Ons :

Docker learnings needed and need to learn document:

Perfect. Now we do this properly — senior style.
I'm going to:

1. ✅ Compare your actual Docker journey vs this 25-step list
2. ✔ Tick what you have already completed
3. Mark partial
4. ✖ Mark what's still missing
5. Add additional senior-level items
6. Reframe everything as "Experienced DevOps Docker Competency Model"

No motivation talk. Only clarity.

---

# ✅ DOCKER MATURITY CHECK — ONS PROJECT

---

## Evaluation Against Your 25-Step List

---

### 1. Optimize Dockerfile

✅ **DONE**
You reduced image from ~955MB → ~221MB using multi-stage.

---

### 2. Use multi-stage builds

✅ **DONE (Strong)**
Builder + runtime separation correctly implemented.

---

### 3. Add `.dockerignore`

**PARTIAL / Not deeply discussed**
You used good practices but didn't formalize ignore strategy.

Still should include:

- target/
- .git/
- node_modules/
- logs/

---

## 4. Run container as non-root

✅ **DONE**
Created `springuser` and verified with `whoami`.

---

## 5. Tag images properly (version / commit SHA)

**PARTIAL**
You used tags like `v1`, `v2`.
But not:

- semantic versioning
- git SHA tagging
- latest strategy control

Still need structured tagging policy.

---

## 6. Externalize all config

**PARTIAL**
You used environment variables in compose.
But:

- No centralized config strategy
- No profile-based separation
- No production config layering

---

## 7. Use `.env` + secrets

**PARTIAL**
You used environment variables.
But:

- Secrets are still visible in compose file
- No secret rotation mechanism
- No Docker secrets usage

---

## 8. Add healthchecks

✅ **DONE (Strong)**
Used DB healthcheck + `service_healthy`.

---

## 9. Add restart policies

✅ **DONE**
`restart: on-failure` implemented.

---

## 10. Set CPU & memory limits

❌ **NOT DONE**

Missing:

```
deploy:
  resources:
    limits:
      cpus: "0.5"
      memory: 512M
```

This is important for Kubernetes transition.

---

## 11. Understand volume lifecycle & backups

**PARTIAL**

You:

- Used named volumes
- Understood persistence

But did NOT:

- Backup volume
- Restore volume
- Migrate volume between machines

Still pending.

---

## 12. Isolate networks (internal vs exposed)

**✕ NOT DONE**

You relied on default bridge.
No custom networks created.

Need:

```
docker network create internal_net
```

---

## 13. Remove DB port exposure

**✕ NOT DONE**

Currently DB is exposed to host via 5433/5434.

In production:
DB should NOT expose port to public.

---

## 14. Handle service startup order correctly

**✅ DONE (Advanced)**
Healthcheck + depends_on condition.

---

## 15. Debug using logs / exec / inspect

✅ **DONE (Strong)**
You used:

- logs -f
- exec
- inspect
- lsof
- network prune

This is real debugging.

---

## 16. Build image in CI pipeline

❌ **NOT DONE**

All builds are manual.

---

## 17. Cache Docker layers in CI

❌ **NOT DONE**

Requires CI/CD setup.

---

## 18. Push image to private registry

❌ **NOT DONE**

You pulled from ECR public.
But did NOT:

- Push your own image to private ECR.

---

## 19. Pull & run same image across envs

❌ **NOT DONE**

No dev/stage/prod image promotion.

---

## 20. Simulate container failure scenarios

**PARTIAL**

You faced failures accidentally.
But not intentionally tested:

- kill container
- simulate OOM
- simulate DB crash

---

## 21. Rotate secrets without rebuild

**✕ NOT DONE**

This is production-level maturity.

---

## 22. Scan images for vulnerabilities

**✕ NOT DONE**

Need:

- Trivy
- Docker Scout

---

## 23. Understand Docker → Kubernetes mapping

**PARTIAL**

You conceptually understand, but not applied.

---

## 24. Replace Compose with K8s manifests

**✕ NOT DONE**

---

### 25. Run app in K8s using same image

**❌ NOT DONE**

---

## FINAL STATUS SUMMARY

| Status | Count |
|---|---|
| ✅ Completed | 8 |
| Partial | 6 |
| ❌ Not Done | 11 |

You are strong in:

- Debugging
- Image building
- Optimization
- Compose orchestration
- Health handling

You are missing:

- CI/CD
- Registry strategy
- Production security hardening
- Resource governance
- Kubernetes execution

---

## ADDITIONAL SENIOR DOCKER TOPICS (Beyond 25)

Now I'm adding what senior engineers must know:

---

## 26. Understand Docker storage drivers (overlay2)

---

## 27. Understand Linux namespaces & cgroups

This separates mid-level from senior.

---

## 28. Understand container runtime (containerd vs dockerd)

Important for Kubernetes.

---

## 29. BuildKit advanced usage

- Cache mounts
- Secret mounts
- SSH forwarding

---

## 30. Distroless images

Security hardening.

---

## 31. Rootless Docker mode

---

## 32. Image signing (Cosign / Notary)

---

## 33. Docker registry garbage collection

---

## 34. Multi-arch builds (amd64 + arm64)

---

### 35. Production troubleshooting scenario simulation

Example:

- DB down
- Network partition
- Image corrupted
- Volume full
- Disk full
- Memory leak

---

## What a Senior DevOps Must Be Able to Explain (Docker)

If interviewer asks ANY of these — you should speak clearly:

- How Docker networking works internally
- How container isolation works
- What happens during docker build
- What is layer caching
- Why multi-stage builds matter
- Why containers are immutable
- How to debug container startup failures
- How to secure container images
- How to limit resources
- How Docker maps to Kubernetes
- Why Compose is not production orchestration
- How to scale containers
- How to avoid secrets leakage
- How to migrate persistent data
- How to design container platform architecture

---

## What Is Left For You To Become “Senior-Ready”

Priority order:

1. CI/CD Pipeline automation

2. Private registry push & pull
3. Resource limits
4. Custom networking
5. Image scanning
6. Failure simulation
7. Kubernetes migration

After that:
You won't just "know Docker."

You will understand container platforms.

---

If you want next:

I can now design:

- A **30-day Docker → Kubernetes transition plan**
- Or a **real interview question simulation round**
- Or a **hands-on Kubernetes mapping of your ONS app**

Tell me which direction we go next.