Container Network - Docker Restart Policies Explained

Docker Container Restart Policies

Docker Container Restart Policies
It's often a good idea to run containers with a restart policy. This is a basic form of self-healing that allows the local Docker engine to automatically restart failed containers.

Restart policies are applied per container. They can be configured imperatively on the command line as part of docker run commands or declaratively in YAML files for use with higher-level tools such as Docker Swarm, Docker Compose, and Kubernetes.

In this document, we will get to know about the following restart policies:

always
unless-stopped
on-failure
alt text

Restart Policies Explained
always
The always policy is the simplest. When a container has this restart policy, Docker will automatically restart the container if it stops or encounters an error.

alt text

Demonstration
Start a new interactive container with the --restart always policy and run a shell process:

```
docker run --name neversaydie -it --restart always alpine sh
```

When we started the alpine container, we told it to run the shell(sh). This makes the shell the one-and-only process running inside the container. We can see this by running ps -elf from inside the container. We will see a output similar to this:

```
PID   USER     TIME  COMMAND
  1 root      0:00 sh
  7 root      0:00 ps -elf
```

The first process in the list, with PID 1, is the shell we told the container to run. The second process is the ps -elf command we ran to produce the list. This is a short-lived process that exits as soon as the output is displayed.

Type exit from the shell to kill the container's PID 1 process and stop the container. Docker will automatically restart it because it has the --restart always policy.

Check the container’s status:

```
docker ps
```

We should see that the container is running again.

```
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
9a9fa81a0691   alpine    "sh"      14 minutes ago   Up 5 seconds              neversaydie
```

The container was created 14 minutes ago but has only been up for 5 seconds. This is because the exit command killed it, and Docker restarted it.

You will see a output similar to this:

alt text

Be aware that Docker has restarted the same container and not created a new one. If we inspect it with docker inspect, we can see the restartCount has been incremented.

```
docker inspect neversaydie --format '{{.RestartCount}}'
```

alt text

An interesting feature of the --restart always policy is that if we stop a container with docker stop and then restart the Docker daemon, the container will be restarted.

To illustrate:

Start a new container with the --restart always policy and intentionally stop it with the docker stop command.
The container will be in the Stopped (Exited) state.
Restart the Docker daemon, and the container will be automatically restarted when the daemon comes back up.
unless-stopped
The main difference between the always and unless-stopped policies is that containers with the --restart unless-stopped policy will not be restarted when the daemon restarts if they were in the Stopped (Exited) state.

alt text

Example
Create two new containers:

```
docker run -d --name always --restart always alpine sleep 1d
docker run -d --name unless-stopped --restart unless-stopped alpine sleep 1d
```

Verify both containers are running:

```
docker ps
```

we will see output similar like this:

```
CONTAINER ID   IMAGE     COMMAND      CREATED          STATUS          PORTS     NAMES
b69a8c552269   alpine    "sleep 1d"   47 seconds ago   Up 46 seconds             unless-stopped
f1b6f3e6a555   alpine    "sleep 1d"   47 seconds ago   Up 46 seconds             always
```

Stop both containers:

```
docker stop always unless-stopped
```

Verify both containers are stopped:

```
docker ps -a
```

we will see output similar like this:

alt text

Restart Docker:

```
systemctl restart docker
```

Check the status of the containers:

```
docker ps -a
```

we will see output similar like this:

alt text

The always container has been restarted, but the unless-stopped container has not.

on-failure
The on-failure policy will restart a container if it exits with a non-zero exit code. It will also restart containers when the Docker daemon restarts, even those that were in the stopped state.

alt text

Demonstration
Use the docker run command with the --restart option set to on-failure.

```
docker run -d --name on-failure --restart on-failure alpine sleep 1d
```

Verify the restart policy:

```
docker inspect on-failure --format '{{.HostConfig.RestartPolicy}}'
```

alt text

explain these for a simple child in a very easiest manner in details with basics also correct any error

and what is the difference between these three policies

Understanding Docker Restart Policies in a Simple Way

Imagine you have a toy robot that sometimes stops working because of low battery, a crash, or because you turned it off. You want to set some rules so the robot knows what to do when it stops. Docker restart policies work in a similar way for containers, which are like small programs running inside Docker.

When a container stops, Docker can automatically restart it depending on the restart policy you set. There are three main restart policies:

1. **Always**

2. **Unless-Stopped**
3. **On-Failure**

Now, let's understand each one with examples.

---

1. Always Restart Policy (`` `always` ``)

This policy means:

> "No matter why the container stops, always start it again!"

Example in Real Life:

Imagine you have a toy car that always restarts itself whenever it stops moving, no matter if it ran out of battery, crashed, or if you turned it off.

