DevOps Shack

Docker Best Practices

1. Use Official or Verified Images

- Always prefer official images from Docker Hub or trusted sources.
- Check image signatures and vulnerabilities before using.

2. Use Minimal Base Images

- Use lightweight images like `alpine`, `distroless`, or `scratch` to reduce attack surface.
- Avoid bloated images like `ubuntu` or `debian` unless necessary.

3. Pin Image Versions

- Use specific tags (`node:18.16-alpine`) instead of `latest` to ensure consistency.
- Maintain a manifest of approved images.

4. Reduce Image Layers

- Combine `RUN` commands into a single line to avoid unnecessary layers.

Example:

```
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
```

5. Leverage Multi-Stage Builds

- Reduce final image size by discarding build-time dependencies.

Example:

```
FROM golang:1.19 AS builder
WORKDIR /app
COPY . .
RUN go build -o myapp

FROM alpine
COPY --from=builder /app/myapp /usr/local/bin/myapp
CMD ["myapp"]
```

6. Use `.dockerignore` Efficiently

- Exclude unnecessary files (e.g., `node_modules`, `.git`, logs).

Example `.dockerignore`:

```
.git
node_modules
logs
```

7. Set a Non-Root User

- Avoid running containers as `root` to reduce security risks.

Example:

```
RUN adduser --disabled-password myuser
USER myuser
```

8. Keep Containers Stateless

- Store data externally (e.g., volumes, databases).
- Do not rely on local container storage.

9. Use Environment Variables for Configurations

- Avoid hardcoding secrets; use `ENV` or secrets management.

Example:

```
ENV APP_ENV=production
```

10. Limit Container Resources

- Use `--memory` and `--cpu` flags to prevent resource hogging.

Example:

```
docker run --memory="512m" --cpus="1" myapp
```

11. Optimize Layer Caching

- Place frequently changing lines (e.g., `COPY` commands) at the bottom of the to improve caching efficiency.

12. Keep Images Updated

- Regularly update images and rebuild to include security patches.

Example:

```
docker pull nginx:alpine && docker run nginx:alpine
```

13. Scan Images for Vulnerabilities

- **Use tools like:**
  - **Trivy (`trivy image myimage`)**
  - **Docker Scout**
  - **Snyk**
  - **Anchore**

14. Use Read-Only Filesystem When Possible

- **Reduce attack vectors by making the filesystem immutable.**

Example:

```
docker run --read-only myimage
```

15. Run Only One Process per Container

- **Follow the single-responsibility principle (e.g., don't run both Nginx and MySQL in one container).**
- **Use multiple services via Docker Compose.**

16. Use Multi-Arch Images

- **Ensure compatibility across different platforms (`amd64`, `arm64`).**

Example:

```
docker buildx build --platform linux/amd64,linux/arm64
-t myapp:latest .
```

17. Implement Proper Logging

- Redirect logs to `stdout` and `stderr` for containerized logging.

Example:

```
docker logs mycontainer
```

18. Use Health Checks

- Define `HEALTHCHECK` in to ensure container reliability.

Example:

```
HEALTHCHECK --interval=30s --timeout=5s --retries=3 CMD
curl -f http://localhost || exit 1
```

19. Use Labels for Metadata

- Add metadata for maintainability.

Example:

```
LABEL maintainer="admin@mycompany.com"
LABEL version="1.0.0"
```

20. Enable Container Restart Policies

- Use `--restart` flag to define behavior on craes.

Example:

```
docker run --restart unless-stopped myapp
```

21. Use Private Registries for Internal Images

- Use Docker Hub, AWS ECR, GCP Artifact Registry, or Harbor for private images.

Authenticate via:

```
docker login myregistry.com
```

22. Keep Containers Ephemeral

- Containers ould be disposable; use volumes for persistent data.
- Avoid modifying running containers.

23. Use Volume Mounts for Persistent Data

- Prefer `-v` volumes over bind mounts for better performance.

Example:

```
docker run -v myvolume:/data myapp
```

24. Ensure Port Binding Security

- Avoid exposing unnecessary ports.

