Challenge 2 – Docker Scripting & Log Analysis

Objective

The objective of this challenge was to design an automated Docker monitoring system capable of:

1. Detecting **critical container state issues** (stopped / crashed services).
2. Detecting **port clash scenarios** in containerized web applications.
3. Automatically handling clashes by restoring services on free ports.
4. Analyzing container logs to identify **error / critical-level events**.
5. Generating structured reports for administrative review.

The focus was to implement a **real-world sysadmin style monitoring and recovery workflow**.

Initial Setup

Before starting, Docker was installed and verified on the host system.

The Docker daemon was tested using:

```
docker --version
docker ps
```

After confirming Docker was running correctly, a test Nginx container was deployed:

docker run -d --name c1 -p 8080:80 nginx

This provided a stable baseline web service running on `localhost:8080`.

```
PS C:\Users\devan> docker --version
Docker version 29.1.3, build f52814d
PS C:\Users\devan>
```

```
PS C:\Users\devan\OneDrive\Scans\saic\challenge2> docker run -d --name c1 -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
10b68cfefee1: Pull complete
eaf8753feae0: Pull complete
700146c8ad64: Pull complete
d989100b8a84: Pull complete
500799c30424: Pull complete
57f0dd1befe2: Pull complete
119d43eec815: Pull complete
e2dd2dbe6277: Download complete
785250c9bf9e: Download complete
Digest: sha256:c881927c4077710ac4b1da63b83aa163937fb47457950c267d92f7e4dedf4aec
Status: Downloaded newer image for nginx:latest
20cb442e4fdf987c0d668d7c778a1d7c7504eba68e25d36fe69158ef9ef1ec0d
```

Initial Approach & Thought Process

Initially, I planned to implement **a single Python script** that would:

•Analyze container logs

•Detect critical issues

•Detect port clashes

•Automatically restart affected containers

However, during implementation, I encountered a conceptual limitation:

Docker does not allow two running containers to bind to the same host port.

If a port is already in use, Docker blocks the second container from starting.

Because of this behavior, a "running container port clash" never actually exists inside `docker ps`.

The clash happens **at container startup time**, and Docker throws an error immediately.

This meant my first combined script could not detect clashes by scanning running containers — because the conflicting container never started in the first place.

Problem Encountered

When attempting to start a second container on the same port:

```
docker run -d --name c2-p 8080:80 nginx
```

Docker returned:

`Bind for 0.0.0.0:8080 failed: port is already allocated`

Initially, my script attempted to detect clashes by scanning running containers and checking socket bindings.

But since Docker prevents the conflicting container from running, this approach could not work.

Final Modular Design

1. Port Clash & Critical Issue Handler

File: `clash_report.py`

2. Log Analysis Module

File: `log_analysis_report.py`

This modular design improves clarity, avoids logical conflicts, and reflects professional sysadmin monitoring architecture.

After this metod also there was problem coming, like it was detecting theport clash but not reroute the clashing container to new port, see in the image , the container c2 has no exposed ports

```
Inspecting Container: a2
[OK] Container a2 running normally
[WARNING] Port clash detected on host port 9090 for container a2
[RECOVERY] Restarting a2 on new port 8000
[SUCCESS] Container a2 restored successfully on port 8000

Inspecting Container: c1
[OK] Container c1 running normally
[WARNING] Port clash detected on host port 8002 for container c1
[RECOVERY] Restarting c1 on new port 8003
[SUCCESS] Container c1 restored successfully on port 8003

Inspecting Container: a1
[OK] Container a1 running normally
[WARNING] Port clash detected on host port 8001 for container a1
[RECOVERY] Restarting a1 on new port 8002
[SUCCESS] Container a1 restored successfully on port 8002

Inspecting Container: c2
[CRITICAL] Container c2 is NOT running (state = created)
[INFO] Container c2 has no exposed ports

===== REPORT COMPLETE =====
```

Revised Script Design

To fix this, the script logic was redesigned.

