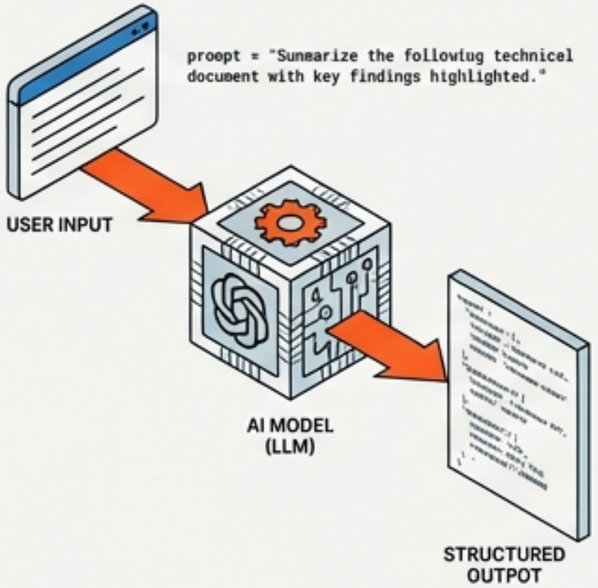
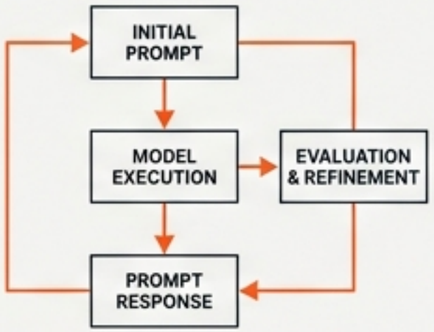
Logic

Prompt Engineering

Part I: The Interface of Intelligence

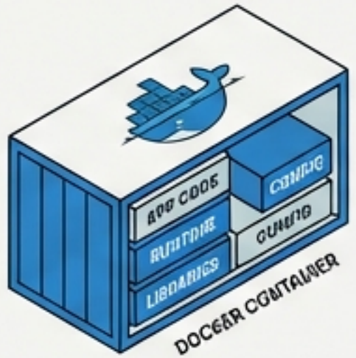


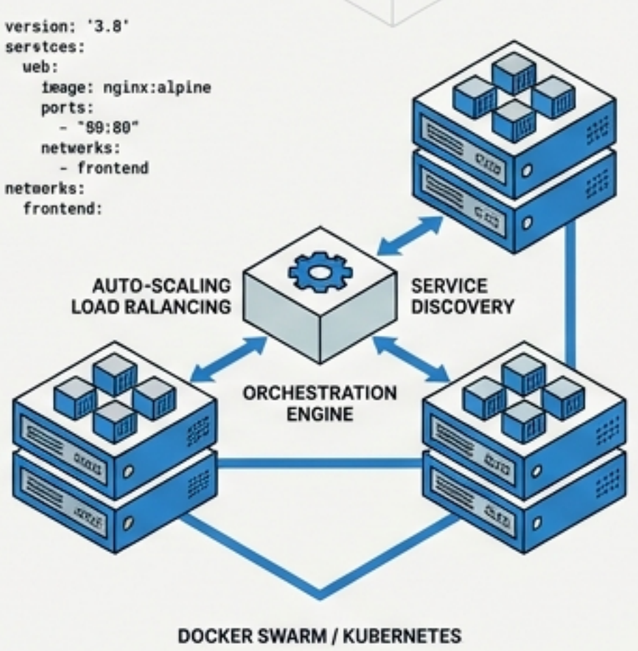


Structure

Advanced Docker Orchestration

Part II: The Infrastructure of Scale





Essential Skills for the Modern AI Developer
A Guide for Undergraduates
Trainer: Anshul – AIML Trainer, Skilloceans

The Roadmap

Zone 1

Part I: Mastering Inputs

1. Defining the Paradigm: **Intent vs. Output**
2. The LLM Mechanism: **Input Processing**
3. Anatomy of a Perfect Prompt (**RTCFC**)
4. The Template Library & Troubleshooting

Zone 2

Part II: Mastering Infrastructure

5. The Infrastructure Shift: **Scaling Up**
6. The Conductor: **Docker Compose**
7. The Nervous System: **Networking & Volumes**
8. Synthesis: **The Full Stack Skillset**

Goal: Bridging the gap between generating code and running applications.

