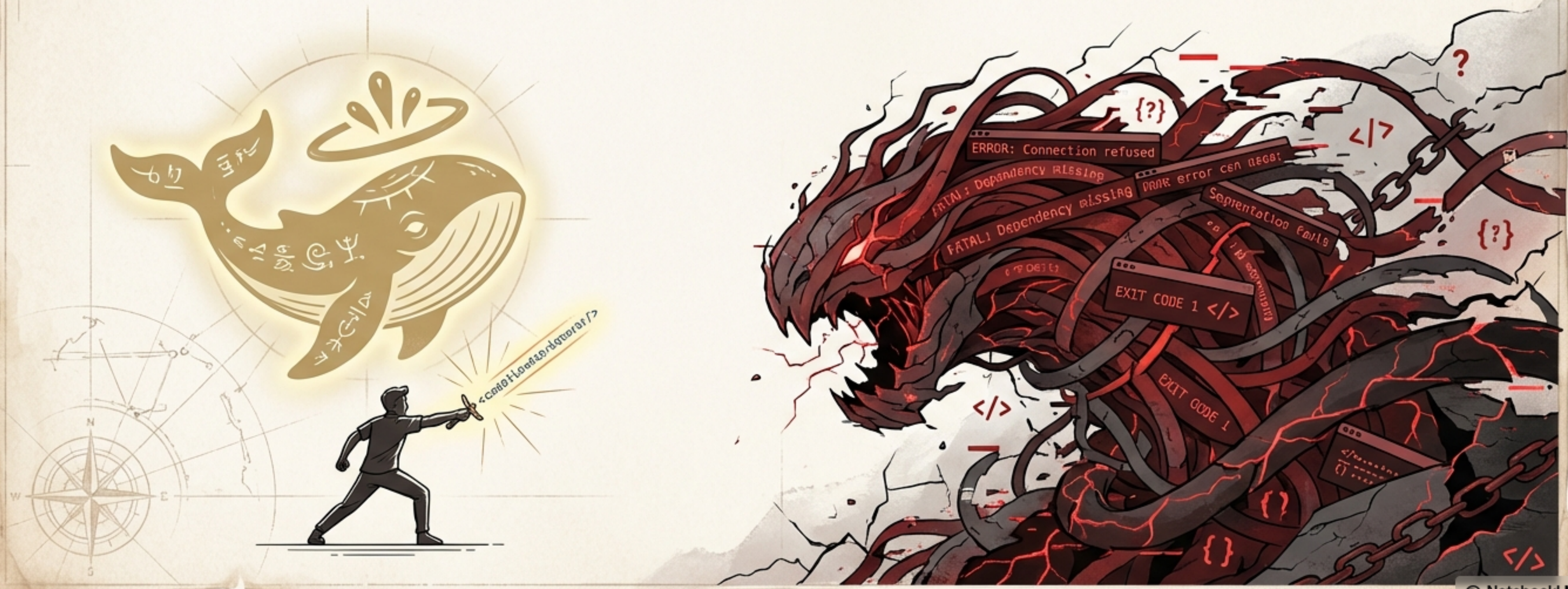
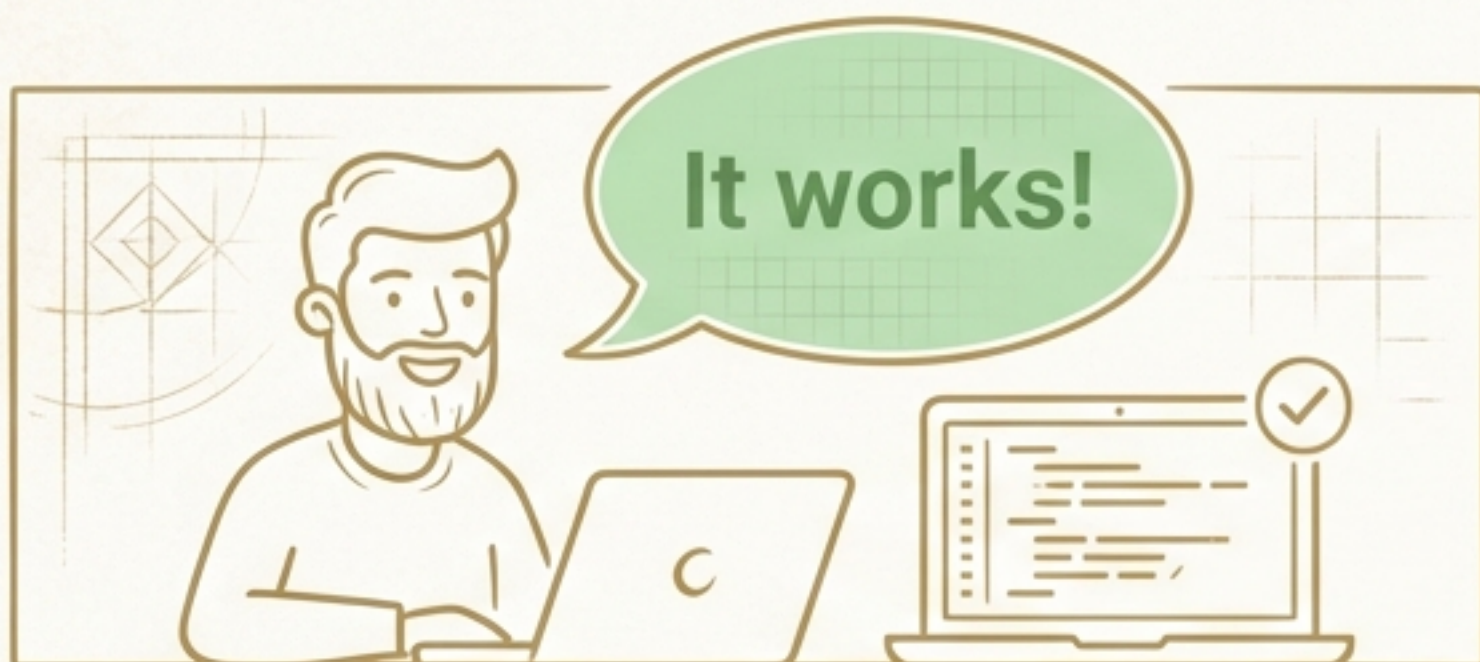


داکر: غول مرحله آخر رو اینجوری شکست بده!

راهنمای ۷ مرحله‌ای و داستانی برای رام کردن هیولای دیپلوی (Deployment)



«چرا پروژه‌ام روی سیستم خودم کار می‌کنه، ولی جای دیگه نه؟»



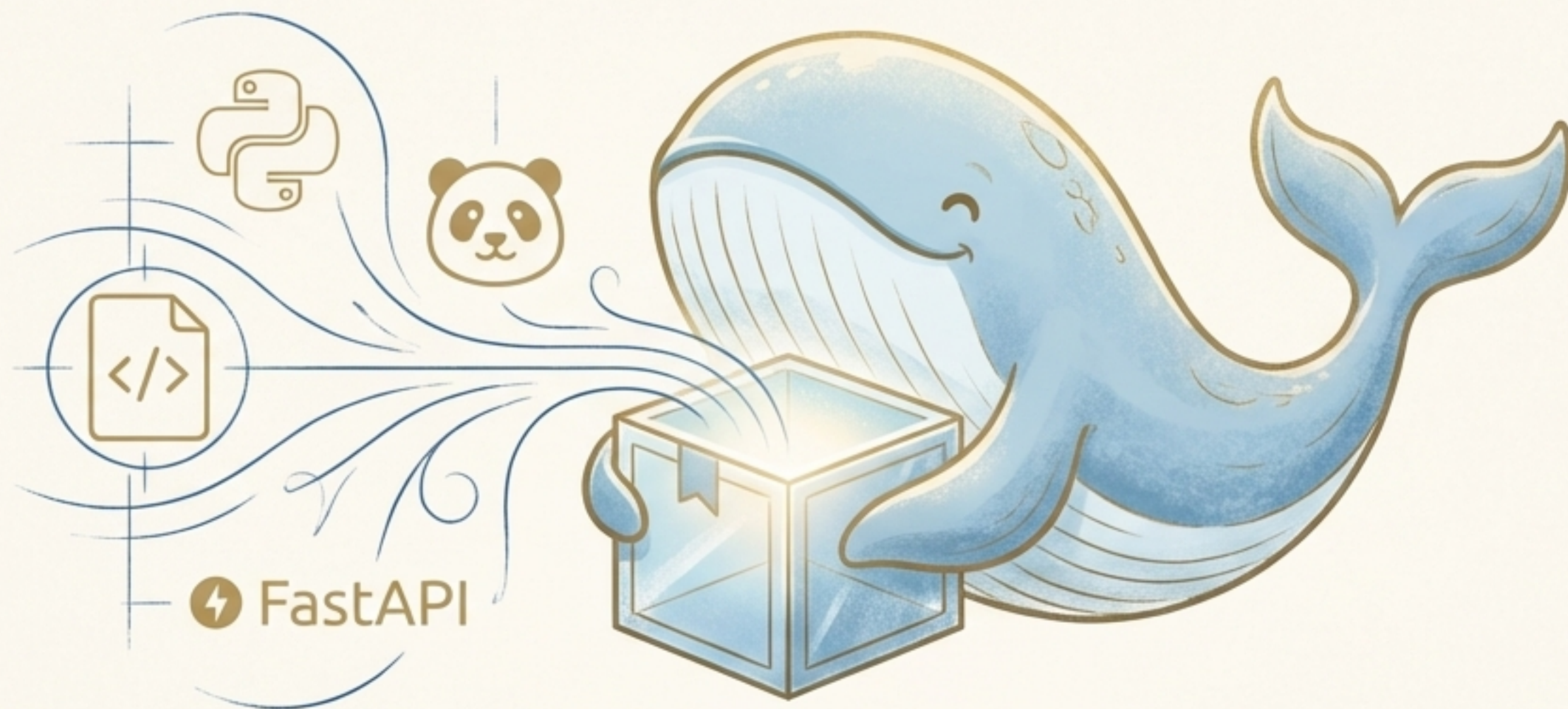
این سوال کابوس خیلی از ماهاست. ساعت‌ها کد می‌زنیم، همه چیز عالی کار می‌کنه، ننه، ولی وقتی می‌خوایم پروژه رو به کس دیگه‌ای نشون بدیم یا روی سرور بذاریم، همه‌چی به به هم می‌ریزه. انگار نفرین شده!



