUT, and DELETE to perform operations on resources. They typically use JSON or XML as the message format and are lightweight and easy to use.

2. **Advantages:**

- **Simplicity:** RESTful web services are simple and easy to use, making them ideal for lightweight applications and client-side development.
- **Performance:** REST services have lower performance overhead compared to SOAP services due to their lightweight message format and stateless nature.
- **Browser Support:** REST is well-supported by web browsers and can be easily consumed by client-side applications using JavaScript.
- **Scalability:** RESTful web services are scalable and can handle a large number of concurrent requests, making them suitable for high-traffic applications.

3. **Disadvantages:**

   - **Security:** REST does not provide built-in security features like encryption or authentication, requiring additional measures to secure the communication between clients and servers.
   - **Complex Operations:** RESTful services can be limited in handling complex operations or transactions that require multiple steps or state management.
   - **Versioning:** REST services can be challenging to version and maintain over time, leading to compatibility issues between clients and servers.
   - **Data Integrity:** REST does not provide built-in mechanisms for ensuring data integrity or consistency, requiring developers to implement custom solutions.

4. **Example:** A RESTful web service for a social media application that allows users to create, read, update, and delete posts using JSON messages over HTTP.

5. **Use Cases:** RESTful web services are commonly used in mobile applications, IoT devices, and web applications that require lightweight communication between clients and servers.

6. **Comparison:** SOAP vs. REST

   - **Protocol:** SOAP is a protocol, while REST is an architectural style.
   - **Message Format:** SOAP uses XML, while REST uses JSON or XML.
   - **Security:** SOAP supports security features, while REST requires additional measures for security.
   - **Complexity:** SOAP messages can be complex, while REST messages are simple and lightweight.
   - **Performance:** SOAP services have higher performance overhead, while REST services are more efficient.

3.7 Web service Development using Python

Python is a popular programming language for developing web services due to its simplicity, readability, and versatility. Python provides libraries and frameworks for building web services using different protocols like SOAP and REST. Some popular libraries and frameworks for web service development in Python include Flask, Django, and FastAPI.

3.8 FastAPI

1. **Description:** FastAPI is a modern web framework for building APIs with Python 3.6+ based on standard Python type hints. It is fast, easy to use, and provides automatic validation, serialization, and documentation of API endpoints.
2. **Features:**
   - **Fast:** FastAPI is one of the fastest web frameworks for Python, providing high performance and low latency for API requests.
   - **Easy to Use:** FastAPI is easy to learn and use, with a simple and intuitive syntax that leverages Python type hints for defining API endpoints.
   - **Automatic Validation:** FastAPI automatically validates request data based on type hints and provides detailed error messages for invalid inputs.
   - **Automatic Documentation:** FastAPI generates interactive API documentation based on type hints, allowing users to explore and test API endpoints in real-time.
3. **Example:** A simple web service using FastAPI that exposes endpoints for creating, reading, updating, and deleting user data.
4. **Installation:** FastAPI can be installed using pip, the Python package manager, by running the command `pip install fastapi`.
5. **Usage:** FastAPI provides a built-in development server for running web services locally and supports deployment to cloud platforms like Heroku, AWS, and Azure.
6. **Resources:** FastAPI documentation, tutorials, and examples are available on the official website at https://fastapi.tiangolo.com/.
7. **Comparison between Flask, Django, and FastAPI:**
   (a) **Flask:** Flask is a lightweight web framework for Python that is easy to use and flexible. It is suitable for small to medium-sized web applications and APIs.
   (b) **Django:** Django is a full-featured web framework for Python that provides a complete set of tools and libraries for building complex web applications. It is suitable for large-scale projects with high traffic and data requirements.
   (c) **FastAPI:** FastAPI is a modern web framework for Python that is fast, easy to use, and provides automatic validation and documentation of API endpoints. It is suitable for building high-performance APIs with minimal code.

