Software Engineering 2023, 10(1): 1-6
DOI: 10.5923/j.se.20231001.01

Cost Effective Cloud ERP Integrations

Mohd Iqbal Ashraf

Architect, Giant Eagle Inc, Pittsburgh, PA, USA

Abstract Cloud integration platforms offered by leading cloud providers can sometimes be expensive, making it challenging for industries to maintain due to high operational costs. To address this issue, this document provides an overview of an alternate cloud integration approach using Python. This approach can help quickly develop and consume integration APIs and is highly cost-effective. The document discusses the use case of integrating with Oracle Cloud ERP using Python and how this approach can also be applied to other cloud ERPs. The document is organized into six sections: 1) Introduction, 2) Methodology, 3) Technical Details for Building the Integration, 4) Use-cases, 5) Tools, and 6) Conclusions. In the introduction and methodology section, the concept of cloud integration and its importance for businesses are discussed. The Technical Details for Building the Integration section explains the steps and considerations involved in building the integration using Python. The Use-Cases section discusses about different use cases. The Tools section provides an overview of the tools and technologies required for the integration process. Finally, the Conclusion section summarizes the key points covered in the document and the potential benefits of using Python for cloud integration.

Keywords Cloud integration, Python, Oracle Cloud ERP, APIs, Data mapping, Security measures, Libraries, Frameworks, Cost savings, Efficiency

1. Introduction

1.1. Background

Cloud integration has become an increasingly important aspect of modern business operations, enabling organizations to streamline their processes, improve data sharing, and enhance collaboration between disparate systems. With the growing adoption of cloud-based enterprise resource planning (ERP) systems, such as Oracle Cloud ERP, the need for efficient and cost-effective integration solutions has become a critical concern for many organizations.

Traditionally, cloud integration has been achieved through the use of cloud integration platforms offered by leading cloud providers, such as Oracle, Salesforce, and Microsoft. While these platforms offer a range of features and functionalities, they can sometimes prove to be expensive, making it challenging for organizations to maintain due to high operational costs.

To address this issue, an alternate approach to cloud integration has emerged, leveraging the power and versatility of the Python programming language. Python is an open-source, high-level programming language known for its simplicity and scalability, making it an ideal tool for developing and consuming integration APIs.

Python provides a range of tools and libraries for building and consuming APIs, as well as powerful data processing and analytics capabilities, making it an attractive option for organizations seeking to achieve cloud integration in a cost-effective and efficient manner.

1.2. Problem Statement

The high operational costs associated with cloud integration platforms provided by leading cloud providers can hinder the ability of businesses to effectively integrate their systems with the cloud. This results in increased costs, decreased efficiency, and difficulty in achieving their integration goals.

To address this challenge, this document proposes a cost-effective alternative solution by using Python for cloud integration. The document outlines the key elements of the cloud integration process, including design overview, technical details for implementation, and necessary tools and technologies. The aim of the document is to provide businesses with a step-by-step guide to using Python for cloud integration and to highlight the potential benefits of this approach, including cost savings and increased efficiency. The ultimate goal is to help businesses overcome the challenges posed by the high operational costs of cloud integration platforms and achieve seamless integration with the cloud.

* Corresponding author:
iqbal7ashraf@gmail.com (Mohd Iqbal Ashraf)
Received: Feb. 1, 2023; Accepted: Feb. 13, 2023; Published: Feb. 14, 2023
Published online at http://journal.sapub.org/se

2. Methodology

2.1. Evaluation

The use of Python offers several benefits, including being open source, scalable and easy to code, having in-memory processing capabilities, and having a rich set of open-source libraries for complex tasks, as well as having machine learning and AI capabilities and robotics process automation capabilities.

By using Python, industries can easily consume SAAS APIs using the 'Requests' open-source library, and store data in Pandas Dataframes, which are highly efficient for data analysis and mass data manipulation. This allows for the efficient and cost-effective development and consumption of integration APIs.

Furthermore, the approach discussed in the article can be applied to other cloud ERPs, not just Oracle Cloud ERP, making it a versatile and adaptable solution. In summary, the use of Python for cloud integration offers a cost-effective and versatile solution for industries, and the methodology outlined in the article provides a clear and detailed guide for successful cloud integration.

