



# Getting up and Going with Docker and Containers

# Your Instructor



**Ajeet Singh Raina**

- Developer Advocate at Docker
- Former Docker Captain
- Docker Community Leader
- Distinguished ARM Ambassador
- Worked @ Dell EMC, VMware, Redis & CGI

# Agenda

## **Lab 1: Inner-Loop Dev Workflow (45 min)**

- Overview
- Docker Developer Workflow
- Image Building Best Practices
- Speed: Docker Init
- Speed: Compose File Watch

## **Lab 2: Container-first Dev Workflow ( 35 min)**

- Overview
- Building a sample app using React, Node and Mongo
- Integrate app with the real-cloud Service

## **Lab 3: Container-supported Dev Workflow (40 min )**

- Overview
- Building a sample app tech stack using emulated-cloud service
- Bringing up the services
- Validating the results
- Wrapping up

# Before we get started

Docker Desktop

## The #1 containerization software for developers and teams

Your command center for innovative container development

[Get Started](#) [Download for Mac - Apple Silicon ▾](#)

[Download for Mac - Intel Chip](#)

[Download for Windows](#)

[Download for Linux](#)

ⓘ Commercial use of Docker Desktop is available for organizations with more than \$10 million in annual revenue.

ⓘ Docker Desktop is available for individuals, small businesses, or organizations with up to 10 employees OR more than \$10 million in annual revenue, or Business).

<https://www.docker.com/products/docker-desktop/>



# Access the Workshop

```
$ docker extension install \
```

```
dockerdevrel/wad-docker-extension:2.0
```

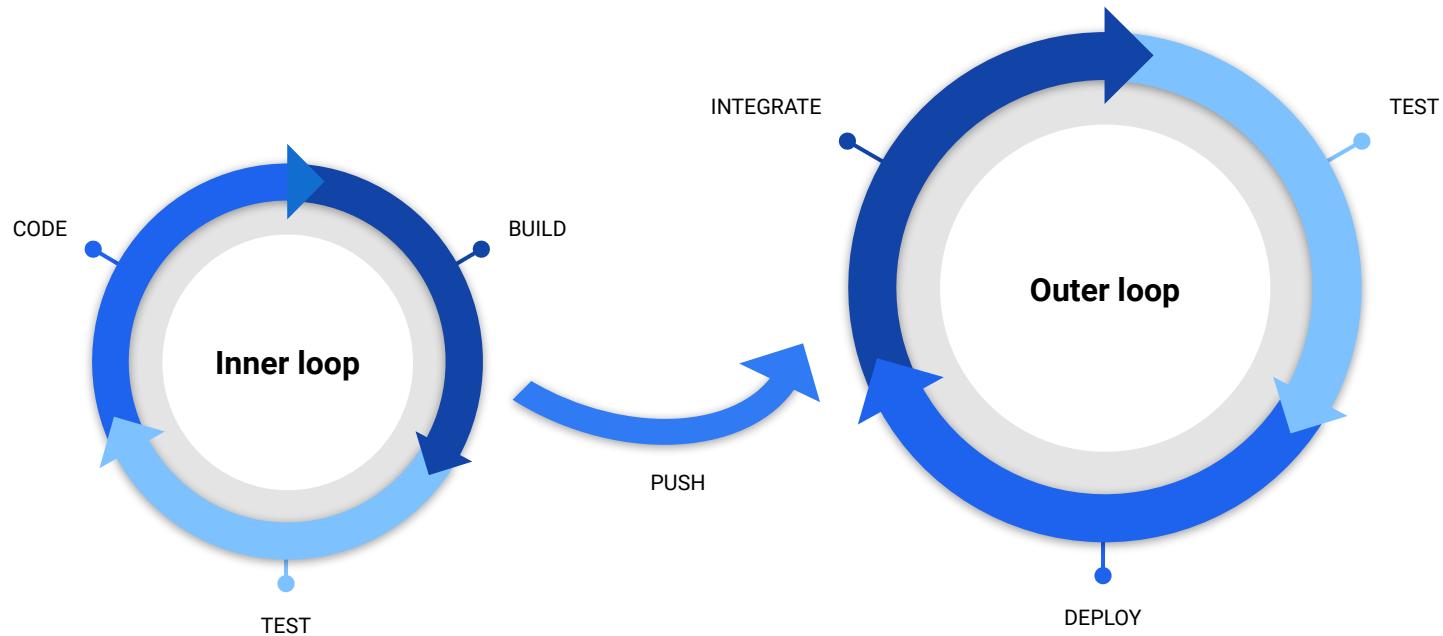


# Access the Workshop

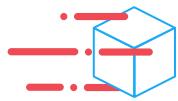
<https://dockerworkshop.vercel.app/>



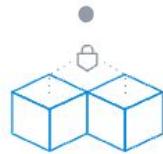
# Docker is uniquely focused on developer success



# What Modern Developers Success Look Like?



Speed

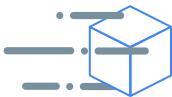


Security



Choice

# What Modern Developers Success Look Like?



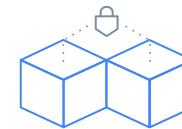
## Speed

- Faster Development
- Rapid Prototyping
- Efficient Builds
- Faster Deployment



## Security

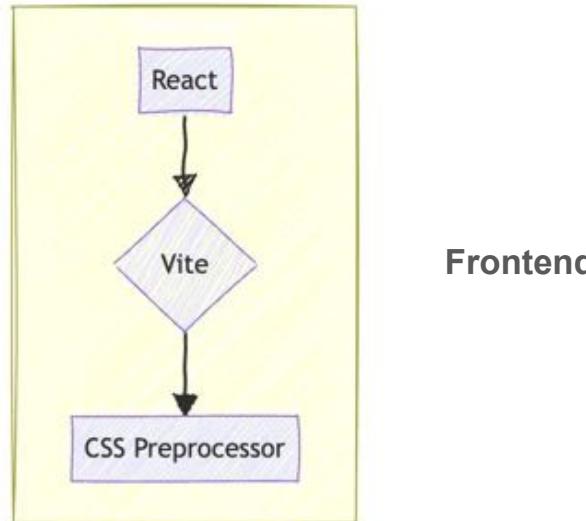
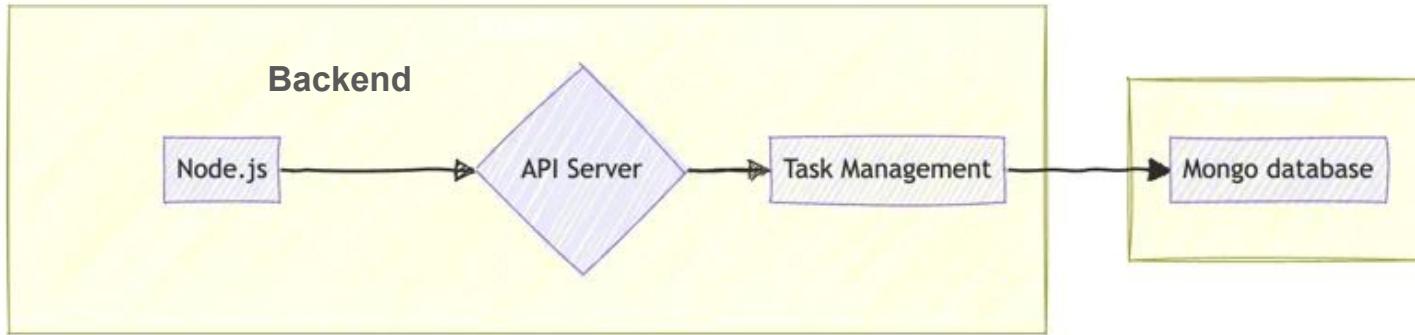
- Supply Chain Security
- Isolation
- Image Security



