AI in a Box: Mastering Containerization for AI Computer Vision

Learn how to package and deploy your computer vision models using containers.

Hossam Tarek
Software Engineer @ DevisionX
htarek@devisionx.com

Agenda

1. Introduction to Containerization
2. Why Containerization for AI and Computer Vision
3. Key Tools & Concepts (Docker, Images, Containers, etc.)
4. Building a Docker
5. Tips for GPU Acceleration
6. Workflow & Best Practices
7. Performance Considerations & Hardware Optimizations
8. Tuba Workflow Use Case
9. Quick Demo
10. Additional Resources & Next Steps
11. Q&A

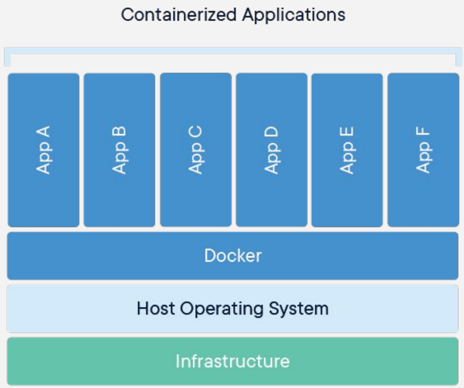
What is Containerization?

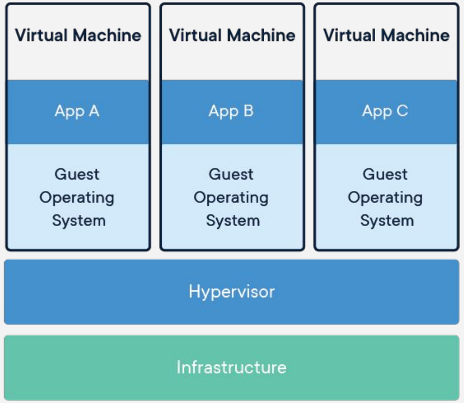
Definition & Core Concept:

- **Containerization** is a method of packaging software so that it can run consistently across different computing environments.
- Encapsulates code, runtime, system tools, libraries, and settings.

Benefits:

- **Consistency:** Works the same in development, testing, and production.
- **Isolation:** Each container is self-contained and doesn't affect others.
- **Portability:** Containers can run on any platform that has Docker (or another container runtime).
- **Efficiency:** Faster spin-up times compared to traditional virtual machines.





Why Containerization for AI + Computer Vision?

Dependency Management

- CV frameworks (TensorFlow, PyTorch, OpenCV) often have complex dependencies.
- Containers ensure version consistency across machines.

Reproducibility

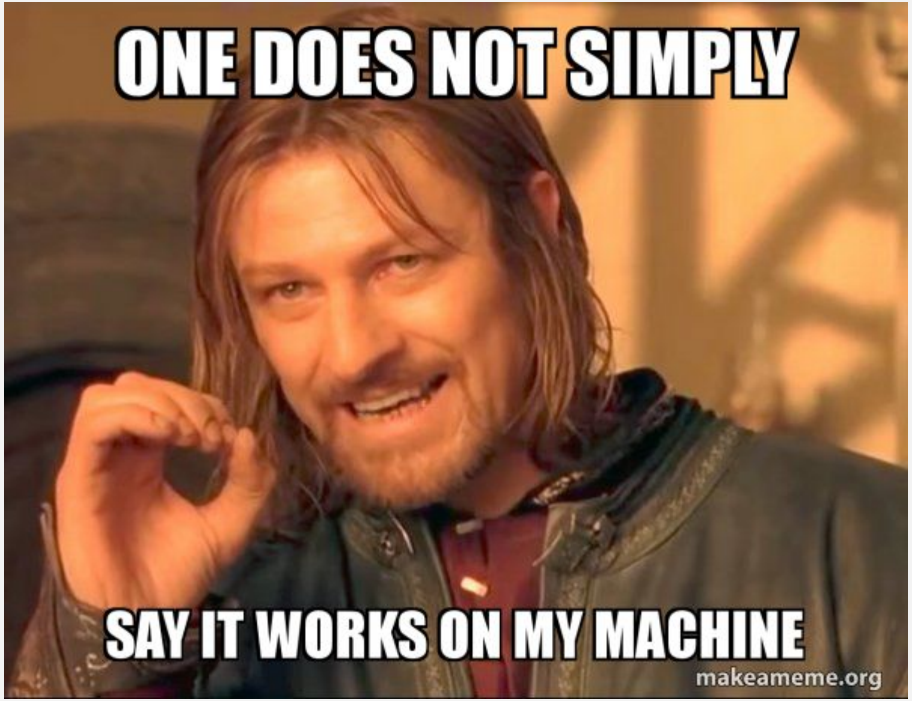
- No “it works on my machine” problem.
- Same environment from training to production.