4 Steps / Procedure to follow for web service deployment using any PaaS tool

4.1 Creation of the Web Service

```python
import uvicorn
from app import app

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8181)
```

```python
from fastapi import FastAPI
from fastapi.responses import FileResponse

from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    item_id: int


@app.get("/")
async def root():
    return {"message": "Hello World, to: there all along react native app."}


@app.get('/favicon.ico', include_in_schema=False)
async def favicon():
    return FileResponse('favicon.ico')


@app.get("/item/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}


@app.get("/items/")
async def list_items():
    return [{"item_id": 1, "name": "Foo"}, {"item_id": 2, "name": "Bar"}]


@app.post("/items/")
async def create_item(item: Item):
    return item
```

4.2 Requirements File To be downloaded on Cyclic

```
anyio==3.7.0
certifi==2023.7.22
click==8.1.3
dnspython==2.3.0
email-validator==2.0.0.post2
exceptiongroup==1.1.1
fastapi==0.96.0
h11==0.14.0
httpcore==0.17.2
httptools==0.5.0
httpx==0.24.1
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
orjson==3.9.0
pydantic==1.10.8
python-dotenv==1.0.0
python-multipart==0.0.6
PyYAML==6.0
```

```
sniffio==1.3.0
starlette==0.27.0
ujson==5.7.0
uvicorn==0.22.0
uvloop==0.17.0
watchfiles==0.19.0
websockets==11.0.3
pytest
```

