Data Engineering and MLOps in Business

MLOps: Preparing for Production and Upscaling

Primoz Konda

AAUBS

March 24, 2025

pk@business.aau.dk

Outline

Where did we end yesterday?

- ?
- Questions?

Introduction - Starting point

- We have a working (still internal) ML pipeline; for example our Penguin Detector
- our models are working well
- the solution is ready for the next stage...

Types of ML Projects

- **Internal for Company:**
  - Used internally to optimize business processes.
  - Examples: Internal forecasting, process automation.
- **Open for Society (Free to Use):**
  - Publicly accessible for societal benefit.
  - Examples: COVID-19 spread prediction models.
- **Tailored for a Specific Customer:**
  - Customized to meet a specific client's needs.
  - Examples: Personalized recommendation engines.

Types of ML Projects (2)

- **Commercial for General Market:**
  - Designed for sale to multiple customers.
  - Examples: SaaS products, subscription-based ML services.
- **Open Source ML Models:**
  - Models developed and shared for community use.
  - Examples: OpenAI's GPT models.
- **Research and Experimental:**
  - Developed for scientific or technical exploration.
  - Examples: New NLP architectures.

Exercise

Sit in groups and discuss:

- What do you do with your repository?
- How do you adjust your code?
- How do you adjust your technical solutions?

Technical Scaling

Different parts of our pipeline needs to be upgraded for upscaling.

Upgrading the Pipeline for Scaling (Examples)

- **Database:**
  - **From SQLite to PostgreSQL / MySQL**: SQLite is single-threaded and not suited for high concurrency.
  - PostgreSQL/MySQL enable higher throughput and better transaction handling.
- **API:**
  - **From Uvicorn to Gunicorn + Nginx**: Uvicorn is single-threaded.
  - Gunicorn with worker processes + Nginx enables load balancing and better request handling.
- **Frontend:**
  - **From GitHub Pages to CDN + Reverse Proxy**: GitHub Pages are static.
  - CDN + Reverse Proxy enables dynamic scaling and caching.

Security Considerations

- Internal Threats:
  - Access control and role-based permissions.
  - Data encryption (in transit and at rest).
- API Security:
  - Authentication and authorization.
  - Rate limiting and monitoring.
- Infrastructure Security:
  - Container isolation.
  - Secrets management.

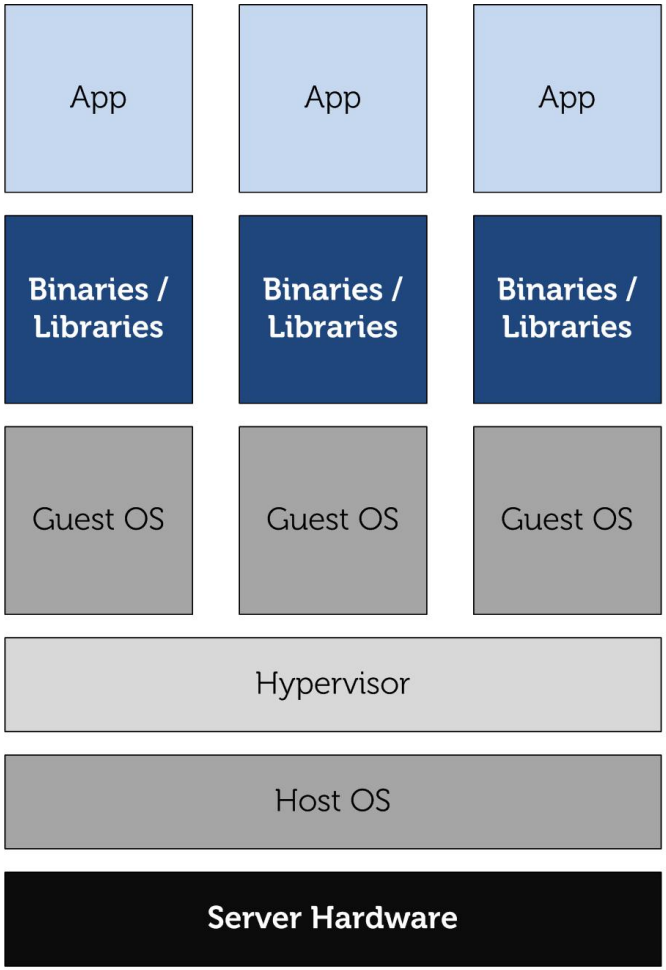
Containerization

- Why Use Containers?
  - Ensure consistency across environments.
  - Scale horizontally with Kubernetes.
  - Simplify deployment and rollback.
- Tools:
  - Docker.
  - Kubernetes (for orchestration).
  - Helm (for managing K8s).

