Introduction to Docker and Containerization

What is Containerization?

Lightweight virtualization

Runs applications in containers

Shares host OS kernel

Packages code, dependencies, settings

Why Containers?

Consistency: Solves 'works on my machine'

Portability: Run anywhere

Efficiency: Faster, low resource

Isolation: Self-contained environments

What is Docker?

Open platform for containerized apps

Separates apps from infrastructure

Written in Go

Launched: March 20, 2013

Key Docker Concepts

Docker Engine: Build and run containers

Image: Read-only snapshot

Container: Running image

Dockerfile: Build script

Docker Hub: Public image registry

Why Docker Over VMs?

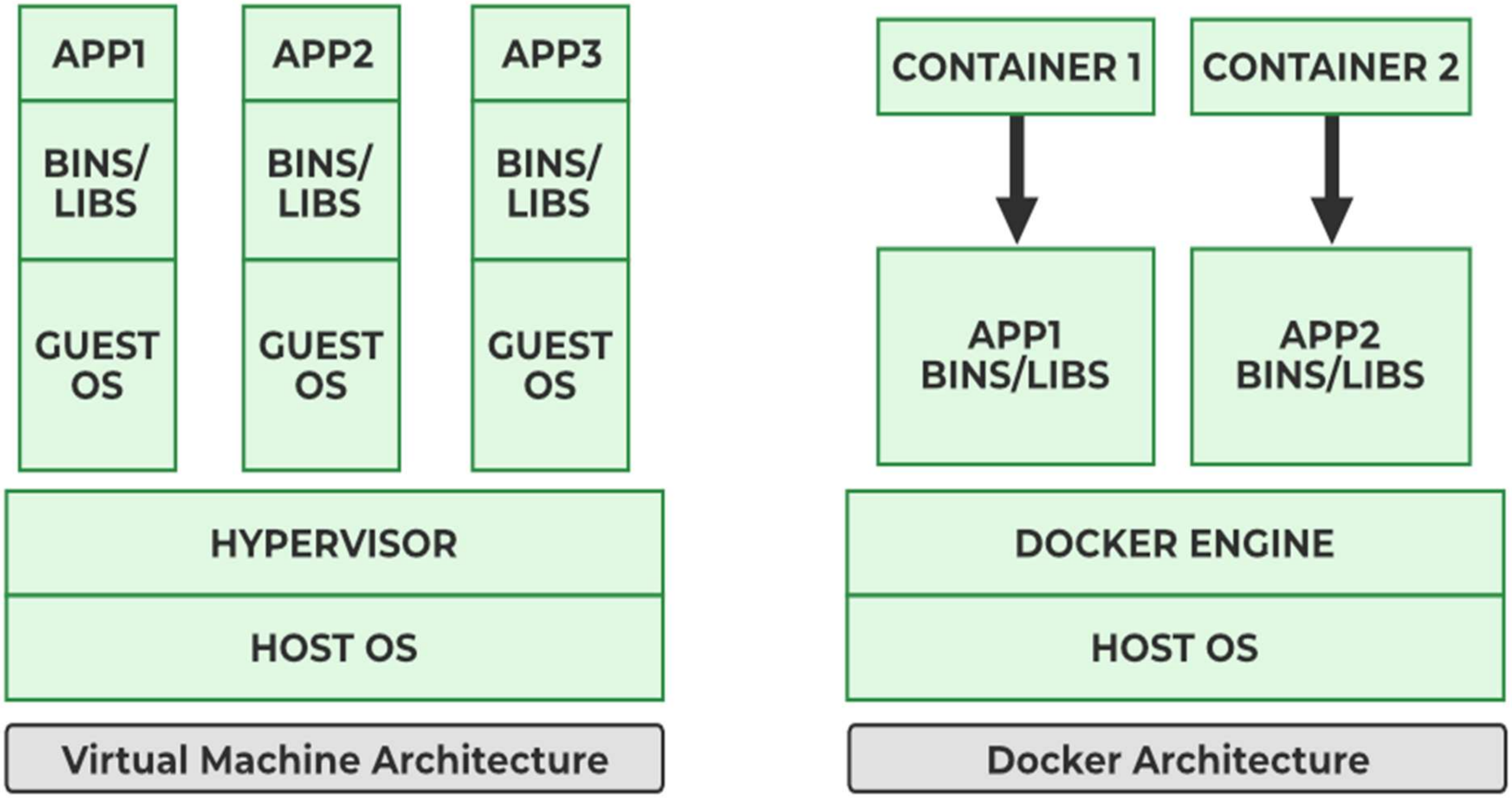
VMs are heavy, containers are light

Containers start in seconds

Easier to scale

Higher portability and efficiency

