



# The Core Problem in Al/ML The "Works on My Machine" Nightmare (2)

- Dependency Hell: Al models rely on complex, often conflicting dependencies (TensorFlow, PyTorch, CUDA, specific Python versions, OS libraries).
  - Example: Model trained with PyTorch 1.12 fails on a production server with PyTorch 2.0.
- Environmental Drift: Differences between the development, staging, and production environments.
- Irreproducibility: Inability to recreate the exact environment used to train and validate a model, leading to inconsistent results.

#### Key Docker Solution: Image Isolation

• Docker packages the application and its environment.



Code + Environment = Immutable Artifact 🕡

- Guaranteed Environment: The Dockerfile specifies the exact OS, Python version, and library versions, ensuring the model's runtime environment is identical everywhere.
  - Analogy: A shipping container where the contents (model, code) are protected by the reinforced shell (OS, dependencies).
- Version Control for Infrastructure: When you commit your Dockerfile and requirements.txt (or a similar lock file) to Git, you are versioning your entire runtime environment.
- Model Auditability: Knowing the precise software stack a model runs on is critical for security and regulatory compliance.

## Benefit 2: Seamless Deployment and Portability

### Deploy Anywhere, Instantly -

**Cloud Agnostic:** A Docker image can run unmodified on any major cloud platform (AWS, Azure, GCP) or on-premises servers.

 No Re-tooling: Transitioning from an on-premises GPU server to an AWS EC2 instance requires no code or environment changes.

CI/CD Integration (MLOps): Containers are the standard unit of deployment in MLOps pipelines.

- Automated Testing: Test the container image in a staging environment.
- Fast Rollbacks: Roll back to a previous, stable model version by simply deploying an older container image tag.

**Simplified Scaling:** Container orchestration tools (Kubernetes) easily replicate your model's container to handle thousands of requests per second.

### Benefit 3: Security and Resource Isolation

#### Secure by Design: From Build to Runtime 🔒

- Resource Management: Docker isolates processes, preventing an overzealous model or process from consuming all host resources. You can cap CPU and memory usage.
- Minimized Attack Surface: Using multi-stage builds (like the example in your Dockerfile) ensures the final image is slim, containing only the runtime components, not build tools or development libraries.
- Non-Root Execution: Running the Al service as a non-root user (as seen in your Dockerfile with appuser) significantly limits the damage an attacker can do if a vulnerability is exploited.

#### Your Dockerfile Example:

- Build-Stage Security: The pip-audit step ensures vulnerable dependencies are caught before deployment.
- Runtime Security: Creating and switching to the appuser limits privileges in the final image.

Slide 6: Summary & Call to Action

Docker: The Essential MLOps Wrapper

Benefit Impact on AI Workflow

Consistency Eliminates "Works on My Machine" errors and ensures model parity.

Portability Enables easy migration across cloud providers and environments.

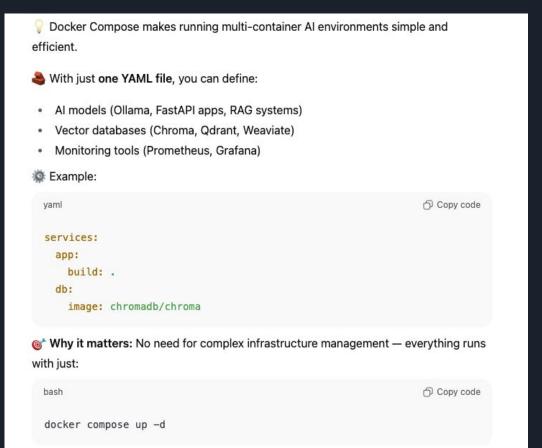
**Security** Reduces attack surface through minimal images and non-root users.

Scalability Standardizes deployment for Kubernetes/orchestration.

Export to Sheets

Call to Action: Implement multi-stage Docker builds and dependency auditing in your next Al project to secure and simplify your MLOps pipeline.

# Why Docker Compose for AI Workloads?



## Perfect for Rapid Prototyping

- 🚀 When working with Al projects, speed matters.
- **Compose allows you to:** 
  - Spin up multiple services instantly
  - Test LLMs, APIs, and databases together
  - Rebuild and reset your environment quickly
- Example use cases:
  - Testing new embeddings in Chroma
  - Switching between models like Mistral or Llama
  - Integrating RAG pipelines rapidly
- ✓ In Al labs, time-to-iteration is everything and Docker Compose delivers that.

## Lightweight Alternative to Kubernetes

Kubernetes is powerful, but heavy for small AI projects.

Docker Compose gives 80% of the functionality with 20% of the complexity.

Perfect for students, startups, and prototype labs before scaling to Kubernetes.

Kubernetes	Docker Compose
Complex	2 mins
High	Low
Production clusters	Local & small-scale AI workloads
Steep	Easy
Large	Minimal
	Complex  High  Production clusters  Steep

## Easy Integration with AI Tools

- **compose integrates easily with AI components like:** 

  - Chroma / Qdrant → vector databases for retrieval
  - FastAPI → lightweight AI agent server
  - Grafana + Prometheus → system observability
- Each service is isolated, reproducible, and restartable.
- You can rebuild specific containers without touching others.
- Example: Update your LLM version in seconds while keeping Chroma's data safe.

## Real Benefits for AI Engineers

- Why Docker Compose belongs in every Al Engineer's toolkit:
  - 1. Modular Architecture isolate Al agents, databases, and APIs.
- 3. Grant Security Controls can add .env and network isolation.
- 4. Observability pair with Prometheus + Grafana easily.
- 5. Focus on AI, not Infrastructure spend time training and deploying, not debugging Kubernetes YAMLs.
- "Docker Compose bridges the gap between AI experimentation and production deployment."
- At Remoder, we use it to teach engineers how to design real-world AI systems fast, secure, and scalable.

## QUESTIONS???

Contact us & come learn with us  $9(9_{\circ})^{\circ}$ 

- https://www.linkedin.com/company/remoder
- https://www.linkedin.com/in/sanjars/
- https://www.youtube.com/@remoder-inc
- remoder.com
- Full walkthrough of this project is available on our "Master Al Deployment " course