# Building Scalable REST APIs with FastAPI

## Introduction

FastAPI is a modern, high-performance web framework for building APIs with Python.
It is designed for speed and ease of use, leveraging Python's type hints to provide automatic validation and API documentation. It is widely used for building scalable and maintainable web applications, microservices, and real-time data processing systems.

## Use Cases of FastAPI

1. Building Authentication & Authorization Systems
2. Developing CRUD APIs for Web and Mobile Apps
3. Creating Microservices in a Scalable Architecture
4. Data Processing and Real-Time APIs
5. Machine Learning Model Deployment as an API

## Use Case 1: Authentication & Authorization

FastAPI makes it easy to build secure authentication systems using JWT (JSON Web Tokens).
By implementing OAuth2 and JWT authentication, developers can protect API endpoints and manage
user authentication seamlessly.

### Sample Code: JWT Authentication

```
from fastapi import FastAPI, Depends
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt

app = FastAPI()
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/protected")
async def protected_route(token: str = Depends(oauth2_scheme)):
    try:
        payload = jwt.decode(token, "secret", algorithms=["HS256"])
        return {"message": "Access granted", "user": payload}
    except JWTError:
        return {"error": "Invalid token"}
```

## Deployment Best Practices

To deploy a FastAPI application efficiently, follow these best practices:

- Use Docker for containerized deployments.
- Implement Gunicorn with Uvicorn workers for better performance.
- Use NGINX as a reverse proxy to handle load balancing.
- Secure the application with HTTPS & environment variables.