Docker Fundamentals

Containers for DevOps

Prakhar Jain - Teaching Assistant

IIIT Hyderabad

Introduction

What is Docker?

- Open-source containerization platform
- Packages application, runtime, libraries, and dependencies
- Ensures consistency across environments
- Core technology in modern DevOps

Why Docker Exists

- *"Works on my machine"* problem
- Dependency conflicts
- Slow VM-based deployments
- Poor scalability

Virtualization vs Containers

Virtual Machines vs Containers

	Virtual Machines	Containers
Guest OS	Required	Not required
Startup time	Minutes	Seconds
Resource usage	High	Low
Isolation	Strong	Process-level

Docker Architecture

Docker Architecture

- Docker Client
- Docker Daemon (`dockerd`)
- Docker Images
- Docker Containers
- Docker Registries

Docker Images

What is a Docker Image?

- Immutable blueprint for containers
- Built using layered filesystem
- Versioned using tags
- Stored locally or in registries

Dockerfile Example

```
FROM python:3.10
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Image Characteristics

- Read-only
- Cached layers
- Reusable across projects

Docker Containers

What is a Container?

- Running instance of an image
- Isolated process space
- Shares host kernel
- Ephemeral by design

Basic Docker Commands

```
docker run nginx
docker ps
docker ps -a
docker stop <container_id>
docker rm <container_id>
```

Docker Networking

Docker Networking Modes

- Bridge (default)
- Host
- None
- User-defined networks

Port Mapping

```
docker run -p 8080:80 nginx
```

Docker Volumes

Why Docker Volumes?

- Containers are ephemeral
- Data must persist
- Volumes survive restarts

Volume Example

```
docker volume create appdata
docker run -v appdata:/data busybox
```

Docker Compose

Why Docker Compose?

- Multi-container applications
- Declarative YAML configuration
- Single-command startup

docker-compose.yml Example

```yaml
version: "3.9"
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: root
```

Docker in DevOps

Docker in CI/CD

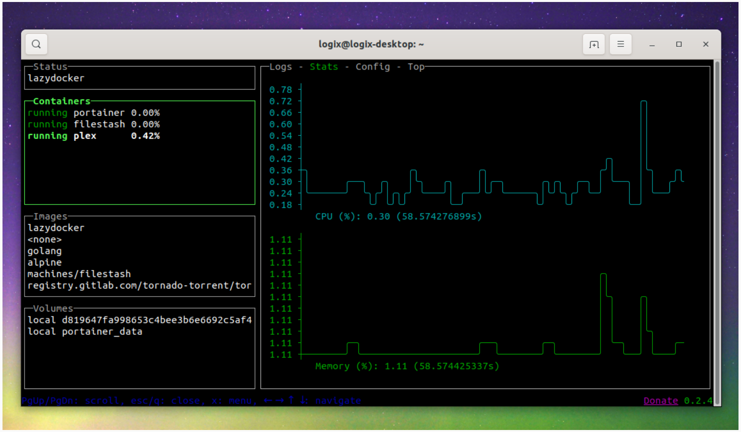
- Reproducible builds
- Isolated test environments
- Standardized deployment artifacts

Lazydocker

What is Lazydocker?

- Terminal-based Docker UI
- Written in Go
- Inspired by lazygit
- Improves developer productivity

Lazydocker User Interface



```
Status
lazydocker

Containers
running portainer 0.00%
running filestash 0.00%
running plex      0.42%

Images
lazydocker
<none>
golang
alpine
machines/filestash
registry.gitlab.com/tornado-torrent/tor

Volumes
local d819647fa998653c4bee3b6e6692c5af4
local portainer_data
```

```
Logs - Stats - Config - Top

CPU (%): 0.30 (58.574276899s)

Memory (%): 1.11 (58.574425337s)

PgUp/PgDn: scroll, esc/q: close, x: menu, ← → ↑ ↓: navigate          Donate 0.2.4
```

Why Use Lazydocker?

- Avoids long Docker commands
- Fast inspection of containers
- Easier debugging
- Excellent for local development

Installing Lazydocker

```
# Homebrew (macOS / Linux)
brew install lazydocker

# Linux install script
curl https://raw.githubusercontent.com/jesseduffield/
    lazydocker/master/scripts/install_update_linux.sh
    | bash
```

Running Lazydocker

```
lazydocker
```

What Lazydocker Can Manage

- Containers
- Images
- Volumes
- Networks
- Docker Compose services

When Lazydocker is NOT Appropriate

- Production servers
- Automated CI/CD pipelines
- Headless environments

Best Practices and Security

Docker Best Practices

- Use minimal base images
- One process per container
- Avoid running as root
- Use `.dockerignore`

Security Considerations

- Scan images for vulnerabilities
- Drop unnecessary privileges
- Keep Docker updated

Conclusion

Summary

- Docker simplifies deployment
- Containers improve consistency
- Lazydocker improves developer productivity