Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: DOCK • **Updated:** 01 Nov 2026
https://www.mockattempt.com/courses/docker-training

DOCK • CONTAINERS, BUILDKIT & CLOUD-NATIVE PACKAGING | **BEGINNER TO SENIOR DOCKER & CONTAINER SECURITY ENGINEER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Docker Training

Master Docker from Linux Kernel Namespaces, cgroups v2, containerd/runc & WSL2 Internals to BuildKit Cache Mounts, Distroless Multi-Stage Dockerfiles (Python, Java, PHP, Ruby, Scala), Overlay2 Storage, Docker Compose v2 Healthchecks & OCI Registry Signing — with 200 Mock Tests & Voice Gemini AI Interviews at ₹2,999.

Duration: 14–16 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

Fixed Non-Negotiable Tuition Fee: Base Fee **₹2,999** + 18% GST (₹540) = **Total Incl. GST: ₹3,539**

Live Batch Tracks: Weekday Live Online Batch (Mon – Fri) (Monday to Friday (5 Live Interactive Classes / Week) • 1.5 Hours / Day (60 Mins Live Architecture & Coding + 30 Mins Hands-On Lab) • 17–19 Weeks Fast-Track) | Weekend Intensive Live Online Batch (Sat – Sun) (Saturday & Sunday (Designed for Working Professionals & College Students) • 3.0 Hours / Day (2 Hrs Deep-Dive Live Coding + 1 Hr Guided Project Build) • 19–21 Weeks Weekend Track)

Core Technologies & Engineering Stack: Docker Engine 27 & containerd / runc • BuildKit & Multi-Stage Dockerfiles • Linux Namespaces, cgroups v2 & Seccomp • Overlay2 Copy-on-Write Storage • Docker Compose v2 & Bridge Networks • Distroless / Alpine Non-Root Images • OCI Registries, Trivy & Cosign

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Explain how Linux Kernel Namespaces (`pid`, `net`, `mnt`, `uts`, `ipc`, `user`) and `cgroups v2` isolate containers without guest hypervisor OS overhead
2. Author tiny, vulnerability-minimal Multi-Stage `Dockerfile`s across Python, Java 21, PHP-FPM, Ruby, and Scala using BuildKit cache/secret mounts and non-root `USER`s
3. Orchestrate multi-service local and staging stacks using Docker Compose v2 (`service_healthy` dependencies, named volumes, isolated bridge networks) and push multi-arch OCI images
4. Clear 200 Timed Mock Exams (10,000 Docker & Container Security MCQs) and 200 Spoken Voice + Google Gemini AI Container Engineering Interviews
5. Linux namespaces, cgroups, OverlayFS layer mechanics, and deterministic `.dockerignore` build contexts.
6. Minimal Multi-Stage Dockerfiles (`alpine`/`distroless`), build-cache ordering, BuildKit secrets, and non-root `USER` execution.
7. Multi-container Docker Compose stacks: custom bridge networks, named volumes, healthcheck `depends_on`, and resource limits.
8. Container supply-chain security: vulnerability scanning, read-only root filesystems, and private registry tagging.

Eligibility & Prerequisites: Basic terminal/shell navigation — starts from container vs virtual machine fundamentals from scratch • Any Windows 10/11 (WSL 2), macOS, or Linux laptop/PC capable of running Docker Engine or Docker Desktop • No prior Docker industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED MODULE & TOPIC-WISE SYLLABUS (16 MODULES • 42 LIVE TOPICS)

MODULE 1 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: OS-Level Virtualization, Linux Kernel Primitives & Core Docker Features

Understand why containers boot in milliseconds by sharing the host Linux kernel instead of booting heavyweight guest OS hypervisors, and explore OCI standardization, immutability, and cgroups v2 resource controls.

Objectives: ✓ Contrast Type-1/Type-2 Hypervisor Virtual Machines against shared-kernel Linux Containers • ✓ Map the 6 Linux Namespaces (`pid`, `net`, `mnt`, `ipc`, `uts`, `user`) and `cgroups v2` resource limits to container isolation • ✓ Leverage immutable content-addressable SHA-256 OCI image layers to

eliminate "works on my machine" drift

1.1 Docker Fundamentals — Containers vs Virtual Machines, Shared Host Kernel & OCI Standards [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Fundamentals — Containers vs Virtual Machines, Shared Host Kernel & OCI Standards in real Docker systems — specifically: Why VMs duplicate...

1.2 Docker Core Features — Linux Namespaces, `cgroups v2`, Seccomp/Capabilities & Layer Deduplication [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Core Features — Linux Namespaces, `cgroups v2`, Seccomp/Capabilities & Layer Deduplication in real Docker systems — specifically: How Docke...

