Lesson 44: Cloud Deployment

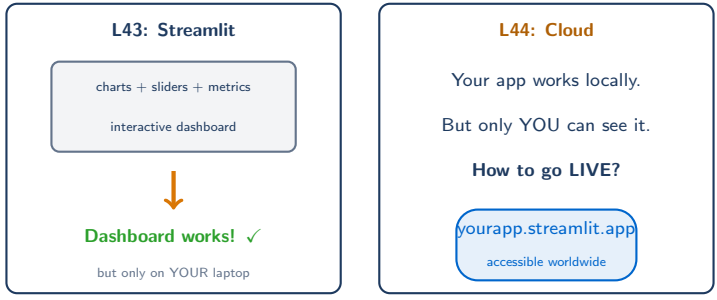
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Previously: Models People Can See



L43 built the dashboard. Now: make it accessible to anyone, anywhere, anytime.

Local works. Cloud scales. The final step in the deployment pipeline.

Learning Objectives

The Problem: Your Streamlit app works on your laptop. But only YOU can see it. How do you put it on the internet so anyone with a URL can access it?

After this lesson, you will be able to:

- Deploy apps to Streamlit Cloud (free, 3 steps)
- Containerize applications with Docker
- Set up CI/CD pipelines with GitHub Actions
- Manage secrets and monitor deployed applications

Finance Application: Sharing prediction dashboards with clients, deploying ML services

Local Works. Cloud Scales.

Why Cloud Deployment?

- **Accessibility:** Anyone with the URL can use your app
- **Reliability:** Server runs 24/7, not just when your laptop is open
- **Scalability:** Handle 100 users, not just you
- **Professionalism:** A live URL beats a Jupyter notebook on a resume

Analogy – Home Kitchen to Restaurant:

- Local development = cooking at home (only you can eat)
- Cloud deployment = opening a restaurant (anyone can visit)
- The cloud = the building, utilities, and staff (you bring the recipe)

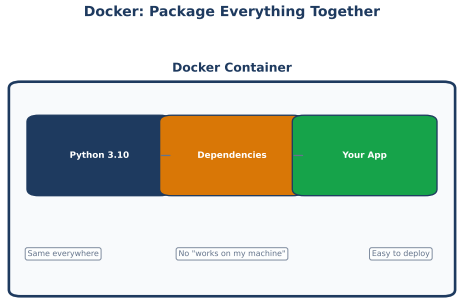
For Students: A deployed app URL in your portfolio is worth more than 10 notebooks.

Cloud deployment = your app accessible to anyone, anywhere, anytime

Docker: Packaging Your App

The “It Works on My Machine” Problem

- Your app needs Python 3.11, specific packages, OS libraries
- Docker packages EVERYTHING into a single container
- Container runs identically on any machine



Docker container = your app + all dependencies, runs the same everywhere

Dockerfile: The Recipe

A Dockerfile Tells Docker How to Build Your Container:

- `FROM python:3.11-slim`
- `WORKDIR /app`
- `COPY requirements.txt .`
- `RUN pip install -r requirements.txt`
- `COPY . .`
- `EXPOSE 8501`
- `CMD ["streamlit", "run", "app.py"]`

Build and Run:

- `docker build -t my-app .`
- `docker run -p 8501:8501 my-app`
- Visit `http://localhost:8501` – same as local!

Why `python:3.11-slim`? Smaller image = faster deployment. Full image is 1GB+, slim is 150MB.

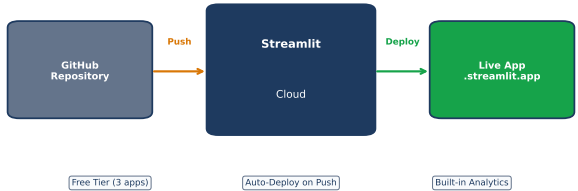
Dockerfile = recipe. docker build = cook. docker run = serve. That simple.