۹۰

جالبه بدونی ۹۰ درصد تازه‌کارها دقیقاً با همین دست و پنجه نرم می‌کنن. تو تنها نیستی قهرمان!

آشنا شو با متحد قدرتمندت: داکر



داکر مثل یک کپسول جادویی. تو کل پروژه و تمام خرت و پرت‌هایش (کتابخانه‌ها، تنظیمات، خود کد) رو داخل این کپسول بسته‌بندی می‌کنی. حالا این کپسول رو روی هر سیستمی که ببری، دقیقاً مثل روز اول کار می‌کنه. بدون هیچ بهونه‌ای! دیگه خبری از «ولی رو سیستم من کار می‌کرد» نیست.

نقشه گنج: ۷ خوان خوان برای ساختن اولین قلعه‌ی داکری





خوان اول: انتخاب زمین مناسب

```
# 1. Use the official lightweight Python base image  
FROM python:3.11-slim
```



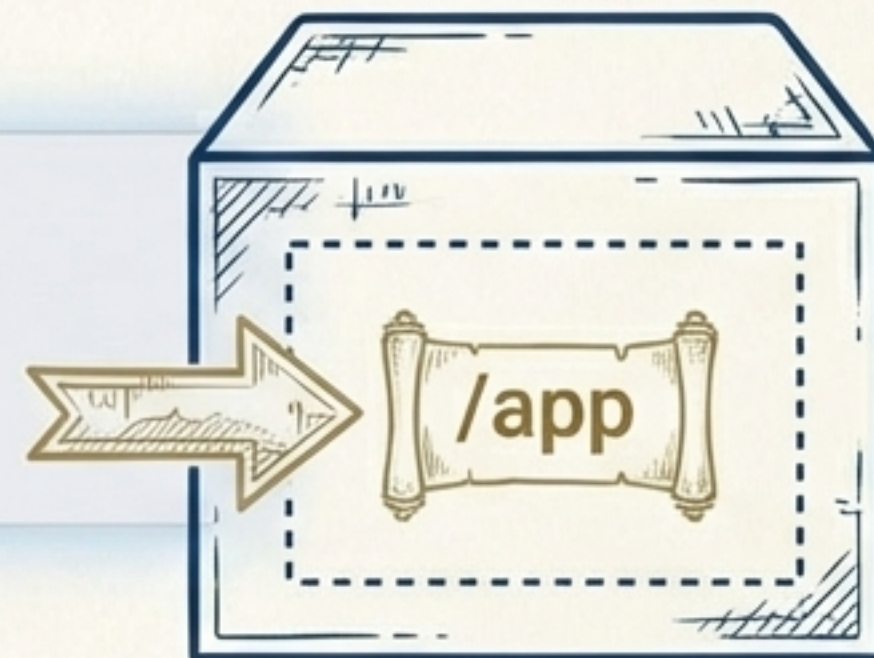
هر قلعه‌ای یه زمینی می‌خواد. ما هم برای پروژه پایتونی‌مون، به یه زمین آماده با پایتون نیاز داریم.
`FROM` یعنی «بر اساس این بساز».

ما نسخه `3.11-slim` رو انتخاب می‌کنیم چون هم جدید و هم خیلی سبک و جمع‌وجور (slim).
اینجوری قلعه‌ی ما از همون اول بهینه ساخته می‌شه.



خوان دوم: مشخص کردن کارگاه

```
# 2. Set working directory inside the container  
WORKDIR /app
```



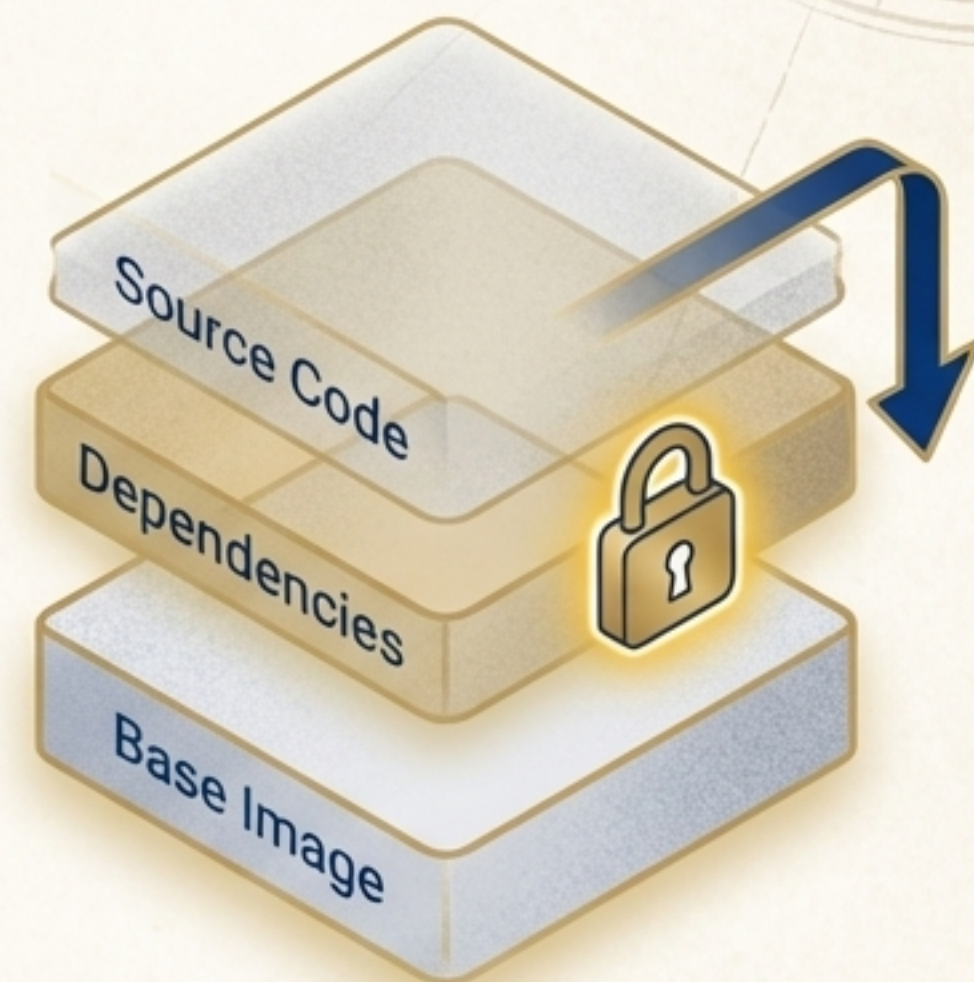
خب، زمین رو داریم. حالا باید بگیم کارگاه اصلی ما کجاست. با `WORKDIR` یه پوشه به اسم `/app` داخل کپسولمون می‌سازیم و به داکر می‌گیم از این به بعد، تمام کارها رو اینجا انجام بده. این به بعد، تمام کارها رو اینجا انجام بده. اینجوری همه چیز مرتب و منظم می‌مونه.



خوان سوم: یک فوت کوزه‌گری برای سرعت!

```
# 3. Copy only dependency file first (for Docker caching)  
COPY requirements.txt .
```

اینجا به حرکت هوشمندانه می‌زنیم. قبل از اینکه کل کد رو بیاریم، فقط لیست ابزارهایی که لازم داریم (requirements.txt) رو کپی می‌کنیم. چرا؟ چون داکر خیلی باهوشه و لایه‌ها رو «گش» می‌کنه. اگه بعداً فقط کد اصلی رو تغییر بدی، داکر دیگه نیازی نداره دوباره همه‌ی کتابخونه‌ها رو نصب کنه و ساخت قلعه‌ات چند برابر سریع‌تر می‌شه!





خوان چهارم: مسلح کردن کارگاه

4. Install Python dependencies

```
RUN pip install --upgrade pip \  
&& pip install -r requirements.txt \  
&& apt-get clean && rm -rf /var/lib/apt/lists/*
```



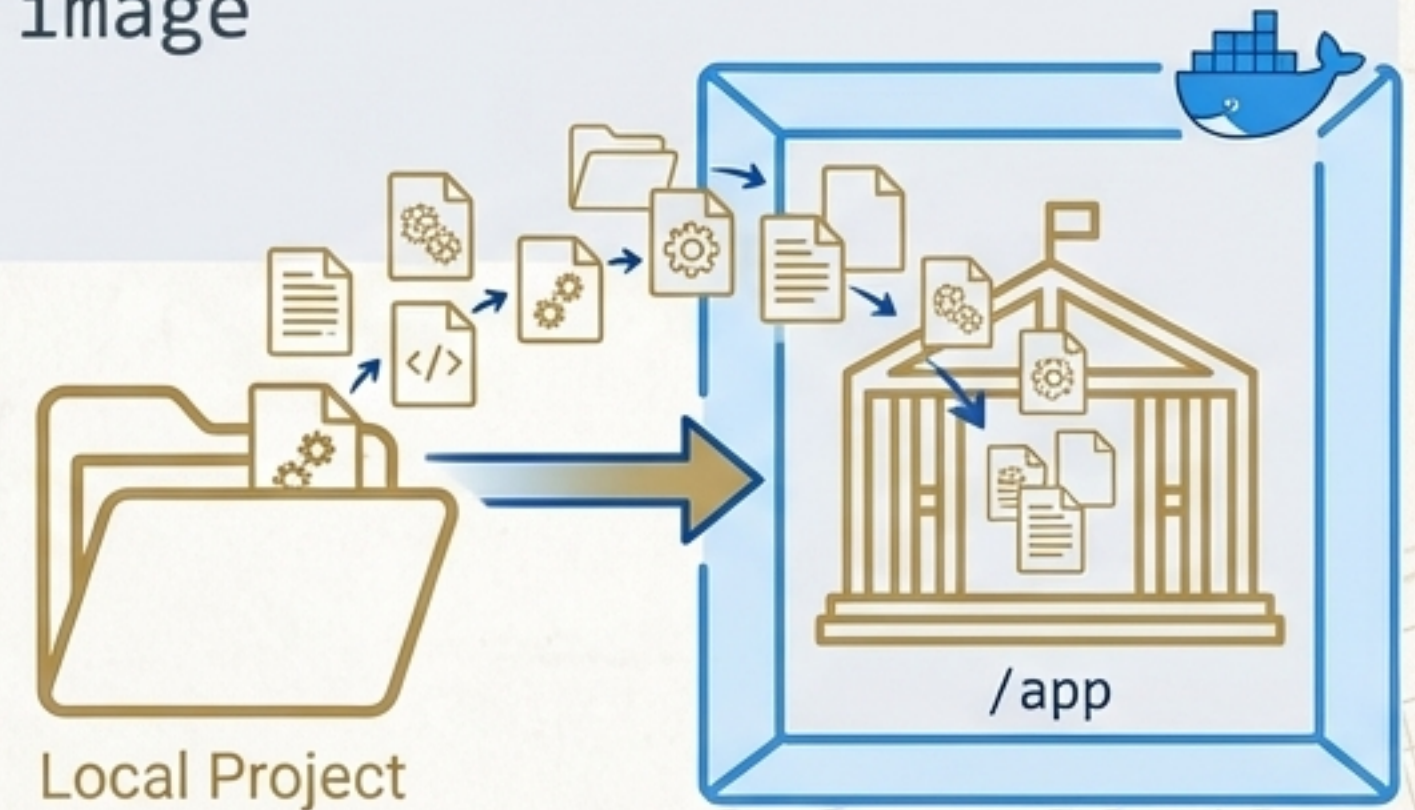
حالا که لیست ابزارها رو داریم، وقتشه نصبشون کنیم. با دستور RUN به داکر می‌گیم این دستورات رو در ترمینال اجرا کن. اول `pip` رو آپدیت می‌کنیم، بعد همه چیز رو از روی requirements.txt نصب می‌کنیم. اون دو خط آخر هم برای تمیزکاریه تا کیپسول نهایی مون سبک بمونه.



