



Improve Your Inner-loop development Workflow and Productivity using Docker

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 less than \$10 million in annual revenue. Docker Desktop is not available for organizations with more than 50 employees OR more than \$10 million in annual revenue (e.g., Government, or Business).

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



Access the Workshop

```
$ docker extension install \
```

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

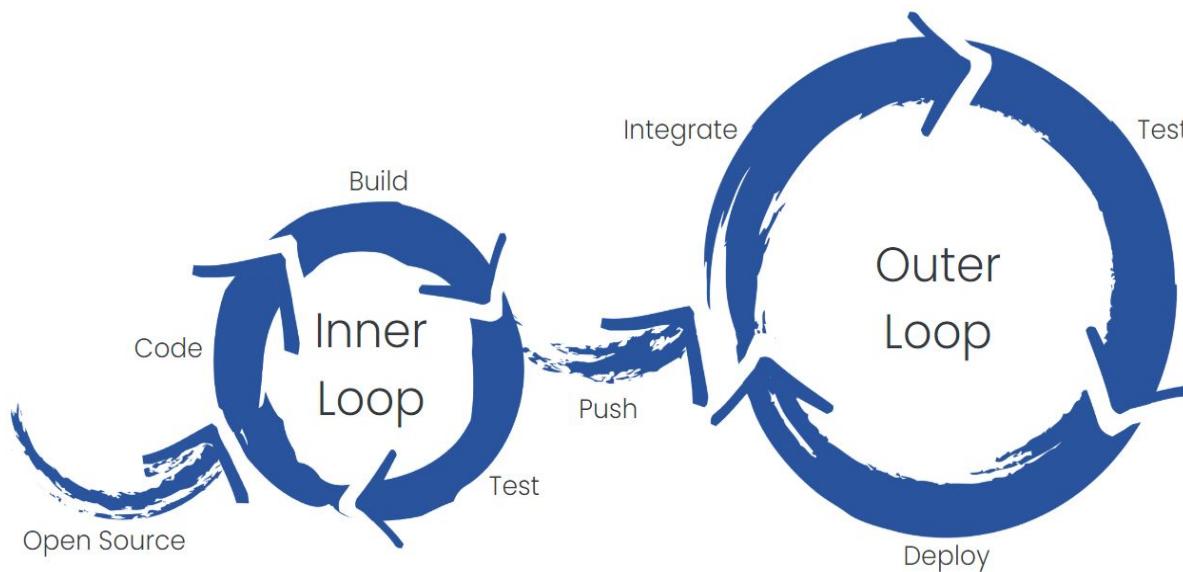


Access the Workshop

<https://wad2024-workshop.vercel.app/>



Docker is Uniquely Focused on Developer Success



Trusted Images

Docker Desktop

Docker Ecosystem