## Choice

- Platform Compatibility
- Ecosystem Expansion
- Technology Agnostic

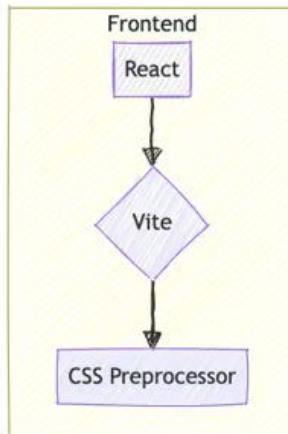
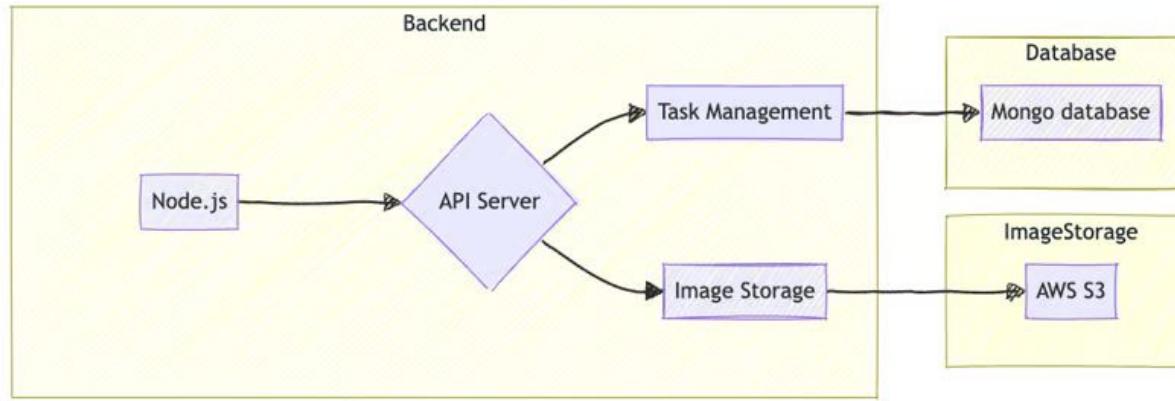
# Tech Stack #1 – A Simple todo-List App



# Challenges

- **Environment inconsistencies:** Different developers may have varying operating system versions, software installations, and configurations, leading to discrepancies in application behavior.
- **Dependency management challenges:** Keeping track of and managing dependencies across frontend and backend can be time-consuming and error-prone.
- **Slow development cycles:** Setting up new development environments for each team member can be lengthy, hindering productivity.
- **Deployment complexities:** Replicating the development environment in production can be difficult due to environment-specific configurations.

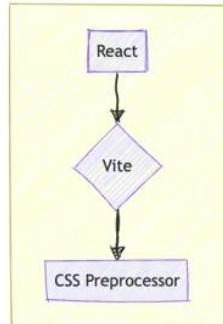
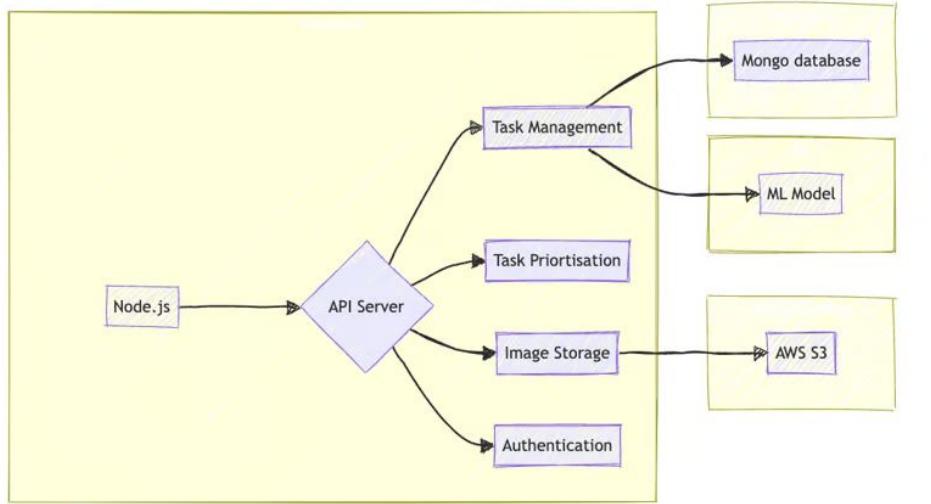
# Tech Stack #2 – Using Cloud Service



# Challenges

- **Complex environment setup:** Requires setting up Node.js, database, image storage (AWS S3 or similar), and potentially other dependencies.
- **Dependency management:** Managing dependencies for the application, database, and image storage can be challenging.
- **Inconsistent environments:** Different developers may have different setups, leading to discrepancies.
- **Deployment complexities:** Deploying to production involves configuring multiple services.

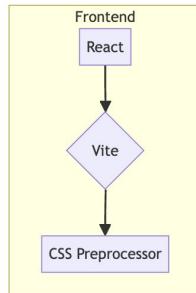
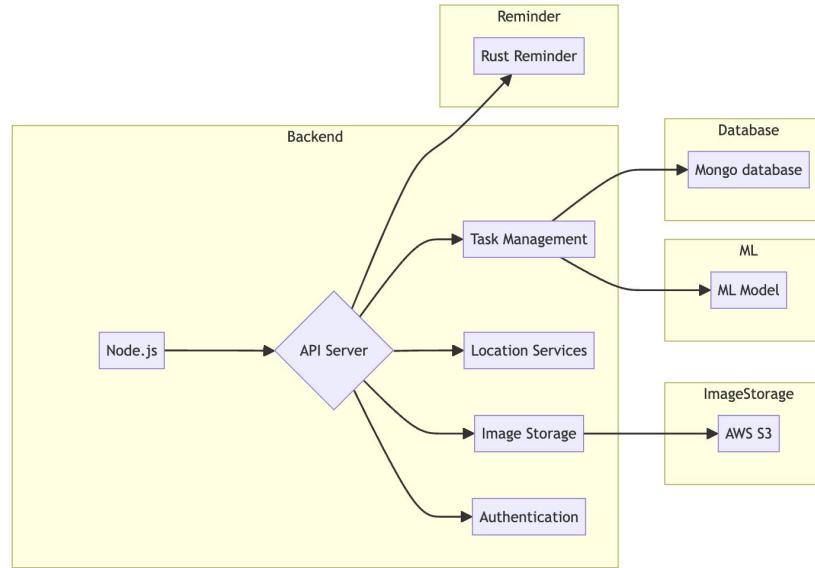
# Tech Stack #3 – Task Prioritization using AI



# Challenges

- **Integration with existing systems:** Deploying the model into a production environment can be complex.
- **Complex workflows:** Managing multiple experiments, hyperparameter tuning, and model iterations can be challenging.
- **Reproducibility:** Ensuring consistent results across different environments is crucial.
- **Tracking model performance:** Monitoring model metrics and detecting performance degradation is essential.
- **Model updates:** Retraining the model with new data to maintain accuracy is necessary.