خوان پنجم: برپا کردن اسکلت اصلی

5. Copy the entire project into the image
COPY . .

ابزارها آماده‌ان. حالا وقتشه که بنای اصلی، یعنی تمام فایل‌ها
فایل‌هه و کدهای پروژه‌مون رو بیاریم داخل کارگاه
نحی پروژه‌میای کدهام (/app): دستور `COPY . .` یعنی
«هرچیزی که اینجا هست رو کپی کن اونجا». به همین سادگی!



خوان ششم: باز کردن دروازه‌ی اصلی

```
```dockerfile
6. Expose FastAPI port
EXPOSE 8000
```
```

قلعه‌ی ما ساخته شده، ولی الان کاملاً مهر و موم شده. اگه پروژه‌مون یه سایت یا API هست (مثل FastAPI یا Flask)، باید یه دروازه برای ارتباط با دنیای بیرون باز کنیم. با `EXPOSE` به داکر می‌گیم که این کیپسول قراره روی پورت `8000` جواب بده.



خوان هفتم: دمیدن روح در کالبد قلعه

```
# 7. Run the FastAPI app using uvicorn  
CMD ["python", "-m", "uvicorn", "src.app.main:app",  
      "--host", "0.0.0.0", "--port", "8000"]
```

این آخرین و مهم‌ترین دستوره. `CMD` به داکر می‌گه «وقتی کپسول رو روشن کردی، این دستور رو اجرا کن».

اینجا ما داریم سرور `uvicorn` رو برای اپلیکیشن FastAPI خودمون اجرا می‌کنیم. این همون دستوریه که قبلاً توی ترمینال خودت می‌زدی، ولی حالا خود داکر این کار رو انجام می‌ده.



طومار جادویی تو آماده‌ست!

تبریک می‌گم قهرمان! تو هر ۷ خوان رو پشت سر گذاشتی. این چیزیه که با هم ساختیم.
یک دستورالعمل کامل و بی‌نقص برای زنده کردن پروژه‌ات در هر کجای دنیا.