The Art of Translating Intent

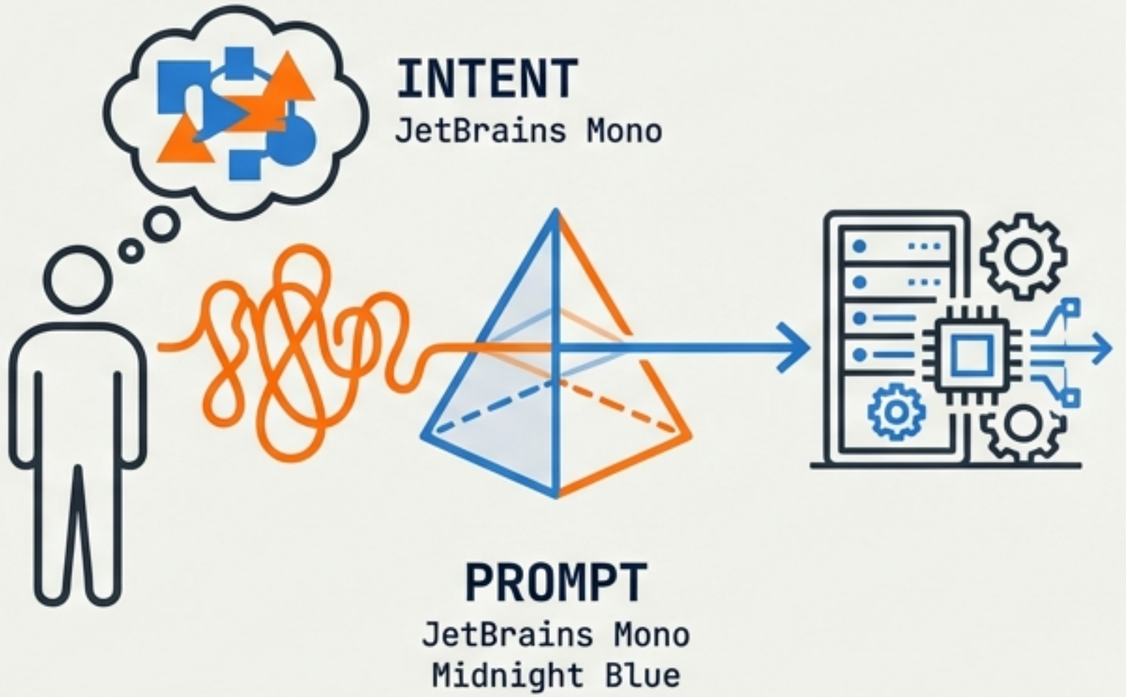
Defining Prompt Engineering

Prompt Engineering is the practice of designing inputs for Generative AI tools that result in optimal outputs. Think of it as **programming in English**.

Why it Matters:

- **Precision**: Moves results from random to deterministic.
- **Efficiency**: Reduces coding and debugging time by over 50%.
- **Relevance**: A critical soft skill becoming a technical requirement.

Student Impact: Essential for accelerating research, debugging complex code, and learning new frameworks rapidly.