2.2. Other Approaches

Compared to other approaches, the use of Python for cloud integration has several advantages and benefits. For example, using proprietary cloud integration platforms offered by leading cloud providers can sometimes be expensive and challenging for industries to maintain due to high operational costs. This can be overcome by using Python, which is open source and free, making it a cost-effective solution for industries.

In addition, Python is a highly scalable and easy to code language, with in-memory processing capabilities and a rich set of open-source libraries for complex tasks, as well as having machine learning and AI capabilities and robotics process automation capabilities. This makes Python a versatile and adaptable solution for cloud integration.

Another advantage of using Python for cloud integration is its ability to easily consume SAAS APIs using the 'Requests' open-source library and store data in Pandas dataframes, which are highly efficient for data analysis and mass data manipulation.

In contrast, other approaches may not offer the same level of versatility, scalability, and cost-effectiveness as Python. They may also require proprietary tools and technologies, making them less adaptable and potentially more expensive for industries.

In conclusion, the use of Python for cloud integration offers a cost-effective and versatile solution for industries, setting it apart from other approaches.

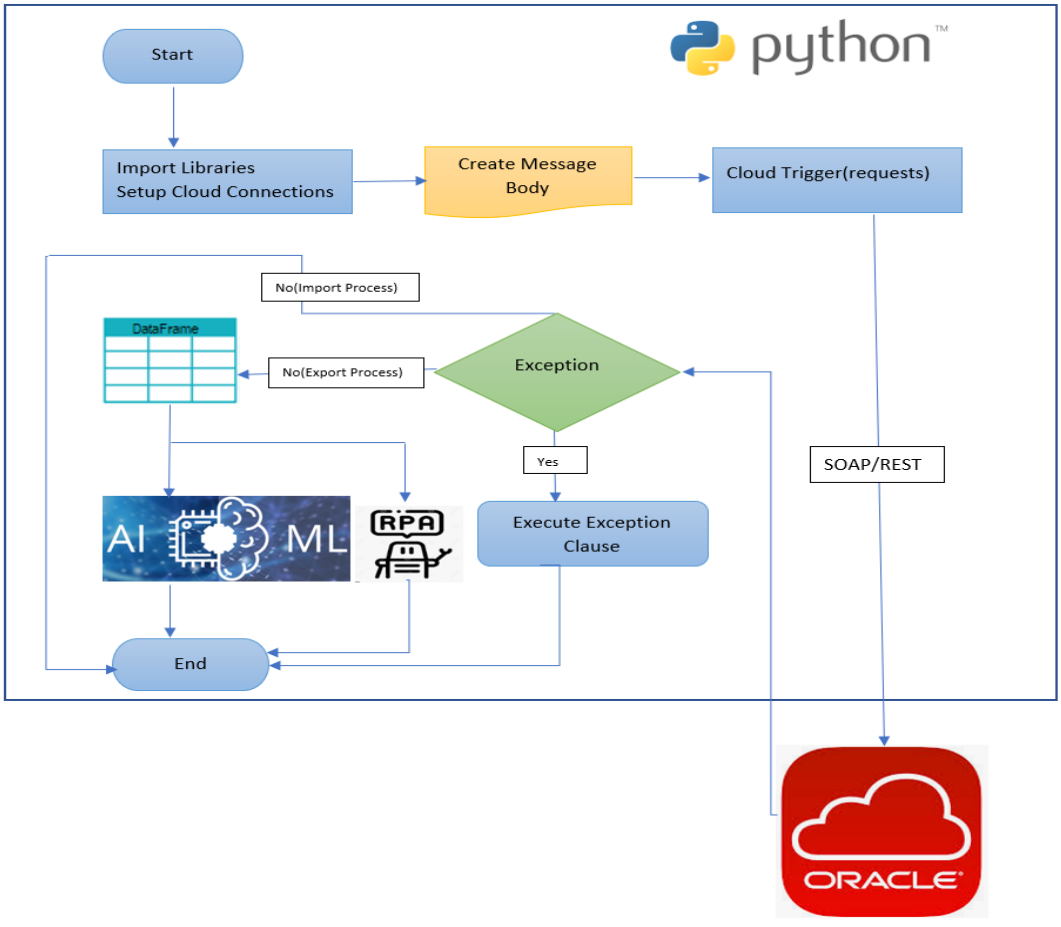

Figure 1. Architecture Design

2.3. Design Overview

This document explores the use of Python for cloud integrations, offering a powerful and cost-effective solution for businesses. Let's first look at some of the advantages of using Python for cloud integrations:

- Open-source and free to use
- Scalable and easy to code
- In-memory processing for high-speed data processing
- Capabilities in machine learning and artificial intelligence
- Access to open-source libraries for complex tasks
- Robotics process automation support

Cloud ERP systems rely on APIs, and Python can be a great choice for managing APIs efficiently. With the help of the open source 'Requests' library, Python can easily consume SAAS APIs, and data can be stored in Pandas Dataframes for efficient data massaging and analytics. The Python code can be reused, scaled, and leveraged for multiple business processes.

The following flow diagram provides a quick overview of the import/export process, which will be discussed in further detail in the next section.

3. Technical Details for Building the Integration

3.1. Prerequisites

