Documentation: Custom PDF Generation REST API

**Summary**

**DOCX to PDF REST API – Project Overview**

This document describes the design, technology stack, and architecture of a REST-based service for converting Microsoft Word documents (.docx) to PDF format in a secure, cloud-hosted environment.

**Goal:** Build a lightweight, secure, and cloud-hosted REST API that converts .docx documents to .pdf.

**Why:**

* Many enterprise systems (e.g. ServiceNow) generate or handle DOCX files
* Converting to PDF ensures consistency, security, and read-only distribution

**Key Benefits:**

* Fully API-driven
* Fast and reliable
* No file storage
* Easy to integrate into ServiceNow workflows

**Technology Stack**

**Detailed Overview**

The solution has been built with a carefully selected technology stack that ensures reliability, performance, maintainability, and security. Here's a breakdown of each component and its purpose:

**Python 3 + Flask**

* Python is a powerful, high-level programming language known for its readability and wide adoption in enterprise automation, scripting, and backend services.
* Flask is a lightweight, minimalistic web framework for Python that’s ideal for building fast and efficient REST APIs.
* Flask is used to expose the /convert/docx/to/pdf POST endpoint
* Accepts binary .docx content and streams back PDF output
* Used to handle:
  + File upload via POST method
  + Interfacing with the document conversion layer
  + Streaming PDF output as a response
* Flask keeps the application lean and avoids unnecessary complexity or overhead

**LibreOffice (Headless Mode)**

* LibreOffice is a powerful open-source office suite that supports a wide variety of document formats.
* In this implementation, LibreOffice is run in headless mode, which means:
  + No graphical user interface (GUI) is launched
  + All conversions are performed via command line (libreoffice --headless)
  + Ideal for server environments where no desktop is available
* It ensures high-accuracy and quality conversion of .docx documents to .pdf with layout, fonts, images, and formatting preserved — without requiring Microsoft Word or any software.

**Docker**

* Docker is used to package the entire application into a portable container, including:
  + Python runtime
  + Flask application
  + LibreOffice installation
  + All dependencies
* Benefits:
  + Environment consistency: The application runs the same in development, staging, and production
  + Isolation: The container has no access to host machine files or memory
  + Portability: Can be deployed anywhere Docker is supported

**Render**

* [Render](https://render.com/) is a modern cloud platform that:
  + Builds and runs Docker-based applications
  + Automatically deploys from GitHub on every push
  + Provides a stable HTTPS endpoint
  + Offers free plans that are ideal for small microservices
* Benefits:
  + Zero infrastructure management — no need to configure or maintain servers
  + Automatic scaling — only pays for what you use (if upgraded)
  + Code pushed to GitHub triggers automatic builds and deployments in Render

**GitHub**

* GitHub is used to version-control the entire codebase, including:
  + app.py (main logic)
  + Dockerfile (container instructions)
  + render.yaml (deployment blueprint)
  + requirements.txt (Python dependencies)
* Benefits:
  + History tracking: Every change is recorded and reversible
  + Team collaboration: Easily extendable by other developers or teams
  + Integration: Works smoothly with Render’s automatic deployment pipeline

**Summary of Technology Stack**

|  |  |
| --- | --- |
| **Component** | **Role Architecture** |
| Python + Flask | Backend logic, request handling, PDF streaming |
| LibreOffice | Actual .docx to .pdf conversion engine (headless, secure) |
| Docker | Contains the full environment and dependencies |
| Render Web Service Hosting | Host and serves the Docker container via HTTPS endpoint |
| GitHub | Code storage, version control, and automatic deployment integration |

**Solution Architecture**

**Request Flow:**

* Receives a .docx file via POST method (Content-Type: application/octet-stream)
* Saves it inside the container
* Uses LibreOffice (headless mode) to convert .docx → .pdf
* Streams the PDF back to the client (via send\_file)
* Deletes both files immediately after conversion from the container using a finally block

**Files involved:**

* app.py: Flask app handling the logic
* Dockerfile: Defines how to build the image
* render.yaml: Configures the deployment to Render
* requirements.txt: Python dependencies (Flask)

**Security & Data Privacy**

This API is designed for sensitive documents and enterprise use cases:

* Files are not saved and stored permanently
* All files are removed in a finally block after processing
* No content is logged or retained
* Files are stored only in-memory or in isolated container paths
* No third-party dependencies or logging services are used

**Integration with ServiceNow**

**How it's used:**

* ServiceNow sends .docx file via REST (using sn\_ws.RESTMessageV2)
* API returns the converted .pdf file
* PDF is attached back to the target record (e.g. incident, HR case)

**Use cases:**

* Outbound REST Messages
* Business Rules
* Script Include
* Scheduled Jobs
* Flow Designer custom steps
* UI Actions or Workflows

**How the Docker Image Works**

* Based on python version 3.
* Installs LibreOffice in headless mode
* Flask app starts on the port specified by Render ($PORT)
* No dependencies on host system
* Fully isolated and portable

**Deployment Strategy**

**A diagram of a webpage

AI-generated content may be incorrect.**

* Hosted on Render Web Service Hosting via containerized build
* Triggered via git push from GitHub
* Uses Docker + render.yaml for repeatable deployment
* No infrastructure or server maintenance needed
* Runs on the Free plan (can be upgrade if needed)