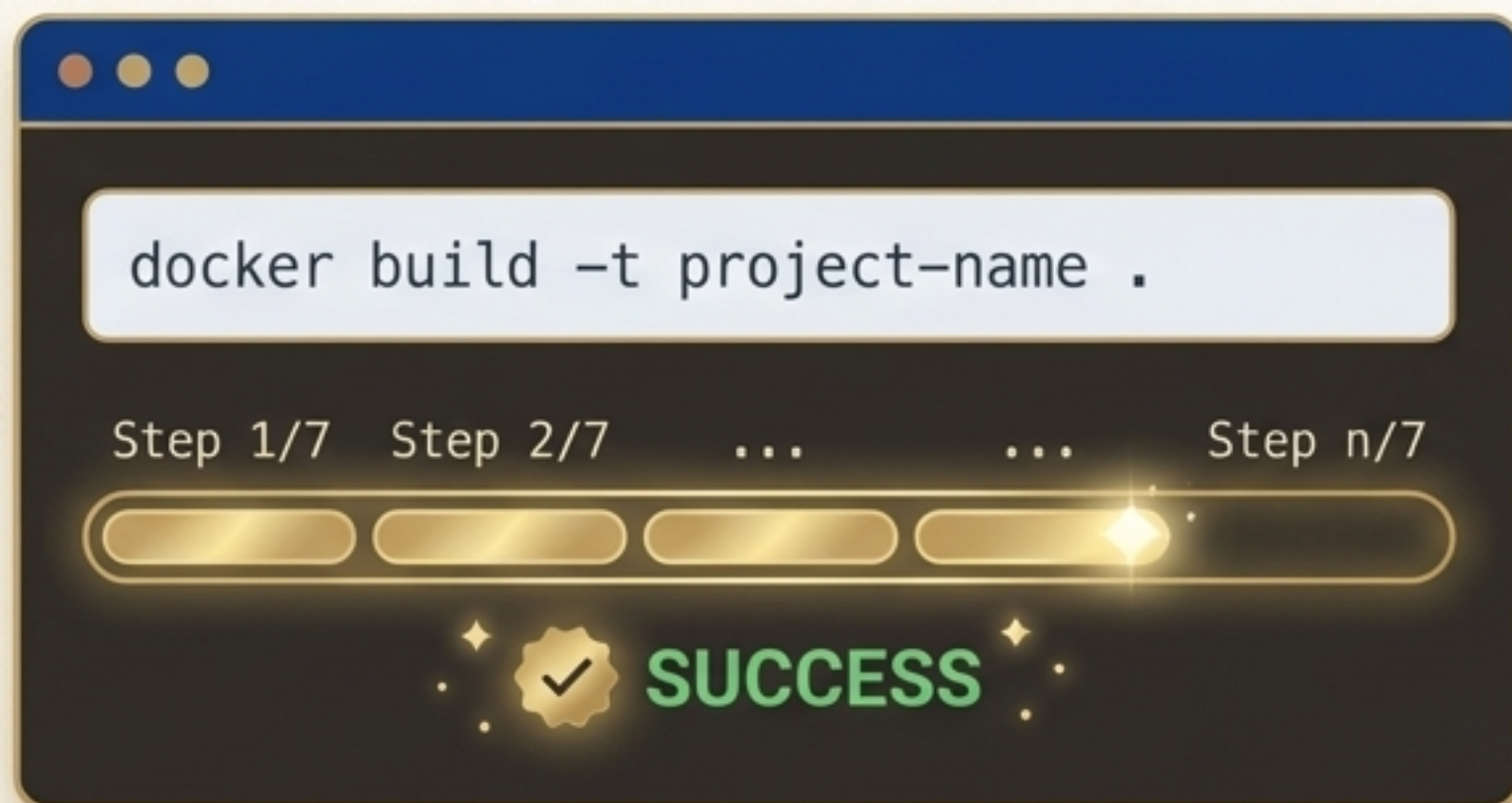
```
# 1. Use the official lightweight Python base image
FROM python:3.11-slim
# 2. Set working directory inside the container
WORKDIR /app
# 3. Copy only dependency file first (for Docker caching)
COPY requirements.txt .
# 4. Install Python dependencies
RUN pip install --upgrade pip \
    && pip install -r requirements.txt \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
# 5. Copy the entire project into the image
COPY . .
# 6. Expose FastAPI port
EXPOSE 8000
# 7. Run the FastAPI app using uvicorn
CMD ["python", "-m", "uvicorn", "src.app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```



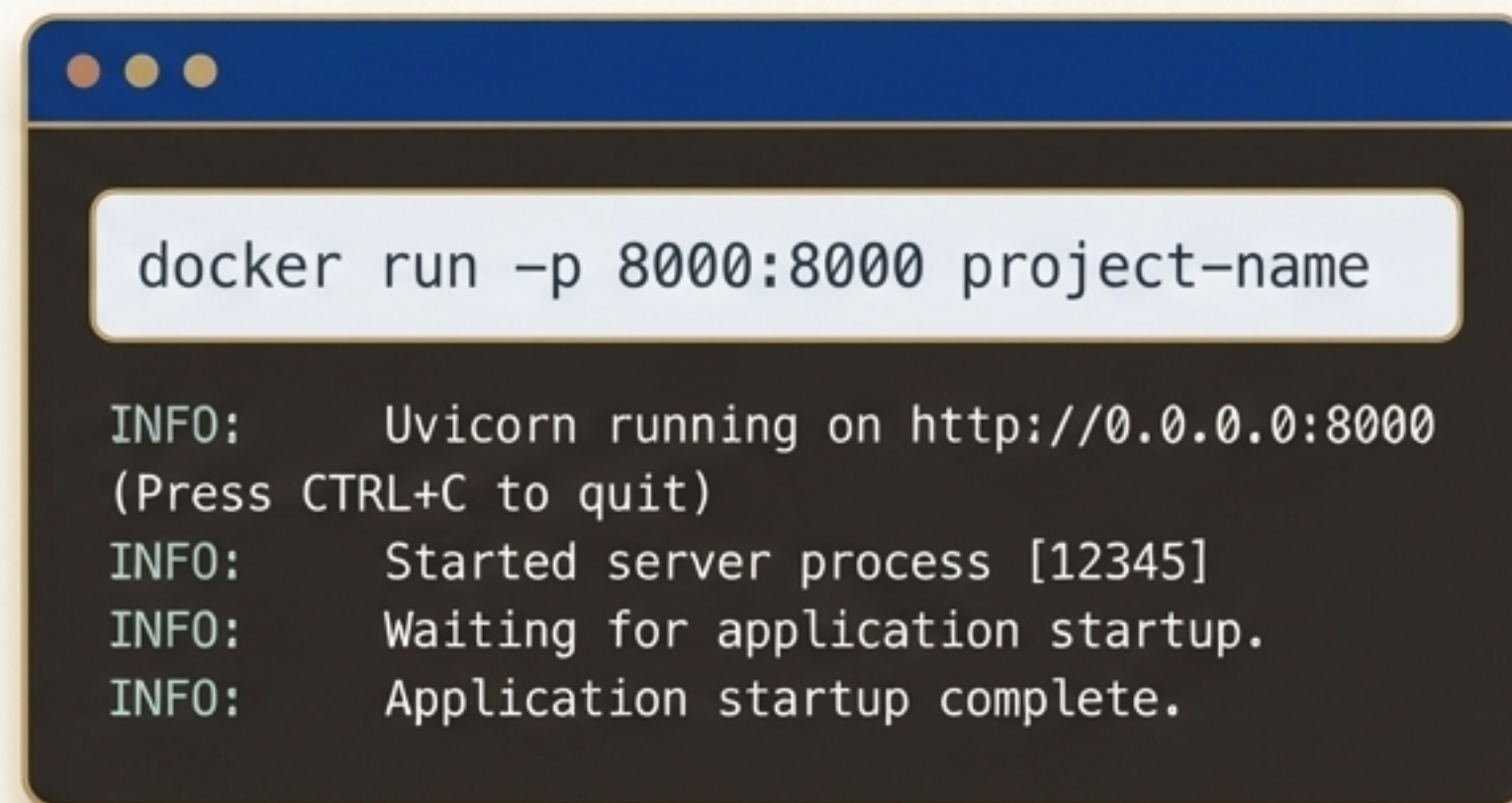
وقتشه آفرینشت رو زنده کنی!

با این دو ورد جادویی در ترمینال، می‌تونی از روی طومارت، قلعه رو بسازی و روشنش کنی.

ساختن ایمیج (قلعه)



اجرای کانتینر (روشن کردن قلعه)



این دستور فایل 'Dockerfile' رو می‌خونه و «ایمیج» رو می‌سازه.
`-t` هم برای اینکه به اسم باحال ('project-name') براش انتخاب کنی.

حالا از روی اون ایمیج، یه نمونه‌ی زنده به اسم «کانتینر» اجرا می‌شه.
د `p` هم دروازه‌ی سیستم تو رو به دروازه‌ی قلعه وصل می‌کنه.

گنجینه اسرار: وردهای پیشرفته برای ماجراجویی بعدی

تو اصول اولیه رو یاد گرفتی. حالا با این ابزارهای پیشرفته می‌تونی قلعه‌های بزرگتر و پیچیده‌تری بسازی.



Docker Compose

برای مدیریت چندتا قلعه با هم (مثلاً API و دیتابیس).
دیگه لازم نیست چندتا دستور `run` بزنی.



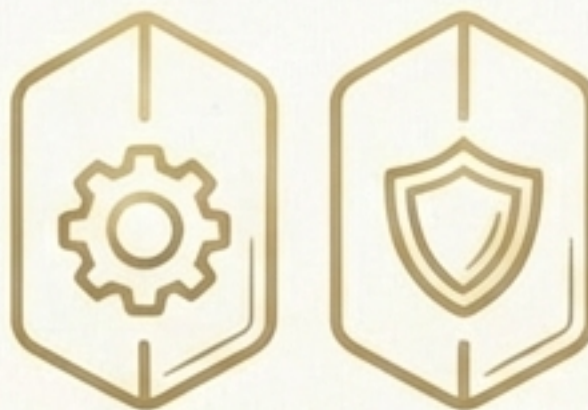
Docker Build Cloud

برای پروژه‌های خیلی بزرگ، ساخت
ایمیج رو تا ۳۹ برابر سریع‌تر
می‌کنه!



docker init

این دستور خودش به صورت
خودکار `Dockerfile` و فایل‌های
لازم دیگه رو برات می‌سازه.
مثل یه دستیار هوشمند.



Dev vs Prod Containers

برای محیط توسعه و محصول نهایی،
کپسول‌های جدا و بهینه بساز.

هیولای دیپلوی رام شد.

حالا تو معمار دنیای خودتی. برو و هرچیزی که تو ذهنته رو بساز.

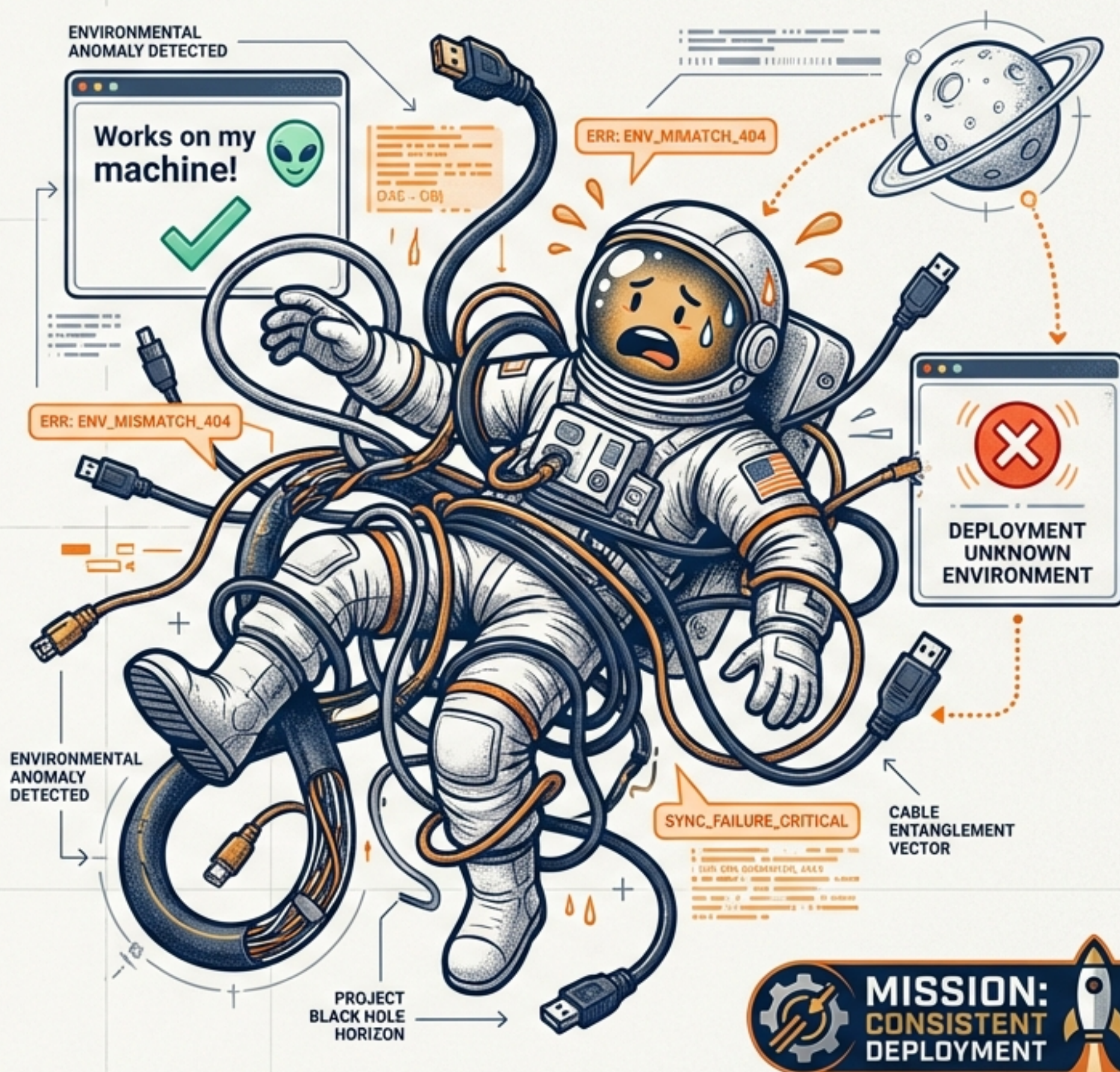


چرا ۹۰٪ فزانونردان تازه‌کار در اولین پرواز گم می‌شوند؟

اینجا با کابوس همیشگی برنامه‌نویس‌ها روبرو می‌شیم: جمله معروف «اپلیکیشن من فقط روی کامپیوتر خودم کار می‌کند!».

این مشکل، مثل یک سیاه‌چاله، پروژه‌ها رو به خاطر ناهماهنگی محیط‌ها و dependency به درون خودش می‌کشد. (این همون دلیلیه که همون دلیلیه که «۹۰٪ مبتدی‌ها گیج می‌شن»).

ماموریت ما اینه که یک سفینه بسازیم که سفر اپلیکیشن شما رو هر بار، به هر مقصدی، امن و یکسان تضمین کنه.



نقشه ساخت سفینه: ۷ مرحله تا پرتاب

این نقشه راه ما برای ساخت سفینه است. هر مرحله، بخش مهمی از فرآیند ساخت و سازه:

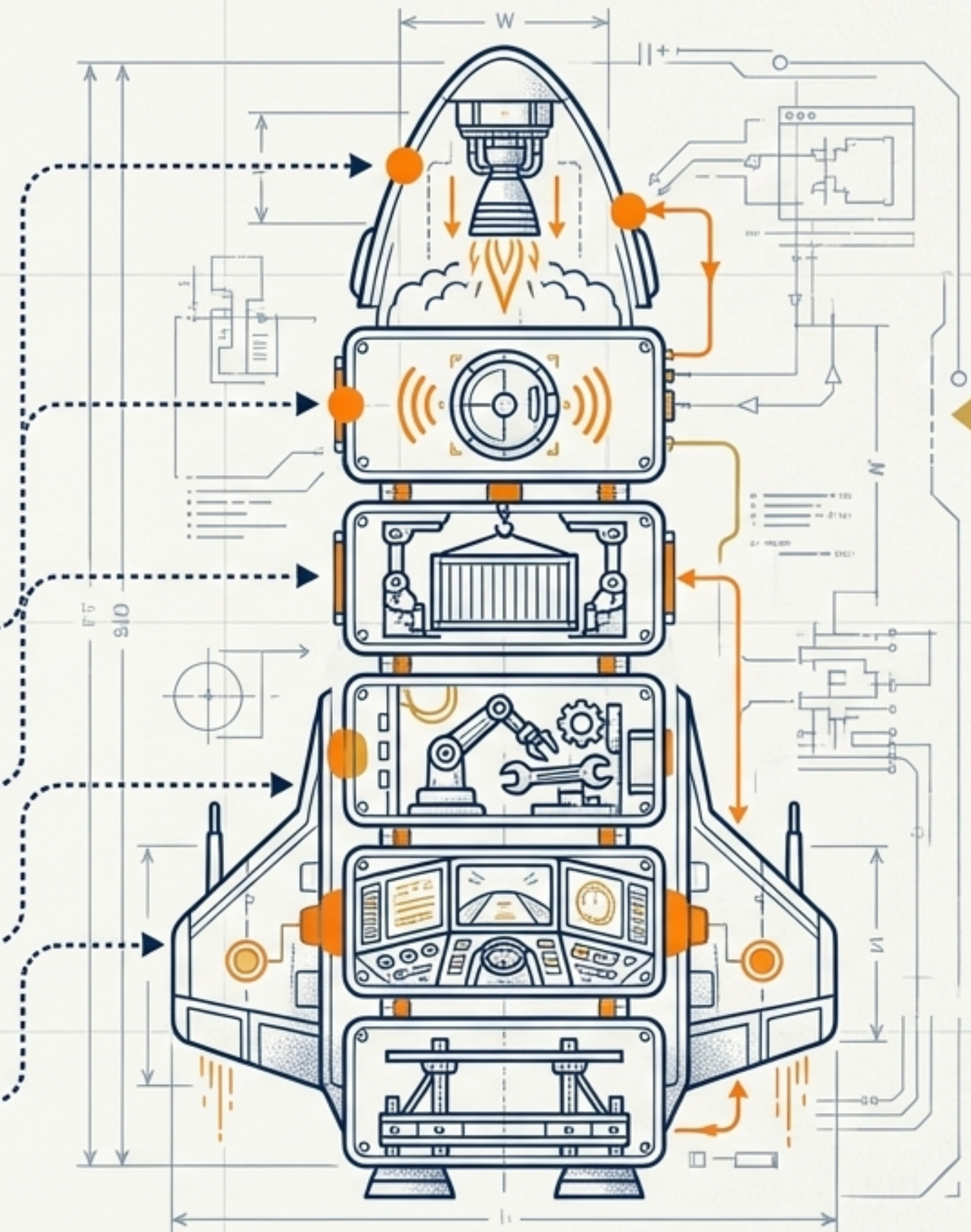
۷. باز کردن دريچه ارتباطی (Define CMD)

۴. نصب و آماده‌سازی ابزارها (Install Project Files)

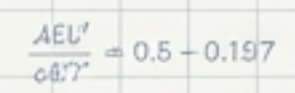
۳. بارگیری لیست ابزارها* (Copy Dependency File)

۲. تعیین اتاق فرمان* (Working Directory)

۱. انتخاب شاسی اصلی** (Base Image)