Module 1 Hands-On Capstone Project (100 Marks): Zero-Drift Immutable Microservice Artifact & Kernel Resource Boundary Spec — Package a stateless HTTP health probe into a minimal non-root OCI image pinned by SHA-256 digest and verify strict CPU/memory cgroup boundaries at runtime.

MODULE 2 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Docker Engine Daemon Architecture (`dockerd` / `containerd` / `runc`) & Linux Installation

Trace every `docker` CLI REST call through `dockerd`, `containerd`, `containerd-shim`, and `runc`, and harden Linux Engine installations with `daemon.json` log rotation and rootless security.

Objectives: ✓ Trace the decoupled Engine stack (`docker` CLI -> Unix socket `/var/run/docker.sock` -> `dockerd` -> `containerd` -> `containerd-shim-runc-v2` -> `runc`) • ✓ Understand why `containerd-shim` enables daemonless live-restore so running containers survive `systemctl restart docker` • ✓ Install Docker Engine from official GPG-signed apt/dnf repositories and configure `/etc/docker/daemon.json` log rotation (`max-size`, `max-file`)

2.1 Docker Architecture — Client REST API, `dockerd`, `containerd`, `containerd-shim` & `runc` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Architecture — Client REST API, `dockerd`, `containerd`, `containerd-shim` & `runc` in real Docker systems — specifically: Why modern Docke...

2.2 Installing Docker Engine on Linux — GPG Apt Repos, Rootless Mode & `docker` Group Security [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Installing Docker Engine on Linux — GPG Apt Repos, Rootless Mode & `docker` Group Security in real Docker systems — specifically: Removing conflic...

Module 2 Hands-On Capstone Project (100 Marks): Production `/etc/docker/daemon.json` Hardening & Disk-Exhaustion Guard — Configure a production Linux Docker Engine daemon spec that prevents unbounded JSON container logs from filling `/var/lib/docker`, enables `live-restore`, and enforces BuildKit + `overlay2`.

MODULE 3 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Docker on Windows (`WSL 2` Architecture) & Image vs Container Layer Mechanics

Optimize Docker Desktop on Windows 11 with WSL 2 native `ext4` filesystems (avoiding `/mnt/c` 9P bottlenecks), and master read-only image layers vs the thin writable container layer.

Objectives: ✓ Configure WSL 2 backend (`~/.wslconfig` RAM limits) and store source repos inside Linux `~/projects` instead of `/mnt/c/...` • ✓ Distinguish immutable multi-layer OCI Images (Manifests + SHA-256 tarballs) from mutable runtime Container instances • ✓ Inspect image layer history (`docker history`), port publishing (`-p 127.0.0.1:8080:80`), and `docker inspect` metadata

3.1 Docker on Windows 10/11 — WSL 2 Lightweight Linux Kernel, `.wslconfig` & Avoiding `/mnt/c` I/O Traps [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker on Windows 10/11 — WSL 2 Lightweight Linux Kernel, `.wslconfig` & Avoiding `/mnt/c` I/O Traps in real Docker systems — specifically: How Do...

3.2 Docker Images vs Containers — Read-Only Content-Addressable Layers vs Thin Writable R/W Layer [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Images vs Containers — Read-Only Content-Addressable Layers vs Thin Writable R/W Layer in real Docker systems — specifically: Class vs Obje...

Module 3 Hands-On Capstone Project (100 Marks): High-Speed WSL 2 Developer Workspace & Multi-Arch Image Manifest Audit — Eliminate 10x Windows bind-mount I/O slowdowns using WSL 2 native `ext4` storage + `.wslconfig` bounds, and inspect multi-architecture (`linux/amd64`, `linux/arm64`) OCI image manifests.

MODULE 4 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: `docker run` Production Flags, Signal Handling (`PID 1`) & BuildKit `Dockerfile` Craft

Run containers safely with explicit loopback port bindings, `--init` zombie reaping, and resource caps, and write cache-optimal `Dockerfile`s (`COPY` vs `ADD`, `CMD` vs `ENTRYPOINT`, `.dockerignore`).

Objectives: ✓ Master `docker run` flags (`-d`, `--rm`, `-p 127.0.0.1:8080:8080`, `--init`, `--read-only`, `--tmpfs /tmp`, `--env-file`) • ✓ Prevent firewall bypasses (Docker `DOCKER-USER` iptables chain) and `PID 1` zombie/SIGTERM hangs (`exec` form vs shell form) • ✓ Order `Dockerfile` layers from least-frequently-changing (lockfiles + install) to most-frequently-changing (source code) with `.dockerignore`

4.1 Mastering `docker run` — Lifecycle (`create` + `start`), `127.0.0.1` Port Binding, `--init` & Resource Caps [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Mastering `docker run` — Lifecycle (`create` + `start`), `127.0.0.1` Port Binding, `--init` & Resource Caps in real Docker systems — specifically:...