Streamlit Cloud: The Easiest Option

Free Deployment in 3 Steps

- Push code to GitHub, connect to Streamlit Cloud, deploy
- No Docker, no servers, no configuration

Streamlit Cloud: GitHub to Live App



Streamlit Cloud: push to GitHub, click deploy, get a URL. That's it!

Streamlit Cloud: Step by Step

What You Need in Your Repo:

- `app.py` – your Streamlit application
- `requirements.txt` – Python dependencies (pinned versions)
- `model.joblib` – saved model (if small, <100MB)
- Data files or external data API connection

requirements.txt:

- `streamlit==1.28.0`
- `pandas==2.1.0`
- `scikit-learn==1.3.0`
- `plotly==5.17.0`
- Generate: `pipreqs . --force` (only imports your code uses)

Limitations (Free Tier):

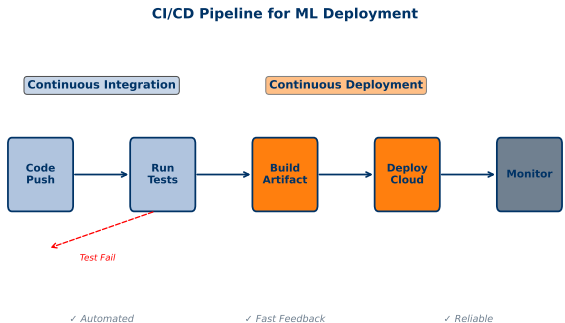
- 1 GB RAM, limited CPU. App sleeps after inactivity.
- Public repos only (private repos need paid plan)

Pin versions with == for reproducibility. Use pipreqs to auto-generate.

CI/CD: Automatic Deployment

Continuous Integration / Continuous Deployment

- Push code → tests run automatically → deploy if tests pass
- No manual deployment steps – code push triggers everything



CI/CD: push to GitHub = automatic test + deploy. No manual steps.

GitHub Actions: CI/CD Made Simple

Automate Testing on Every Push:

- `.github/workflows/test.yml`:

Example Workflow:

```
name: Test
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-python@v5
    - run: pip install -r requirements.txt
    - run: pytest tests/
```

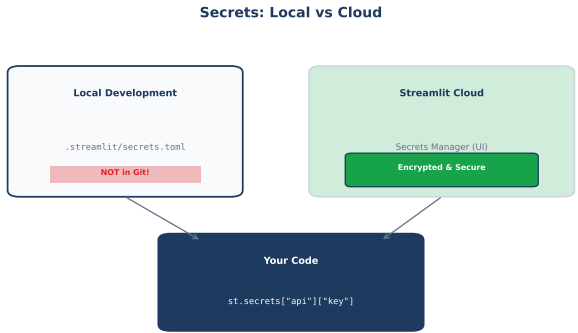
Result: Every push runs your tests. Green check = safe to deploy.

GitHub Actions: free CI/CD. Define in YAML, runs automatically on every push.

Secrets: Never Hardcode API Keys

Keep Sensitive Data Safe

- API keys, database passwords, tokens must stay SECRET
- Never commit them to GitHub – anyone can see public repos!



NEVER commit secrets to git! Use environment variables or secrets management.

Secrets: Implementation

Streamlit Secrets:

- In Cloud: Settings → Secrets → paste TOML config
- In code: `api_key = st.secrets["API_KEY"]`
- Local: `.streamlit/secrets.toml` (add to .gitignore!)

Environment Variables (Docker/Cloud):

- `import os`
- `api_key = os.environ.get("API_KEY")`
- Docker: `docker run -e API_KEY=secret my-app`
- GitHub Actions: stored in repo Settings → Secrets

.gitignore Essentials:

- `.streamlit/secrets.toml`
- `.env`
- `*credentials*`
- `__pycache__/`

Three rules: never commit secrets, use .gitignore, use environment variables

Checkpoint: Why Docker?