```
FROM python:3.11-slim
```

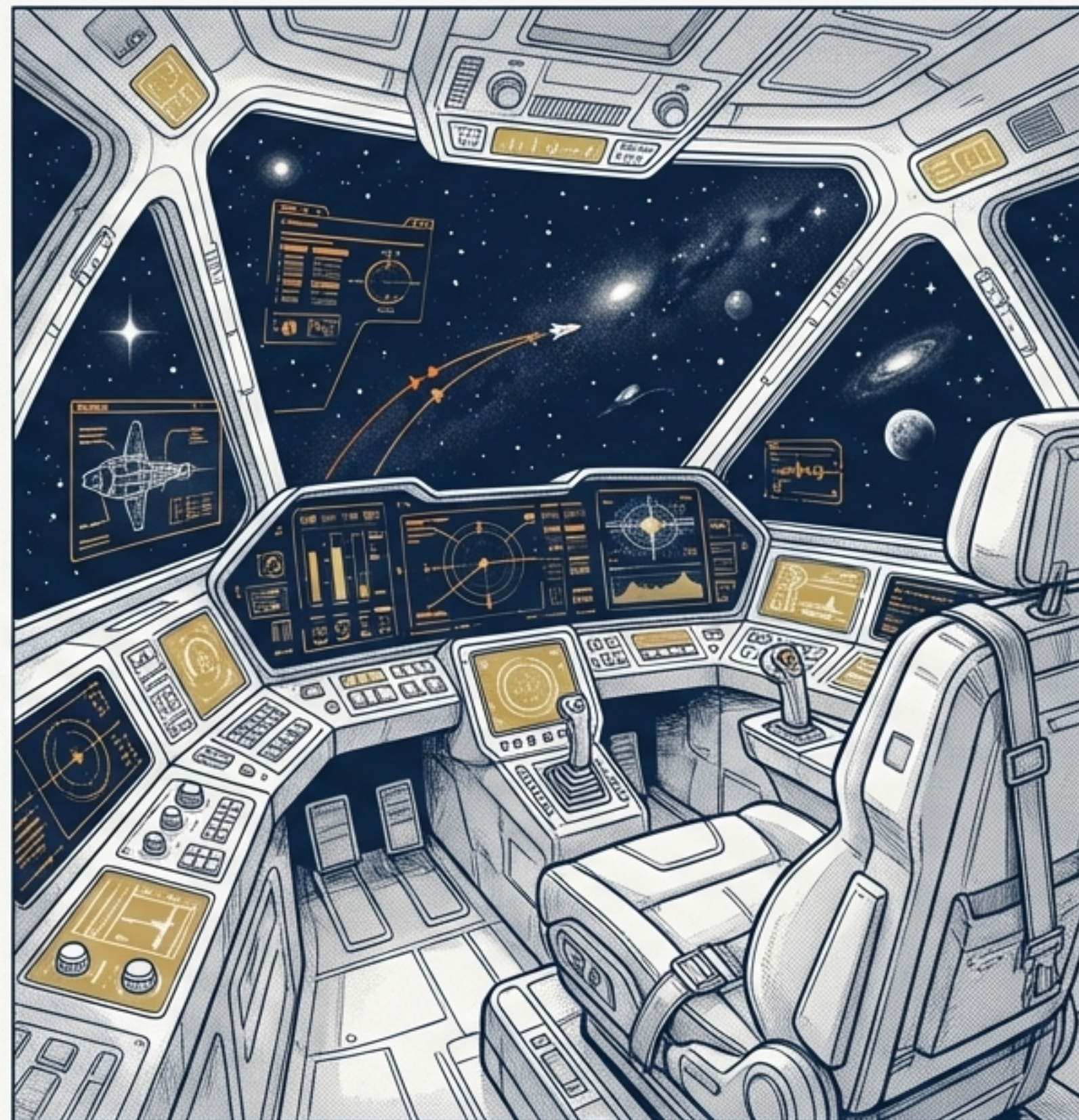


MISSION:
**CONSISTENT
DEPLOYMENT**

مرحله ۲: تعیین اتاق فرمان (WORKDIR)

اینجا مرکز فرماندهی ماست. تمام عملیات‌ها و کدهای ما از این نقطه اجرا میشن. با این دستور به داکر می‌گیم که اتاق فرمان ما در سفینه، پوشه 'app' هست. از این به بعد، هر دستوری داخل این پوشه اجرا می‌شه.

```
WORKDIR /app
```



مرحله ۳: بارگیری لیست ابزارها (requirements.txt)

قبل از اینکه خود ابزارها رو به سفینه بیاریم، فقط لیستشون وارد می‌کنیم. چرا؟ این یه ترفند هوشمندانه برای داکره! اگر لیست ابزارها (فایل `requirements.txt`) تغییر نغیر نکنه، داکر از حافظه پنهان (کش) خودش استفاده می‌کنه و ، مرحله بعدی (نصب ابزارها) رو دوباره از اول اجرا نمی‌کنه. نتیجه؟ ساخت سفینه شما خیلی سریع‌تر می‌شه!

```
COPY requirements.txt .
```

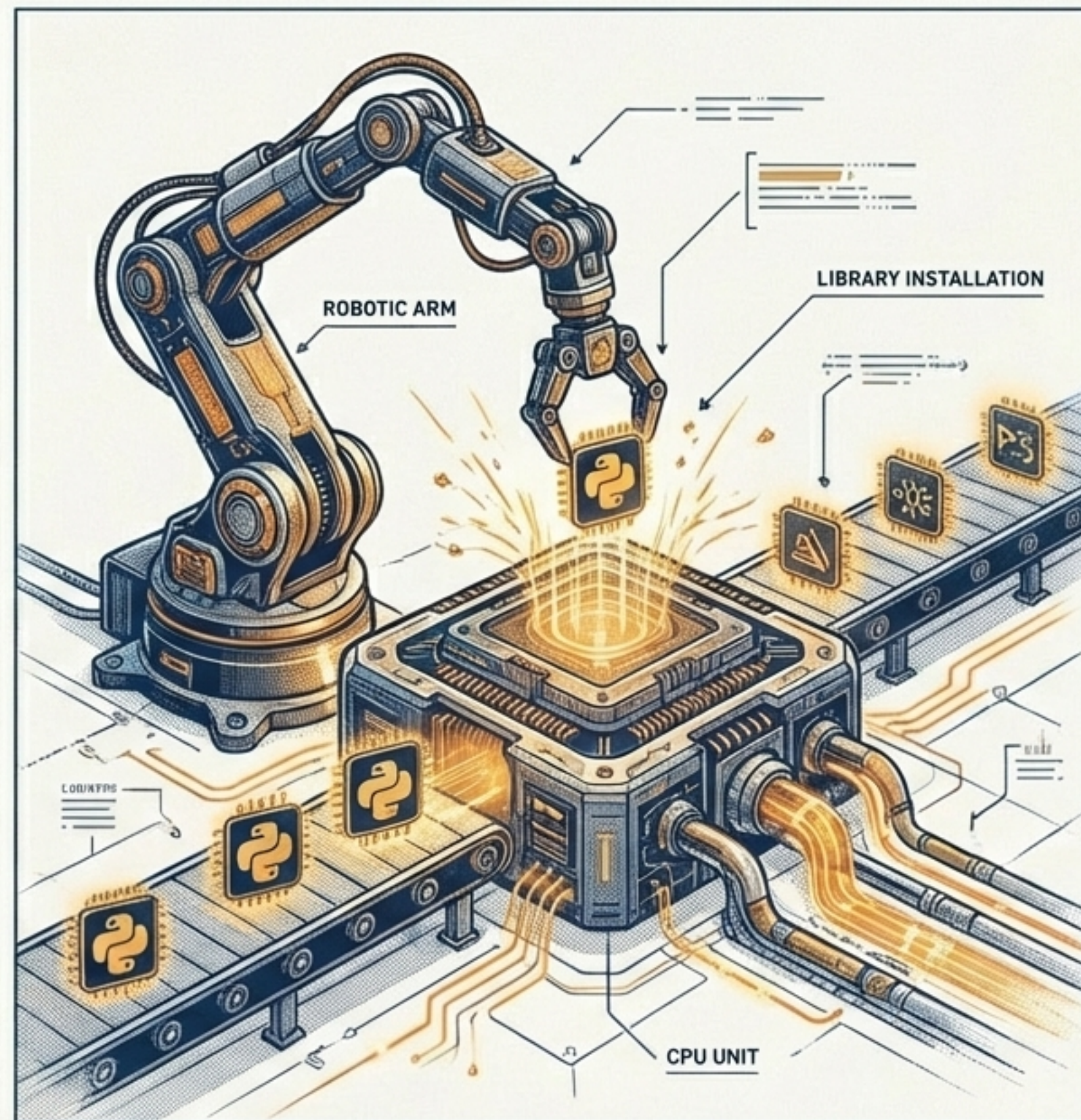


MISSION: CONSISTENT DEPLOYMENT

مرحله ۴: نصب و آماده‌سازی ابزارها

حالا که لیست رو داریم، موتورخانه شروع به ساخت ساخت و نصب تمام ابزارهای لازم از روی لیست می‌کنه. `pip` مثل بازوی رباتیک ماست که همه کتابخانه‌های پایتون رو دقیق و سریع نصب می‌کنه.