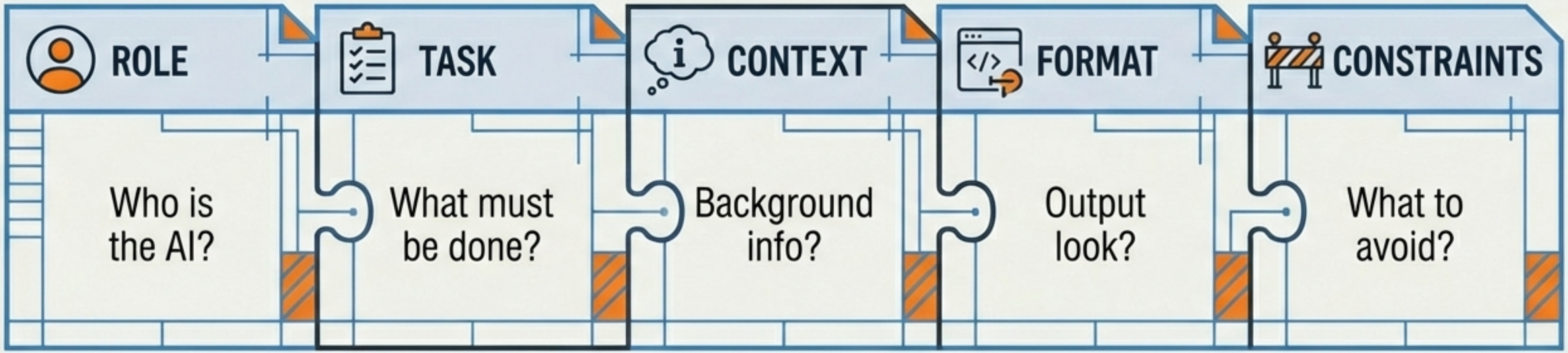
The Mechanism: How LLMs Process Input

Case Study: Specificity Matters

Bad Input	Write code for a login page.	Generic, likely insecure, wrong language.
Good Input	Write a Python Flask login route using JWT authentication with error handling.	Specific, usable, industry-standard.

Anatomy of a Perfect Prompt

The RTCFC Framework



Treat these components as variables in a function. Missing variables lead to undefined behavior.

The Builder’s Template Library

Type: Question-Based

Use for extracting facts or explanations.

```
Explain Docker Volumes as if I am a beginner.
```

Type: Instruction-Based

Use for executing specific actions on data/code.

```
Refactor this code to improve time complexity O(n).
```

Type: Role-Based

Use for simulating specific expertise.

```
Act as a tech interviewer. Ask me 3 hard SQL questions.
```

Type: Step-by-Step

Use for complex logic or debugging (Chain of Thought).

```
Think step-by-step to find the bug in this error log.
```

Why Prompts Fail

Common Pitfalls & Fixes

	Vague Instructions	Issue: Leaving the ‘how’ open to interpretation. Fix: Be explicit about the methodology and tools.
	Information Overload	Issue: Confusing the context window with irrelevant data. Fix: Curate the context. Only paste relevant code snippets.
	No Clear Format	Issue: Receiving paragraphs when you need code. Fix: Specify the structure: ‘`Return a JSON object`’ or ‘`List format`’.
	Unrealistic Expectations	Issue: Asking for private data or impossible logic. Fix: Provide all necessary data in the `Context` block.

Lab Session: Optimizing the Input

Scenario: Create a Python web scraper

The Weak Prompt

> “Write me a scraper.”

- ✕ No language specified
- ✕ No target defined
- ✕ No library preference

The Master Prompt

Role → Act as a **Python Developer.** Write a script using BeautifulSoup to scrape article titles from 'example.com'. Handle connection errors and save the output to a CSV file. Provide code only.