# Tech Stack #4 - Securing and Authentication



# Challenges

- **Misconfiguration:** Incorrectly configured servers and applications can expose vulnerabilities.
- **Outdated Software:** Difficulty in keeping up with security patches and updates for all dependencies.
- **Complex Authentication Implementations:** Developing and maintaining custom authentication systems can be time-consuming and error-prone.
- **Session Management:** Managing user sessions and preventing session hijacking is challenging
- **Shared Hosting Environments:** Potential security risks from neighboring applications on shared servers.

# Docker Developer Tools

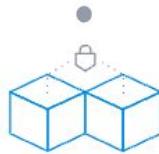


# Built for Developers, by Developers



## Speed

- Docker init
- Compose File Watch
- VirtioFS Support
- VPNKit => gVisor
- Docker Build Cloud



## Security

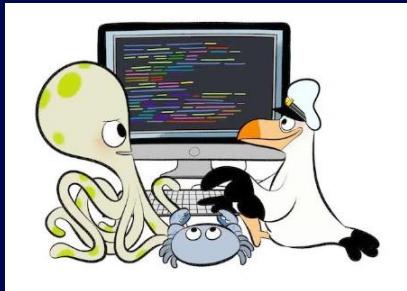
- Docker Scout
- Attestations



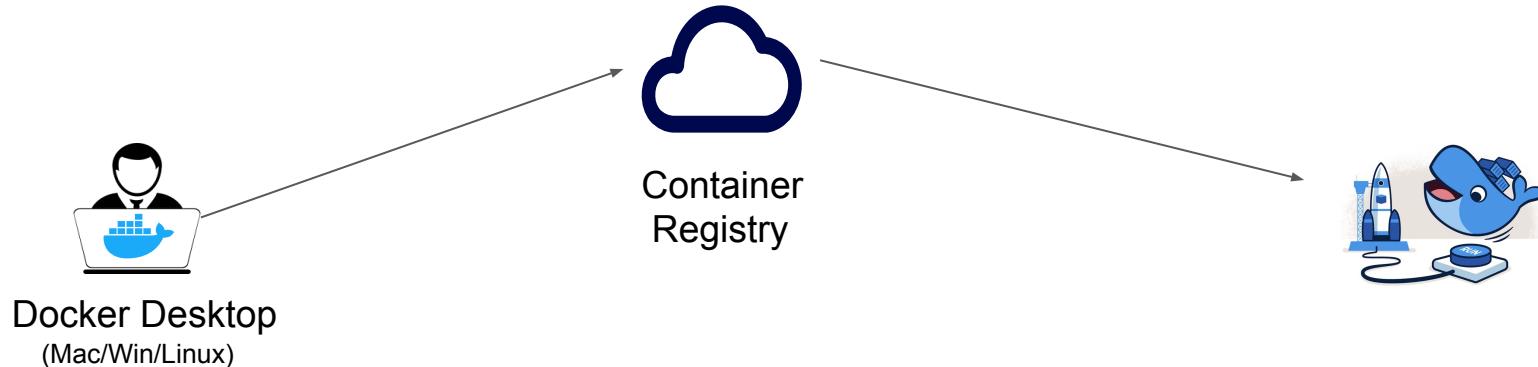
## Choice

- Docker Extensions
- Docker Sponsored Open Source Projects
- Rosetta 2
- WebAssembly

# Speed for developers



# A 30,000 ft View



## BUILD

- Package applications as portable container images
- Create Multi-container apps using Docker Compose

```
$ docker build
```

## SHARE

- Collaborate and distribute via Registry
- Shareable application with clear interface for operators

```
$ docker push
```

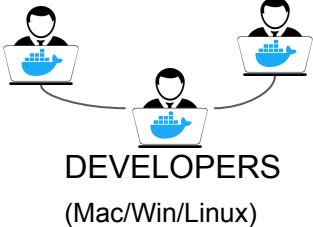
## RUN

- Run multiple versions of the same application and manage pre-environment settings
- Launch your applications locally and on the cloud with AWS ECS and Azure ACI.

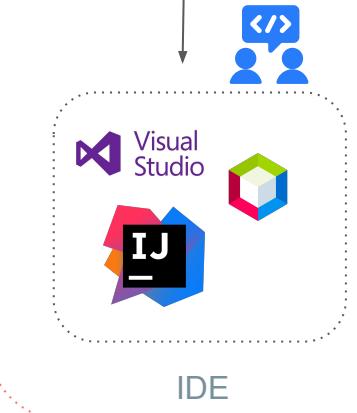
```
$ docker run
```

# Inner-Loop Developer Workflow

## BUILD



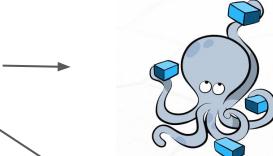
Code Commit



Source Control



## SHARE



docker push

Define services

```
php:  
  build: php  
  port: "80:80"  
  ports: "441:443"  
  volumes: "/path/www:/var/www/html"  
  links: db
```