Delivery Platforms

20M+ Active Developers

450+ Trusted Partners

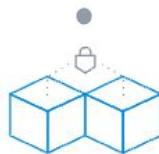


Built for Developers, by Developers



Speed

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



Security

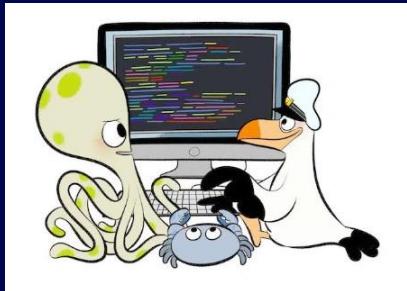
- Docker Scout
- Attestations



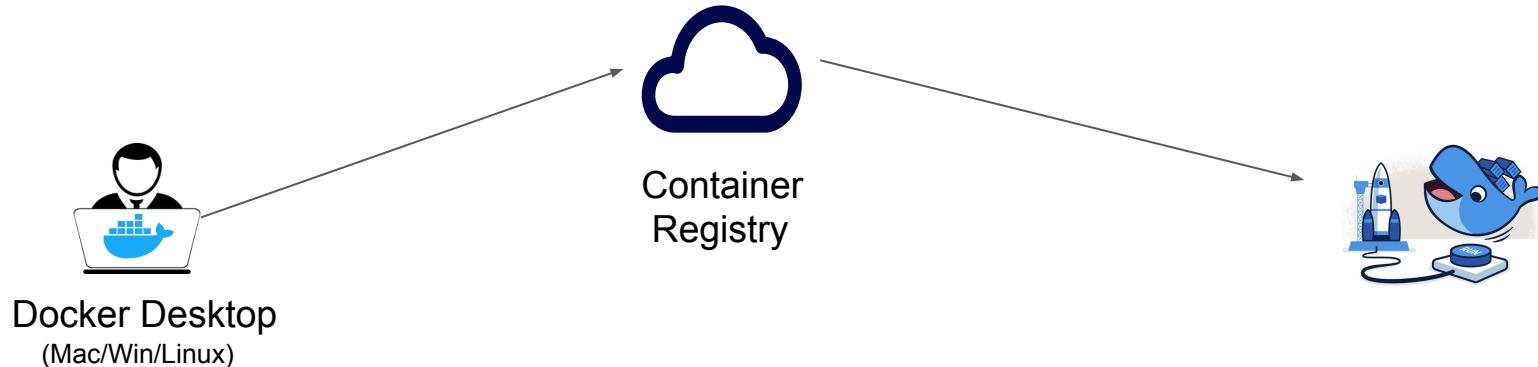
Choice

- 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
```

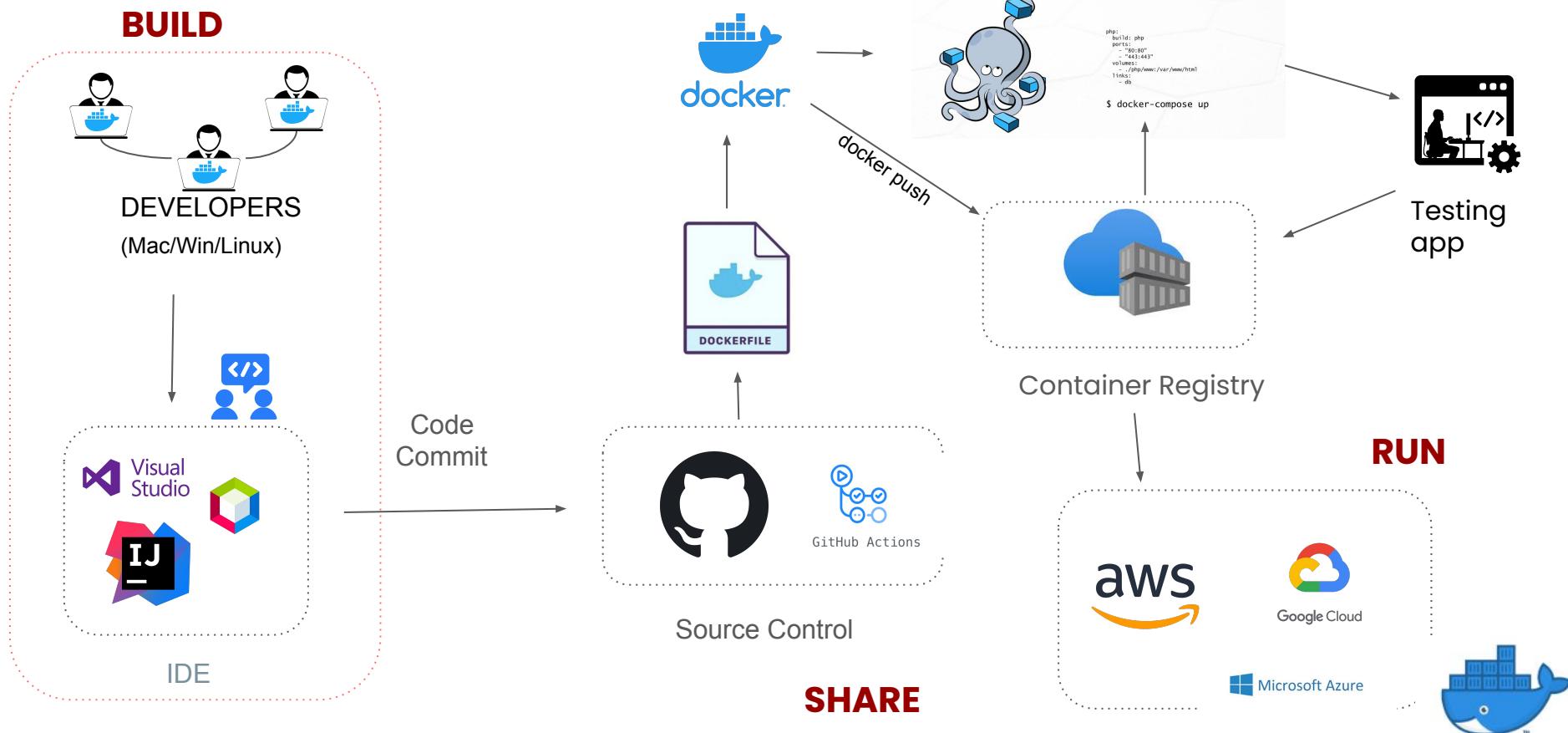
RUN

- 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 and Deploy a Simple Node Application



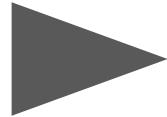
```
git clone https://github.com/.../..
cd [path to your node-docker directory]
npm init -y
npm install
touch server.js
node server.js