How it Works in Docker:

1. You run a container with the `` `always` `` restart policy:

   ```
   scss

   docker run --name neversaydie -it --restart always alpine sh
   ```

2. The container starts running.
3. If you stop the container by typing `` `exit` ``, it will **automatically restart**.
4. Even if you restart your computer or Docker itself, the container will **start again**.

Key Points:

✔️ It restarts the container **for any reason** it stops.
✔️ Even if you manually stop it (`` `docker stop` ``), restarting Docker will bring it back.
✔️ The only way to prevent it from restarting is by removing it (`` `docker rm` ``).

---

2. Unless-Stopped Restart Policy (`` `unless-stopped` ``)

This policy means:

> "Restart the container unless I stopped it myself."

Example in Real Life:

Imagine you have a toy drone that automatically restarts when it crashes, but if you turn it off with a switch, it stays off until you turn it on again.

How it Works in Docker:

1. You start a container with this policy:

```
perl

docker run -d --name unless-stopped --restart unless-stopped alpine sleep 1d
```

2. If the container **crashes or stops unexpectedly**, Docker will **restart it**.
3. However, if you **manually stop** it using:

```
arduino

docker stop unless-stopped
```

   it will **stay stopped**, even if you restart Docker or your computer.

Key Points:

✔️ Restarts automatically **only if it didn't stop manually**.
✔️ If you stop it yourself, it will **not** restart when Docker or the system restarts.
✔️ Useful if you want some containers to run automatically, but still want to control them.

---

3. On-Failure Restart Policy (`on-failure`)

This policy means:

> "Restart the container only if it **crashed with an error**."

Example in Real Life:

Imagine a toy robot that only restarts itself if it crashes. If you turn it off normally, it won't restart.

How it Works in Docker:

1. You start a container with this policy:

```
nginx

docker run -d --name on-failure --restart on-failure alpine sleep 1d
```

2. If the container **stops normally** (e.g., `exit` command), Docker **does not** restart it.
3. If the container **crashes with an error** (a non-zero exit code), Docker **restarts it**.

Key Points:

✔️ Only restarts if the container **stops due to an error**.
✔️ If you stop it manually (`docker stop`), it will **not** restart.
✔️ Useful for programs that may fail due to temporary issues.

Comparison Table of Restart Policies

Policy	Restarts if Crashes?	Restarts if Manually Stopped?	Restarts on Docker Restart?
`always`	✅ Yes	✅ Yes	✅ Yes
`unless-stopped`	✅ Yes	❌ No	❌ No (if manually stopped)
`on-failure`	✅ Yes (only if error)	❌ No	✅ Yes

Which Restart Policy Should You Use?

- Use `always` if you want the container to run all the time, no matter what.
- Use `unless-stopped` if you want the container to restart only if it wasn't manually stopped.
- Use `on-failure` if you only want the container to restart when it **fails due to an error**.

This way, you can control how your Docker containers behave when they stop! 🚀

what is the difference between this:
docker run --name neversaydie -it --restart always alpine sh
and this:
docker run --name neversaydie -it --restart always alpine

Great question! Let's break it down in a very simple way.

You asked about the difference between:

```bash
docker run --name neversaydie -it --restart always alpine sh
```

and

```bash
docker run --name neversaydie -it --restart always alpine
```

✅ What's the Same in Both?

- `--name neversaydie`: Names the container "neversaydie".
- `-it`: Runs the container in interactive mode with a terminal.
- `--restart always`: Sets the restart policy to "always".
- `alpine`: This is the name of the image used (a tiny Linux OS).

❓ So, What's the Difference?

1️⃣ First Command:

```bash
docker run --name neversaydie -it --restart always alpine sh
```

- This **tells the container to run the `sh` command** (the shell).
- Inside the Alpine image, you're saying:
  "Run Alpine and **start a shell** (sh) so I can type commands."

📌 This will open an interactive shell where you can type commands like `ls`, `pwd`, etc.

2️⃣ Second Command:

```bash
docker run --name neversaydie -it --restart always alpine
```

- This runs the Alpine container **without telling it what to do**.
- If you don't specify a command, it uses Alpine's **default command**, which is `/bin/sh` — but it **does not run interactively**, so it usually just **starts and stops** instantly (unless `-it` is used, which it is here, so it may still open a shell).

📌 In practice, **it may still work the same** because Alpine's default command is `sh`. But **you shouldn't rely on the default** unless you're sure.

🧠 Final Verdict:

They both may end up starting a shell inside the Alpine container **because Alpine's default command is `sh`**, but...

✅ The **first version is better and more explicit** — you're clearly saying "run `sh`."
⚠️ The **second version might confuse someone** or behave differently if Alpine ever changes its default command.

So, it's **good practice** to use the first version:

```bash
docker run --name neversaydie -it --restart always alpine sh
```

Let me know if you'd like to test the behavior difference or dig into what "default command" really means!