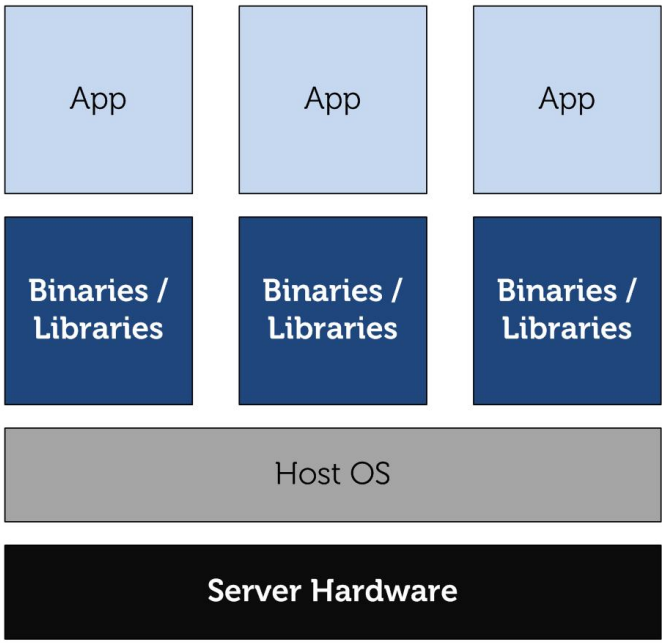
What is Docker?

- Docker is an open-source platform that automates the deployment of applications inside lightweight containers.
- Allows applications to run in the same manner on any system that supports Docker.
- Simplifies configuration, increases reproducibility, and eases isolation of dependencies.

Virtual Machines VS Containers



Virtualization



Containers

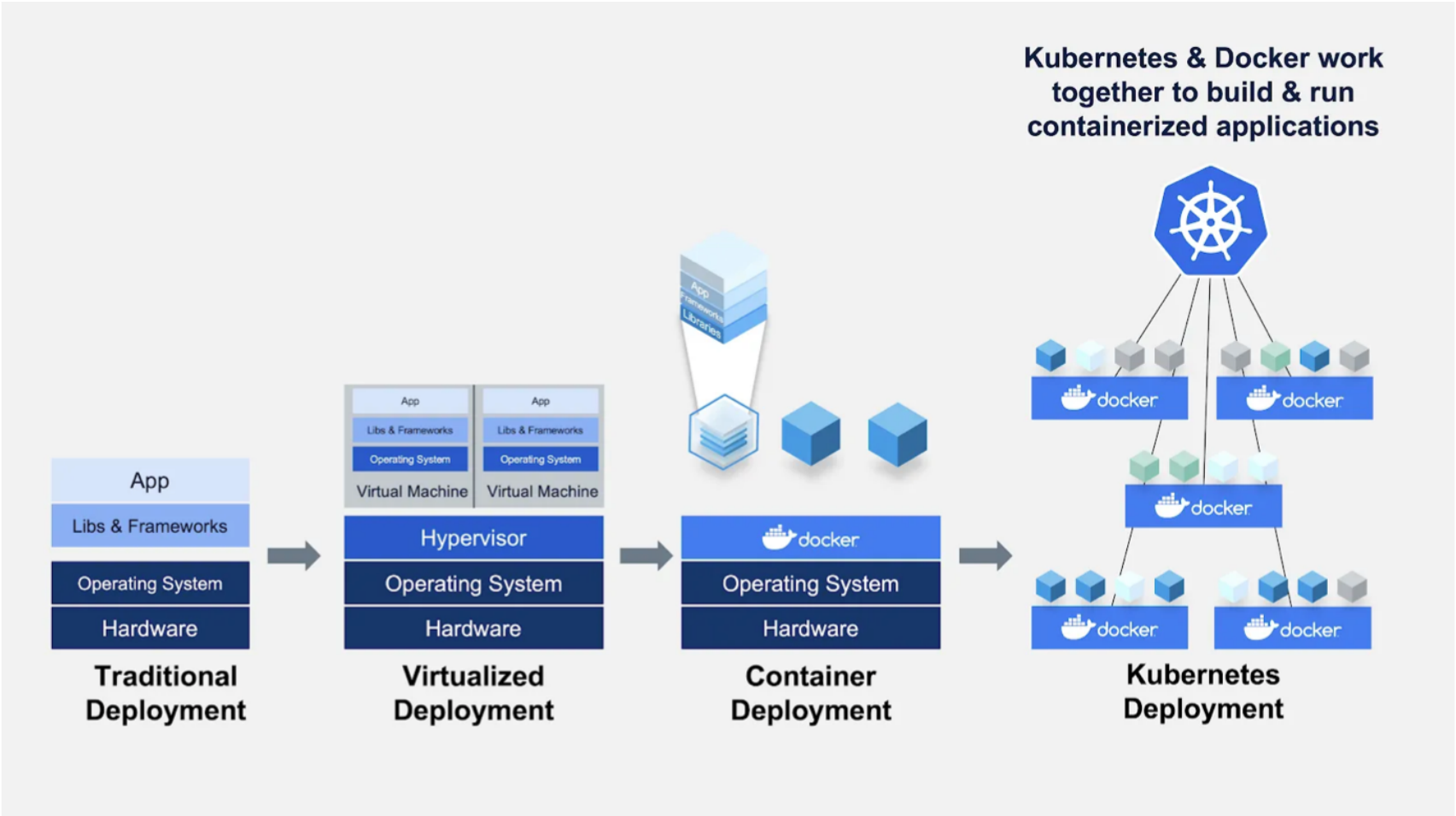
Why Docker?