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

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

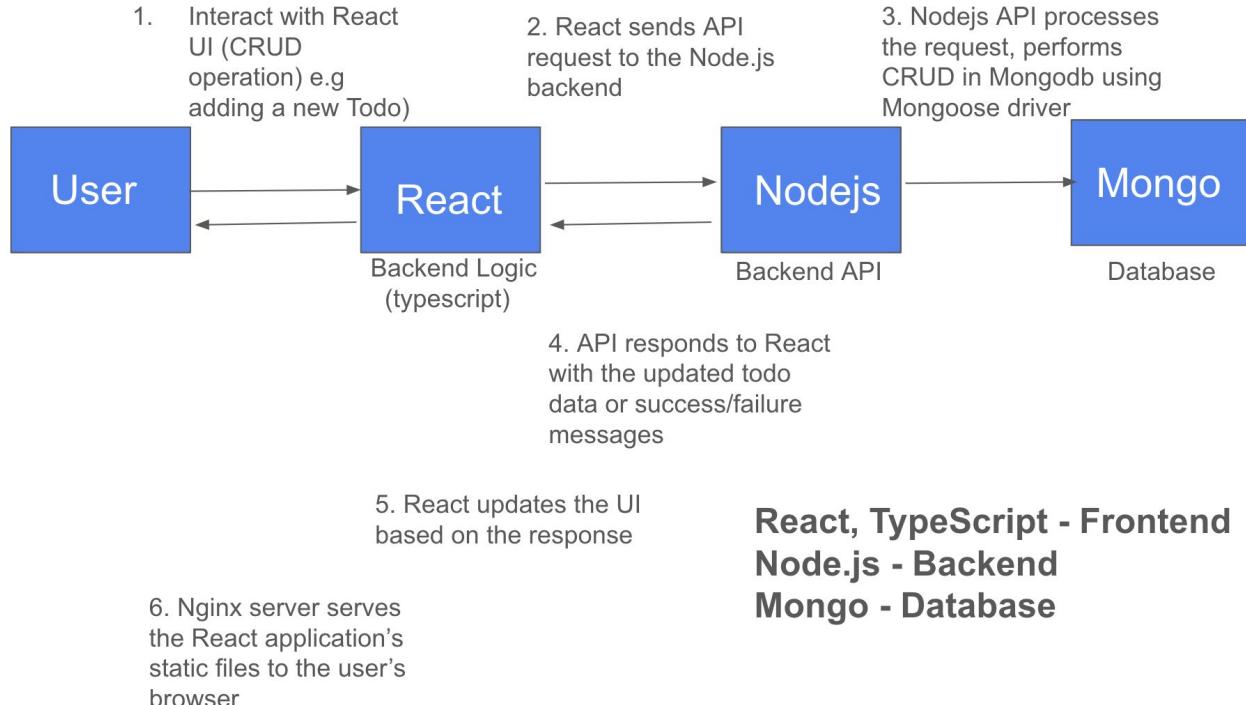
COPY . .

