# Redundancy Testing with Docker

Rehearse the failover and failback of your solution on one computer: the redundant pair you deploy on two servers runs the same way in two Docker containers.



**Version 10.1.6+**

## Overview

The FrameworX runtime runs on .NET 10, and the same installation runs on Windows and on Linux. That portability makes a full hot-standby test possible on the computer you already develop on: two Linux containers, each with its own network address, form a real redundant pair, with the same switch-over, synchronization and replication as two physical servers.

With this setup you can:

- validate the redundancy design of a solution while you build it, not on site;
- repeat the same failure tests after every change to the solution;
- show the behavior to your customer, for example in a factory acceptance test, without reserving two servers.

**Prerequisites**

- Windows with Docker Desktop running Linux containers
- The FrameworX installation of the version you deliver
- A solution set to **Multiplatform** (**Solution  Settings  Target Platform and Product**)

What This Test Covers
How the Pair Is Built
Set Up the Pair
Watch the Pair
What to Test in Your Solution
Troubleshooting
Stop and Reset
Related Pages

## What This Test Covers

The containers run the real runtime, so the redundancy logic under test is the one you deploy. Some parts of a field installation are not in the picture, and they still need a test on the target servers.

| Covered on one computer | Still to test on the target servers |
|---|---|
| • Pair formation, from the redundancy settings of your solution<br>• Failover after a crash, a power loss, a frozen server or a network loss<br>• Failback, automatic or commanded<br>• Tag values and retentive values on the new active server<br>• Script tasks, alarms and historian across a switch<br>• Updates of the solution on the standby server | • Real network latency and firewalls between the two servers<br>• Device communication, when the field devices are not reachable from the containers<br>• Windows-only features: a solution with a Windows target does not run in a Linux container<br>• The licensed production configuration: an unlicensed runtime runs in Evaluation mode, 4 hours per start |

## How the Pair Is Built

Two runtimes on the same Windows computer cannot form a pair, because both would listen on the same address. Docker solves it with a user-defined bridge network. Each container gets a fixed address on that network, `172.20.0.5` for the primary and `172.20.0.4` for the secondary. Each address is local to its own container, which is the condition for a node to take its role in the pair. See Redundancy in Docker Bridge Networks.

Each container runs two processes from your installation:

- the Solution Center service, through which the standby downloads the active server's solution;
- the runtime, started with the redundancy parameters.

On your Windows computer, the two runtimes answer on ports `3101` (primary) and `3102` (secondary).

## Set Up the Pair

### 1. Prepare the Solution

In **Runtime  Startup**, enable redundancy with IP1 `172.20.0.5` and IP2 `172.20.0.4`, both on port `3101`. Set Main Port to `3101` and clear **Auto Ports**. These container addresses are for the test: on site, IP1 and IP2 are the two servers.

## 2. Build an Image from Your Installation

The image copies the `net10.0` and `fx-10\net10.0` folders of your installation, so the pair runs exactly the version you deliver. Save the two files below in an empty folder.

**Dockerfile**

```
FROM mcr.microsoft.com/dotnet/runtime:10.0-noble
RUN apt-get update && apt-get install -y --no-install-recommends tini curl procps libfontconfig1 \
 && rm -rf /var/lib/apt/lists/*
COPY --from=fxroot net10.0/       /app/FactoryStudio/
COPY --from=fxroot fx-10/net10.0/ /app/FactoryStudio/fx-10/
COPY entrypoint.sh /app/entrypoint.sh
RUN sed -i 's/\r$//' /app/entrypoint.sh && chmod +x /app/entrypoint.sh \
 && mkdir -p /app/Documents/FrameworX/Solutions /app/Documents/FrameworX/MachineSettings
EXPOSE 10108 3101
HEALTHCHECK --interval=10s --timeout=3s --start-period=20s CMD curl -fsS http://127.0.0.1:10108/health ||
exit 1
WORKDIR /app/FactoryStudio
ENTRYPOINT ["/usr/bin/tini", "--", "/app/entrypoint.sh"]
```

**entrypoint.sh**

```
#!/bin/bash
# The standby downloads the active server's solution through the Solution Center service,
# which serves only the folders listed in SearchPaths.txt.
SP=/app/Documents/FrameworX/MachineSettings/SearchPaths.txt
grep -qsF "/app/Documents/FrameworX/Solutions/," "$SP" || echo "/app/Documents/FrameworX/Solutions/,True,
True" >> "$SP"
exec dotnet /app/FactoryStudio/TWebServices.dll /docker /keepRunning
```

On Linux the installation has no `net10.0` subfolder: the Solution Center service goes to `/app/FactoryStudio` and the runtime to `/app/FactoryStudio/fx-10`. With this layout, the product's Documents folder is `/app/Documents/FrameworX`, which holds Solutions, MachineSettings and TraceLogs.

In that folder, build the image, pointing `fxroot` at the FrameworX folder of your installation:

```
docker buildx build --load --build-context fxroot="<installation folder>\FrameworX" -t fx-pair:10.1.6 .
```

## 3. Start the Two Containers

Save the Compose file in the same folder and start it. Each node keeps its solution, databases and trace files in its own volume.

