Docker Incident Management Runbooks

Purpose of Docker

Docker is a platform designed to make it easier to create, deploy, and run applications using containers. Containers allow a developer to package an application with all of its dependencies into a standardized unit for software development.

Function of Docker

Docker provides a lightweight, portable, and consistent runtime environment. It uses the host OS kernel but isolates the application at the process level using namespaces and control groups (cgroups).

Key Components:

- **Docker Engine**: The runtime that runs containers.
- **Docker Images**: Blueprints of the container environment.
- **Docker Containers**: Instances of Docker images.
- **Dockerfile**: Script containing instructions to build an image.
- **Docker Hub**: Public registry for images.
- **Docker Compose**: Tool to define and run multi-container Docker applications.

Example Use Cases

- Running a microservice as a container
- Packaging a Node.js or Java application with NGINX and Redis
- CI/CD pipelines that build and test in containers for consistency
- Running local development environments that match production

Example Dockerfile for a Node.js App:

FROM node:20

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

```
COPY . .

EXPOSE 3000

CMD ["node", "app.js"]
```

Why These Failure Modes Were Chosen

These failure modes were selected because they represent **frequent, high-impact issues** across key stages of container lifecycle and infrastructure health. They're aligned with core responsibilities in DevOps, Site Reliability Engineering (SRE), and Production Support, helping ensure high availability and operational resilience.

Included Failure Modes:

1. Docker container won't start in development
2. Docker containers crash in production
3. Docker image pull failure
4. Docker network failure
5. Docker daemon crash or unresponsiveness
6. Docker volume mount failure
7. Container resource exhaustion (CPU/Memory)
8. Container restart loop

Runbook: Docker Container Won't Start in Development

Failure Mode: Docker container fails to start or crashes on startup during development.

Symptoms

- OCI runtime error, permission denied, or mount failures
- docker ps shows container exits immediately
- docker logs <container> shows stack trace
- docker-compose up exits prematurely

Diagnostics

- Run docker logs <container>
- Inspect with docker inspect <container>
- Check Dockerfile, docker-compose.yml, and mounted volumes
- Run container interactively with --entrypoint /bin/bash

Causes

- Invalid image or corrupt build
- Volume path doesn't exist or has restricted permissions
- Conflicting host ports
- Application crash or missing dependency
- Docker daemon misconfigured

Actions

- Rebuild image using --no-cache
- Check file/volume permissions
- Isolate by simplifying run command
- Test with a different or base image
- Restart Docker and verify daemon

Remediation

- Fix and rebuild Dockerfile
- Validate environment variables
- Use .dockerignore correctly
- Use docker-compose config for file validation

DevOps/SRE Notes

- Enforce base image validation
- Automate Dockerfile linting
- Use preconfigured Compose templates

Runbook: Docker Containers Crash or Fail in Production

Failure Mode: Docker containers fail intermittently or consistently, disrupting production services.

Symptoms

- 5xx errors or service outage
- High container restart count
- Health check failures
- Spike in resource usage (CPU/memory)

Diagnostics

- Use docker logs and centralized logs
- Inspect containers, exit codes
- Run docker stats or check Prometheus metrics
- Use docker events for real-time insight

Causes

- OOM kills or memory leaks
- Disk full (Docker can't write logs or layers)
- External service dependency failures
- Kernel or cgroup violations
- Misconfigured resource limits

Actions

- Scale replicas or restart containers
- Prune unused images: docker system prune
- Adjust memory or CPU limits
- Check and fix health checks

Remediation

- Set CPU/memory limits in Compose/K8s

- Configure log rotation and storage thresholds
- Use observability tools (Grafana, Prometheus)
- Apply Infrastructure-as-Code for consistency

DevOps/SRE Notes

- Tag and scan images in pipelines
- Automate health probes and self-healing
- Apply chaos engineering principles

Runbook: Docker Image Pull Failure

Failure Mode: Docker is unable to pull an image from a registry.