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

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

A Simple to-do List app

(React/TypeScript + Node.js + Mongo)



Do's

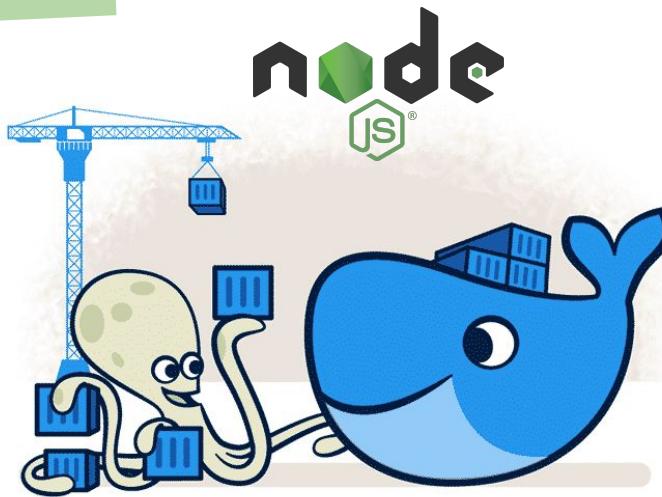
✓

- 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 is a sidebar with options: Containers, Images, Volumes, Dev Environments (BETA), Extensions (BETA), Disk usage, slim.ai, and Add Extensions. The main area is titled "Docker Desktop" and shows a search bar with "node". Below it, a card for "Docker Hub Public node latest" is displayed, with "Explore" and "Compare" buttons. The "FILE EXPLORER" tab is selected, showing "Instructions for All 9 Layers" and "Container Startup Files". At the bottom, a table titled "File Changes for All 9 La..." lists file changes across 32,193 objects. The table includes columns for Name, Added, Modified, Deleted, Count, Mode, and Size. The "Count" column is highlighted in blue. The table shows entries for /, bin, boot, and dev. The bottom status bar indicates RAM 3.57GB, CPU 33.73%, and "Connected to Hub". The version v4.12.0 is also visible.

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



node:alpine3.16 ← 49 MB

node:lts-slim ← 62 MB

node:16.17 ← 333 MB

node:16.17.0-slim ← 62 MB



#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



#4

Quickly identify and fix vulnerabilities during the build time using Scout

Scout interface showing vulnerabilities for Docker images:

Images (6) Vulnerabilities (321) Packages (6)

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

Compare images



#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

The screenshot shows a dark-themed instance of Visual Studio Code. In the Explorer sidebar, a folder named 'REACT-E...' is expanded, revealing subfolders like '.docker' and 'backend'. Inside 'backend', there are several files and folders: '.dockerignore' (which is selected and highlighted in blue), 'Dockerfile', 'Dockerfile-multi', 'package-lock.json', 'package.json', 'README.md', 'server.js', 'frontend', '.gitignore', 'compose-multi.yaml', 'docker-compose.yaml', 'output.png', and 'README.md'. The 'backend' folder is also listed in the main editor area under the heading 'backend > .dockerignore'. The editor tab displays the contents of the '.dockerignore' file, which contains two lines: 'node_modules' and '2'. The status bar at the bottom right indicates the file is part of a 'react-express-mongodb' project.