4.2 Production `Dockerfile` Engineering — Layer Cache Ordering, `COPY` vs `ADD`, `ARG` vs `ENV` & `ENTRYPOINT` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Production `Dockerfile` Engineering — Layer Cache Ordering, `COPY` vs `ADD`, `ARG` vs `ENV` & `ENTRYPOINT` in real Docker systems — specifically:...

Module 4 Hands-On Capstone Project (100 Marks): Cache-Optimal BuildKit `Dockerfile` & Firewall-Safe Runtime Launch Spec — Refactor a naive slow-rebuilding `Dockerfile` so source-only edits rebuild in `<1s` via layer caching, exclude `.git`/`node_modules`/`.env` via `.dockerignore`, and handle `SIGTERM` cleanly...

MODULE 5 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Essential Docker Operations CLI & Production Python Containerization

Debug live containers using `docker exec`, `logs`, `stats`, `events`, and `system df/prune`, and package fast, non-root Python (`uv` / Gunicorn / FastAPI) microservices cleanly.

Objectives: ✓ Diagnose live container bottlenecks and disk bloat using `docker stats`, `docker logs --since`, `docker inspect`, and `docker system df -v` • ✓ Set critical Python container environment flags (`PYTHONDONTWRITEBYTECODE=1`, `PYTHONUNBUFFERED=1`) so stdout logs stream in real time • ✓ Build slim multi-stage Python 3.12 containers (`python:3.12-slim`) isolating C-compiler headers (`gcc`, `libpq-dev`) to the builder stage

5.1 Essential Docker CLI Commands — `exec`, `logs`, `stats`, `cp`, `events` & Safe `system prune` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Essential Docker CLI Commands — `exec`, `logs`, `stats`, `cp`, `events` & Safe `system prune` in real Docker systems — specifically: Daily operati...

5.2 Containerizing Python Applications — `python:3.12-slim`, `PYTHONUNBUFFERED=1` & Multi-Stage `venv` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Containerizing Python Applications — `python:3.12-slim`, `PYTHONUNBUFFERED=1` & Multi-Stage `venv` in real Docker systems — specifically: Why `pyt...

Module 5 Hands-On Capstone Project (100 Marks): Multi-Stage Non-Root Python 3.12 Gunicorn/Uvicorn Production Image — Containerize a Python 3.12 web API using a two-stage virtual-environment build so the final runtime image contains zero `gcc`/`pip` build toolchains, streams unbuffered JSON logs, and runs a...

MODULE 6 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Enterprise JVM (`Java 21` `jlink` / Layered JARs) & `PHP-FPM + Nginx` Containers

Containerize Java 21 / Spring Boot 3 services with cgroup-aware `MaxRAMPercentage` and layered JREs, and architect production PHP 8.3 (`php-fpm` + OPcache + Nginx) container topologies.

Objectives: ✓ Size JVM container memory accurately (`-XX:MaxRAMPercentage=75.0`) leaving 25% headroom for Metaspace, Thread Stacks, Direct NIO, and CodeCache • ✓ Extract Spring Boot Layered JARs (`dependencies`, `spring-boot-loader`, `application`) onto a minimal `eclipse-temurin:21-jre` or `jlink` runtime • ✓ Separate `php:8.3-fpm-alpine` (with `opcache.validate_timestamps=0` in immutable images!) from stateless Nginx FastCGI handling

6.1 Containerizing Java 21 & Spring Boot — `MaxRAMPercentage`, Non-Heap Headroom & Layered JREs [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Containerizing Java 21 & Spring Boot — `MaxRAMPercentage`, Non-Heap Headroom & Layered JREs in real Docker systems — specifically: Why shipping a...

6.2 Containerizing PHP 8.3 & Laravel — Multi-Stage Composer, `php-fpm` Tuning & Immutable `OPcache` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Containerizing PHP 8.3 & Laravel — Multi-Stage Composer, `php-fpm` Tuning & Immutable `OPcache` in real Docker systems — specifically: Installing...

Module 6 Hands-On Capstone Project (100 Marks): Cgroup-Aware Java 21 Layered JRE & Immutable OPcache PHP 8.3 Build Suite — Create production multi-stage `Dockerfile`s for (1) a Java 21 service protected against cgroup `OOMKilled 137` via `MaxRAMPercentage=75.0` and (2) an immutable PHP 8.3 FPM container with pre...