4.3 Make Repository on Github

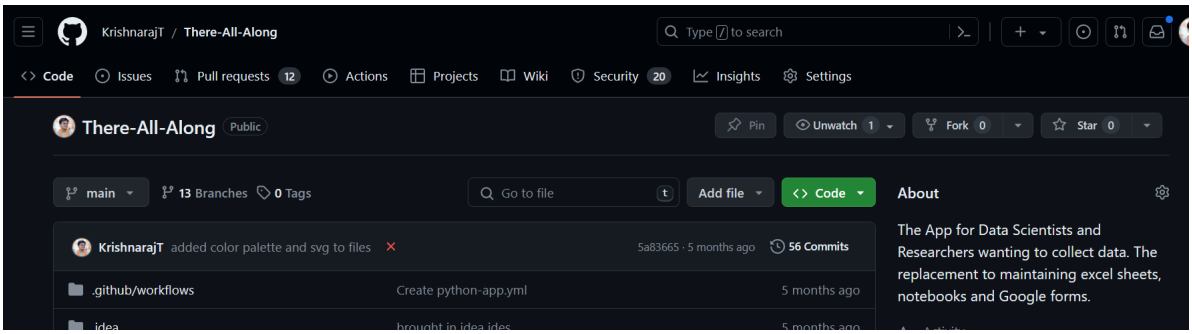

Figure 1: Repository of the App on Github

4.4 Upload on Cyclic

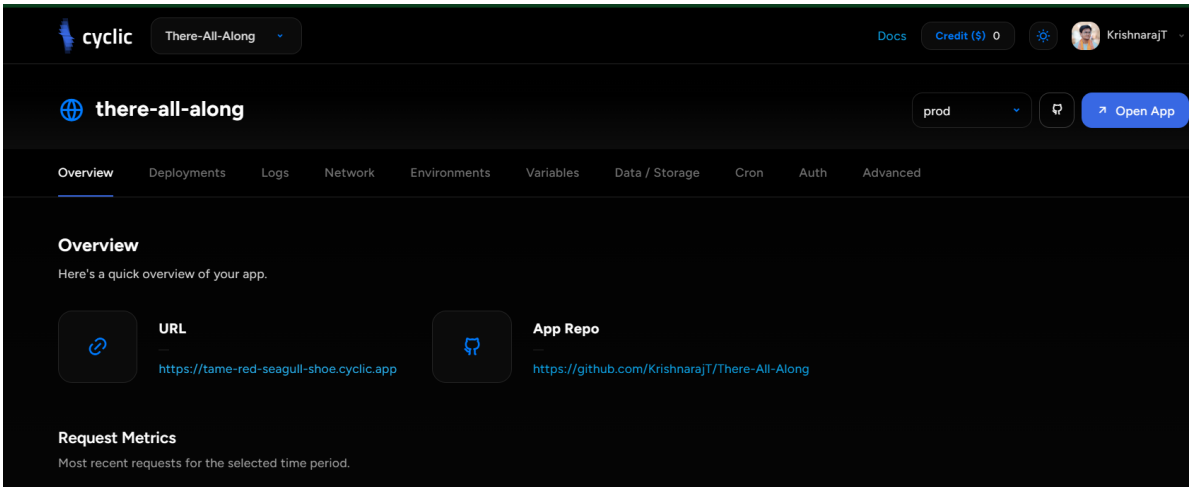

Figure 2: App on Cyclic after Connecting Github Repository

4.5 Deployment after commit on Cyclic

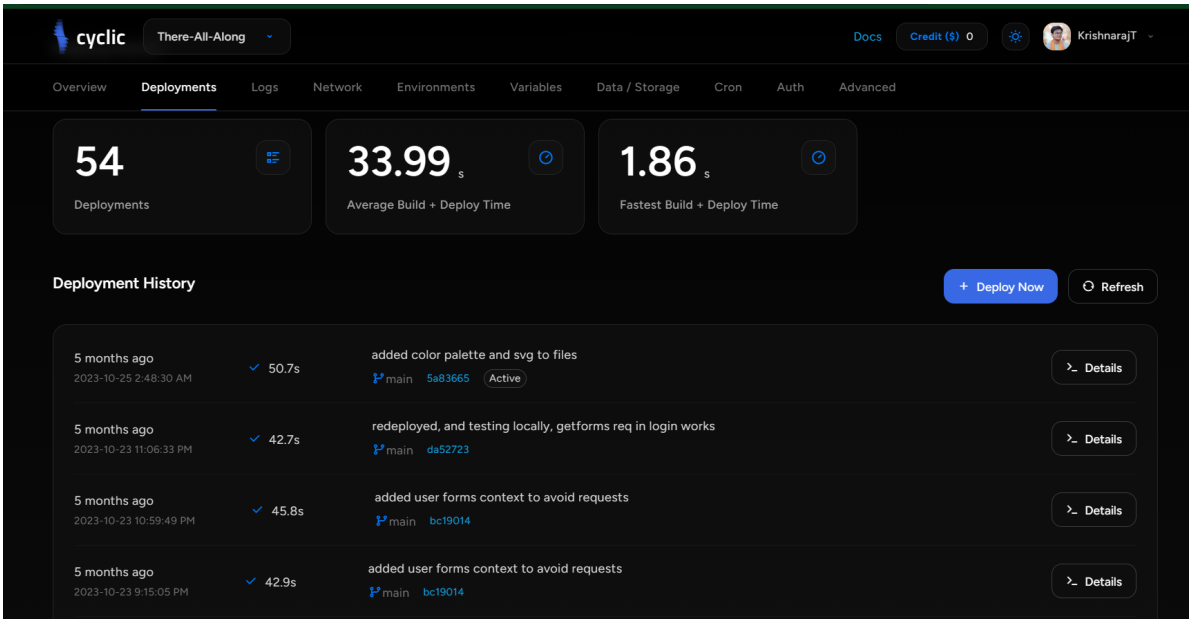

Figure 3: After every commit, cyclic auto deploys the new service after building it.

