

# Cloud Foundations & Orchestration

**CS 203: Software Tools and Techniques for AI**

Prof. Nipun Batra, IIT Gandhinagar

# Beyond "Localhost"

Works on my machine != Works in production.

## Production Needs:

1. **Uptime:** 24/7 availability.
2. **Scalability:** Handle 1 user or 10,000 users.
3. **Security:** SSL, Firewalls, Secrets management.
4. **Coordination:** Multiple services (API, DB, Cache) talking to each other.

## Today's Stack:

- **Docker Compose:** Orchestrating multi-container apps.
- **Cloud Providers:** AWS/GCP/Azure vs. PaaS (Render/Railway).
- **IaC:** Infrastructure as Code concepts.

# Multi-Container Apps

Real AI apps aren't just one Python script.

- **Frontend:** Streamlit/React.
- **Backend:** FastAPI.
- **Database:** Postgres/MongoDB.
- **Vector DB:** Chroma/Weaviate.
- **Cache:** Redis.

**Problem:** Starting them all manually is painful.

```
python api.py ... npm start ... redis-server ...
```

**Solution:** Docker Compose.

# Docker Compose

Define your entire stack in one YAML file ( `docker-compose.yml` ).

```
version: '3.8'

services:
  api:
    build: ./api
    ports:
      - "8000:8000"
    depends_on:
      - redis
    environment:
      - REDIS_URL=redis://redis:6379

  redis:
    image: "redis:alpine"

  frontend:
    build: ./frontend
    ports:
      - "8501:8501"
```

# Networking in Compose

## Magic DNS:

- Services can talk to each other by service name.
- Inside the `api` container, `redis` resolves to the IP of the redis container.
- No need to hardcode IPs.

## Volumes:

- Persist database data even if container dies.

```
db:
  image: postgres
  volumes:
    - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

# Cloud Computing Models

## 1. IaaS (Infrastructure as a Service):

- **AWS EC2, Google Compute Engine.**
- You get a virtual machine (Linux box).
- You install Docker, Nginx, Python.
- **Pros:** Full control. **Cons:** You manage OS updates, security.

## 2. PaaS (Platform as a Service):

- **Heroku, Render, Railway, Google App Engine.**
- You push code/Dockerfile.
- They handle servers, load balancing, SSL.
- **Pros:** Easy. **Cons:** More expensive, less control.

# Deploying to a PaaS (Render/Railway)

The easiest path for student projects.

1. **Push to GitHub.**
2. **Connect Repo** in Render/Railway dashboard.
3. **Detects Dockerfile** automatically.
4. **Deploys.**
5. **Provision DB:** Click "Add PostgreSQL" -> get connection string -> add to Environment Variables.

## Why use PaaS?

- Free tiers often available.
- HTTPS (SSL) out of the box.
- CI/CD built-in (deploys on push)

# Deploying to IaaS (AWS EC2)

The "Standard" Industry Way.

1. **Launch Instance:** Select Ubuntu, t3.micro (Free tier).

2. **SSH Access:** `ssh -i key.pem ubuntu@1.2.3.4`

3. **Setup:**

```
sudo apt update
sudo apt install docker.io docker-compose
git clone https://github.com/my/repo.git
cd repo
docker-compose up -d
```

4. **Security Group:** Open ports 80 (HTTP) and 443 (HTTPS).

**Challenge:** Keeping it running (systemd), Logs, Updates.



# Infrastructure as Code (IaC)

**Problem:** Clicking buttons in AWS console is manual and error-prone.

**Solution:** Define infrastructure in code (Terraform, Pulumi).

```
# Terraform Example (Conceptual)
resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "AI-App-Server"
  }
}
```

## Benefits:

- Reproducible environments.
- Version control your infrastructure.

# Secrets Management

NEVER commit API keys to Git.

Bad:

```
api_key = "sk-123456" # ❌
```

Good:

```
import os  
api_key = os.getenv("OPENAI_API_KEY") # ✅
```

In Production:

- Use `.env` files (but don't commit them!).
- Inject via Cloud Provider's "Environment Variables" settings.
- Use Secret Managers (AWS Secrets Manager HashiCorp Vault) for enterprise

# Lab: Full Stack Deployment

**Objective:** Deploy the Week 9 RAG App to the cloud.

**Steps:**

1. **Docker Compose:** Create `docker-compose.yml` for App + VectorDB (if local) or just App.
2. **Environment Variables:** Move API keys to `.env`.
3. **Deploy:**
  - **Option A (Easy):** Deploy to Render.com.
  - **Option B (Hard):** Provision an AWS EC2 instance, SSH in, and run `docker-compose up`.
4. **Verify:** Access the public URL.

# Resources

- **Docker Compose Docs:** [docs.docker.com/compose](https://docs.docker.com/compose)
- **The Twelve-Factor App:** [12factor.net](https://12factor.net) (Principles for cloud apps)
- **AWS Free Tier:** [aws.amazon.com/free](https://aws.amazon.com/free)
- **Render Docs:** [render.com/docs](https://render.com/docs)

**Questions?**