MODULE 7 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Base Image Selection (`Ubuntu` / `Alpine` / `Distroless`) & `Overlay2` Storage Internals

Choose accurately between `ubuntu:24.04` (`glibc`), `alpine` (`musl`), and `distroless` base images, and understand Linux `overlay2` Copy-on-Write (`lowerdir`, `upperdir`, `merged`) vs persistent Docker Volumes.

Objectives: ✓ Run non-interactive reproducible `apt-get` builds in `ubuntu:24.04` (`DEBIAN_FRONTEND=noninteractive`, `--no-install-recommends`) • ✓ Trace `overlay2` union mounts (`lowerdir`, `upperdir`, `workdir`, `merged`) and the Copy-on-Write (`copy_up`) latency penalty on large database files • ✓ Bypass `overlay2` completely for stateful I/O using Managed Named Volumes (`docker volume create`) vs Bind Mounts vs `tmpfs`

7.1 Ubuntu & Debian Base Images — `DEBIAN_FRONTEND=noninteractive`, Single-Layer `apt-get` & Distroless Trade-Offs [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Ubuntu & Debian Base Images — `DEBIAN_FRONTEND=noninteractive`, Single-Layer `apt-get` & Distroless Trade-Offs in real Docker systems — specifical...

7.2 Docker Storage Drivers — `overlay2` (`lowerdir` / `upperdir` / `merged`), Copy-on-Write & Named Volumes [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Storage Drivers — `overlay2` (`lowerdir` / `upperdir` / `merged`), Copy-on-Write & Named Volumes in real Docker systems — specifically: How...

Module 7 Hands-On Capstone Project (100 Marks): CVE-Minimal Ubuntu/Distroless Base Benchmark & `Overlay2` vs Volume I/O Lab — Build a non-interactive hardened Ubuntu 24.04 LTS utility container with zero leftover apt caches, and verify why PostgreSQL/MySQL data directories MUST reside on named Docker Volumes rather...

MODULE 8 OF 16 • 11 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Multi-Container `Docker Compose v2` Stacks & OCI Registry Supply-Chain Security

Wire multi-tier application stacks with `compose.yaml` (`condition: service_healthy`, isolated custom bridge DNS, Compose `secrets`), and publish multi-arch signed images to Docker Hub / GHCR / ECR.

Objectives: ✓ Eliminate boot-order race conditions using `depends_on` with `condition: service_healthy` + container `healthcheck`s • ✓ Isolate frontend and backend tiers on custom user-defined bridge networks (`internal: true` for DB/Redis) • ✓ Authenticate safely (`docker login --password-stdin`), build multi-arch `linux/amd64,linux/arm64` images via `buildx`, scan with Trivy, and push immutable tags

8.1 Docker Compose v2 — `service_healthy` Readiness, Embedded `127.0.0.11` DNS, Profiles & Isolated Networks [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker Compose v2 — `service_healthy` Readiness, Embedded `127.0.0.11` DNS, Profiles & Isolated Networks in real Docker systems — specifically: Mo...

8.2 Pushing to OCI Registries — `--password-stdin`, Immutable SemVer+SHA Tags, Multi-Arch `buildx` & SBOMs [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Pushing to OCI Registries — `--password-stdin`, Immutable SemVer+SHA Tags, Multi-Arch `buildx` & SBOMs in real Docker systems — specifically: Regi...

Module 8 Hands-On Capstone Project (100 Marks): 3-Tier Health-Gated `compose.yaml` Stack (`internal: true` DB Network) & Multi-Arch Buildx Push — Orchestrate a Web + API + PostgreSQL stack in `compose.yaml` where the API waits for `pg_isready` (`service_healthy`) and the database sits on an `internal: true` network with zero outbound...

MODULE 9 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Managed Cloud Container Platforms & Docker vs Kubernetes Architecture Trade-Offs

Deploy containers to serverless/managed cloud runtimes (AWS ECS Fargate, Cloud Run, Docker Build Cloud, Registry Mirrors), and clarify how Docker (`containerd` packaging) complements Kubernetes orchestration.