Docker vs. Virtual Machines



Docker Architecture

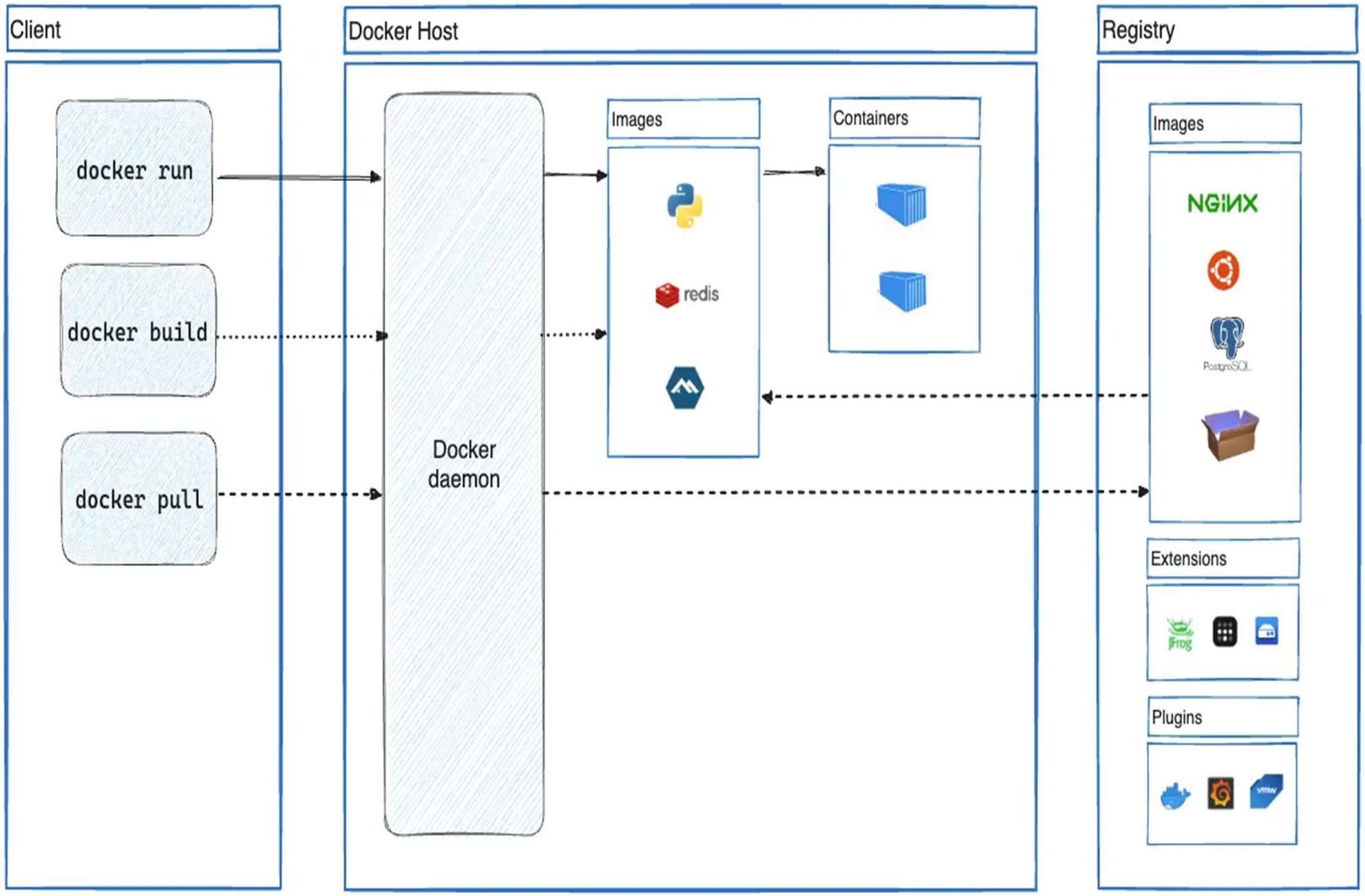
Docker Engine: Client, Daemon, REST API

Image: Template

Container: Executable image

Registry: Stores images

Volume: Persistent storage


Client
Docker Host
Registry
docker run
docker build
docker pull
Docker daemon
Images
Containers
redis
Images
NGINX
PostgreSQL
Extensions
JFrog
Plugins
vmw

Impact of Docker

Streamlines DevOps (CI/CD)

Supports microservices

Cloud-optimized

Boosts developer productivity

Real-World Use Cases

Dev/test environments

CI/CD pipelines

Cloud deployments

Microservices

Edge/IoT

Future Outlook

Kubernetes integration

Serverless and Edge computing

Hybrid cloud

Cloud-native design

Summary

Docker vs VMs: Lightweight, portable

Solves VM limitations

Key components: Image, Container, etc.

Next: Build and deploy with Docker