Fast API

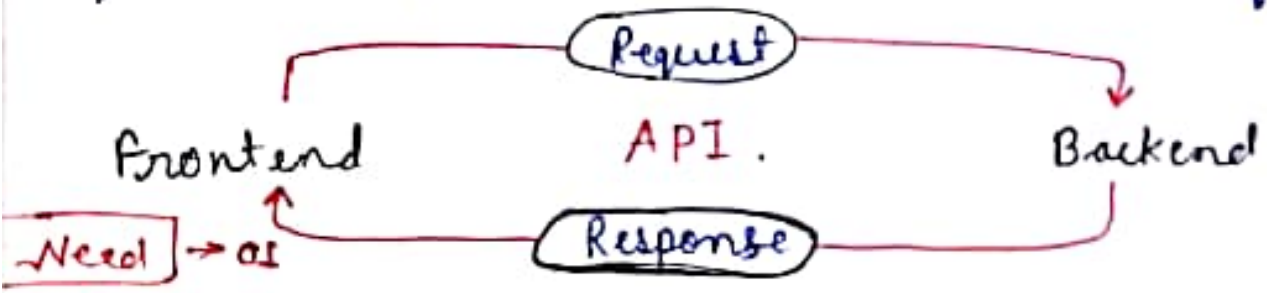
Lecture -01

23-08-2025

FAST_API.

What is an API?

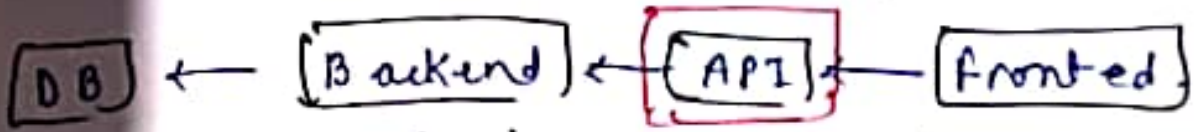
API's are mechanism that enable two s/w components — such as the frontend and backend of an application — to communicate with each other using a define set of rules, protocals and formats.



Before API's application built in Monolithic Architecture whole application (frontend + backend + database) within a single folder.

The problem is Tightly Coupled Nature We can't share data from anyone in this application because it tightly coupled.

API use in decoupled arch. like:-

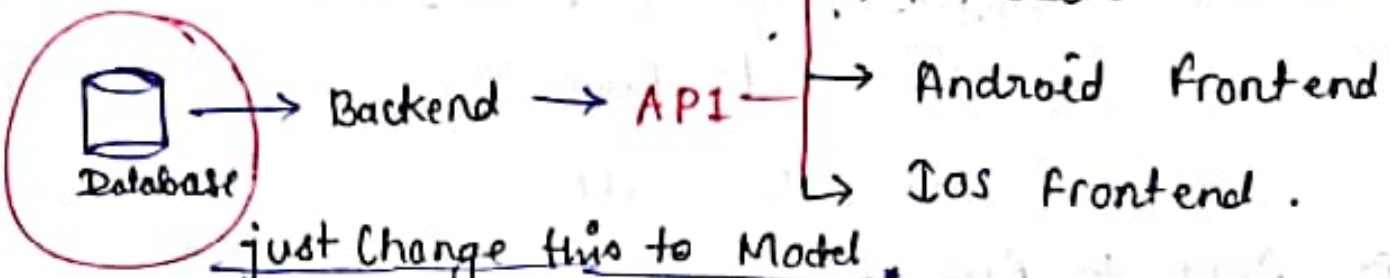


The communication between these s/w under a some protocals that is we follow a HTTP Protocal.

After fetching data via API the response or info fetch return in JSON.

It is universal data format every programming can take and understand.