\$ docker-compose up



## RUN



Microsoft Azure

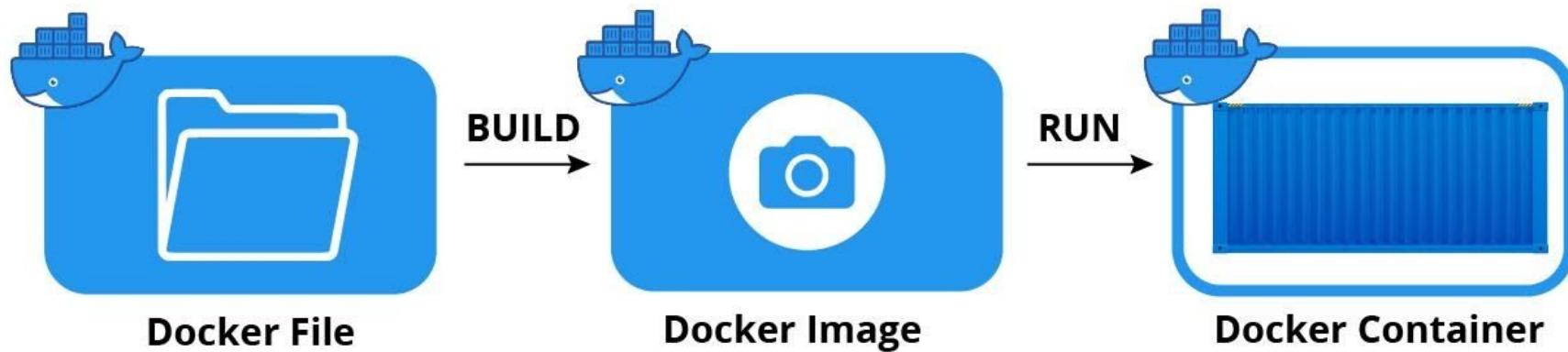
# Build and Deploy a Simple Node Application



```
git clone https://github.com/.../..
cd [path to your node-docker directory]
npm init -y
npm install
touch server.js
node server.js

curl --request POST \
  --url http://localhost:8000/test \
  --header 'content-type: application/json' \
  --data '{"msg": "testing"}'
```

# A Typical Docker Workflow



# Application

```
git clone https://github.com/.../..
cd [path to your node-docker directory]
npm init -y
npm install
touch server.js
node server.js

curl --request POST \
  --url http://localhost:8000/test \
  --header 'content-type: application/json'
\
  --data '{ "msg": "testing" }'
```

# Dockerfile

```
FROM node

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

```
$ docker build -t node-docker .
$ docker run -d -p 8080:800 node-docker
```

## Do's

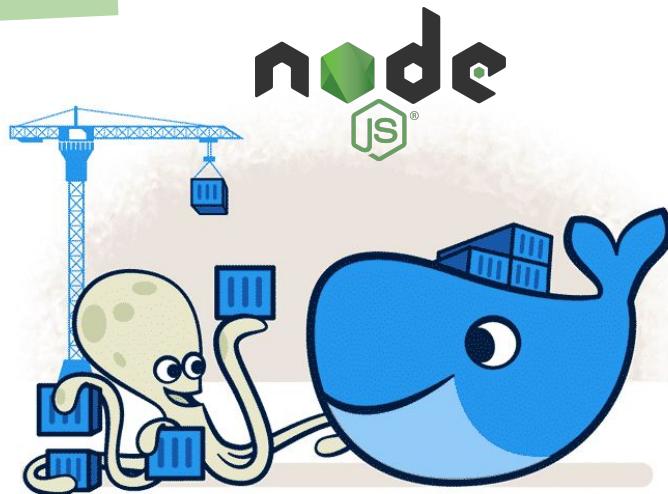
✓

- node:<tag>
- USER node
- MEM LIMIT
- HEALTHCHECK
- SCANNING
- COPY
- Multi-Stage
- .dockerignore
- buildx -platform

## Don'ts

✗

- node:latest
- ADD
- USER root
- EXPOSE db\_port
- node\_modules
- SIGTERM



# 9 Tips for Containerizing a NodeJS Application

## 1 Use a specific base image tag instead of latest

The "latest" Docker image tag is unpredictable, hence try to use specific image tags whenever possible.

## 2 Use Multi-Stage Build

Multi-Stage build makes the final image more secure and smaller in size.

## 3 Fix security vulnerabilities in your Node image

Always run security scans on the images to detect any known vulnerabilities.

## 6 Run container as non-root user

Running app with user privileges is safe since it helps mitigates risks.

## 5 Use .dockerignore

To increase build performance, it is recommended to use .dockerignore file in the same directory as your Dockerfile.

## 4 Leverage HEALTHCHECKS

Test a container and confirm that it's still working.

## 7 Favor Multi-arch Docker Image

Always build the Docker image with Multi-Architecture Platform Support.

## 8 Explore graceful shutdown options for Node

Ensure that your app is handling the ongoing requests and cleaning up resources in a timely fashion.

## 9 Measure Node Performance

Use the OpenTelemetry API to ensure that apps are faster and more performant.



# #1 Use explicit Base Image reference instead of latest

```
FROM node:latest  
  
WORKDIR /usr/src/app  
  
COPY package*.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 8080  
CMD [ "node", "server.js" ]
```



```
FROM node:16.17  
  
WORKDIR /usr/src/app  
  
COPY package*.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 8080  
CMD [ "node", "server.js" ]
```



# Downsides of latest Images

- Inconsistent Docker Image Builds
- Non-deterministic Behavior

The screenshot shows the Docker Desktop interface. On the left, the sidebar includes 'Containers', 'Images' (selected), 'Volumes', 'Dev Environments (BETA)', 'Extensions (BETA)' (with 'slim.ai' listed), 'Disk usage', and '+ Add Extensions'. The main area displays the Docker Hub Public 'node' image, specifically the 'latest' tag. The 'FILE EXPLORER' tab is active, showing 'Instructions for All 9 Layers' and 'Container Startup Files'. Below this, a table titled 'File Changes for All 9 La...' lists directory changes across 32,193 objects. The table has columns for Name, Added, Modified, Deleted, Count, Mode, and Size. Key entries include the root directory '/' with 28,572 files (mode drwxrwxrwx, size 934.5 MB) and sub-directories 'bin', 'boot', and 'dev'. The bottom status bar shows RAM 3.57GB, CPU 33.73%, and 'Connected to Hub'. The version v4.12.0 is visible in the bottom right.

Name	Added	Modified	Deleted	Count	Mode	Size
/	0	-	-	28,572	drwxrwxrwx	934.5 MB
bin	0	3,4	-	83	drwxr-xr-x	5.3 MB
boot	0	-	-	0	drwxr-xr-x	0
dev	0	-	-	0	drwxr-xr-x	0

## #2 Prefer Leaner Docker Images

```
FROM node:latest  
  
WORKDIR /usr/src/app  
  
COPY package*.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 8080  
CMD [ "node", "server.js" ]
```



```
FROM node:lts-slim  
  
WORKDIR /usr/src/app  
  
COPY package*.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 8080  
CMD [ "node", "server.js" ]
```



# Downsides of Bigger Image

- Larger Download Size
- Higher Exposure to Vulnerabilities
- Increases Resource Consumption

Images on disk

Last refresh: Never    14 images    Refresh to see disk size    Clean up

Images [Give Feedback](#)

LOCAL    REMOTE REPOSITORIES

node

In use only

NAME ↑	TAG	IMAGE ID	CREATED	SIZE
node	latest	ebe3af0adc87	about 10 hours ago	940.68 MB

```
services:  
  
  llm: &llm  
    image: ollama/ollama:latest  
    profiles: ["linux"]  
    networks:  
      - net  
  
  llm-gpu:  
    <<: *llm  
    profiles: ["linux-gpu"]  
    deploy:  
      resources:  
        reservations:  
          devices:  
            - driver: nvidia  
              count: all  
              capabilities: [gpu]
```

```
$ docker compose -profile linux up
```

```
$ docker compose -profile linux-gpu up
```

## The Power of Compose Profile

# #3

## Use Multi-Stage Build