Scaling

- Easier to scale services in container orchestration systems (Kubernetes, ECS, etc.).

Deployment

- Rapid deployment of AI-based microservices or APIs.



Key Tools & Concepts

Docker

- Open platform for developing, shipping, and running applications in containers.
- Core Commands: `docker build`, `docker run`, `docker push`, `docker pull`.

Dockerfile

- Text file containing instructions for building a Docker image.
- Specifies base image, dependencies, code, environment variables, etc.

Image vs. Container

- **Image:** The blueprint of your application (read-only).
- **Container:** A running instance of that image.

Container Registries

- Where images are stored and shared (Docker Hub, GitHub Container Registry, etc.).

Building a Docker

High-Level Steps:

1. **Choose a Base Image**
   - E.g., `python:3.9-slim` or `nvidia/cuda:11.3.1-cudnn8-runtime-ubuntu20.04` for GPU.
2. **Install Dependencies**
   - Python packages: `PyTorch`, `TorchVision`, `OpenCV`, etc.
   - System libraries if needed.
3. **Copy Your Code**
   - Model files, scripts, or your entire project folder.
4. **Set Up the Entry Point**
   - The command that runs when the container starts (e.g., `python app.py`).

```
FROM python:3.9-slim

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    libglib2.0-0 \
    libsm6 \
    libxext6 \
    libxrender-dev && \
    rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy requirements file and install
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy your application code
COPY . .

# Expose any ports if you run a web service
EXPOSE 8080

# Define entry point or default command
CMD ["python", "app.py"]
```

GPU Acceleration (Optional but Important)

NVIDIA Container Toolkit

- Required for GPU access inside containers.
- Allows the container to see the GPU resources.

Base Images

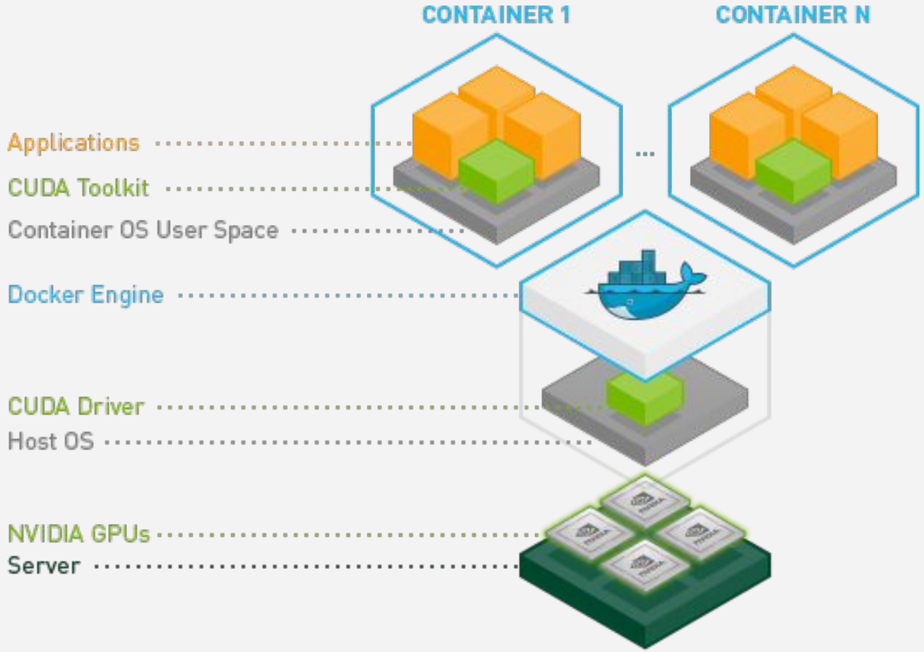
- `nvidia/cuda` images are commonly used.

Runtime Flag

- `docker run --gpus all my-image` to allocate all GPUs.

Performance Considerations

- Ensure your frameworks (PyTorch, TensorFlow) are compiled with GPU support.



Workflow & Best Practices

- **Start from Minimal**
  - Use python:3.9-slim or even alpine if compatible.
- **Order Your RUN Commands**
  - Capitalize on Docker's layer caching—place frequently changing steps at the bottom.
- **Avoid Installing Unnecessary Packages**
  - Smaller image ⇒ faster builds & deploys.
- **Security**
  - Update and clean up packages (apt-get update && apt-get install -y, then remove apt lists).
- **Version Control**
  - Keep your Dockerfile in Git; create a branch for major dependency changes.

Performance Considerations & Hardware Optimization

GPU Tuning

- Match CUDA/CUDNN versions to your AI framework.