#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

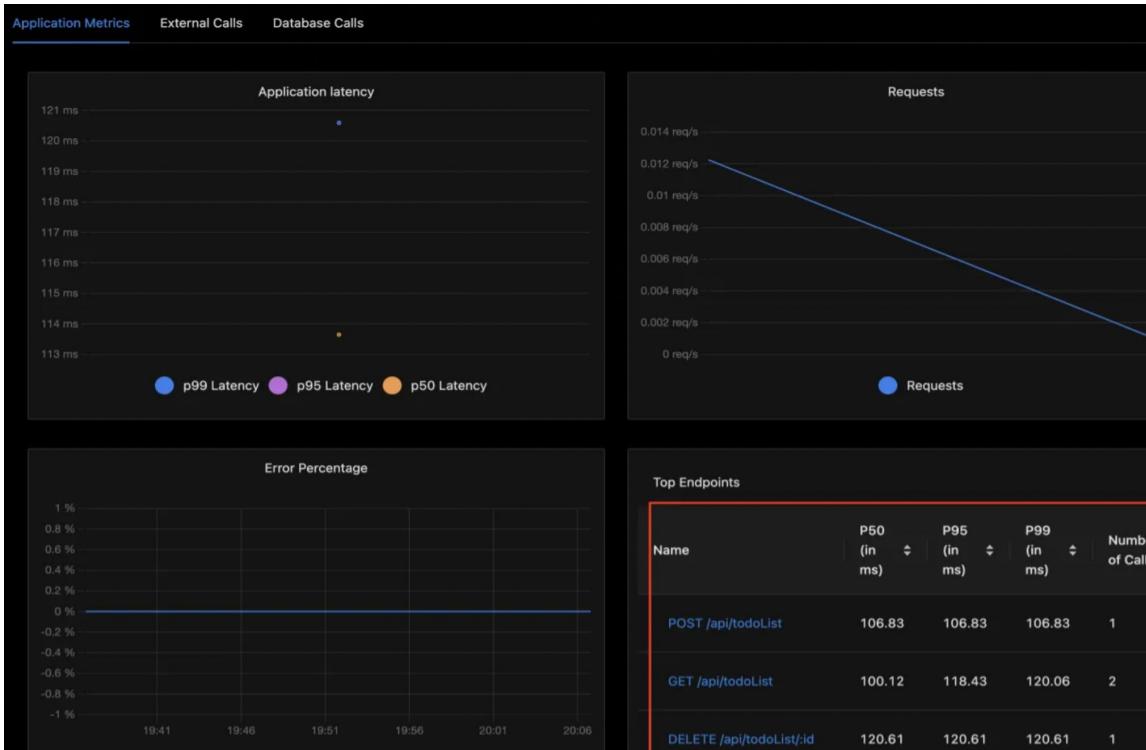
Favour Multi-Architecture Docker Images

```
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

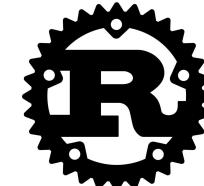
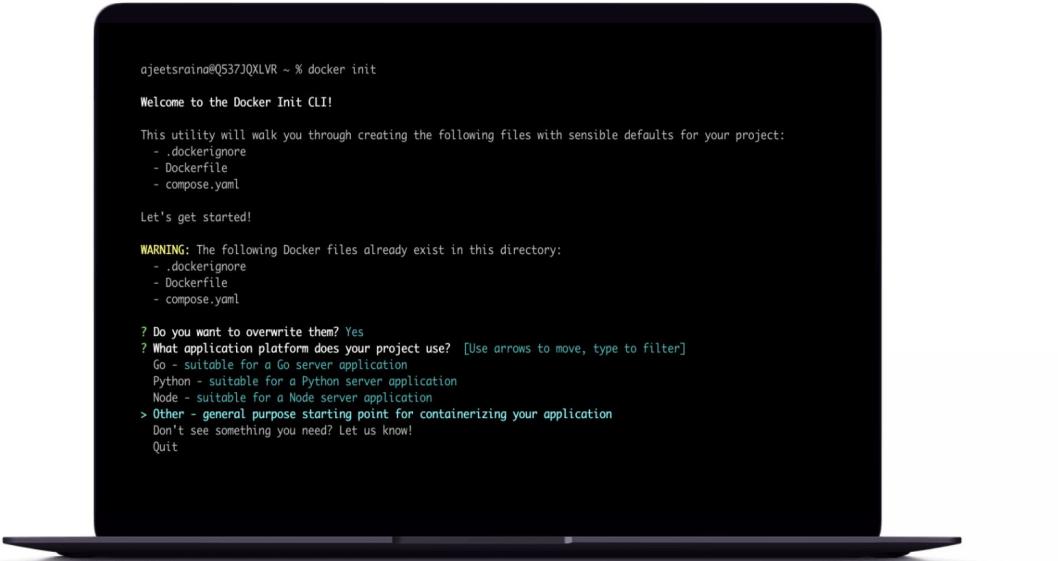


Docker Init



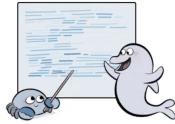
\$ docker init

- 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

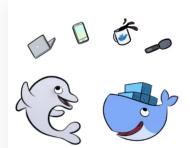
Simplified Docker Assets Creation



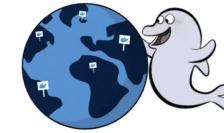