```
FROM node:lts-buster-slim AS builder
WORKDIR /usr/src/app

COPY package.json /usr/src/app/package.json
COPY package-lock.json /usr/src/app/package-lock.json
RUN npm ci

COPY . /usr/src/app

EXPOSE 3000

CMD [ "npm", "run", "dev" ]

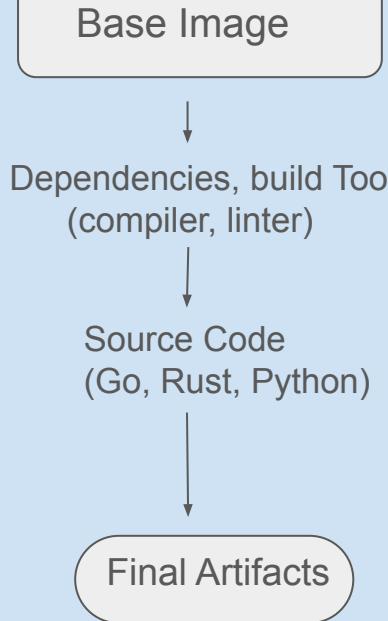
FROM builder as dev-envs
RUN <<EOF
apt-get update
apt-get install -y --no-install-recommends git
EOF

# install Docker tools (cli, buildx, compose)
COPY --from=gloursdocker/docker / /
```

BUILD STAGE

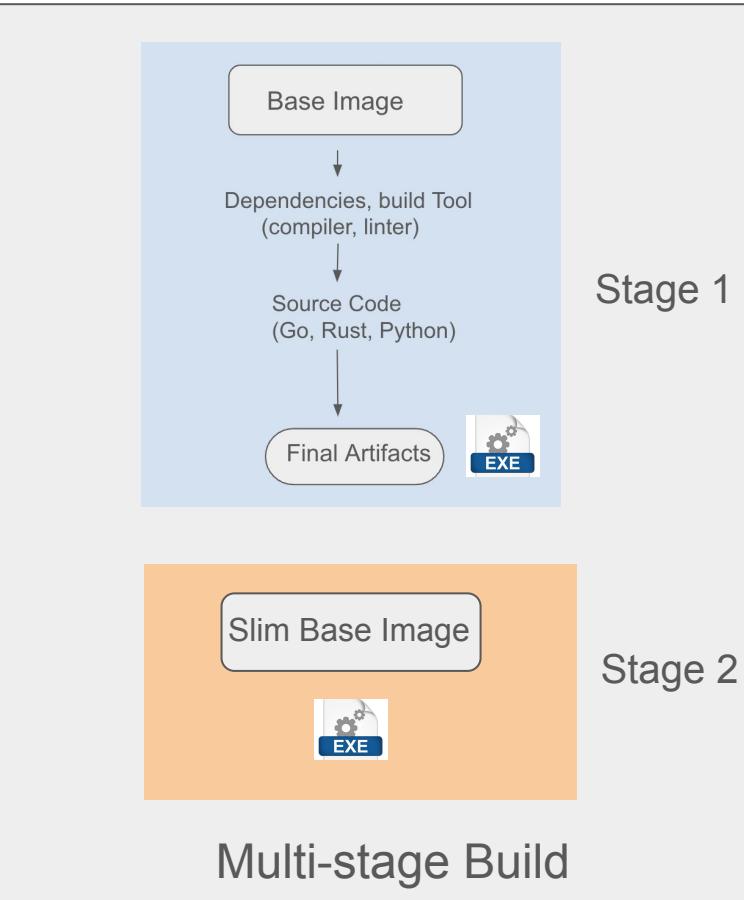
DEVELOPMENT STAGE





Traditional Build

Gigs



#4

# Quickly identify and fix vulnerabilities during the build time using Scout

The screenshot shows the Scout application interface. At the top, there are three tabs: 'Images (6)', 'Vulnerabilities (321)', and 'Packages (6)'. The 'Vulnerabilities' tab is selected, indicated by a blue underline.

The main table displays vulnerabilities across different packages:

Package	Vulnerabilities
> ubuntu/pcre2 10.34-7	3 C 0 H
> ubuntu/libksba 1.3.5-2	2 C 0 H
> ubuntu/curl 7.68	accounts-api
> ubuntu/zlib 1:1.2	Tags OS Vulnerabilities
> ubuntu/pam 1.3.	Latest Linux 3 C 6 H 12 M 11 L
> ubuntu/krb5 1.17	1.7.0 Linux 3 C 6 H 12 M 11 L
	1.6.0 Linux 3 C 6 H 12 M 11 L

A modal window is open over the table, centered on the 'curl' package. The modal has a dark background and contains the package name 'accounts-api' and a 'Compare images' button. It also includes tabs for 'Tags', 'OS', and 'Vulnerabilities'. Under 'Vulnerabilities', it shows four categories: Critical (3 C), High (6 H), Medium (12 M), and Low (11 L). Below these categories, there are two rows of vulnerabilities for the 'curl' package, each with a checkbox, a tag icon, a version number, an OS type (Linux), and the same set of vulnerability counts.

# #5

## Add HEALTHCHECK