Need → 02



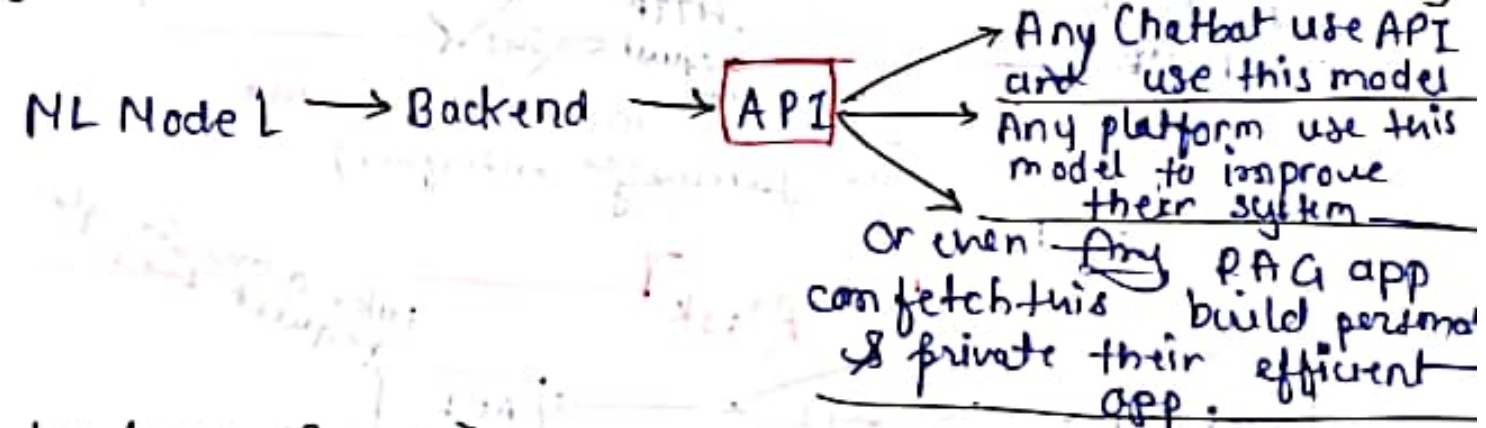
just Change this to Model.

API – ML Perspective →

ML Model → Backend → Frontend

The problem since Monolithic Architecture [Tightly Coupled nature]

The Solution →



Lecture - 02 →

FAST_API → Fast_API is a modern, high-performance web framework for API's with python

Starlette + Pydantic → fast_API

Starlette: manages how your API receives request and sends back response

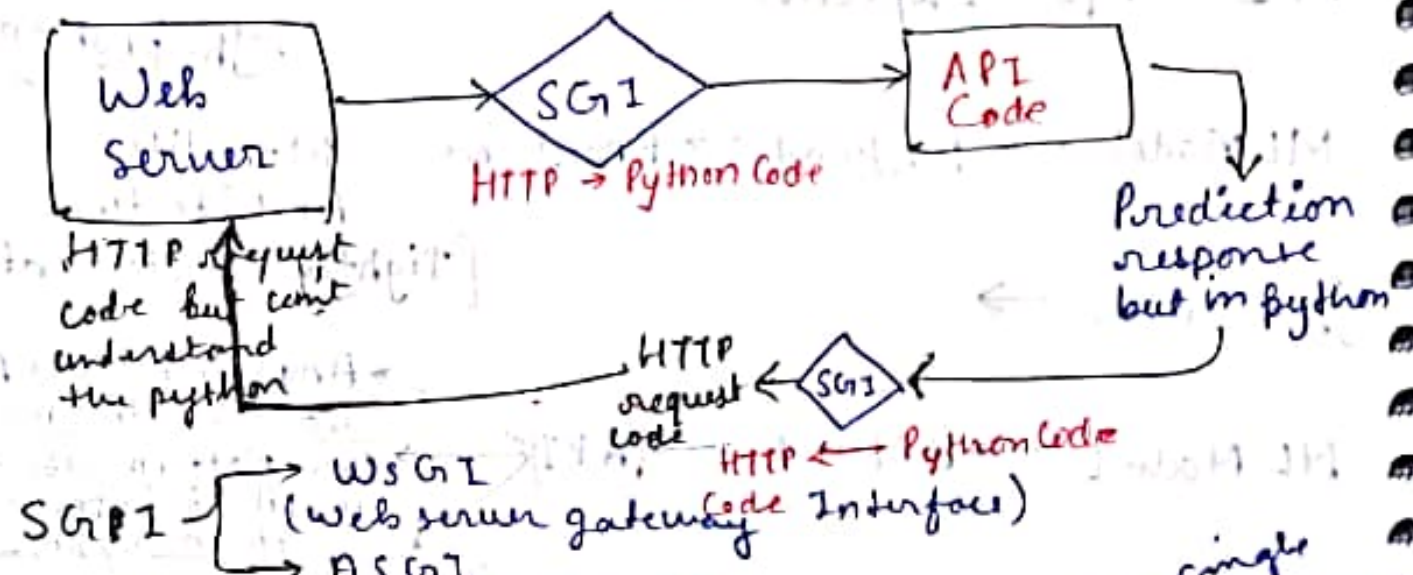
+

Pydantic used to check if the data coming into your API is correct and in the right format.