- **Consistency:** Ensures consistent environments from development through to production.
- **Speed:** Containers can be spun up in seconds, making deployments and scaling faster.
- **Isolation:** Applications and their dependencies are isolated in containers, reducing conflicts.
- **Portability:** Containers can run anywhere - on a developer's laptop, on physical or virtual machines, in data centers, or in the cloud.
- **Microservices:** Facilitates the microservices architecture by allowing each service to be containerized and scaled independently.

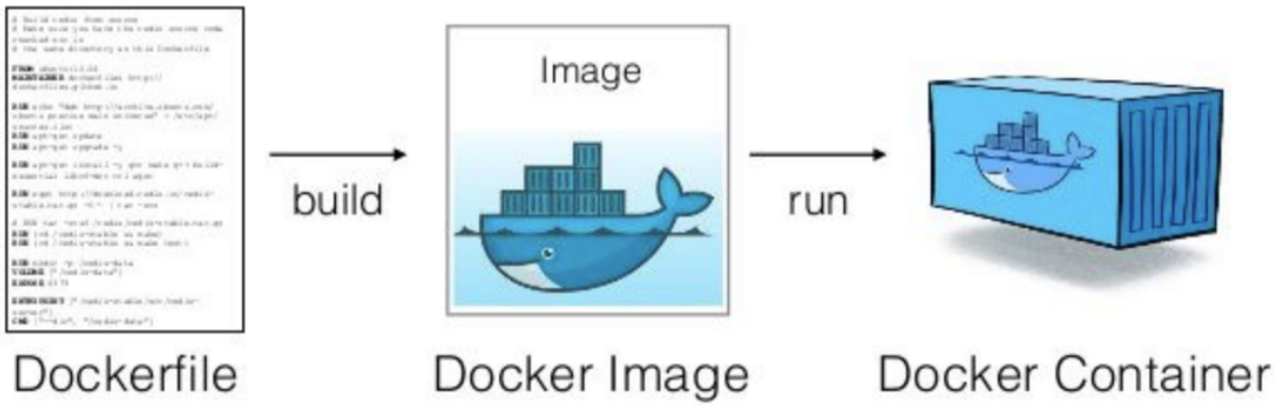
Docker in Practice

- **Development:** Developers use Docker to run and test applications in environments identical to production.
- **CI/CD:** Docker simplifies the continuous integration and delivery process by ensuring consistent environments at all stages.
- **Application Deployment:** Easily deploy applications on any platform that supports Docker, ensuring reliability and consistency.

Conteiner system



What is Docker?



Dockerfile

- A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image.
- Acts as a blueprint for building Docker images.
- Specifies the base image, software installations, environment variables, network ports, and other configurations needed for the container.
- **Build command:** `docker build -t my-image-name .`

How to structure Dockerfile? Example

```
# Use the official Python image from the Docker Hub
FROM python:3.8-slim

# Set the working directory inside the container to '/app'.
WORKDIR /app
COPY core/requirements.txt .
RUN pip install –no-cache-dir -r requirements.txt
COPY app/ .

# Instruct Docker to listen on port 8501 at runtime.
EXPOSE 8501

# Set the command to run the app using Streamlit.
CMD ["streamlit", "run", "app/streamlit_app.py", "–server.port=8501",
"–server.address=0.0.0.0"]
```

Managing Multiple Processes in Docker with Supervisord

Problem: Docker containers are designed to run a single process. However, sometimes you need to run multiple processes within the same container (e.g., a web server + background job).

Solution: Use `supervisord` to manage multiple processes within a Docker container.

Why Supervisord?

- Manages the lifecycle of multiple processes.
- Automatically restarts failed processes.
- Handles logging and process isolation.

Docker Image

- A **Docker Image** is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and config files.
- Images are built from the instructions in a Dockerfile.
- Images are immutable, meaning once built, they do not change.
- Can be stored in a Docker registry such as Docker Hub, allowing them to be shared and deployed anywhere Docker is supported.
- **Usage:** Images become containers when they run on Docker Engine.

Docker Container

- A **Docker Container** is a runtime instance of a Docker image.
- Containers encapsulate the application and its environment at the runtime.
- Provide isolation from other containers and the host system, yet allow for network and storage connectivity.
- Can be started, stopped, moved, and deleted independently.
- Containers ensure that the software runs uniformly and consistently across any platform.

Q & A

Questions?