```
FROM node:lts-buster

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080
HEALTHCHECK CMD curl --fail
http://localhost:3000 || exit 1
CMD [ "node", "server.js" ]
```

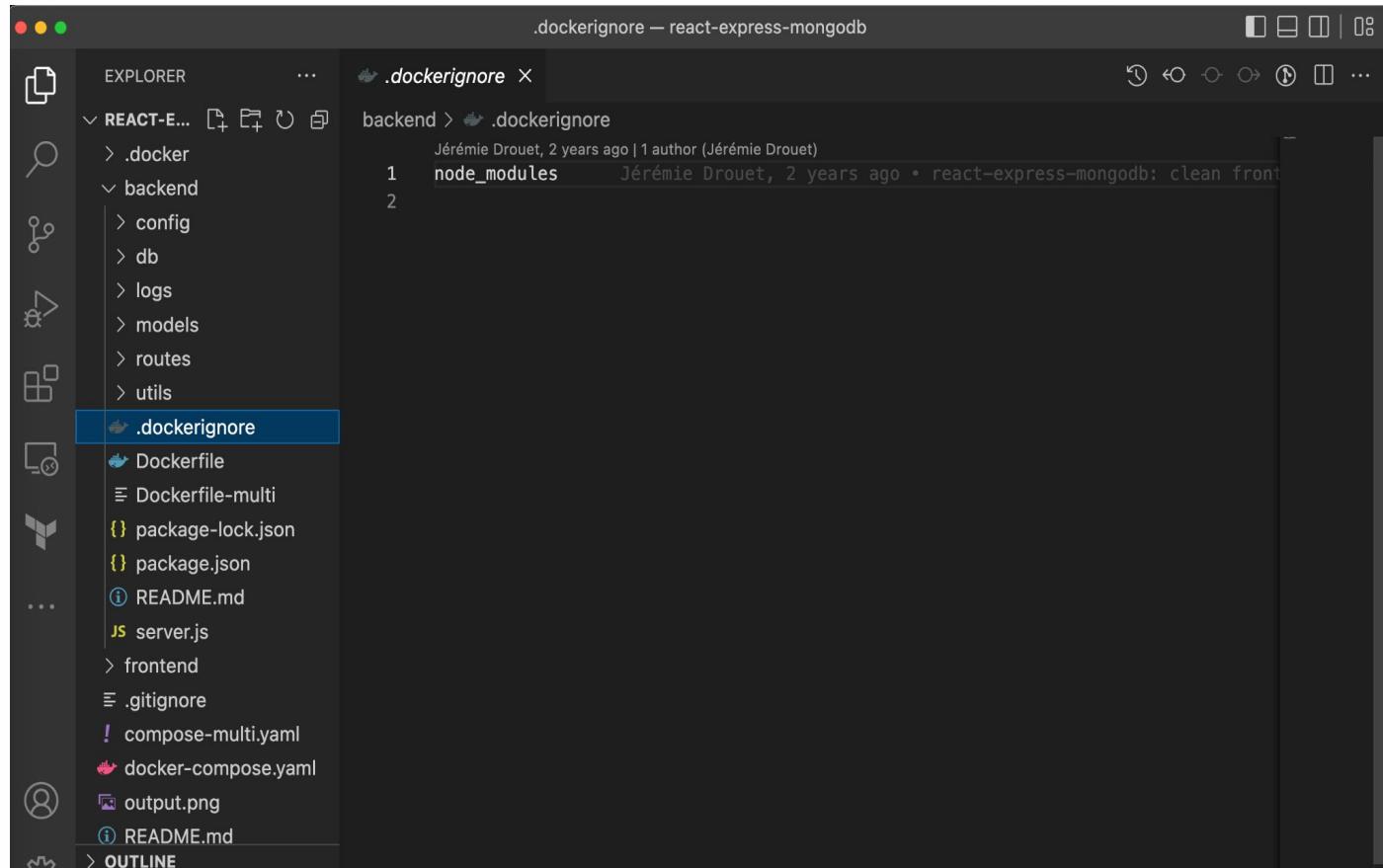
# #5

## Add HEALTHCHECK

```
docker ps
CONTAINER ID      IMAGE
COMMAND           CREATED          STATUS
PORTS             NAMES
1d0c5e3e7d6a    react-express-mongodb-frontend
"docker-entrypoint.s..."   23 seconds ago   Up 21
seconds (health: starting)
0.0.0.0:3000->3000/tcp   frontend
```

#6

# Use .dockerignore



#7

## Run as non-root user for security purpose

```
FROM node:lts-buster AS development
WORKDIR /usr/src/app
COPY package.json /usr/src/app
COPY package-lock.json /usr/src/app
RUN npm ci
COPY . /usr/src/app
EXPOSE 3000
CMD ["npm", "start"]

FROM development as dev-envs
RUN <<EOF
apt-get update
apt-get install -y --no-install-recommends git
EOF

RUN <<EOF
useradd -s /bin/bash -m vscode
groupadd docker
usermod -aG docker vscode
EOF

# install Docker tools (cli, buildx, compose)
COPY --from=gloursdocker/docker / /
```

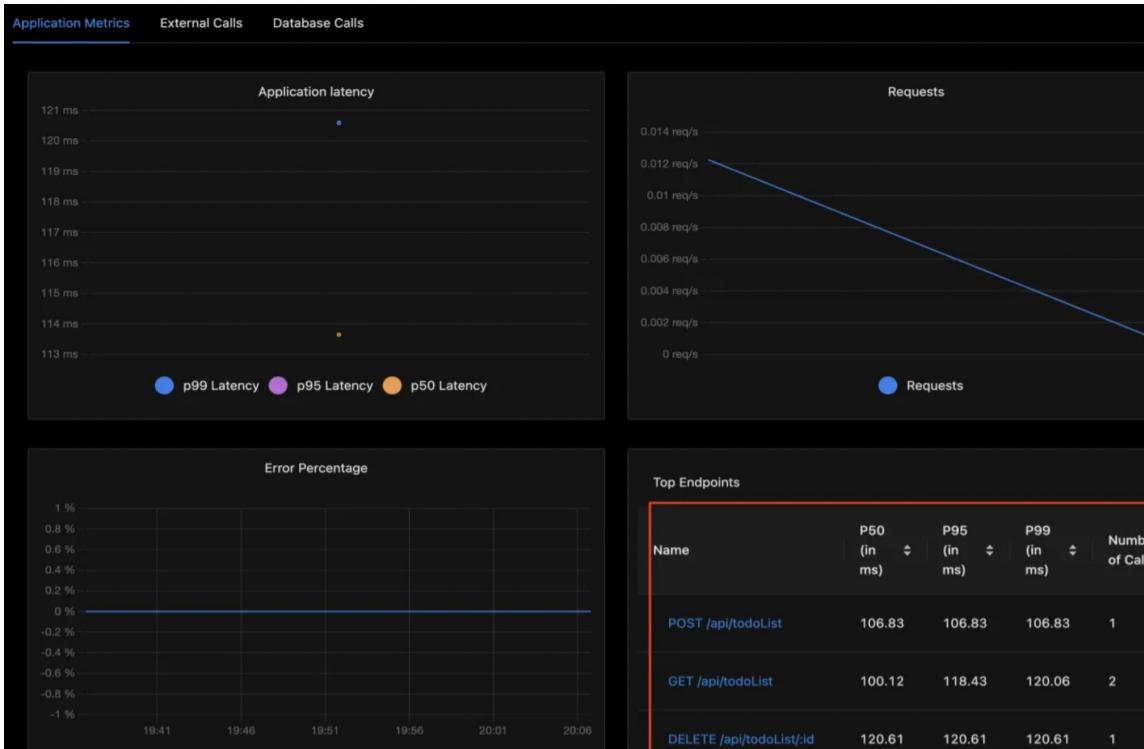
#8

## Favour Multi-Architecture Docker Images

```
docker run --rm mplatform/mquery node:lts-buster
Unable to find image 'mplatform/mquery:latest'
locally
d0989420b6f0: Download complete
af74e063fc6e: Download complete
3441ed415baf: Download complete
a0c6ee298a93: Download complete
894bcacb16df: Downloading
[======>
] 3.146MB/3.452MB
Image: node:lts-buster (digest:
sha256:a5d9200d3b8c17f0f3d7717034a9c215015b7aae70cb2
a9d5e5dae7ff8aa6ca8)
 * Manifest List: Yes (Image type:
application/vnd.docker.distribution.manifest.list.v2
+json)
 * Supported platforms:
 - linux/amd64
 - linux/arm/v7
 - linux/arm64/v8
```

# #9 Use Open Telemetry API to measure App performance

- Metrics(what)
- Logs(why)
- Trace(how)



#10

## Using HEALTHCHECK in Docker Compose