4.6 Get request from Browser

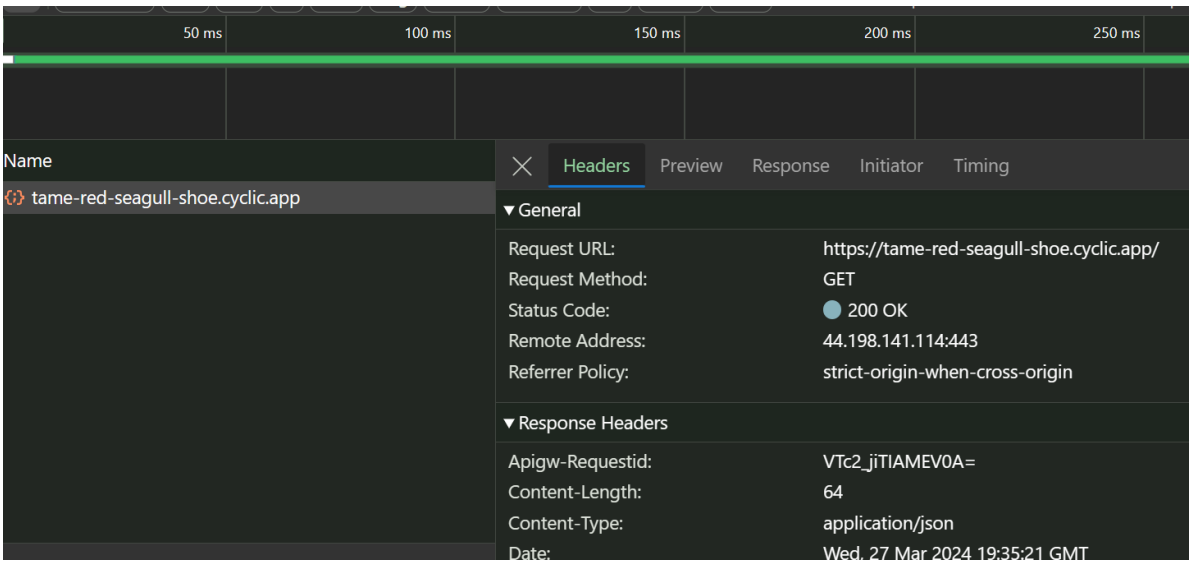

Figure 4: Sending get request from browser

4.7 Response fromn the Web Service

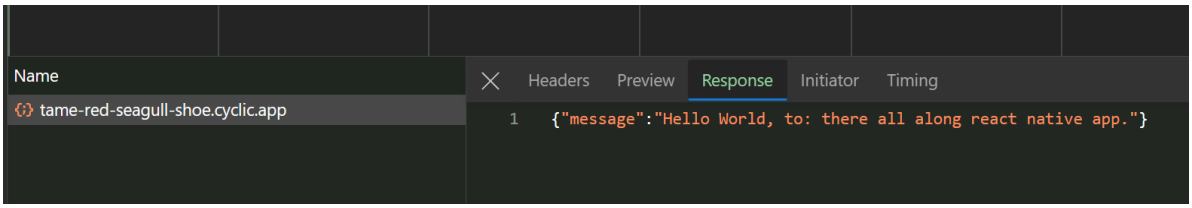

Figure 5: Response from server

4.8 Stored on Amazon S3 with Dynamodb

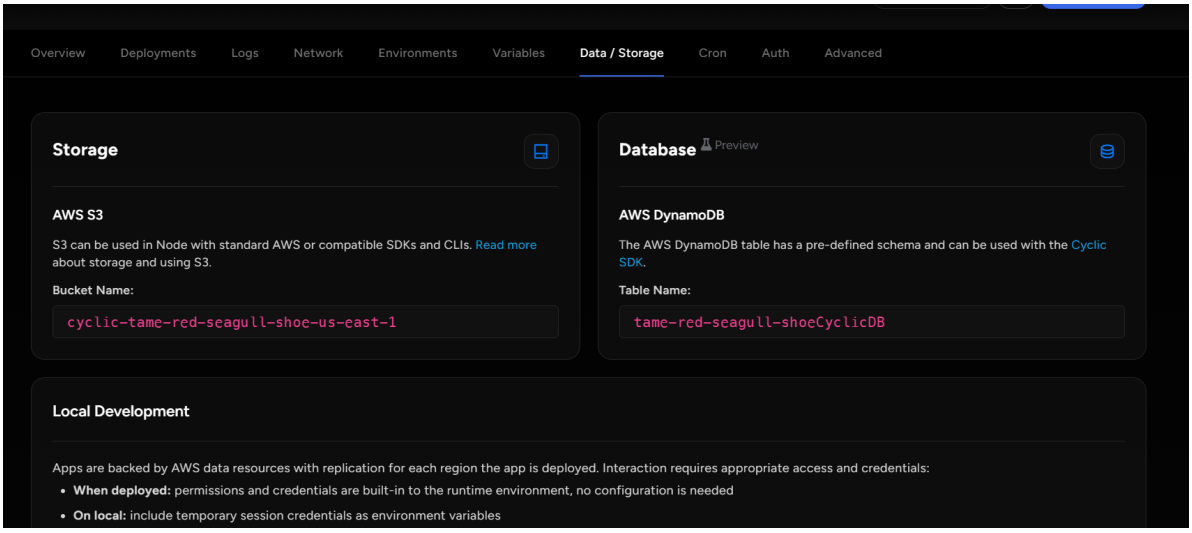

Figure 6: Url and s3 bucket

5 Platform

Operating System: Windows 11
IDEs or Text Editors Used: Visual Studio Code

6 FAQs

1. **Explain the differences between SOAP and RESTful web services in terms of architecture, communication protocols, and data formats?**

   - **Architecture:**
     - **SOAP (Simple Object Access Protocol):** SOAP is a protocol-based standard for exchanging structured information in the implementation of web services. It relies on XML for message format and typically uses WSDL (Web Services Description Language) for service description.