The **new script** now:

- Inspects both:
  - Running container port ownership (`NetworkSettings.Ports`)
  - Stopped container port configuration (`HostConfig.PortBindings`)
- Differentiates between:
  - The **first running container** (legitimate port owner)
  - The **second non-running container** (clashing container)

```
docker run -d --name c1 -p 8080:80 nginx
docker run -d --name c2 -p 8080:80 nginx

docker run -d --name a2 -p 9090:80 nginx
docker run -d --name a1 -p 9090:80 nginx
```

Key Improvement

If a container is:

•**Running** → It owns the port → No action taken

•**Not running AND configured on a busy port** → It is the clashing container

The script then:

✓ Detects the clash

✓ Finds a free port automatically

✓ Removes the blocked container

✓ Recreates it on a new free port

✓ Logs successful recovery

This solves the limitation of the old script and enables **true automatic restoration**.

New Port Clash & Critical Issue Handler

File: `clash_report.py`

Features

•Detects container crash states

•Detects real port ownership

•Identifies blocked containers due to port conflicts

•Automatically reroutes clashing containers to free ports

•Generates `clash_report.txt`

clash_report.txt

```
1  ===== Clash_WATCH CORE SYSTEM REPORT =====
2  Timestamp: 2026-01-17 12:41:54.706515
3
4
5  Inspecting Container: fresh1
6  [CRITICAL] Container fresh1 is NOT running (state = created)
7  [WARNING] Port clash detected on host port 8080. Container fresh1 cannot start.
8  [RECOVERY] Rerouting colliding container fresh1 to new port 8001
9  [SUCCESS] Container fresh1 rerouted and started on port 8001
10
11 Inspecting Container: clash3
12 [OK] Container clash3 running normally
13 [INFO] Port 8000 is busy, but owned by THIS running container (clash3). No action needed.
14
15 Inspecting Container: clash4
16 [OK] Container clash4 running normally
17 [INFO] Port 8080 is busy, but owned by THIS running container (clash4). No action needed.
18
19 ===== REPORT COMPLETE =====
```