```
healthcheck:  
  test: ["CMD", "curl", "-f",  
"http://localhost"]  
  interval: 1m30s  
  timeout: 10s  
  retries: 3  
  start_period: 40s
```

# Best Practices – A Developers' Love-Hate Relationship



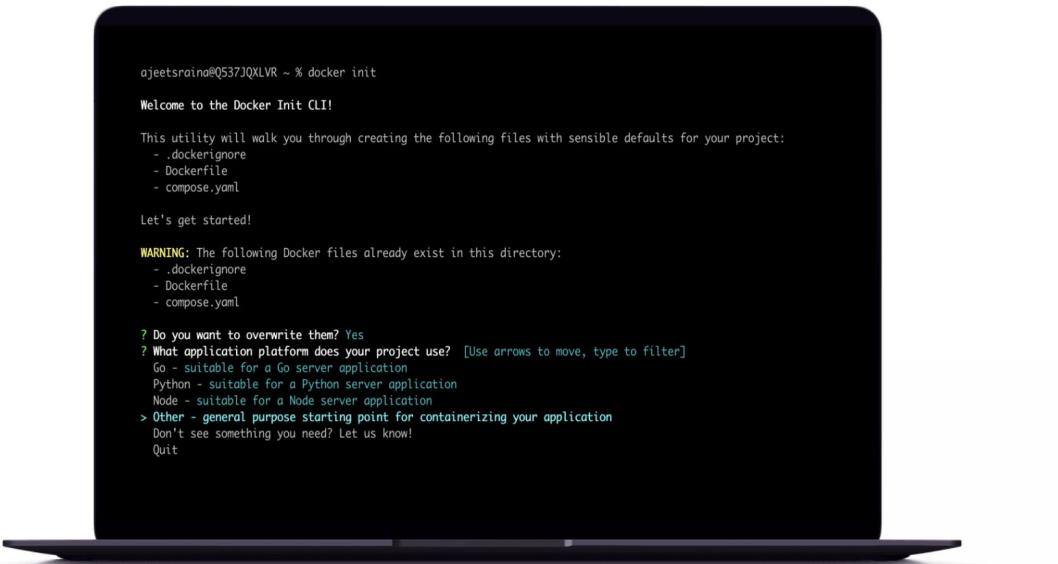
# Introducing



## Docker Init

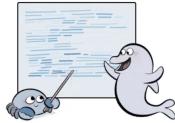
# \$ docker init

- Introduced for the first time in [Docker Desktop 4.18](#)
- Generates Docker assets for projects
- Allows you to choose application platform
- Makes it easier to create Docker images and containers



# \$ docker init

Simplified Docker Assets Creation



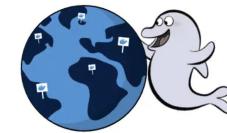
Saves Time and Effort



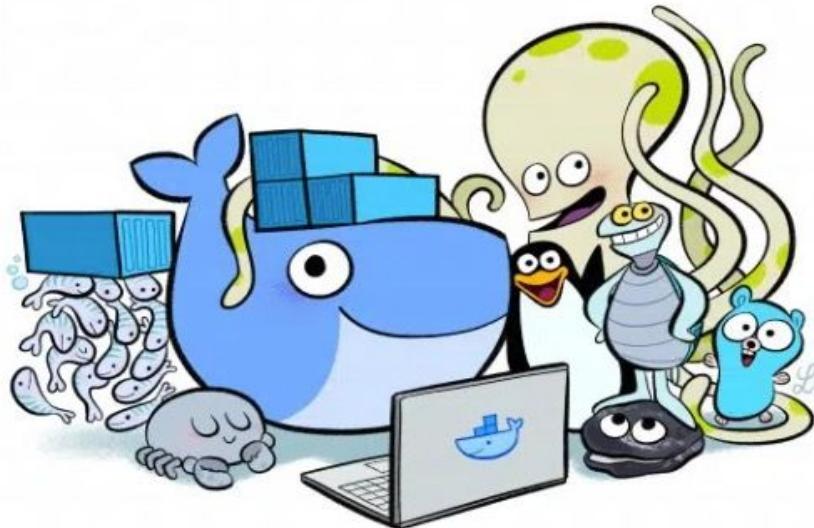
Better Project Organization



Enhanced Portability



# It's Demo Time



<https://dockerworkshop.vercel.app/lab1/docker-init/>

# Compose Watch



COMPOSE  
FILE WATCH



# Compose Watch

- Automatically updates your compose service containers while you work
- Blazing-fast file synchronization supporting live update

## How it works?

- Automatically builds a new image with BuildKit and replaces the running service container
- Add a develop section to your services in the compose.yaml file
- Configure it with a list of paths to watch and actions to take
- Watch rules allow ignoring specific files or entire directories within the watched tree.