Quick Check:

Your colleague says: "Why not just copy my Python files to the server? Why do I need Docker?"

What's wrong with "just copy files"?

1. Server might have different Python version
2. Server might be missing system libraries (libgomp for sklearn)
3. `pip install` might install different package versions
4. OS differences (Windows → Linux) can break code

Docker solves ALL of these:

- Container includes Python, all packages, all libraries
- Same container runs identically on any machine
- "It works on my machine" → "It works in my container"

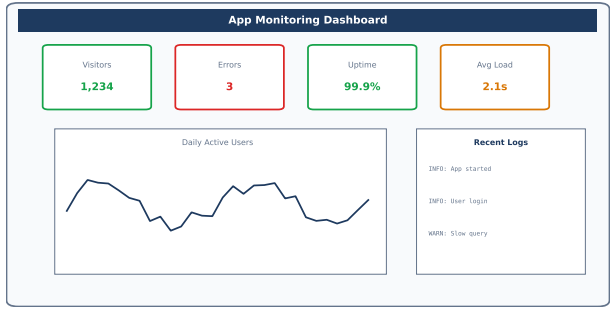
Docker: same environment everywhere. No more "works on my machine" surprises.

Monitoring Your Deployed App

Keeping Your App Healthy After Launch

- Track uptime, errors, response times, resource usage
- Set alerts for failures before users notice

Monitor Your Deployed App



Deploy is not the end. Monitoring is what keeps your app alive.

Common Deployment Problems

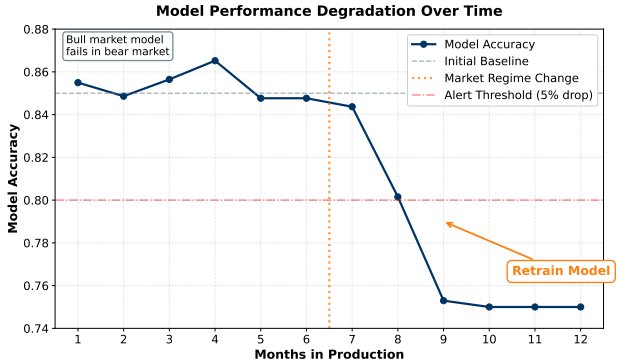
Things That Go Wrong (and How to Fix Them):

- **App crashes on deploy:**
  - Missing package in requirements.txt – run `pipreqs`
  - Wrong Python version – specify in `runtime.txt`
- **App is too slow:**
  - Large data files – use caching and external storage
  - Missing `@st.cache_data` – add caching decorators
- **Memory errors:**
  - Free tier: 1GB limit – optimize data loading
  - Load only needed columns: `pd.read_csv(..., usecols=[...])`
- **Model file too large:**
  - Use joblib compression: `compress=3`
  - Store on S3/GCS, download at startup

Most deployment failures: missing dependency or memory limit. Check logs first.

Model Drift: When Models Go Stale

The Problem: Your model was great in January. It's June now. Is it still good?



Solution: Monitor prediction distributions. Retrain when drift is detected.

Models degrade over time. Monitor and retrain regularly.

A/B Testing Models in Production

Comparing Model Versions Safely:

- Route 90% of traffic to current model (v1)
- Route 10% to new model (v2) as a test
- Compare predictions, accuracy, latency over time
- If v2 is better: gradually shift traffic

Implementation Pattern:

```
import random
if random.random() < 0.1:
    pred = model_v2.predict(features)
else:
    pred = model_v1.predict(features)
```

- Log which model served each prediction

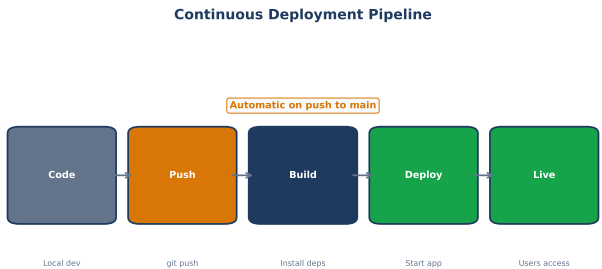
Finance: Never switch a trading model without A/B testing first – the cost of a bad model is measured in dollars.