Use explicit port binding:

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

25. Limit Container Privileges

- **Run with `--cap-drop=ALL` and add only required capabilities.**

Example:

```
docker run --cap-drop=ALL --cap-add=NET_ADMIN myapp
```

26. Avoid `ADD` When `COPY` Is Sufficient

- **`COPY` is preferred unless you need URL extraction or tarball auto-extraction.**

27. Keep Secrets Secure

- **Use `--secret` flag or a secrets manager.**
- **Avoid adding secrets in or `ENV`.**

28. Use CI/CD for Automated Builds

- **Automate builds and deployments with GitHub Actions, GitLab CI/CD, Jenkins.**

29. Secure the Docker Daemon

- **Use TLS to secure the Docker API.**
- **Restrict access using `iptables` or `firewalld`.**

30. Regularly Prune Unused Containers and Images

- **Clean up old images and containers to free disk space.**

Example:

```
docker system prune -af
```

31. Validate Configurations with Linter

- **Use `hadolint` to check for best practices.**

Example:

```
hadolint
```

32. Use `docker-compose` for Multi-Service Apps

- **Maintain `docker-compose.yml` for better management.**

Example:

```
version: "3"
services:
  app:
    image: myapp
    ports:
      - "8080:80"
```

33. Consider Rootless Docker for Security

- Running Docker in rootless mode reduces privilege risks.

34. Use Namespaces for Isolation

- Utilize cgroups and user namespaces to restrict access.

35. Set Up Network Security

- Use Docker networks instead of `--host` networking.

Example:

```
docker network create mynetwork
docker run --network=mynetwork myapp
```

36. Use Immutable Tags for Deployment

- Instead of `latest`, use commit-based tags for reproducible builds.

Example:

```
docker build -t myapp:v1.2.3 .
```

37. Enable AppArmor or SELinux Policies

- Use security profiles to restrict access to resources.

Example (AppArmor):

```
docker run --security-opt apparmor=myprofile myapp
```

38. Avoid Privileged Mode

- Never run containers with `--privileged` unless absolutely necessary.
- It grants full host access, which is a major security risk.

39. Use Host Networking Only When Necessary

- `--network=host` bypasses Docker's network isolation.
- Prefer bridge or overlay networks instead.

40. Restrict Container Process Capabilities

- Drop all unnecessary Linux capabilities using `--cap-drop=ALL`.

Example:

```
docker run --cap-drop=ALL --cap-add=NET_ADMIN myapp
```

41. Use Health Check Retries to Avoid False Positives

- Increase `--retries` value to ensure accurate health status.

Example:

```
HEALTHCHECK --interval=10s --timeout=5s --retries=5 CMD
curl -f http://localhost || exit 1
```

42. Use tmpfs for Temporary Data

- Instead of writing to disk, use `--tmpfs` for temporary storage.

Example:

```
docker run --tmpfs /tmp:size=100m myapp
```

43. Avoid `--pid=host` for Security

- **aring the host's PID namespace increases risks of process injection.**
- **Use default process isolation.**

44. Regularly Audit and Configs

- **Automate security checks using tools like:**
  - **Docker Bench for Security (`docker bench security`)**
  - **Trivy, Anchore, or Snyk for scanning**

45. Ensure Logs Are Managed Properly

- **Avoid letting logs grow indefinitely.**

Use log rotation:

```
docker run --log-driver=json-file --log-opt
max-size=10m --log-opt max-file=3 myapp
```

46. Enforce Image Provenance & Signature Verification

- **Use Docker Content Trust (DCT) to sign and verify images.**

Example:

```
export DOCKER_CONTENT_TRUST=1
docker pull mysecureimage
```

47. Use CI/CD Pipelines for Automated Security Scans

- Integrate security scanning into Jenkins, GitHub Actions, GitLab CI/CD.

Example GitHub Action:

```
jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: aquasecurity/trivy-action@master
        with:
          image-ref: 'myapp:latest'
```

48. Prefer Build-Time Arguments Over Environment Variables

- Use `ARG` instead of `ENV` to avoid persisting sensitive values in the final image.

Example:

```
ARG API_KEY
RUN echo "API_KEY=${API_KEY}"
```

49. Avoid Mounting the Docker Socket (`/var/run/docker.sock`)

- Exposing the Docker daemon socket to containers can lead to privilege escalation.
- If necessary, use tools like gVisor or sysbox for isolation.

50. Enable Seccomp Profiles

- **Reduce attack surface by enforcing syscall restrictions.**
- **Example:**

```
docker run --security-opt seccomp=default.json
myapp
```