Philosophy of Fast_API

- Fast : to Run
- Fast to Code

① Fast to Run ?



Flask protocols

Flask
Web Server
SGI
API Code
WSGI-werkzeg
Synchronous End point
take a one single request time

Gunicorn (based on WSGI) HTTP saver

Fast_API

FastAPI
Web Server
SGI
API Code
Use feature [Async-Await]

uvicorn
ASGI - Starlette (Asynchronous server gateway Interface)
Asynchronous End point

It is high performance & concurrent process support In this.

② Why FAST-API is fast to code ? [Pydantic support]

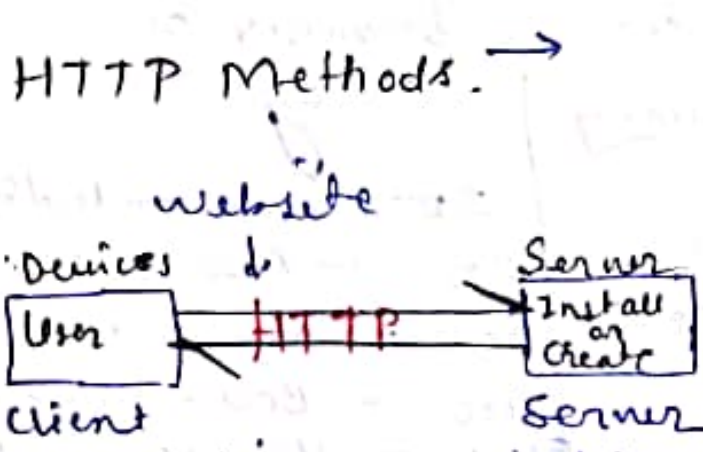
1. Automatic input validation_
2. Auto Generated Interactive Documentation_
3. Seamless Integration with Modern Ecosystem [ML/DL libraries, OAuth, JWT, SQLAlchemy, Docker, Kubernetes etc]

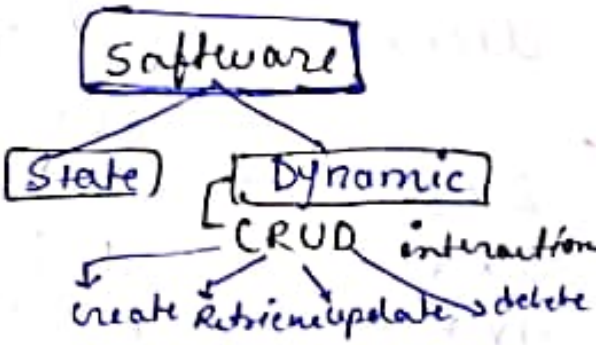
Lecture -03

HTTP Methods In Fast-API.

23-08-

HTTP Methods. →





For taking or retrieve something from website — GET.

For Send — POST

For Update — PUT

For delete — DELETE

} HTTP Methods.

Lecture -04

Path & Query Params (Parameter)

24-08

Path Params → they are dynamic segments of a URL path used to identify a specific resources.

localhost : 8000/ view / .3 → Dynamic segment of resources.

Page or section

Path() → The Path() function in FAST API used to provide metadata, validation, rules and documentation hints for path parameters in API.

Title, Description, Example, ge, gt, Le, lt

Min_length, Max_length, regex

HTTP Status Codes → They are 3 digit numbers returned by a web server (like FAS API) to include the result of a client's request (like from a browser or API consumer)

		Eg.
2XX	→ Success	200-OK — Standard success
3XX	→ Redirection	201 — Resource created
4XX	→ Client error	204 — No content
5XX	→ Server error	400 — Bad request
		401 — Unauthorized
		403 — Forbidden
404	- Page Not found	502 - Bad gateway
500	- Internal server error	
503	- Service Unavailable	

HTTP-Exception → It is special built in exception in Fast API used to return custom HTTP error responses when something goes wrong in API

Instead of returning a normal JSON or crashing the server, you can gracefully raise an error —
- a proper HTTP status code
- a custom error message
- (optional) extra headers

Query Params → They are optional key-value pairs appended to the end of URL used to pass additional data to the server in an HTTP request. They are typically employed for operations like filtering, sorting, searching, and pagination, without altering the path itself. asc↑ desc↓

/patients? city = Delhi & sort_by = age, order

→ ? · mark start : the query params-
→ Each param is key value · pair (like sort_by = asc)
→ Multiple param are seperated by &

Query( ) → is a utility func provide by FAST API to declare, validate and document query parameters in your API endpoints.