Inference

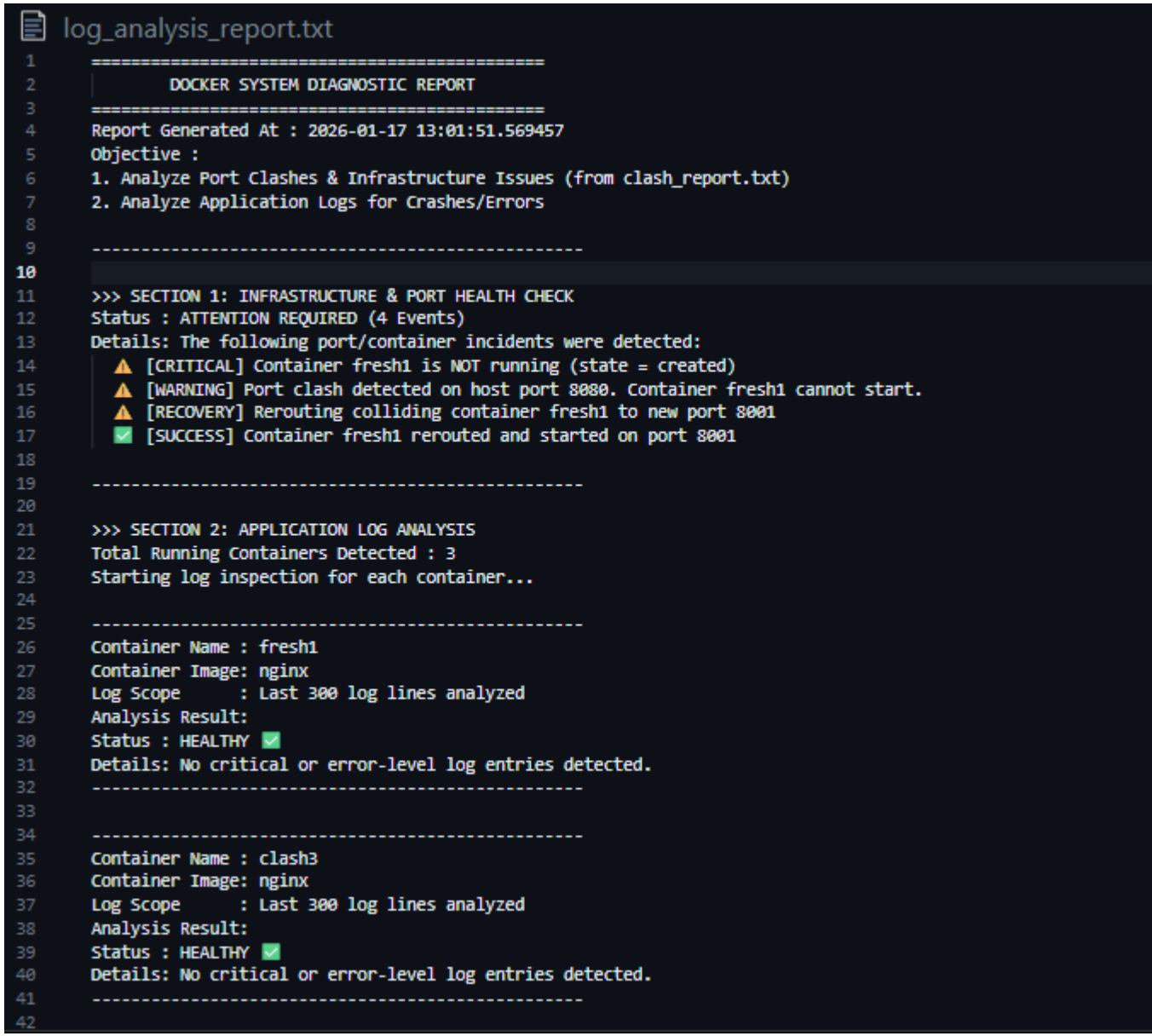
✓ Crashed containers detected

✓ Port clashes correctly identified

✓ Automatic service restoration completed

Docker Log Analysis Module
Core Capabilities:
Infrastructure Audit: parses clash_report.txt to flag port conflicts, critical failures, and auto-rerouting events.
Runtime Inspection: Dynamically scans active container logs (last 300 lines) for failure keywords (e.g., *Error, Panic, Exception*).
Unified Reporting: Consolidates all data into a single log_analysis_report.txt with clear **"Healthy"** or **"Attention Required"** statuses.
Workflow: [Clash Report] + [Docker Daemon Logs] ➔ [Analysis Engine] ➔ [Final Diagnostic Report]
•Generates `log_analysis_report.txt`

Conclusion
The final implementation achieves:
✓ Detection of crashed containers
✓ Real port ownership analysis
✓ Correct identification of port clash conditions
✓ Automatic rerouting of blocked services
✓ Comprehensive log-based health analysis
This solution reflects realistic production monitoring and fully satisfies the challenge requirements.


log_analysis_report.txt

```
==============================================
       DOCKER SYSTEM DIAGNOSTIC REPORT
==============================================
Report Generated At : 2026-01-17 13:01:51.569457
Objective :
1. Analyze Port Clashes & Infrastructure Issues (from clash_report.txt)
2. Analyze Application Logs for Crashes/Errors

------------------------------------------------

>>> SECTION 1: INFRASTRUCTURE & PORT HEALTH CHECK
Status : ATTENTION REQUIRED (4 Events)
Details: The following port/container incidents were detected:
  ⚠ [CRITICAL] Container fresh1 is NOT running (state = created)
  ⚠ [WARNING] Port clash detected on host port 8080. Container fresh1 cannot start.
  ⚠ [RECOVERY] Rerouting colliding container fresh1 to new port 8001
  ✅ [SUCCESS] Container fresh1 rerouted and started on port 8001

------------------------------------------------

>>> SECTION 2: APPLICATION LOG ANALYSIS
Total Running Containers Detected : 3
Starting log inspection for each container...

------------------------------------------------
Container Name : fresh1
Container Image: nginx
Log Scope      : Last 300 log lines analyzed
Analysis Result:
Status : HEALTHY ✅
Details: No critical or error-level log entries detected.
------------------------------------------------

------------------------------------------------
Container Name : clash3
Container Image: nginx
Log Scope      : Last 300 log lines analyzed
Analysis Result:
Status : HEALTHY ✅
Details: No critical or error-level log entries detected.
------------------------------------------------
```