Resource Limits

- Docker flags --memory and --cpus help manage resources in production.

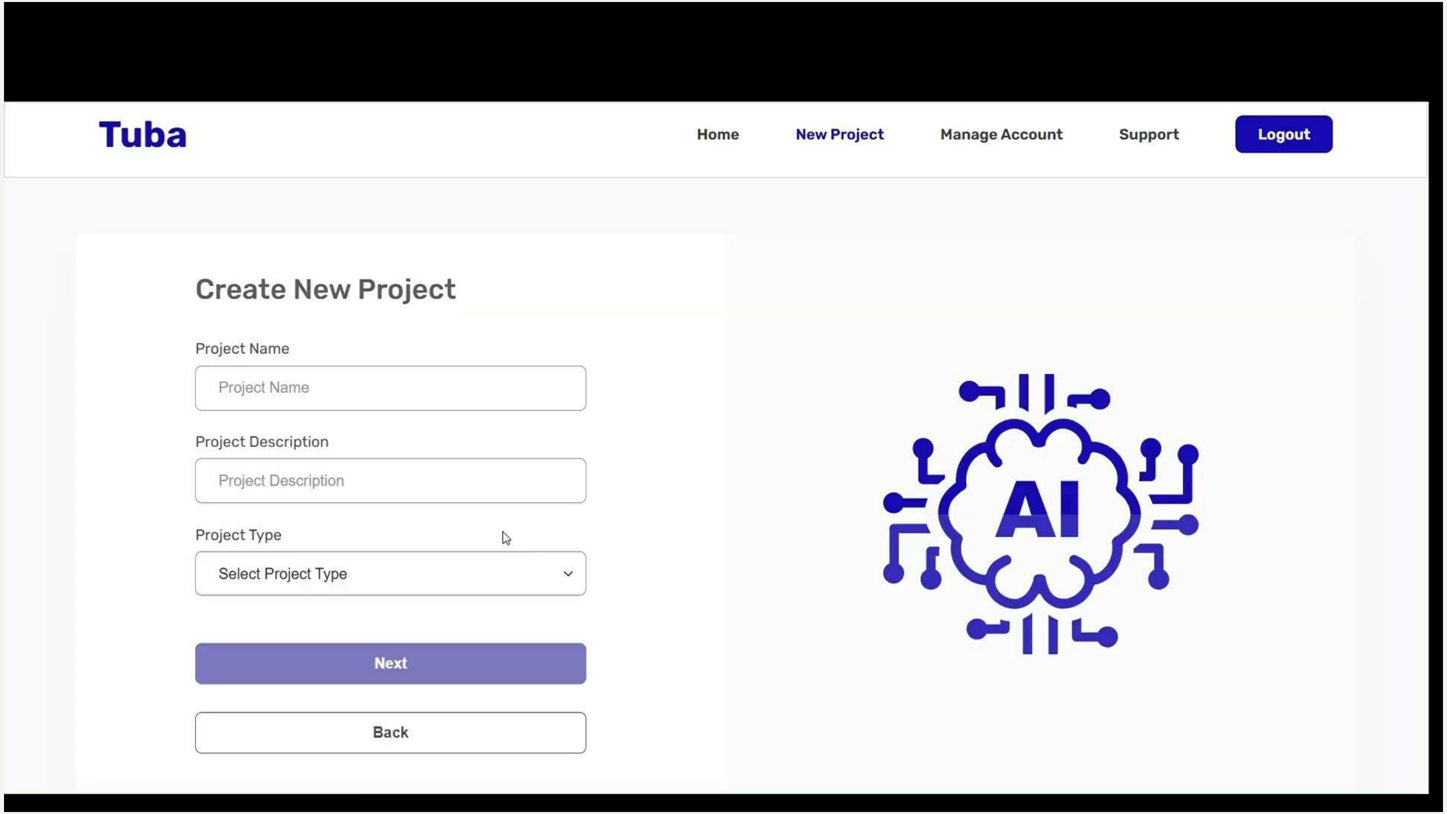
Caching Data & Model Artifacts

- Use volumes or multi-stage builds for large models to avoid expensive re-downloads.

Parallel Deployments

- Docker Compose or Kubernetes can scale up multiple containers for inference tasks or microservices.

Tuba Workflow Use Case



Quick Demo

Resources & Next Steps

Repository Access

- https://github.com/HossamTarek-bits/AI-in-a-Box

Documentation

- **Docker Docs:** https://docs.docker.com/
- **NVIDIA Container Toolkit Docs:** https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html

Suggested Learning Path

- Explore **Docker Compose** for multi-container workflows.
- Learn **Kubernetes** or other orchestration for scaling.

Q&A