Objectives: ✓ Compare serverless container runtimes (AWS ECS Fargate, Google Cloud Run, Azure Container Apps) and remote BuildKit (`Docker Build Cloud`) • ✓ Resolve the classic misconception behind Kubernetes deprecating `dockershim` (Docker-built OCI images run 100% natively on Kubernetes `containerd`!) • ✓ Choose pragmatically between single-host Docker Compose, Docker Swarm, Managed Serverless Containers, and full Kubernetes clusters

9.1 Managed Cloud Containers — AWS ECS Fargate, Cloud Run, Pull-Through Cache Mirrors & Remote BuildKit [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Managed Cloud Containers — AWS ECS Fargate, Cloud Run, Pull-Through Cache Mirrors & Remote BuildKit in real Docker systems — specifically: Running...

9.2 Docker vs Kubernetes — Packaging (`OCI` Build) vs Cluster Orchestration & Why `dockershim` Removal Changed Nothing for Dockerfiles [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Docker vs Kubernetes — Packaging (`OCI` Build) vs Cluster Orchestration & Why `dockershim` Removal Changed Nothing for Dockerfiles in real Docker...

Module 9 Hands-On Capstone Project (100 Marks): Cloud-Ready 12-Factor Container Contract & Orchestrator Decision Matrix — Package a stateless 12-Factor container meeting strict Cloud Run / AWS Fargate / Kubernetes runtime contracts (`PORT` env binding, stdout structured JSON logging, `<10s` graceful `SIGTERM` d...

MODULE 10 OF 16 • 10 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Polyglot Production Packaging — `Ruby on Rails` (`Jemalloc`/Puma) & `Scala` (`sbt` Layered JVM)

Complete your polyglot container mastery by packaging memory-efficient Ruby 3.3 / Rails 7+ (`jemalloc` + Puma) and fast-caching Scala 3 / `sbt` multi-stage JVM microservices.

Objectives: ✓ Build slim multi-stage Ruby 3.3 containers (`BUNDLE_DEPLOYMENT=1`, `BUNDLE_WITHOUT="development:test"`, `LD_PRELOAD=libjemalloc.so.2`) • ✓ Cache slow `sbt update` Ivy/Coursier dependency resolution separately from Scala source compilation in BuildKit • ✓ Ship minimal distroless/JRE Scala runtime images without the `1.2 GB` `sbt` + JDK compiler toolchain

10.1 Containerizing Ruby & Rails — Multi-Stage Bundler (`BUNDLE_DEPLOYMENT=1`), `libjemalloc2` & Puma [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Containerizing Ruby & Rails — Multi-Stage Bundler (`BUNDLE_DEPLOYMENT=1`), `libjemalloc2` & Puma in real Docker systems — specifically: Why compil...

10.2 Containerizing Scala & `sbt` — Coursier Cache Mounts, `sbt stage` vs Fat `assembly` JARs & JRE Runtime [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Containerizing Scala & `sbt` — Coursier Cache Mounts, `sbt stage` vs Fat `assembly` JARs & JRE Runtime in real Docker systems — specifically: Why...

Module 10 Hands-On Capstone Project (100 Marks): Polyglot Multi-Stage Build Suite (`Ruby 3.3 + jemalloc` & `Scala 3 sbt-native-packager`) — Author production multi-stage `Dockerfile`s for a Ruby 3.3 Puma web service using `jemalloc` to slash heap fragmentation by ~35% and a Scala 3 service stripping `sbt` from the runtime image.

MODULE 11 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

BuildKit Advanced Caching, Multi-Arch (AMD64/ARM64) & Distroless Images

Production-grade Docker Training engineering module covering BuildKit Advanced Caching, Multi-Arch (AMD64/ARM64) & Distroless Images with hands-on labs and capstone build.

Objectives: ✓ BuildKit Cache Mounts (--mount=type=cache) & Build Secret Mounts — Purpose: Speed up npm/pip/Maven builds 6x without ever baking private tokens into image layers!. • ✓ Multi-Stage Builds into Distroless / Scratch / Alpine Runtimes — Purpose: Strip compilers and shells out of prod images, slashing size and CVE surface by 85%. • ✓ Cross-Compiling linux/amd64 + linux/arm64 Images with docker buildx — Purpose: Run identical container tags on Apple Silicon laptops and AWS Graviton servers. • ✓ Layer Ordering Discipline & Strict .dockerignore Hygiene — Purpose: Ensure editing one source file never invalidates cached dependency layers.

11.1 BuildKit Cache Mounts (--mount=type=cache) & Build Secret Mounts [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply BuildKit Cache Mounts (--mount=type=cache) & Build Secret Mounts in real Docker systems — specifically: Speed up npm/pip/Maven builds 6x without e...

11.2 Multi-Stage Builds into Distroless / Scratch / Alpine Runtimes [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Multi-Stage Builds into Distroless / Scratch / Alpine Runtimes in real Docker systems — specifically: Strip compilers and shells out of prod image...