```
RUN pip install --upgrade pip && pip  
install -r requirements.txt
```

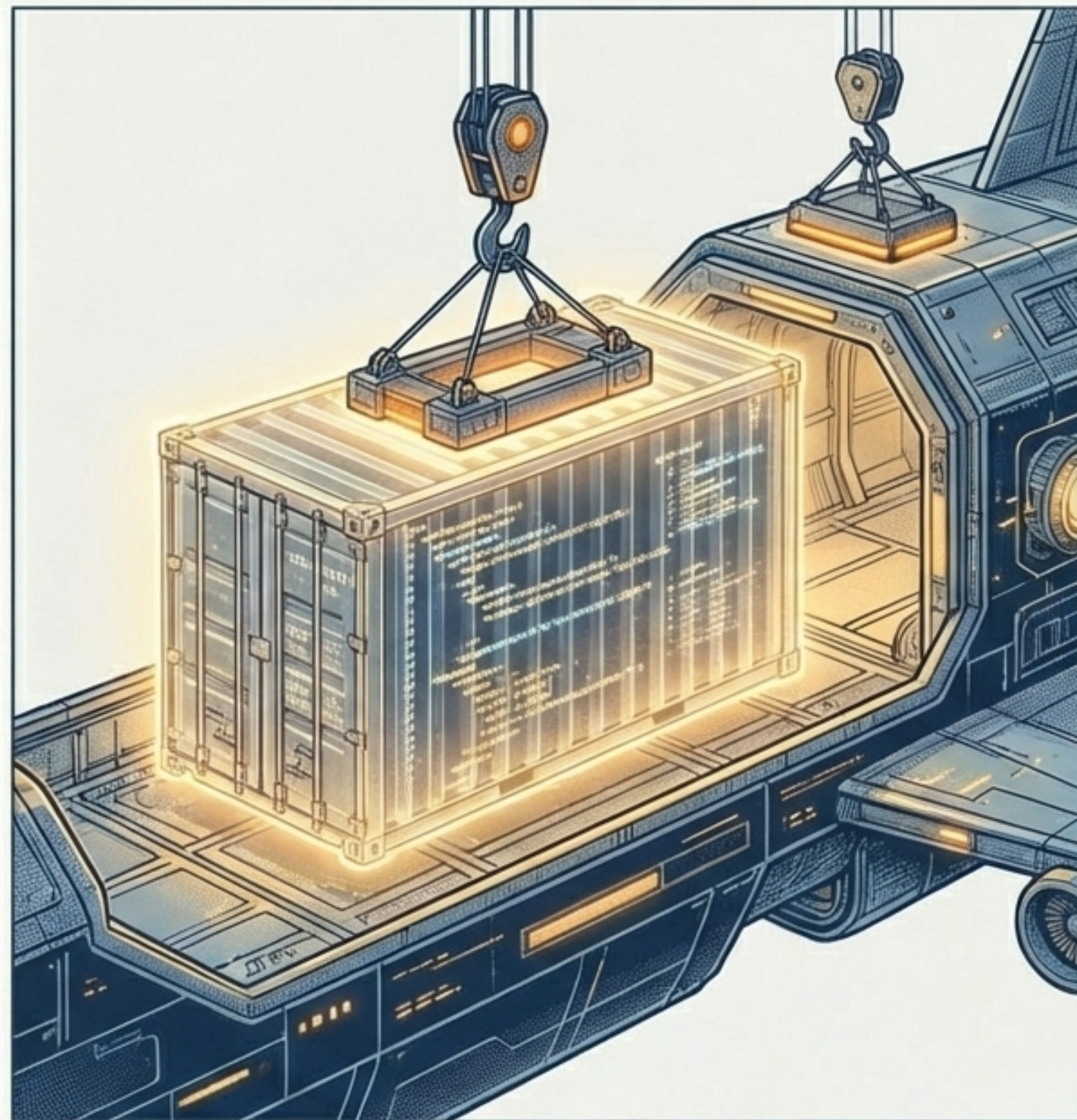


MISSION: CONSISTENT DEPLOYMENT

مرحله ۵: انتقال محموله اصلی (کد پروژه)

شاسی آماده‌ست، ابزارها هم نصب شدن. حالا وقتشه که بار اصلی- یعنی کد ارزشمند پروژه شما- رو به داخل سفینه منتقل کنیم. این دستور تمام فایل‌ها و پوشه‌های پروژه رو به اتاق فرمان (/app) کپی می‌کنه.

COPY . .

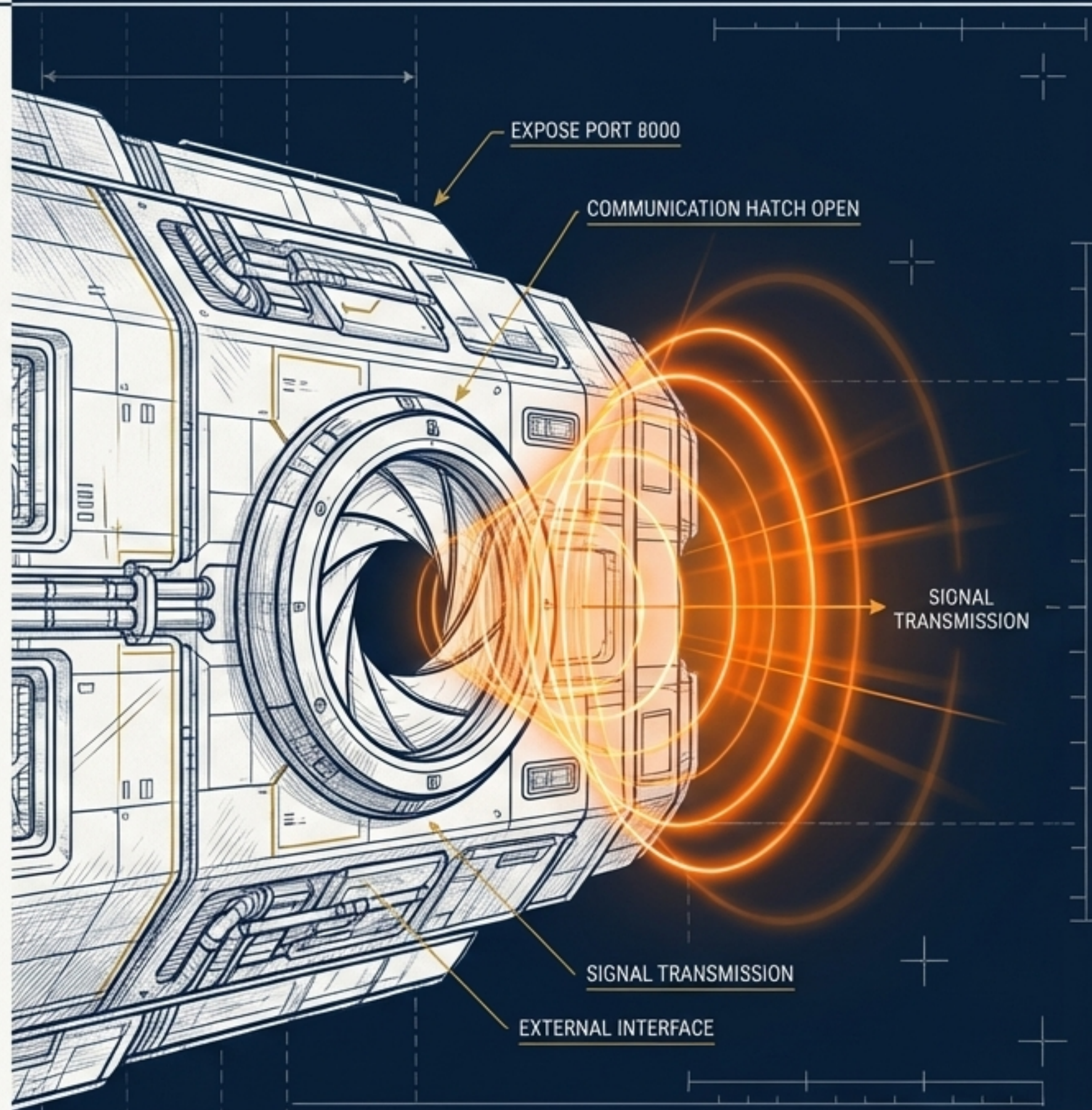


MISSION: CONSISTENT DEPLOYMENT

مرحله ۶: باز کردن دریچه ارتباطی (EXPOSE)

سفینه ما برای ارتباط با دنیای بیرون به یک درگاه یا پورت نیاز دارد. با این دستور، ما به همه اعلام می‌کنیم که اپلیکیشن ما داخل سفینه، از طریق پورت ۸۰۰۰ آماده صحبت کردن و پاسخگویی است. این فقط یک اعلامه است، اتصال نهایی موقع پرتاب انجام می‌شود.

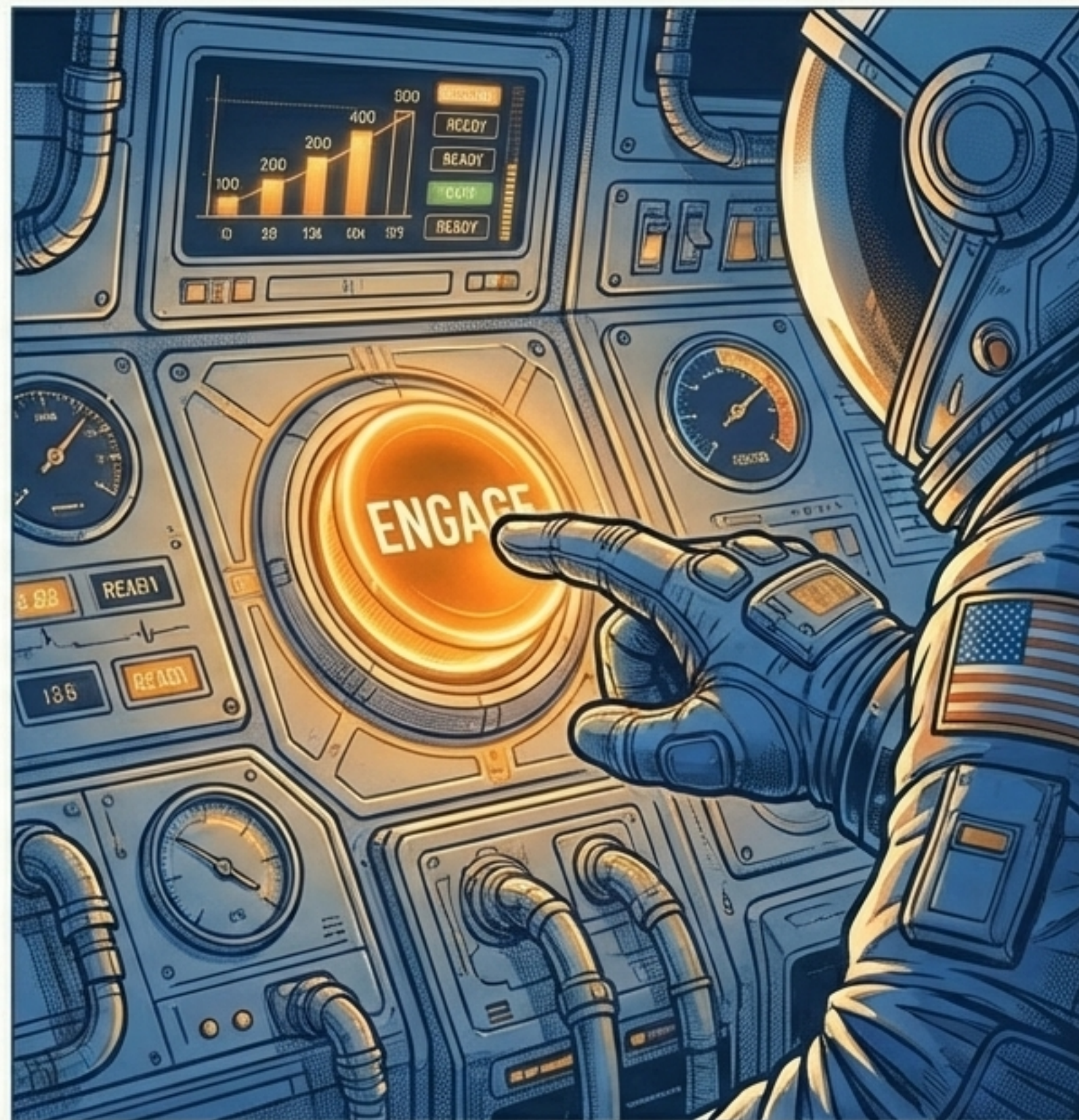
EXPOSE 8000



مرحله ۷: فعال سازی موتور اصلی (CMD)

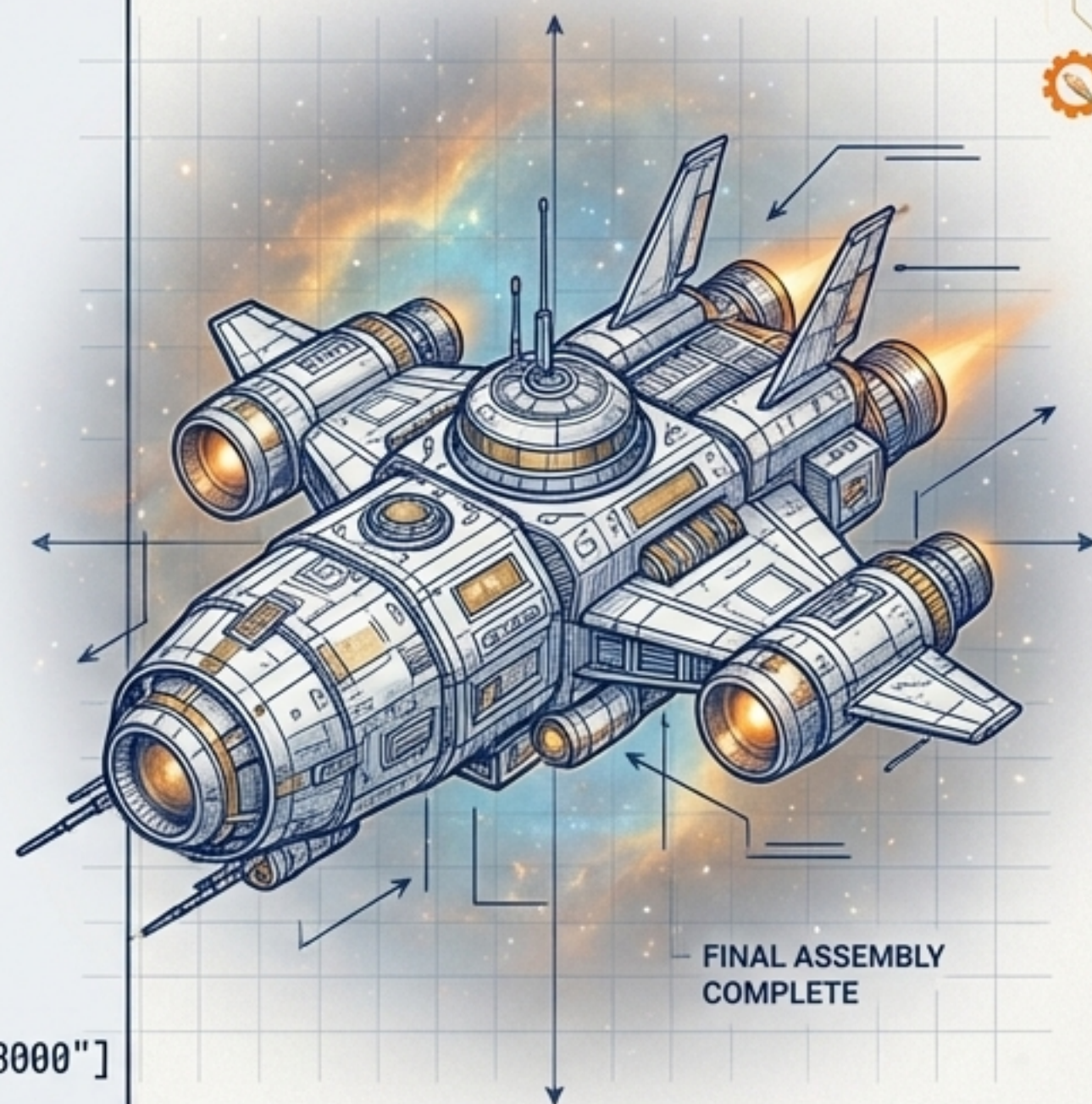
این دکمه پرتابه! با این دستور به سفینه می گیم که به محض محض روشن شدن، باید چه دستوری رو اجرا کنه تا زنده بشه. اینجا ما موتور وب سرور `uvicorn` رو روشن می کنیم تا اپلیکیشن FastAPI ما شروع به کار کنه و از طریق دریاچه ۸۰۰۰ قابل دسترس باشه.