It allows you to : ◦ Set default values
◦ Enforce validation
◦ Add metadata → like title, description, ex

---

default → Set default value
title → Display in API docs
description → Detailed explanation in swagger
example / examples → Provide simple I/P
min_length, max_length → Validate str length
ge, gt, le, lt → validate numeric bounds
regex → Pattern match for strings

Lecture - 08 **Pydantic** 25-08

1. Define a pydantic model (class) that represents the ideal schema of the data.
◦ This includes the expected fields, their types and any validation constraints
2. Instantiate the model with raw input data (usually a dict or json like structure)
3. Pass the validated model object to functions or use it throughout your codebase

Optional field → If its not filled then show the none.

It provide validation

- Type validation
  - : int
  - : str
  - : List[str]
- Data validation
  - : EmailStr
  - : AnyUrl

Field : for data validation (gt (greater than) 20, lt = 120

Field

~~Field~~ (gt=0, lt=120) / (max_length=5)

→ It is use for attach metadata for adding description, title, examples, default = None

Annotated → It is use adding multiple constrain of field at a time on a single place

strict = True / False

If we use this constrain in field then we can't change the type of var than there is no type conversion

eg - Pydantic is smart enough to 75 = '75' and print it but when this is true it will give error

Field validator → For personalise or business related data validation that you mode custom as per your requirements

basically customise field object for personal use

In this feature there is mode = 'before' / 'after'
before type conversion its after (also default)

Model validator →
when we want to validate more than one field by their corresponding means validate interrelated form (combine)
so this model validator.

computed fields → that fidd are created by computer not user but by using their info.

Eg. weight = 1.81 m
height = 55 K
→ user created field.

→ But we want BMI
so we define formula using computed field decorator.

@computed field
@property
def calculate_bmi(self) → float:
    bmi = self.weight / (self.height)^2
    return bmi

This field bmi is computer

Nested Model →

It use when in model there is type data types or more there

like - 58 house no, bhopal, 462001

↓

in Patient model, but for reading this we create another called Address that it will define in the Patient model -

like class Patient (Base Model):

name : str
age : int
address : Address ← Nested Model -

Serialization →

To export existing pydantic model object to json, dict format, file to building API, debugging etc.

patient.model_dump() → dict → format, dtype

patient.model_dump_json() → format - json
dtype - str

include = { } → what want

exclude = { } → what remove

exclude_unset = { } it print only that we define value not whole model values

Lecture - 05

Post request In fast API

25 - 08 -

Request Body → It is the portion of an HTTP request that contains data sent by the client to the server. It is typically used in HTTP method such as POST or PUT to transmit structured data (eg. JSON, XML, form data) for the purpose of creating or updating resources on the server.