A/B testing: safe model upgrades. Test before you commit to a new version.

Complete Deployment Workflow

From Code to Cloud

- Test locally, containerize, push, deploy, monitor
- Each step has a verification checkpoint



Test locally first! Most deployment failures are missing dependencies.

Finance: Production ML Checklist

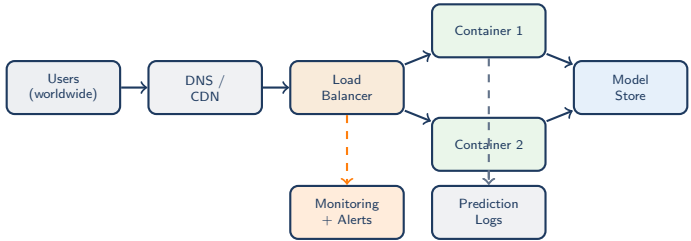
Before Going Live with a Financial ML System:

- ☐ **Model validation:** Tested on out-of-sample data
- ☐ **Version control:** Model artifact versioned with metadata
- ☐ **Monitoring:** Drift detection and performance alerts
- ☐ **Logging:** Every prediction logged with timestamp
- ☐ **Rollback plan:** Can revert to previous model version
- ☐ **Secrets:** API keys in environment variables, not code
- ☐ **Testing:** Unit tests + integration tests + load tests
- ☐ **Documentation:** API docs, model card, deployment runbook

Regulatory: Financial models require audit trails, version histories, and explainability documentation.

Production ML in finance: not just "does it work?" but "can we prove it works?"

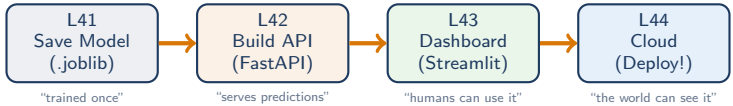
Production Cloud Architecture



Production stack: DNS → load balancer → containerized app → model store. Plus monitoring and logging.

This is what "going to production" looks like. Docker containers behind a load balancer.

Module 9: From Notebook to Production



From notebook to production!

★

Module 9 complete. You can now take any ML model from training to a live, deployed application.

Save → Serve → Show → Ship. The complete deployment pipeline.

Hands-On Exercise (25 min)

Task: Deploy Your Dashboard to the Cloud

1. Take your Streamlit app from L43
2. Create requirements.txt with `pipreqs . --force`
3. Add `.gitignore` (secrets, __pycache__, .env)
4. Push to a GitHub repository
5. Deploy on Streamlit Cloud
6. Share the deployed URL with a classmate

Verification:

- Can your classmate access the URL from their laptop?
- Do all widgets and charts work on the deployed version?

Deliverable: Live deployed URL + screenshot of running app.

Extension: Add a Docker container and deploy to Hugging Face Spaces

Going Live

You are now a full-stack ML engineer. Model + API + dashboard + cloud = production ML.

Module 9 turns you from "data scientist" into "ML engineer who ships"

Lesson Summary

Problem Solved: We can now deploy ML apps to the cloud so anyone with a URL can access them.

Key Takeaways:

- **Streamlit Cloud:** free, easy – push to GitHub, click deploy
- **Docker:** packages app + dependencies into portable container
- **CI/CD:** GitHub Actions automates testing and deployment
- **Secrets:** never commit API keys – use environment variables
- **Monitoring:** deploy is not the end – watch for drift and errors

Module 9 Complete! Save $\rightarrow$ API $\rightarrow$ Dashboard $\rightarrow$ Cloud.

Next: Module 10 – Capstone & Ethics (L45–L48). Apply everything you've learned.

You can now go from trained model to live deployed application. That's ML engineering.