```
CMD ["python", "-m", "uvicorn",  
     "src.app.main:app", "--host",  
     "0.0.0.0", "--port", "8000"]
```



نقشه نهایی سفینه شما

- # 1. Use the official lightweight Python base image
FROM python:3.11-slim (last pushed 1 week ago)
- # 2. Set working directory inside the container
WORKDIR /app
- # 3. Copy only dependency file first (for Docker caching)
COPY requirements.txt .
- # 4. Install Python dependencies (add curl if you use MLflow local tracking URI)
RUN pip install --upgrade pip \
 && pip install -r requirements.txt \
 && apt-get clean && rm -rf /var/lib/apt/lists/*
- # 5. Copy the entire project into the image
COPY . .
- # 6. Expose FastAPI port
EXPOSE 8000
- # 7. Run the FastAPI app using uvicorn (change path if needed)
CMD ["python", "-m", "uvicorn", "src.app.main:app", "--host", "0.0.0.0", "--port", "8000"]



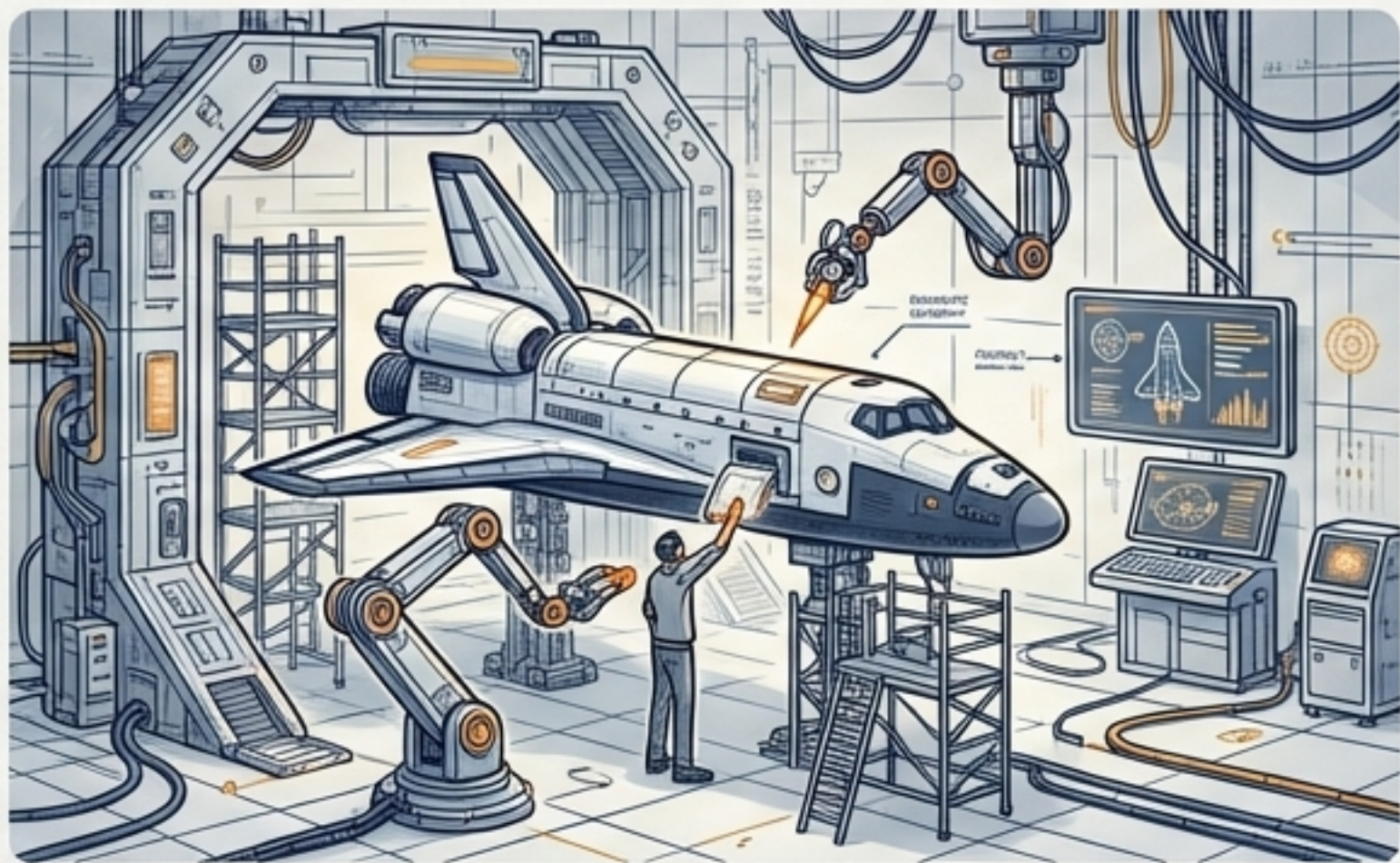
این نقشه کامل و بی نقصه. با این دستور ساخت، شما هر بار دقیقاً همین سفینه رو با تمام امکانات و تنظیماتش تحویل می گیرید. قابل اعتماد، تکرارپذیر و آماده پرواز.



شمارش معکوس برای پرتاب!



ساخت سفینه (Build)



با این دستور، داکر نقشه فنی ما رو می‌خونه و اون رو به یک سفینه واقعی و قابل استفاده تبدیل می‌کنه.