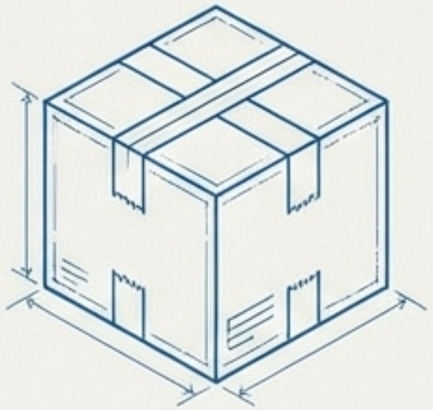
Task

Constraints

From Containers to Orchestration

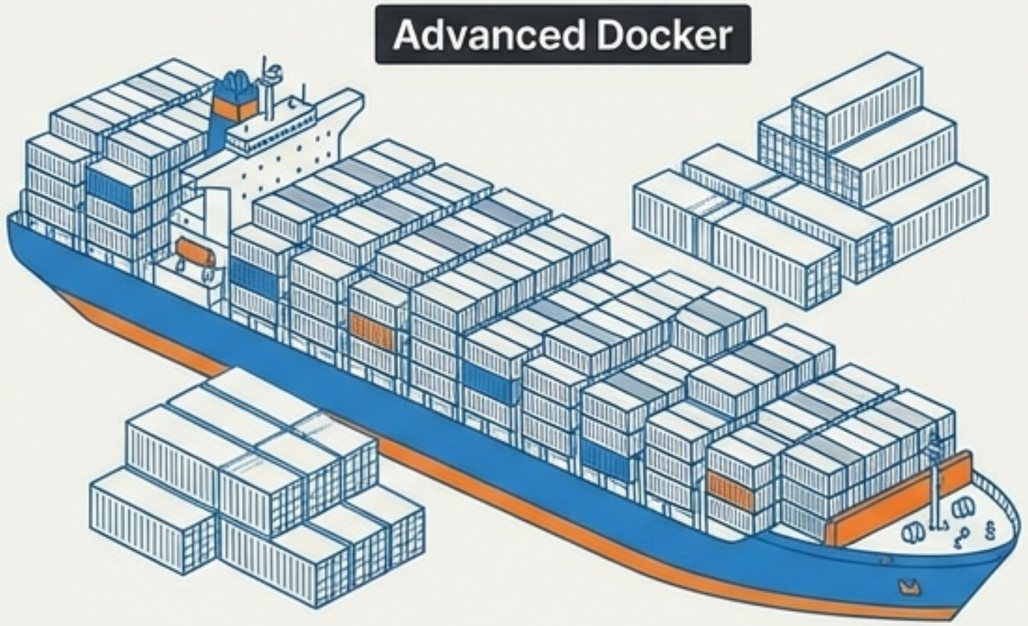
The Shift to Advanced Docker

Basic Docker



Packages one app + dependencies.
Good for simple scripts.

Advanced Docker



Manages a fleet of services (Frontend, Backend, DB).
Essential for Microservices.

Key Insight

- **The Problem:** Modern apps are not single files. They are **ecosystems**.
- **The Solution:** **Orchestration** manages the relationship, networking, and lifecycle of multiple containers simultaneously.

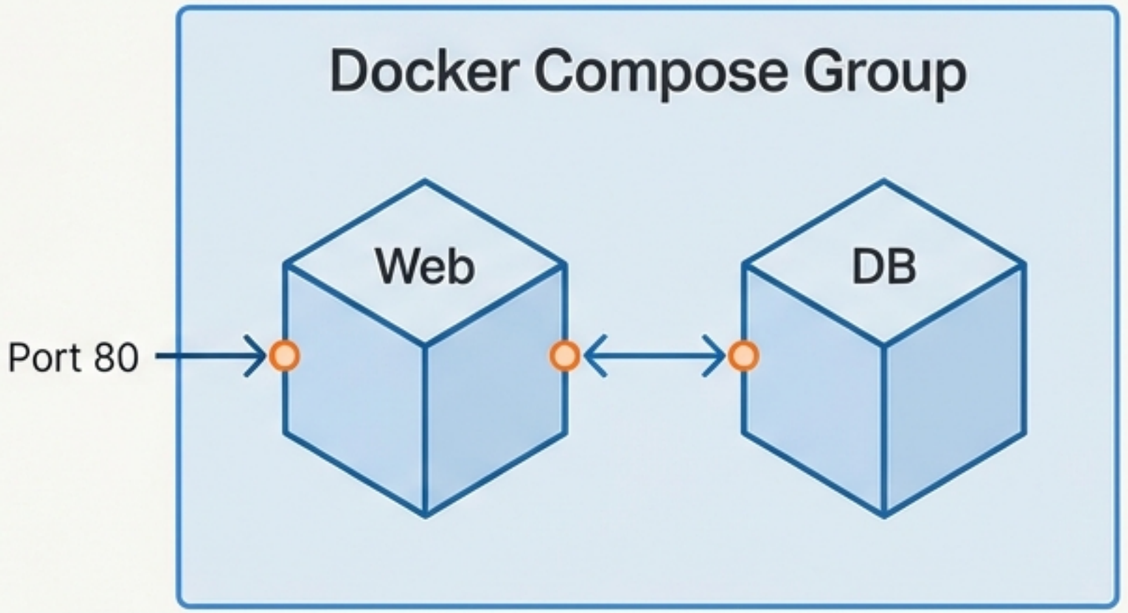
The Conductor: Docker Compose

```yaml
version: '3'
services:
  web:
    image: my-app:latest
    ports:
      - "80:80"
  db:
    image: postgres:13
```