Saves Time and Effort



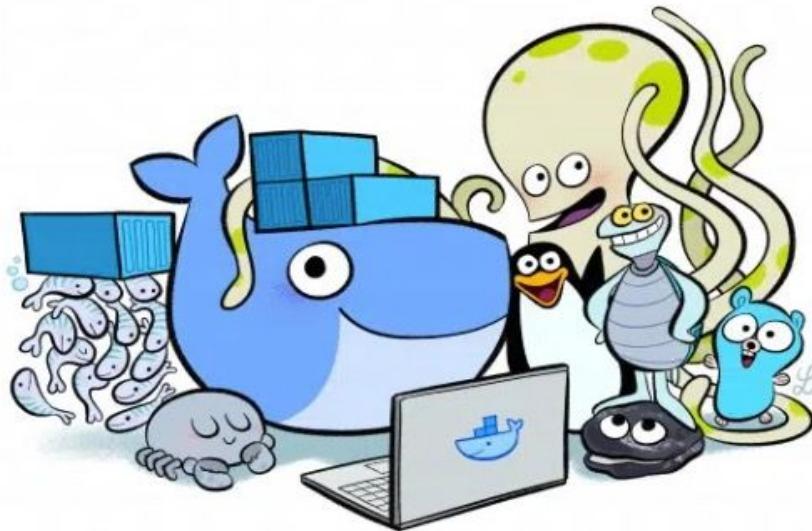
Better Project Organization



Enhanced Portability



It's Demo Time



<https://github.com/dockersamples/docker-init-demos>



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://wad2024-workshop.vercel.app/>

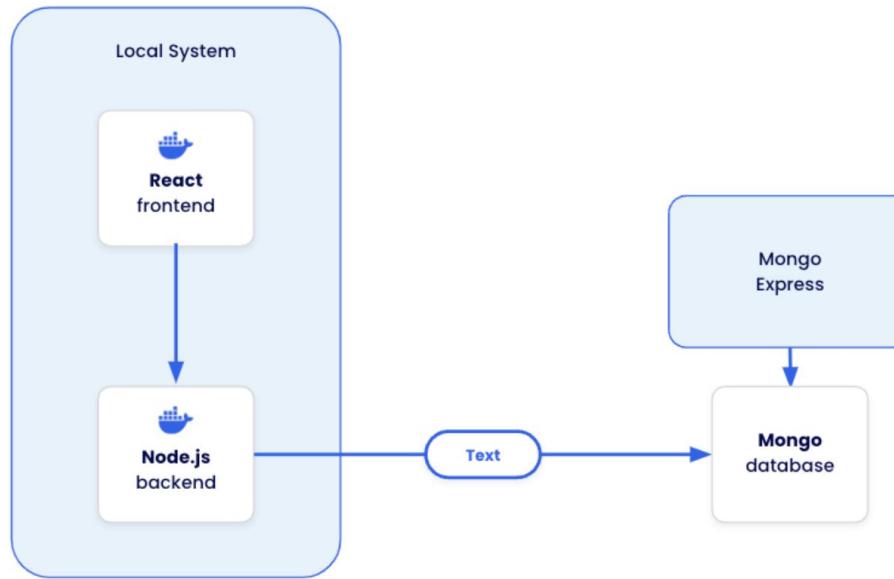


Container-First Development Workflow



Container-first Development Workflow

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



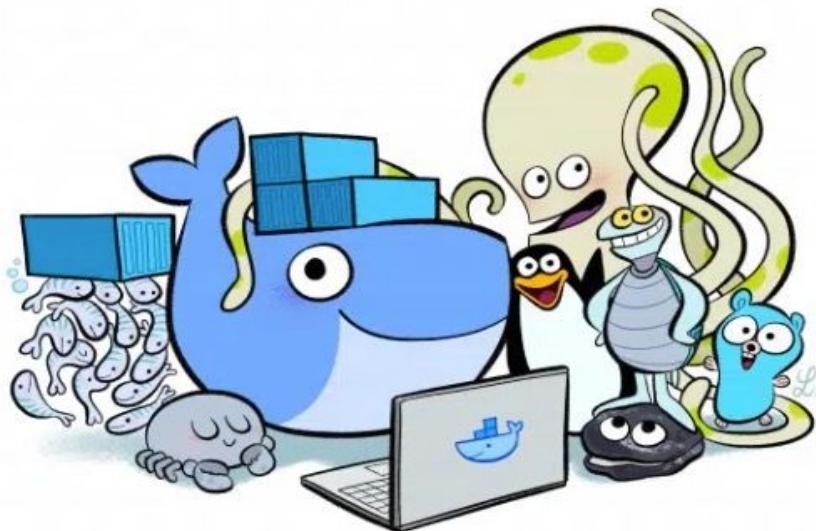
Container-first Development Workflow