11.3 Cross-Compiling linux/amd64 + linux/arm64 Images with docker buildx [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Cross-Compiling linux/amd64 + linux/arm64 Images with docker buildx in real Docker systems — specifically: Run identical container tags on Apple S...

11.4 Layer Ordering Discipline & Strict .dockerignore Hygiene [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Layer Ordering Discipline & Strict .dockerignore Hygiene in real Docker systems — specifically: Ensure editing one source file never invalidates c...

Module 11 Hands-On Capstone Project (100 Marks): Sub-50MB Multi-Arch Production Container Build Pipeline — Design, implement, test, and document 'Sub-50MB Multi-Arch Production Container Build Pipeline' applying every production concept from Module 11 (BuildKit Advanced Caching, Multi-Arch (AMD64...

MODULE 12 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Container Security Hardening: Linux Namespaces, cgroups, Trivy & SBOMs

Production-grade Docker Training engineering module covering Container Security Hardening: Linux Namespaces, cgroups, Trivy & SBOMs with hands-on labs and capstone build.

Objectives: ✓ Linux Namespaces, Control Groups (cgroups v2) & OverlayFS Internals — Purpose: Understand what containers actually are under the hood on the Linux kernel. • ✓ Non-Root USER, --cap-drop=ALL, --read-only & no-new-privileges — Purpose: Contain exploit blast radius even if an application library has a zero-day bug. • ✓ Automated Vulnerability Scanning with Trivy & Syft CycloneDX SBOMs — Purpose: Fail GitHub Actions builds automatically when Critical OS/library CVEs appear. • ✓ Cosign Keyless Image Signing & Private Registry Access Controls — Purpose: Verify cryptographic provenance so clusters run only trusted CI-built images.

12.1 Linux Namespaces, Control Groups (cgroups v2) & OverlayFS Internals [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Linux Namespaces, Control Groups (cgroups v2) & OverlayFS Internals in real Docker systems — specifically: Understand what containers actually are...

12.2 Non-Root USER, --cap-drop=ALL, --read-only & no-new-privileges [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Non-Root USER, --cap-drop=ALL, --read-only & no-new-privileges in real Docker systems — specifically: Contain exploit blast radius even if an appl...

12.3 Automated Vulnerability Scanning with Trivy & Syft CycloneDX SBOMs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Automated Vulnerability Scanning with Trivy & Syft CycloneDX SBOMs in real Docker systems — specifically: Fail GitHub Actions builds automatically...

12.4 Cosign Keyless Image Signing & Private Registry Access Controls [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Cosign Keyless Image Signing & Private Registry Access Controls in real Docker systems — specifically: Verify cryptographic provenance so clusters...

Module 12 Hands-On Capstone Project (100 Marks): Zero-Critical-CVE Hardened Non-Root Container Supply Chain — Design, implement, test, and document 'Zero-Critical-CVE Hardened Non-Root Container Supply Chain' applying every production concept from Module 12 (Container Security Hardening: Linux Names...

MODULE 13 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Production Docker Compose: Healthchecks, Isolated Networks, TLS & Limits

Production-grade Docker Training engineering module covering Production Docker Compose: Healthchecks, Isolated Networks, TLS & Limits with hands-on labs and capstone build.

Objectives: ✓ depends_on: service_healthy with Real DB/Redis Readiness Checks — Purpose: Stop API containers from crashing at boot before PostgreSQL finishes initializing. • ✓ Internal-Only Backend Bridge Networks vs Public Reverse-Proxy Network — Purpose: Keep PostgreSQL and Redis completely unreachable from public host ports. • ✓ Compose Secrets, Named Volume Backups & CPU/Memory Resource Limits — Purpose: Prevent one runaway container from starving host RAM or losing stateful data. • ✓ PID 1 Signal Forwarding (tini / exec Form) for Clean SIGTERM Shutdown — Purpose: Ensure containers stop in 1 second gracefully instead of suffering 10s SIGKILL kills.

13.1 depends_on: service_healthy with Real DB/Redis Readiness Checks [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply depends_on: service_healthy with Real DB/Redis Readiness Checks in real Docker systems — specifically: Stop API containers from crashing at boot b...

13.2 Internal-Only Backend Bridge Networks vs Public Reverse-Proxy Network [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Internal-Only Backend Bridge Networks vs Public Reverse-Proxy Network in real Docker systems — specifically: Keep PostgreSQL and Redis completely...

13.3 Compose Secrets, Named Volume Backups & CPU/Memory Resource Limits [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Compose Secrets, Named Volume Backups & CPU/Memory Resource Limits in real Docker systems — specifically: Prevent one runaway container from starv...