  - **RESTful (Representational State Transfer):** RESTful web services are based on the REST architectural style, which emphasizes a stateless client-server interaction, uniform resource identifiers (URIs) for resource identification, and standard HTTP methods (GET, POST, PUT, DELETE) for communication.
- **Communication Protocols:**
  - **SOAP:** SOAP typically uses HTTP or SMTP (Simple Mail Transfer Protocol) as the communication protocol. It can also work with other protocols such as TCP/IP and JMS (Java Message Service).
  - **RESTful:** RESTful web services primarily use HTTP as the communication protocol. They leverage standard HTTP methods and status codes for CRUD (Create, Read, Update, Delete) operations.
- **Data Formats:**
  - **SOAP:** SOAP messages are typically formatted using XML (eXtensible Markup Language). They can also support other data formats such as JSON (JavaScript Object Notation) and MTOM (Message Transmission Optimization Mechanism) for binary data.
  - **RESTful:** RESTful web services can support various data formats, including XML, JSON, HTML, and plain text. The choice of data format is typically based on the client's requirements and preferences.

2. **Can you describe the typical architecture of a web service and the components involved, including clients, servers, and service endpoints?**

- A typical web service architecture consists of clients, servers, and service endpoints.
- **Clients:** Clients are applications or systems that consume the web service by sending requests and processing responses.
- **Servers:** Servers host the web service and handle client requests. They process incoming requests, perform necessary operations, and send responses back to clients.
- **Service Endpoints:** Service endpoints are URLs (Uniform Resource Locators) or URIs (Uniform Resource Identifiers) that clients use to access the web service. Each endpoint corresponds to a specific operation or resource exposed by the service.

3. **What are some common use cases for web services in various industries and domains, such as e-commerce, finance, healthcare, and IoT?**

- **E-commerce:** Integrating payment gateways, order processing systems, and inventory management.
- **Finance:** Implementing banking APIs for transaction processing, account management, and fraud detection.
- **Healthcare:** Interconnecting electronic health record systems, medical imaging services, and patient management platforms.
- **IoT (Internet of Things):** Collecting sensor data, monitoring device status, and controlling connected devices.

4. **What are Platform-as-a-Service (PaaS) tools like Cloud Foundry, Google App Engine, and OpenShift, and how do they differ from other cloud computing service models?**

    - PaaS tools like Cloud Foundry, Google App Engine, and OpenShift provide platforms for developing, deploying, and managing applications without the complexity of managing underlying infrastructure.
    - Unlike Infrastructure as a Service (IaaS) and Software as a Service (SaaS), which focus on providing infrastructure or ready-to-use software, PaaS offers a complete development and deployment environment.
    - PaaS tools abstract away infrastructure management tasks such as provisioning servers, configuring networks, and managing operating systems, allowing developers to focus on building and deploying applications.

5. **How does Cloud Foundry streamline the process of deploying and managing applications across different cloud environments, including public, private, and hybrid clouds?**

    - Cloud Foundry provides a consistent platform for deploying applications across multiple cloud environments, including public, private, and hybrid clouds.
    - It abstracts away the underlying infrastructure complexity and provides a self-service platform for developers to deploy and manage applications with minimal manual intervention.
    - Cloud Foundry supports automation of application lifecycle management tasks such as deployment, scaling, and monitoring, reducing operational overhead and ensuring consistent application performance.
    - By leveraging containerization and microservices architectures, Cloud Foundry enables developers to build and deploy applications in a modular and scalable manner, enhancing agility and flexibility.

7 Conclusion

In this assignment, we have learned about web services, PaaS tools, and the deployment of web services using Python on Cyclic. We have explored the different types of web services, including SOAP and RESTful services, and the standards and protocols used in web service development. We have also discussed the architecture of web services, the components involved, and the security mechanisms used to protect sensitive data. Finally, we have demonstrated the deployment of a web service using FastAPI and Cyclic, showcasing the process of creating, deploying, and accessing web services on a cloud platform.

References

[1] IBM Cloud. Cloud Computing Concepts Overview. Accessed from: https://www.ibm.com/cloud/learn/cloud-computing-concepts.