client ——— server
POST
JSON
request body

Lecture → 06

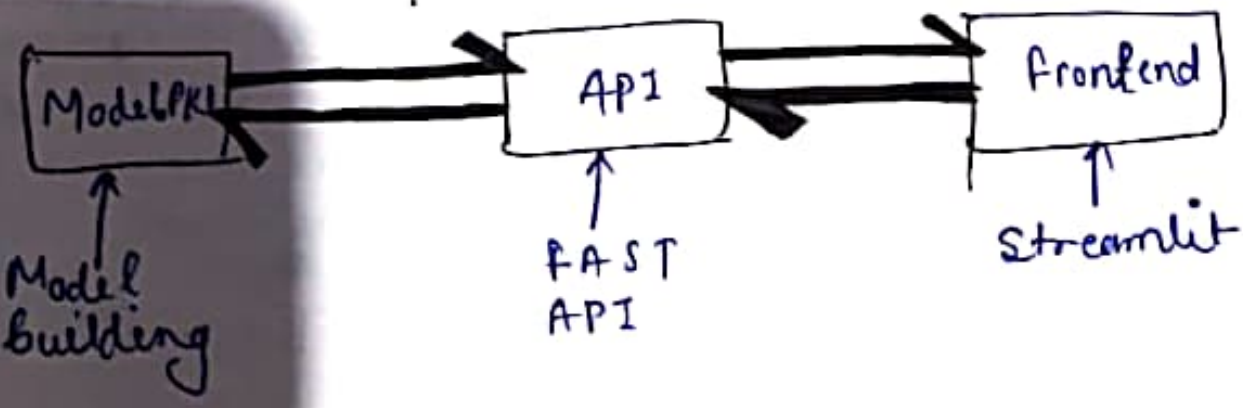
PUT & Delete In fast API

PUT → Update.
Delete → for delete

Project complete all crud operation is used.

Lecture → 07

ML with fast API



Lecture -08

Improving the FASTAPI APZ

1. Create a new folder
2. Field validator for city feature
3. Add routes
   a. Home
   b Health check
4. Add model version
5. Seperation of logic
   a. Pydantic model (class)
   b. city file
   c. ML logic
6. Try Catch
7. Add confidence score
8. Response Model.

. Response Model →
It defines the structure of the data that your API endpoint will return. It helps in:
1 Generating clean API
2. Validating output
3. Filtering unnecessary data.

25-08

Lecture → # Docker

Docker is a platform designed to help developers build, share and run container application.

Why do we need docker. →

→ Consistency Accross.

Problem → Applications often because differently in development, testing, and production env due to variations in configurations - ~~develop~~ dependencies, and infrastructure.

Solution → Docker container encapsule all the neccessary components, ensuring the application runs consistently across env

→ Isolation → Running multiple app on the same host So docker isolated env for each.

→ Scalability.

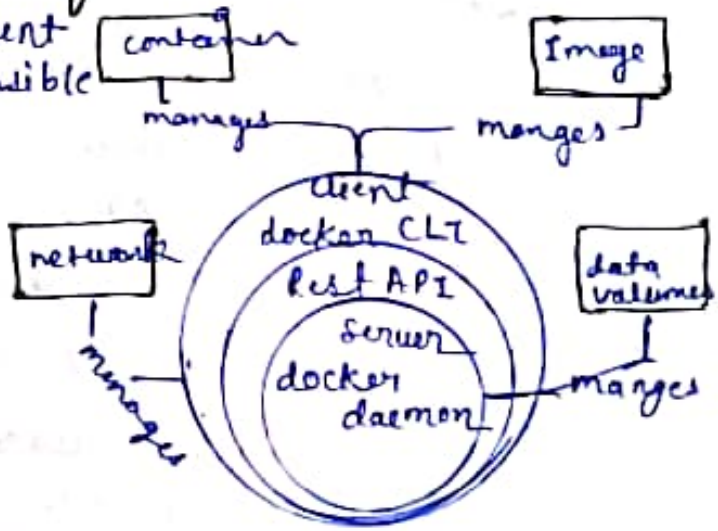
↳ To handled increase load can be challenging requiring, manual intervention and configuration

↳ Makes it easy to scale app horizontally by running multiple container instances allowing for quick and efficient scaling.

→ ## Components of Dockers

Docker Engine is the core component of the docker platform, responsible for creating, running and managing docker containers

It serves as the runtime that powers docker's containerization capabilities, ~~in~~ in depth look at the docker Engine



Docker Image → It is a light weight, stand alone and executable software package that includes everything needed to run a piece of software, such as the code, runtime, libraries, env. variables and config files. Images are used to create Docker containers, which are instances of these images

Components of this →
1. Base Image
2. Application Code
3. Dependencies
4. Metadata

Docker Image lifecycle → creation → Storage → Distribution → Execution

Dockerfile → It is text file that contains a series of instruction used to build a docker image. Each instruction in a dockerfile creates a layer in the image, allowing for efficient image creation and reuse of layers.
Dockerfiles are used to automate the image creation process ensuring consistency and reproducibility.

- Key components in this →
  1. Base Image (FROM)
  2. Labels (LABEL)
  3. Run Commands (RUN)
  4. Copy files (COPY)
  5. Environment variables (ENV)
  6. Work Directory (WORKDIR)
  7. Expose Ports (EXPOSE)
  8. Command (CMD)
  9. Volume (VOLUME)
  10. ARGUMENT (ARG)

Docker Container → is a lightweight portable and isolated env that encapsulates an application and its dependencies allowing it to run consistently across different computing env.. Containers are created from docker Images, which are immutable and contain all the necessary components for the application to run.