- Technical Skills: The implementation of the approach outlined in the article requires a certain level of technical proficiency in Python programming, as well as experience with APIs and cloud technologies.
- Cloud Environment: The approach requires access to a cloud environment, such as Oracle Cloud ERP or other cloud-based services, to implement the integration.
- Integration Tools: The approach requires the use of various tools and technologies, such as Python libraries, frameworks, and data management tools, to develop and implement the integration.
- Data Structure: The data structure of the systems being integrated must be considered and mapped to ensure efficient data transfer and processing.
- Security Measures: The integration must consider security measures, such as data encryption, access controls, and network security, to ensure the confidentiality and privacy of sensitive data.
- Business Processes: The integration must align with the existing business processes and requirements to ensure that it supports the goals and objectives of the organization.
- Maintenance and Support: The approach requires ongoing maintenance and support to ensure that it continues to meet the evolving needs and requirements of the organization.

3.2. Technical Design

In this document, we will examine the use case of integrating with Oracle Cloud ERP and explain how this approach can be applied to other SAAS-based ERP systems.

3.2.1. Import - On-Premises Data Integration to Cloud

A use case of XML file General Ledger data Integration to Oracle ERP Cloud. Same approach can be used for CSV/flat files integration. Below table has logical steps and exact python code:

Table 1. Outbound Integration

Logical Steps	Python Code
Step1 - Import Python Libraries. This is used to invoke REST/SOAP call	import requests
Step2 - Create Message body. Message has actual data to be imported on cloud.	soapBodyRequest="""<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> <soap:Body> <ns1:importJournalsAsync xmlns:ns1="http://xmlns.oracle.com/apps/financials/generalLedger /journals/desktopEntry/journalImportService/types/"> <ns1:interfaceRows xmlns:ns2="http://xmlns.oracle.com/apps/financials/generalLedger /journals /desktopEntry/journalImportService/"> <ns2:BatchName>ORACLEGLBATCH</ns2:BatchName> <ns2:BatchDescription></ns2:BatchDescription> <ns2:LedgerId>10000000123456789</ns2:LedgerId> <ns2:AccountingPeriodName></ns2:AccountingPeriodName> <ns2:AccountingDate>2022-05-05</ns2:AccountingDate> <ns2:UserSourceName>JournalSource</ns2:UserSourceName> <ns2:UserCategoryName>External System</ns2:UserCategoryName>.."""

