RSEG 176: Cloud Computing

Professor: Ari Davidow

Assignment 1. Document: PDF Conversion

Group A

Joshua Mandell

Kevin Altermatt

Touhedul Islam

Content

|  |  |  |
| --- | --- | --- |
| Item |  | Page no. |
| Background |  | 3 |
| Stakeholder |  | 4 |
| Context |  | 5 |
| Demo |  | 6 |
| Future |  | 6 |

**Background**

Document conversion apps have been around for a while now. There are different types of file conversion apps are available in the market. Their job is to convert PDF documents to doc, XLS, PPT, or the other way around. Needless to say, PDF documents get downloaded by businesses all the time and these documents need to get edited. Most PDF viewers or editors has its costs to use or the free version has limited access. But, most businesses usually use Microsoft word or Google docs. Both of them allow its client to edit the document the way they want. This is where our PDF to Docx conversion app comes in. The client can easily upload their documents on the server and get a copy of Docx. file. Now, we can build this PDF converter app in a traditional approach where we have to own our server, write programming language in our local machine, add dependencies, use build software, etc. But this traditional path has its troubles and it affects our workflow and deployment of the software. Development time is high on the traditional approach as the whole software is released in one big package. Operating system dependency can be the major issue that hampers the scalability of the application. It’s important for software to be run on any kind of machine. Release pace is also a big factor when we try to implement the application. In the traditional approach, it would take a huge time to find bugs and then apply the changes. Hard coding, downtime, and low backup capabilities are also some negative aspects of the traditional approach. (Gill, 2022) On the other hand, we can build the app as a cloud Native app which will solve the above problem. We don’t have to deal with operating system dependency since the cloud provider will work as an Infrastructure as Service(IaaS) that provides frameworks, IDE, APIs, and Networking altogether. On top of that, we can save costs as we will pay only for what we use. We don’t have to deal with data storage. We decided to use Amazon web service (AWS) our cloud provider which is the biggest player in the cloud computing market. AWS is a platform that offers computing solutions, storing, and networking in different layers of abstraction. We can host websites, run applications, and run the database. We can have virtual machines where we install needed software (EC2) and Amazon S3 which is an object storage service. (Wittig & Wittig, 2018) With the above-sophisticated features of AWS we can build an app from scratch or we can run an existing application. As I mentioned before, for our group project we had the idea of a web app that can take in written documents in different formats and return them with a changed format. The first step was to turn a PDF file into a .docx file and to host this application on an AWS account.

**Stakeholders**

* **End users:** Individuals or organizations that need files changed from one format to another. These might include publishing groups or any organization that works with large numbers of documents that may need to have their formats changed. Success for these stakeholders means that the application can quickly, efficiently, and accurately – without error – change the format of files, and that these files will be secure and accessible.
* **Developers:** As the creators of the application our measure of success is meeting the needs of the end users and having our application be able to host stably on our web server without errors and without using up more resources than we want to be used. As a developer, Josh, kevin and touhedul worked on to build the app.
* **Tester:** As the director of the whole process, Professor Ari Davidow will test the application to check whether hosted app meets the criteria or not. If any improvement needed he can advice us the path for further improvement. Assignment 1 is the first part of the whole project and we have to add other relevant feature to Assignment 2. Upon approval from tester, we can start working on 2nd part of the project.

**Context**

To accomplish this task, we initially started with an ideal concept framework. Where a front-end user interface acts as a medium between the action of a user uploading and tagging (with meta-data) the document and the back-end manipulation of the document. The back end converts the document by adding a watermark and stores the document on AWS. The user can then retrieve the document using the meta-data or document name in the new converted form. The architecture purposed for this solution is the front-end application API connected with an AWS Lambda function to initiate the business functions and Amazon DocumentDB to store the document. This architecture would run in reverse for document retrieval.

**Issues and Troubleshooting:** At first, we tried using AWS architecture directly and uploading our Python code to a lambda function, building the application using AWS tools piece by piece. However, this approach ran into problems. One of the issues we were facing in the backend was to working with python libraries and other dependencies. We needed to upload the libraries and dependencies in AWS S3 and connect those dependencies as a layer in AWS lambda. And then we created a function in S3where the code runs. On the code console, we can add the layer which consisted of all the libraries and dependencies from AWS S3. But this testing was keep failing as it was getting issues with python libraries and dependencies. One other issue that we found in building this application was a successful front-end application that could receive the document. We started working with Amplify but there were significant issues when trying to attempt to push a basic HTML code written to the console. When this was unable to be resolved other ways were attempted. One such way was creating a React application in Visual Studio however although the code compiled without failure it did not load on the webpage. We had also looked at using Amplify CLI instance instead but were met with another issue that we had found is crashing CLI instances on connection attempts. The biggest hurdle was that the issues slowed our team's progress in developing the connected architecture in AWS. However, we had significant progress in developing the Python application to perform the business functions as well as some success in connecting Python through an EC2 Instance for future front-end application development.

What ultimately solved our difficulties was hosting the application – a python & HTML website using the Flask framework – on an EC2 machine. The Flask framework allows machines to host websites that can be navigated using a web browser, as well as allowing the storage of files on the instance itself. Because of the capabilities of AWS as a platform, the EC2 machine can be enabled or disabled at will, and as the administrator, I can remote into the EC2 machine to bring the website up or down and make changes to the code as needed.

**Demo:**

After having success with AWS flask we can run our application from the EC2 server. It has a general front end where the user can upload the PDF files that need to be converted and in the back end where we can perform the conversion, and file manipulation and passes the finished files back to the front end where the user can download the file. The backend code has been written in Python to perform file conversion from PDF to DocX format. It is possible to perform other file manipulations as well depending upon the user’s needs, which can be added to the application later. Once we run the app on EC2, we can open the public port so that others can access the site to get their PDF converted. We can get the address from the public Ipv4 column. For example, Josh runs his AWS EC2 where he turns on the public IP address so that someone can access the site. He kept his EC2 running and we can access the pdf converter by using the following link <http://18.212.177.201:8080/> or <http://ec2-18-212-177-201.compute-1.amazonaws.com:8080/>

**Future**

As it stands this project is very basic and has only a barebones interface, changing files from .pdf to .docx format. Ideally in the future we would like to improve the interface, add the ability to convert from and to more formats, add security and monitoring features to the application, and perhaps even add OCR (Optical Character Recognition) for the reading of PDFs that do not currently have searchable text. This last feature may be overly ambitious but I would consider it a stretch goal if we can find a way to do it.

**References:**

1. Gill, N. S. (2022, June 1). *Which one to choose: Cloud-native or traditional app development?* Elixirdata. Retrieved November 7, 2022, from <https://www.xenonstack.com/insights/traditional-vs-cloud-native-applications>
2. Wittig, M., & Wittig, A. (2018, September). Amazon Web Services in action, Second edition. O'Reilly Online Learning. Retrieved November 7, 2022, from https://learning.oreilly.com/library/view/amazon-web-services/9781617295119/kindle\_split\_012.html#:-:text=Amazon%20Web%20Services%20(AWS),n%20design%20a%20new%20setup%20from