13.4 PID 1 Signal Forwarding (tini / exec Form) for Clean SIGTERM Shutdown [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply PID 1 Signal Forwarding (tini / exec Form) for Clean SIGTERM Shutdown in real Docker systems — specifically: Ensure containers stop in 1 second gr...

Module 13 Hands-On Capstone Project (100 Marks): Multi-Service App + DB + Redis + Caddy/Nginx TLS Stack — Design, implement, test, and document 'Multi-Service App + DB + Redis + Caddy/Nginx TLS Stack' applying every production concept from Module 13 (Production Docker Compose: Healthchecks, Isol...

MODULE 14 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Container Observability, Log Drivers, Live Forensics & CI/CD Rollouts

Production-grade Docker Training engineering module covering Container Observability, Log Drivers, Live Forensics & CI/CD Rollouts with hands-on labs and capstone build.

Objectives: ✓ Log Rotation (json-file max-size), Fluentd/Loki Drivers & Prometheus cAdvisor — Purpose: Prevent unrotated container stdout logs from filling 100% of server disks!. • ✓ Live Container Diagnostics (docker stats, inspect, events & nsenter) — Purpose: Debug high memory, DNS resolution, and socket exhaustion on live containers. • ✓ GitHub Actions Registry Caching, Semantic SHA Tags & Rollback Scripts — Purpose: Eliminate mutable :latest tag chaos with

immutable commit-SHA deployments. • ✓ Reverse-Proxy Blue/Green Container Cutover & Automated Prune Policies — Purpose: Swap traffic to verified new containers seamlessly and reclaim disk safely.

14.1 Log Rotation (json-file max-size), Fluentd/Loki Drivers & Prometheus cAdvisor [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Log Rotation (json-file max-size), Fluentd/Loki Drivers & Prometheus cAdvisor in real Docker systems — specifically: Prevent unrotated container s...

14.2 Live Container Diagnostics (docker stats, inspect, events & nsenter) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Live Container Diagnostics (docker stats, inspect, events & nsenter) in real Docker systems — specifically: Debug high memory, DNS resolution, and...

14.3 GitHub Actions Registry Caching, Semantic SHA Tags & Rollback Scripts [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply GitHub Actions Registry Caching, Semantic SHA Tags & Rollback Scripts in real Docker systems — specifically: Eliminate mutable :latest tag chaos w...

14.4 Reverse-Proxy Blue/Green Container Cutover & Automated Prune Policies [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Reverse-Proxy Blue/Green Container Cutover & Automated Prune Policies in real Docker systems — specifically: Swap traffic to verified new containe...

Module 14 Hands-On Capstone Project (100 Marks): Monitored Zero-Downtime Container Host with Centralized Logs — Design, implement, test, and document 'Monitored Zero-Downtime Container Host with Centralized Logs' applying every production concept from Module 14 (Container Observability, Log Drivers, L...

MODULE 15 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Supply-Chain Security: BuildKit Secrets, SBOMs, Trivy Scanning & Distroless

Self-explanatory list-and-bullet production mastery module covering Supply-Chain Security: BuildKit Secrets, SBOMs, Trivy Scanning & Distroless in Docker Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ BuildKit Cache Mounts, Secret Mounts (--mount=type=secret) & Layer-Order Mastery — Purpose: Cut CI Docker build times by 70% while ensuring private tokens never bake into image layers. • ✓ Distroless / Wolfi / Alpine Minimal Images, Non-Root UIDs & Dropped Linux Capabilities — Purpose: Shrink attack surface and image megabytes by stripping shells and package managers from runtime. • ✓ Vulnerability Scanning (Trivy), SBOM Generation & Signed Multi-Arch (amd64/arm64) Releases — Purpose: Block vulnerable base images in CI and build images that run on both x86 and AWS Graviton. • ✓ Build, verify, and document 'Zero-Critical-CVE Multi-Arch Distroless Image Pipeline with BuildKit Cache' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + Google Gemini AI Technical Interviews.

15.1 BuildKit Cache Mounts, Secret Mounts (--mount=type=secret) & Layer-Order Mastery [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply BuildKit Cache Mounts, Secret Mounts (--mount=type=secret) & Layer-Order Mastery in real Docker systems — specifically: Cut CI Docker build times...

15.2 Distroless / Wolfi / Alpine Minimal Images, Non-Root UIDs & Dropped Linux Capabilities [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Distroless / Wolfi / Alpine Minimal Images, Non-Root UIDs & Dropped Linux Capabilities in real Docker systems — specifically: Shrink attack surfac...