Step3 - Connection details – POD is ERP instance URL.	url="https://<POD>/fscmService/JournalImportService?wsdl" Headers= {'content-type': 'text/xml','SOAPAction': ''}
Step4 - Initiate SOAP call to post message on Cloud and try to catch exceptions for bad responses or failures. This is final step in data import process.	try: response = requests.post(url,data=soapBodyRequest,headers=Headers, auth=(' user_id','password')) except requests.exceptions.HTTPError as errh: print ("Http Error:",errh) except requests.exceptions.ConnectionError as errc: print ("Error Connecting:",errc) except requests.exceptions.Timeout as errt: print ("Timeout Error:",errt) except requests.exceptions.RequestException as err: print ("Some other error",err)

3.2.2. Export - Cloud Data Extraction for AI/ML/RPA

A use case of extracting real-time data from Oracle Cloud ERP and storing it in Python Data Frames. Once the cloud data is stored in Pandas Dataframes then it can be utilized for:

- Data Analytics – via Scikit-learn.
- Boomerang Integrations of importing back massaged cloud data.
- Robotics Process Automation of any manual steps for same data set – via selenium, browser, keyboard etc.

Below table has logical steps and exact python code:

Table 2. Inbound Integration

Logical Steps	Python Code
Step1 - Import Python Libraries. This is used to invoke REST/SOAP call, xml parsing, base64 conversion.	import pandas as pd import xml.etree.ElementTree as ET import requests import base64 import pandasql import os
Step2 - Connection details – POD is ERP instance URL.	url="https://<POD>/fscmService/JournalImportService?wsdl" Headers= {'content-type': 'text/xml','SOAPAction': ''}
Step3 - SQL Query to be executed on cloud, converted to base64. In following example, we are trying to extract all Journal sources on cloud.	soapQuerySQL = """ select * from gl_je_sources """ soapQuerySQL_bytes = soapQuerySQL.encode('ascii') soapQuerySQLbase64_bytes = base64.b64encode(soapQuerySQL_bytes) soapQuerySQLbase64_message = soapQuerySQLbase64_bytes.decode('ascii')
Step4 - Create Message body. Message has actual query to be run on cloud.	soapQueryRequest="""<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:pub="http://xmlns.oracle.com/oxp/service/PublicReportService" xmlns:pub1="http://xmlns.oracle.com/oxp/service/PublicReportService"> <soapenv:Header/> <soapenv:Body> <pub:runReport> <pub:reportRequest> <pub:attributeFormat>xml</pub:attributeFormat> <pub:attributeLocale>en-US</pub:attributeLocale> <pub:reportAbsolutePath>/Custom/GEI Custom/Integrations/ClientCloudQueryOnly/ClientCloudQueryOnlyReport.xdo </pub:reportAbsolutePath> <pub:parameterNameValues> <pub:item> <pub:name>query1</pub:name> <pub:values> <pub:item>{soapQuerySQLbase64_messageBody}</pub:item> </pub:values> </pub:item>

	</pub:parameterNameValues> </pub:reportRequest> <pub:userID>user_id</pub:userID> <pub:password>password</pub:password> </pub:runReport> </soapenv:Body> </soapenv:Envelope>"""
Step5 - Initiate SOAP call to post message on Cloud and try to catch exceptions for bad responses or failures.	try: response = requests.post(url,data=soapQueryRequest.format (soapQuerySQLbase64_messageBody=soapQuerySQLbase64_message),headers=Headers) except requests.exceptions.HTTPError as errh: print ("Http Error:",errh) except requests.exceptions.ConnectionError as errc: print ("Error Connecting:",errc) except requests.exceptions.Timeout as errt: print ("Timeout Error:",errt) except requests.exceptions.RequestException as err: print ("Some other error",err) st=(response.content) outputXML = open("output.xml", "w") str=st.decode('utf-8') outputXML.write(str) outputXML.close() print("-----XML File Content--------") readXML=open("output.xml","r")

Step6 - Parse response xml data and store data in pandas Dataframes. This is final step in data export process. From this step data can be utilized for AI/ML/RPA/Other Integrations.	for line in readXML.readlines():
	#print (line)
	if 'reportBytes' in line:
	x = line.split('reportBytes')[1]
	x = x.replace('>','')
	x = x.replace('</','')
	# print('ReportOutput--' + x + '\n')
	finalOutput=base64.b64decode(x)
	finalOutput=finalOutput.decode('utf-8')
	outputNewXML = open("finalOutput.xml", "w")
	outputNewXML.write(finalOutput)
	dff = pd.read_xml(finalOutput)
	dff.drop("QUERY1", axis=1, inplace=True)
	CloudDataFrame = dff.drop(0)