```
services:  
  web:  
    build: .  
    command: npm start  
  x-develop:  
    watch:  
      - action: sync  
        path: ./web  
        target: /src/web  
        ignore:  
          - node_modules/  
      - action: rebuild  
        path: package.json
```

# Compose Watch Actions

## Sync

Specifies a path to watch for changes in the host file system, and a corresponding target path inside the container to synchronize changes to.

## Rebuild

The "rebuild" action specifies a path to watch for changes in the host file system, and triggers a rebuild of the container when changes are detected.

## sync+restart

The "sync+restart" action specifies a path to watch for changes in the host file system, and a corresponding target path inside a container to first synchronize changes to and then restart the container

<https://dockerworkshop.vercel.app/>

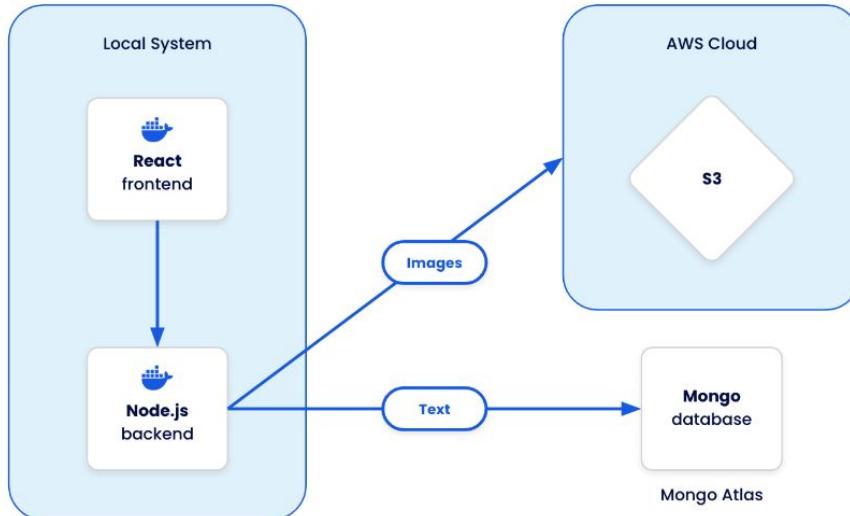


# **Container-First Development Workflow**



# Container-first Development Workflow

Using containers for every aspect of software development, including the application runtime itself.

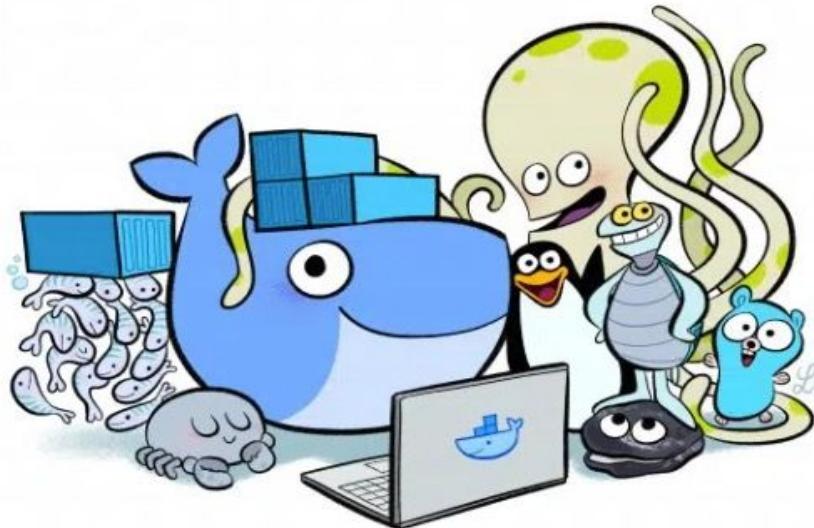


# Container-first Development Workflow

## Benefits for Developers:

- **Extreme Portability:** Developers only need a container engine and an IDE to work on the project. This ensures identical development environments regardless of the underlying operating system or pre-installed software.
- **Faster Setup:** No time wasted installing the application runtime or fiddling with local configurations. Developers can start coding as soon as the containerized environment is up.
- **Improved Isolation:** Each project runs within its own isolated container, preventing conflicts between projects or dependencies from interfering with other applications on the developer's machine.
- **Simplified Collaboration:** Team members can easily share and reproduce development environments using the same container images.

# It's Demo Time



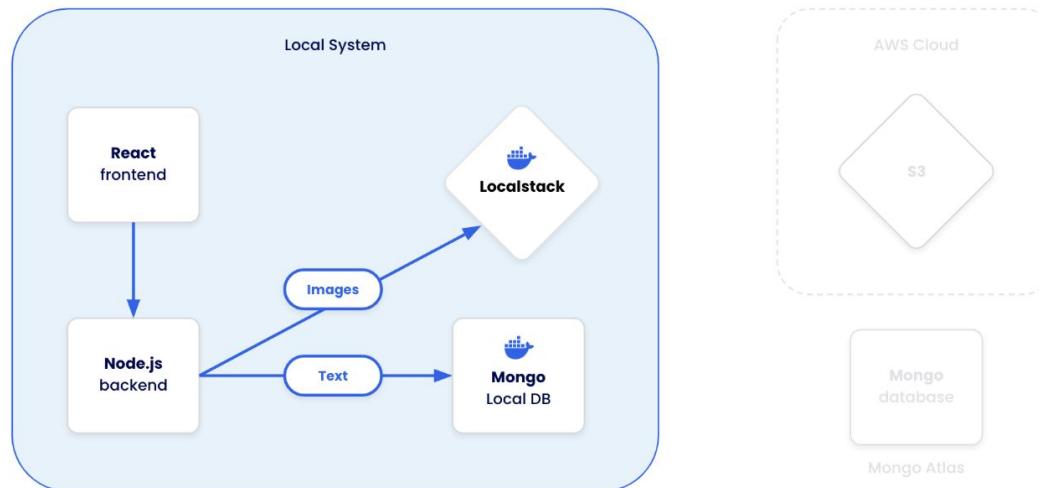
<https://dockerworkshop.vercel.app/lab2/services/>

# **Container-supported Development Workflow**



# Container-first Development Workflow

Using containers to support and enhance development without touching the main application runtime itself.

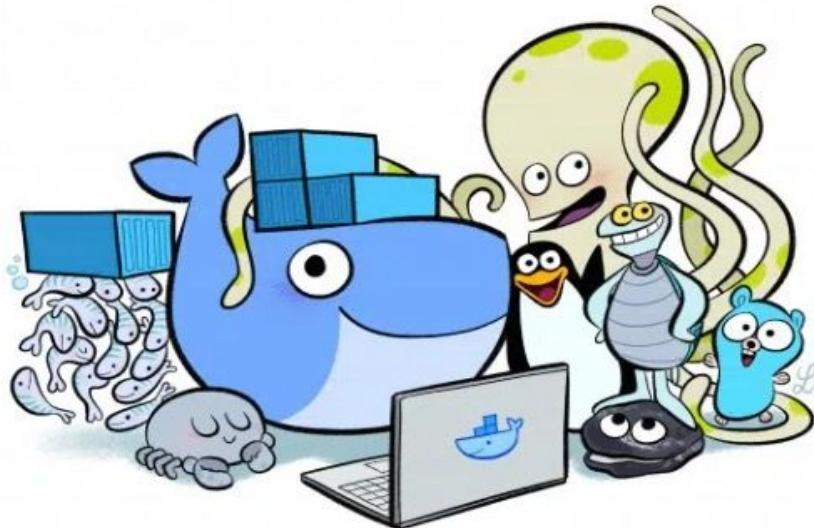


# Container-supported Development Workflow

## Benefits for Developers:

- **Separation of Concerns:** Developers focus on the core application logic using their familiar runtime (JVM, Node, Python etc.), while external dependencies are isolated in containers.
- **Improved Efficiency:**
- Easier setup: Containers simplify dependency management, reducing time spent configuring environments. Version consistency: All developers use the same container image, ensuring consistent dependencies across the team.
- Enhanced Capabilities: Docker allows running local simulations of cloud services and real services within tests, providing a more realistic development environment.
- **Flexibility in Implementation:** There's no single approach. Teams can use:
  - Wrapper scripts for simple container execution.
  - IDE plugins for launching development environments defined in Docker Compose files.
  - Programming interactions with libraries like Testcontainers.
- **Leveraging Shared Resources:** Teams can benefit from pre-built container images from public repositories like Docker Official Images (DOI) and Docker Verified Publishers (DVP).

# It's Demo Time



<https://dockerworkshop.vercel.app/lab3/services/>

# Wrapping up



# References

- Container-supported Development Workflow -  
<https://docs.docker.com/guides/container-supported-development/>
- Docker init Demo - <https://github.com/dockersamples/docker-init-demos>
- Docker Best Practices for NodeJS -  
<https://www.docker.com/blog/9-tips-for-containerizing-your-node-js-application/>
- Docker Best Practices for .NET -  
<https://www.docker.com/blog/9-tips-for-containerizing-your-net-application/>
- Compose File Watch - <https://github.com/dockersamples/avatars>





**Thank you**  
See you again soon.