Symptoms

- Error: image not found, unauthorized, or timeout
- CI/CD pipeline halts
- Local build fails at docker pull step

Diagnostics

- Check Docker Hub or registry URL availability
- Inspect network/DNS configuration
- Confirm image name/tag spelling
- Run docker login to check auth

Causes

- Invalid image tag
- Registry permissions or credentials issue
- Network proxy/firewall blockage
- Image deleted or never existed

Actions

- Validate image name, org, and tag

- Re-authenticate with docker login
- Test access from another host
- Mirror image locally if necessary

Remediation

- Use versioned/tagged images (not latest)
- Host critical images in a private registry
- Automate retries in CI/CD

DevOps/SRE Notes

- Integrate image availability checks in pipeline
- Cache critical images at edge or internally

Runbook: Docker Network Failure

Failure Mode: Containers cannot communicate with each other or external services.

Symptoms

- Services timeout or return connection errors
- ping, curl, or telnet fail inside containers
- docker network inspect shows inconsistencies

Diagnostics

- Check network mode (bridge, host, overlay)
- Use docker network ls and inspect
- Verify firewall or iptables rules
- Use tools like tcpdump for deeper inspection

Causes

- Misconfigured Compose or K8s network policy
- IP address conflicts
- Network plugin errors (CNI or Docker)

- Host-level firewall interference

Actions

- Restart Docker daemon
- Recreate network (docker network rm then create)
- Remove and redeploy affected containers

Remediation

- Define container-specific subnets
- Use DNS-based service discovery
- Monitor container network performance

DevOps/SRE Notes

- Integrate service mesh or overlay networks for scaling
- Centralize network policy management

Runbook: Docker Daemon Crash or Unresponsiveness

Failure Mode: Docker daemon (dockerd) crashes, becomes unresponsive, or fails to restart.

Symptoms

- docker ps, docker start, or docker info hang or error
- Daemon logs show panics or core dumps
- Services fail unexpectedly on host reboot

Diagnostics

- Check daemon logs: /var/log/docker.log
- Run systemctl status docker
- Analyze core dumps or journalctl logs

Causes

- Out-of-memory or kernel panic

- Corrupt Docker metadata or images
- OS-level update incompatible with Docker version
- Filesystem errors on /var/lib/docker

Actions

- Restart Docker service
- Clear corrupted image layers
- Restore Docker from backup metadata
- Isolate cause via strace or debug mode

Remediation

- Keep Docker Engine updated with stable versions
- Back up Docker volumes and metadata
- Use orchestration tools to handle failovers

DevOps/SRE Notes

- Monitor daemon health via system metrics
- Create high-availability clusters

Runbook: Docker Volume Mount Failure

Failure Mode: Containers fail to start or operate correctly due to issues mounting a volume.

Symptoms

- Error: mount denied, no such file or directory, or permission denied
- docker-compose up fails
- Application can't read/write to disk

Diagnostics

- Check volume path on host
- Use docker inspect to confirm mounts

- Review SELinux/AppArmor logs
- Try running without volume to isolate

Causes

- Host path doesn’t exist
- SELinux/AppArmor blocks access
- Mounts conflict with image's internal directory structure
- Wrong volume driver

Actions

- Create missing directory on host with correct permissions
- Test using bind mounts
- Adjust security profiles (e.g., --privileged)

Remediation

- Define volumes properly in docker-compose.yml
- Validate read/write permissions
- Store persistent data outside containers

DevOps/SRE Notes

- Use named volumes for portability
- Automate volume creation and checks in CI

Runbook: Container Restart Loop

Failure Mode: Containers repeatedly start and stop in a loop.

Symptoms

- High restart count in docker ps
- docker logs shows same log segment repeatedly
- docker inspect shows non-zero exit codes

Diagnostics

- Review application logs inside container
- Use docker inspect to check restart policy
- Confirm entrypoint or CMD correctness

Causes

- Application crash or missing dependency
- Environment variable misconfiguration
- Invalid restart policy or script loop

Actions

- Temporarily disable restart policy
- Run image interactively to debug
- Add health checks to prevent auto-restart

Remediation

- Fix application entrypoint
- Improve container readiness detection
- Implement logging/alerting on restarts

DevOps/SRE Notes

- Use exponential backoff in restart policies
- Track container health as SLO metric