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://wad2024-workshop.vercel.app/lab2/services/>

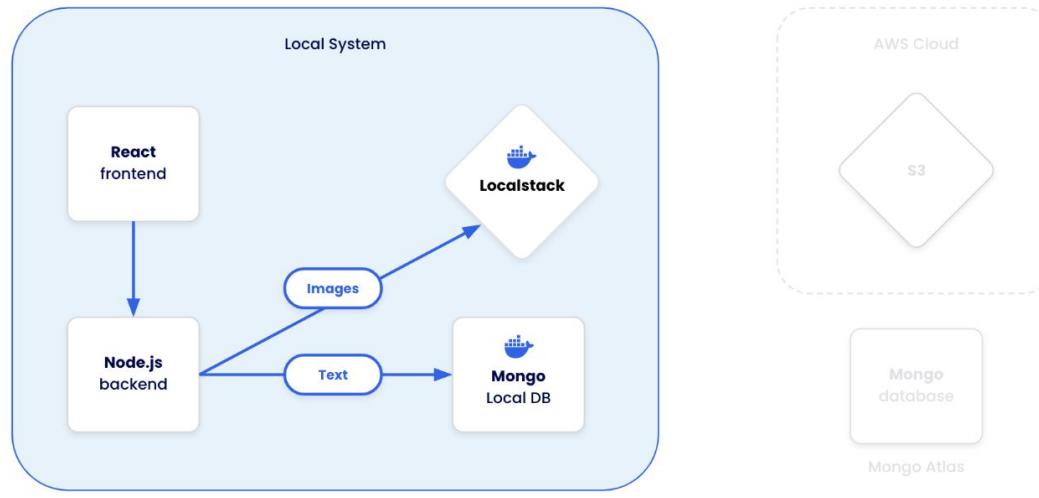


Container-supported Development Workflow



Container-first Development Workflow

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



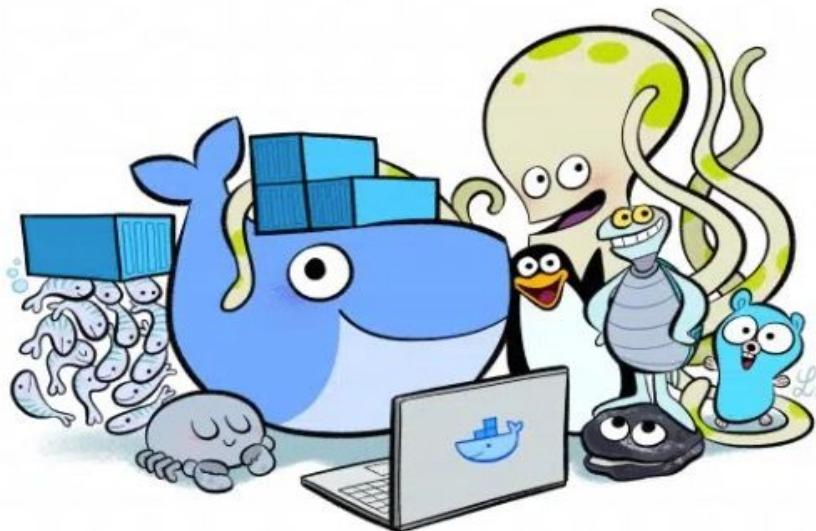
Container-supported Development Workflow

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://wad2024-workshop.vercel.app/lab3/services/>



Wrapping up





Thank you
See you again soon.