4. Use-Cases

The methodology outlined in the article can be applied to various use cases beyond cloud ERP integration, such as:

- Data Integration: Python can be used to integrate data from multiple sources, such as databases, web services, and file systems, into a central repository for analysis and reporting.
- Application Integration: Python can be used to integrate multiple applications, such as accounting, customer relationship management, and enterprise resource planning systems, to streamline processes and improve data accuracy.
- IoT Integration: Python can be used to integrate Internet of Things (IoT) devices and sensors, such as wearable devices, smart home systems, and industrial machinery, to collect and analyze data in real-time.
- Big Data Integration: Python can be used to integrate large data sets, such as log files, social media data, and sensor data, for analysis and reporting purposes.
- Cloud Services Integration: Python can be used to integrate cloud-based services, such as Amazon Web Services, Google Cloud, and Microsoft Azure, to build and deploy cloud-based applications and services.

In each of these use cases, Python can provide a cost-effective and scalable solution for integration, leveraging its open-source libraries, APIs, and in-memory processing capabilities.

5. Tools

- Python: Python is a high-level, interpreted programming language that is widely used for data analysis and web development. It is open-source, easy

to learn, and offers a wide range of libraries and frameworks for various tasks, including API development and data processing.
- Requests Library: Requests is a popular Python library for making HTTP requests to APIs. It is used for consuming APIs in the cloud integration process.
- Pandas: Pandas is an open-source library for data analysis and manipulation. It is used to store and manipulate data obtained from APIs in the cloud integration process.
- Flask: Flask is a lightweight Python framework for web development. It is used to build and host REST APIs in the cloud integration process.
- Django: Django is a high-level Python web framework for full-stack web development. It is used for building complex cloud integration solutions in the integration process.
- Oracle Cloud ERP: Oracle Cloud ERP is a cloud-based enterprise resource planning system offered by Oracle Corporation. It is used as a use case for the integration process in the journal.
- Other Cloud ERPs: The approach discussed in the journal can be applied to other cloud-based ERP systems, such as SAP, Workday, and Microsoft Dynamics, among others.

6. Conclusions

In summary, cloud integration platforms provided by leading cloud providers can present a considerable challenge due to their high operational costs. However, this document proposes a cost-effective alternative solution by using Python for cloud integration. The document outlines the use case of integrating with Oracle Cloud ERP using Python, but the same approach can be applied to other cloud ERPs.

The document delves into the key elements of the cloud integration process, such as the design overview, technical details for implementation, and necessary tools and technologies. The design overview showcases the advantages of cloud integration, while the technical details section offers a comprehensive guide to building the integration, including API usage, data mapping, and security measures. The tools section explains the necessary tools and technologies.

In conclusion, this document highlights the potential benefits of using Python for cloud integration, including cost savings and improved efficiency. With the use of Python, businesses can accomplish seamless cloud integration and reach their objectives with greater efficiency and effectiveness.

REFERENCES

[1] "Python for Data Analysis" by Wes McKinney.

[2] "Flask Web Development with Python Tutorial" by Corey Schafer (YouTube).

[3] "Django for Beginners" by William S. Vincent.

[4] "Integrating Oracle Cloud ERP with External Systems Using REST API" by Oracle.

[5] "Using Python for API Integration" by Nitin Padalia on Medium.

[6] "Integrating Cloud ERP Systems with External Applications: A Guide" by OpenMind Solutions.

[7] "A Beginner's Guide to API Integration" by RapidAPI.

[8] "Cloud Integration for Dummies" by Oracle.

[9] "Python for Finance: Apply powerful finance models and quantitative analysis with Python" by Yves Hilpisch.

[10] "Mastering Python for Finance" by James Ma Weiming.

[11] https://pandas.pydata.org/

[12] https://www.python.org/

[13] https://pypi.org/project/requests/

[14] https://www.oracle.com/erp/financials/