```
docker build -t project-name .
```

پرتاب به فضا (Run)



و با این دستور، سفینه رو به مدار می‌فرستیم و دریچه ارتباطی ۸۰۰۰ اون رو به ایستگاه زمینی خودمون (کامپیوتر) وصل می‌کنیم تا بتونیم باهاش حرف بزنیم.

```
docker run -p 8000:8000 project-name
```



ابزارها و آپگریدهای پیشرفته برای کاپیتانهای حرفه‌ای

Docker Compose

برای مدیریت یک ناوگان کامل (مثلاً سفینه اپلیکیشن + سفینه دیتابیس) و تعریف ارتباط بین اون‌ها.

Docker Build Cloud

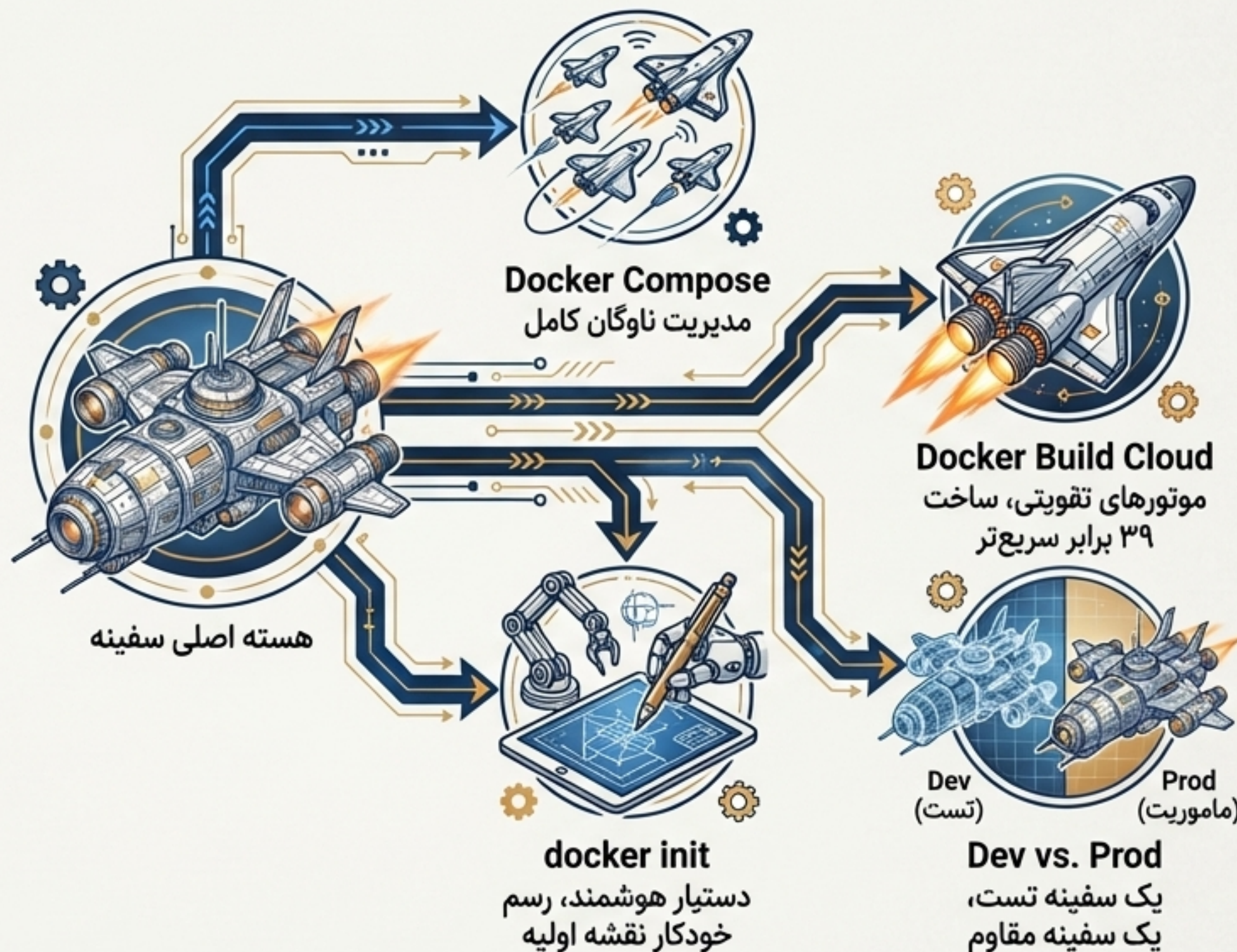
موتورهای تقویتی برای ساخت سفینه‌های غول‌پیکر غول‌پیکر تا ۳۹ برابر سریع‌تر!

docker init

دستیار هوشمندی که نقشه اولیه سفینه و فایل‌های جانبی رو به صورت خودکار برای شما می‌کشد.

Dev vs. Prod

همیشه یک سفینه برای آزمایش و تست (Dev) و یک سفینه مقاوم و بهینه برای ماموریت اصلی (Prod) داشته باشید!



شما با موفقیت اولین سفینه خود را ساختید!

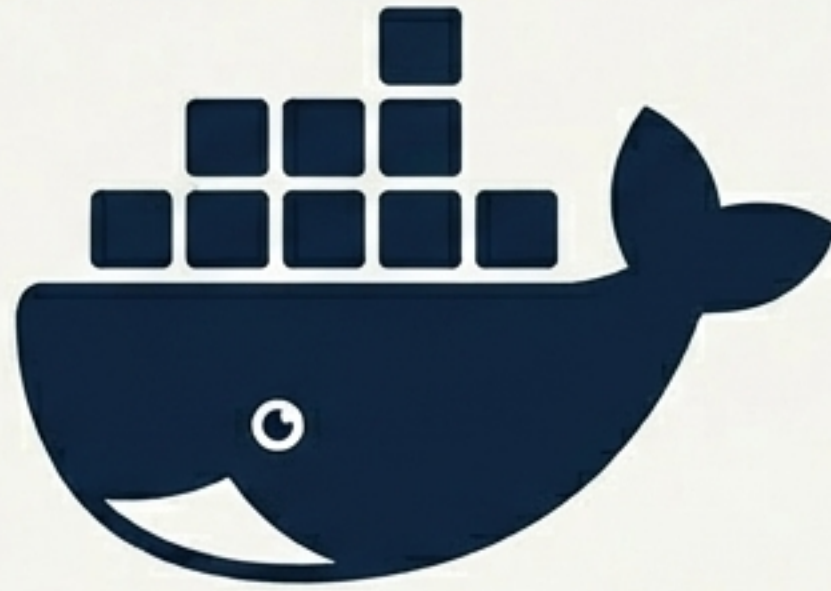
ماموریت با موفقیت انجام شد. ما با یک مشکل
بزرگ (هرج و مرج محیطها) شروع کردیم، از
یک نقشه ۷ مرحله‌ای پیروی کردیم و یک سفینه
قابل اعتماد و قابل حمل برای کدمون ساختیم.

از این به بعد، اپلیکیشن شما می‌تونه در هر
هر کهکشانی، هر زمان و بدون هیچ غافلگیری‌ای
اجرا بشه.



MISSION: CONSISTENT DEPLOYMENT

!Happy Dockering



به کهکشان کانتینرها خوش آمدید