Defines the app ecosystem

Individual containers

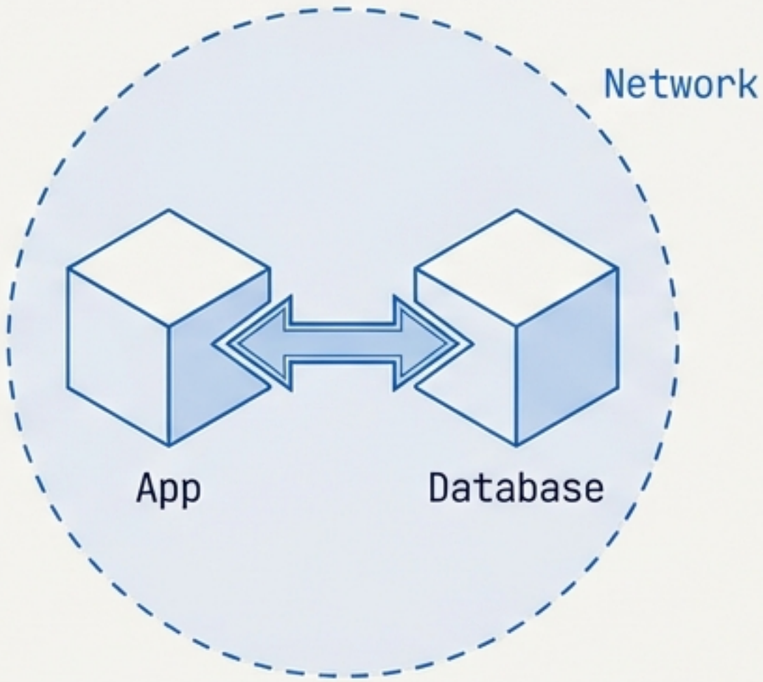
Access points



One Command: ‘`docker-compose up`’ starts the entire architecture instantly.

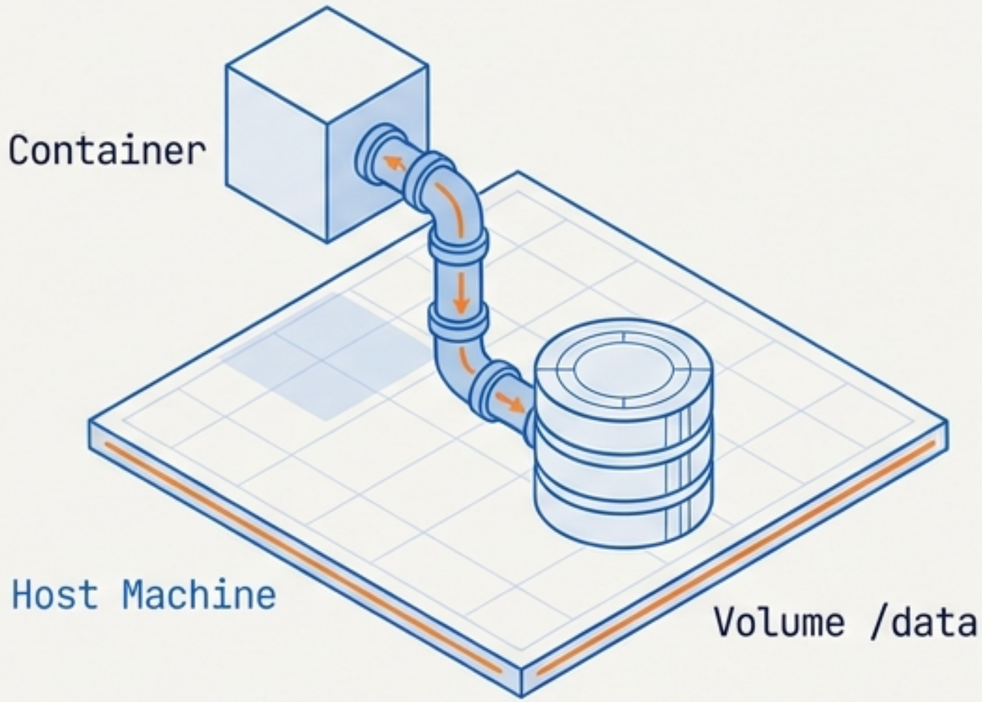
The Nervous System: Networking & Volumes

Networking (Communication)



Isolation & Discovery. Containers in the same network talk by name (e.g., `ping 'db'`). Traffic is secure from outside.

Volumes (Persistence)



Data Survival. Containers are ephemeral (temporary). Volumes store data on the host, so if the container dies, the data survives.

Bind Mounts (Dev link to folder) vs **Volumes** (Prod managed storage)

The Developer’s Toolkit Summary

Prompt Engineering	Docker Orchestration
☐ Did I assign a clear Role?	☐ Are services defined in Compose?
☐ Is the Context sufficient and curated?	☐ Is persistent data mapped to Volumes?
☐ Is the Output Format specified?	☐ Are containers isolated via Networks?
☐ Did I iterate based on the output?	☐ Does it run with a single command?

“Mastering the Prompt gives you the Code. Mastering Docker gives you the Product.”

Resources: docs.docker.com | openai.com/documentation