15.3 Vulnerability Scanning (Trivy), SBOM Generation & Signed Multi-Arch (amd64/arm64) Releases [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Vulnerability Scanning (Trivy), SBOM Generation & Signed Multi-Arch (amd64/arm64) Releases in real Docker systems — specifically: Block vulnerable...

Module 15 Hands-On Capstone Project (100 Marks): Zero-Critical-CVE Multi-Arch Distroless Image Pipeline with BuildKit Cache — Architect, implement, test, and document 'Zero-Critical-CVE Multi-Arch Distroless Image Pipeline with BuildKit Cache' applying every bullet-wise engineering concept from Module 15 (Supply-Ch...

MODULE 16 OF 16 • 14 LIVE + LAB HOURS • 3 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Production Docker Compose Orchestration, Resource Limits & Container Triage

Self-explanatory list-and-bullet production mastery module covering Production Docker Compose Orchestration, Resource Limits & Container Triage in Docker Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ Healthchecks, service_completed_successfully Dependencies & Graceful PID 1 Signals — Purpose: Ensure apps wait for ready databases and receive SIGTERM cleanly without 10-second SIGKILL hangs. • ✓ Memory/CPU Cgroups Limits, Log Rotation Drivers & Isolated Internal Networks — Purpose: Prevent a single runaway container from exhausting host disk space or host RAM. • ✓ Container Filesystem/Overlay2 Internals & Docker Troubleshooting Interviews — Purpose: Inspect namespaces, cgroups, zombie processes, and DNS resolution issues confidently. • ✓ Build, verify, and document 'Health-Gated Reverse-Proxy + App + DB Compose Stack with Log/Memory Limits' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests

and Spoken Voice + Google Gemini AI Technical Interviews.

16.1 Healthchecks, service_completed_successfully Dependencies & Graceful PID 1 Signals [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Healthchecks, service_completed_successfully Dependencies & Graceful PID 1 Signals in real Docker systems — specifically: Ensure apps wait for rea...

16.2 Memory/CPU Cgroups Limits, Log Rotation Drivers & Isolated Internal Networks [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Memory/CPU Cgroups Limits, Log Rotation Drivers & Isolated Internal Networks in real Docker systems — specifically: Prevent a single runaway conta...

16.3 Container Filesystem/Overlay2 Internals & Docker Troubleshooting Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Container Filesystem/Overlay2 Internals & Docker Troubleshooting Interviews in real Docker systems — specifically: Inspect namespaces, cgroups, zo...

Module 16 Hands-On Capstone Project (100 Marks): Health-Gated Reverse-Proxy + App + DB Compose Stack with Log/Memory Limits — Architect, implement, test, and document 'Health-Gated Reverse-Proxy + App + DB Compose Stack with Log/Memory Limits' applying every bullet-wise engineering concept from Module 16 (Productio...

3. LIVE CLIENT PRODUCTION PROJECT, GLOBAL CERTIFICATIONS & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: ContainerForge Multi-Service Production Stack & Hardened Image Supply Chain • **Confirmed Role:** Docker & Container Engineering Intern

Domain & Production Infra: SaaS Platform DevOps Modernization Client (Multi-Stage Dockerfiles + Docker Compose Production + Private Registry + TLS)

- Engineered minimal multi-stage Dockerfiles running as non-root users, cutting production image size by over 65%.
- Orchestrated isolated bridge networks, named persistent volumes, secrets, and health-checked services via Compose.
- Integrated automated image vulnerability scanning, layer caching, and tagged registry releases in CI/CD.
- Deployed the containerized multi-service application stack to a live HTTPS client production host.

Aligned Global Vendor Certifications: **DCA** (Docker Certified Associate (Container Engineering)) • **CKAD** (Certified Kubernetes Application Developer (CKAD)) • **GH-200** (GitHub Actions Certified CI/CD Engineer)

Target Job Roles: Docker & Containerization Engineer, Associate DevOps & Cloud Engineer, Build & Release Automation Engineer, Container-Ready Backend Developer

Realistic Annual CTC Scope: **Fresher / Entry-Level DevOps & Cloud (0–1 Yr):** ₹5.0 LPA – ₹9.5 LPA **Mid-Level Container & DevOps Engineer (1–3 Yrs):** ₹10.5 LPA – ₹20.5 LPA **Senior Cloud-Native / Platform Engineer (3–6+ Yrs):** ₹21.0 LPA – ₹40.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://www.mockattempt.com/courses/docker-training

Try 2 Free Topic-Wise 50-MCQ Mock Tests & Spoken Gemini AI Interviews daily on every topic above before enrolling.