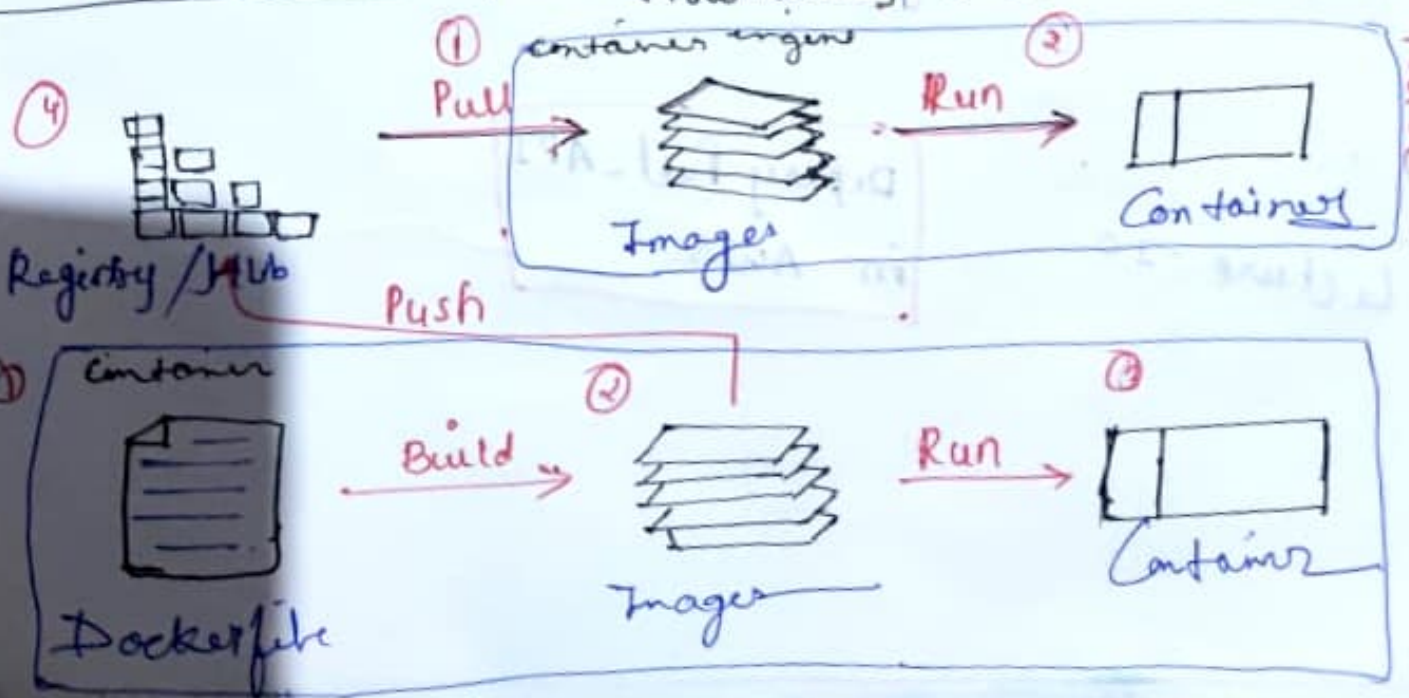
Registry → A docker registry that stores and distributes docker images
It act as a

Key Components
1. Reposatories
2. Tags.

Types of docker registry
1. Docker hub →
   - Description
   - URL
   - Use Cases
2. Private Registries
   - Dese
   - Use Case
3. Thir-Party Registries

Benefits
1. Centralized Image mgmt
2. Version Control
3. Collaboration
4. Security
5. Integration CI/CD.

How exactly docker is used →



Use - Cases →
- MicroServices Architecture
- Continuos Integration and continuos Deployment (CI/CD)
- Cloud Migration
- Scalable Web Applications
- Testing and QA
- M/C Learning & AI
- API development and deployment

To making dockerfile.

base image

work dir

copy

run

port

command

Lecture - 09

FAST-API
Dakerization

Lecture - 10

Deploy Fast-API in AWS