**docker-compose.yml**

```
services:
  fx-primary:
    image: fx-pair:10.1.6
    container_name: fx-primary
    hostname: fx-primary
    networks: { fxred: { ipv4_address: 172.20.0.5 } }
    ports: ["3101:3101", "10109:10108"]
    volumes: ["fx-primary-docs:/app/Documents"]
  fx-secondary:
    image: fx-pair:10.1.6
    container_name: fx-secondary
    hostname: fx-secondary
    networks: { fxred: { ipv4_address: 172.20.0.4 } }
    ports: ["3102:3101", "10110:10108"]
    volumes: ["fx-secondary-docs:/app/Documents"]
networks:
  fxred:
    name: fxred
    ipam: { config: [ { subnet: 172.20.0.0/16 } ] }
volumes:
  fx-primary-docs:
  fx-secondary-docs:
```

```
docker compose up -d
docker ps --filter name=fx-   # both containers show (healthy)
```

### 4. Copy the Solution and Start the Pair

Both nodes run the same solution file. Copy it to both, start the primary, then the secondary:

```
docker cp .\Plant1.dbsln fx-primary:/app/Documents/FrameworX/Solutions/Plant1.dbsln
docker cp .\Plant1.dbsln fx-secondary:/app/Documents/FrameworX/Solutions/Plant1.dbsln

$fxArgs = '/solution:/app/Documents/FrameworX/Solutions/Plant1.dbsln /profile:1 /redundancy /ip1:172.20.0.5
/port1:3101 /ip2:172.20.0.4 /port2:3101 /connectiontimeout:3 /SaveLog:3'
docker exec -d -w /app/FactoryStudio/fx-10 fx-primary   sh -c "sleep infinity | dotnet TStartup.dll $fxArgs"
Start-Sleep 20
docker exec -d -w /app/FactoryStudio/fx-10 fx-secondary sh -c "sleep infinity | dotnet TStartup.dll $fxArgs"
```

- When the runtime is started from the command line, the redundancy parameters go on that command line. Selecting the profile alone does not make the node join a pair.
- `/SaveLog:3` writes each node's trace to disk. This is how you read what the standby does.
- `sleep infinity` keeps the launcher running in the background.
- You can also upload the solution with Solution Center: connect to `http://localhost:10109` for the primary and `http://localhost:10110` for the secondary.

---

## Watch the Pair

Open Property Watch and Trace Window from the `fx-10` folder of your installation, with `/redundancy` and both node addresses. The tools then follow the active node across every switch, as an operator client does. Without `/redundancy`, they stay on the first address after a failure:

```
& "<installation folder>\FrameworX\fx-10\PropertyWatch.exe" /redundancy /ip1:127.0.0.1 /port1:3101 /ip2:
127.0.0.1 /port2:3102
& "<installation folder>\FrameworX\fx-10\TraceWindow.exe"   /redundancy /ip1:127.0.0.1 /port1:3101 /ip2:
127.0.0.1 /port2:3102
```

A standby server accepts no clients, so the tools always show the active node. The standby's side is in its trace files. The redundancy detail is in `ExecutionLog_TServer_*.txt`, in the `Debug_*` folder of each start. The files are UTF-16:

```
docker exec fx-secondary sh -c 'cd /app/Documents/FrameworX/TraceLogs; f=$(ls -t Debug_*
/ExecutionLog_TServer_*.txt | head -1); iconv -f UTF-16 -t UTF-8 "$f" | grep -E "Pair connected|Activated"'
```

---

## What to Test in Your Solution

Work through the table on your own solution. Each row is a question your customer will ask about the redundant system.

| Test | How | Expected |
|---|---|---|
| **The pair forms** | Start both nodes as in the setup. | In Property Watch, `Server.Redundancy.IsStandByActive` is `True`. The standby's trace shows `Pair connected` from `172.20.0.5` and `Activated: False`. |
| **Takeover after a failure** | Inject one failure on the active node (table below). | The standby becomes active within seconds. A process crash is detected fastest; a frozen server or a network loss takes longer, because the standby waits for the connection timeout. |
| **Values continue** | Before the failure, note a few tag values, a retentive tag and a counter that a script increments. | The new active node continues from the synchronized values. Nothing restarts from zero. |
| **Scripts run once** | Watch a script task that writes a tag or a log line. | It runs on the active node only, and resumes on the new active node after the switch. |
| **Alarms and history** | Raise an alarm and log historian values across the switch, with alarm and historian replication enabled. | The new active node has the alarm state and the history recorded before the switch. |
| **Clients follow** | Keep an operator client open during the failure, started with `/redundancy` and both addresses, for example `TRichClient.exe /redundancy /ip1:127.0.0.1 /port1:3101 /ip2:127.0.0.1 /port2:3102`. | The client connects to whichever node is active, and reconnects to the new active node on its own within seconds. Without `/redundancy`, the client stays on the first address. |
| **Failback** | Bring the failed node back (table below). | After a restart, the primary joins as standby, and `@Server.Redundancy.SwitchToStandby()` on the active node returns the active role to it. After a freeze or a network loss, the primary keeps the active role when it is reachable again, and the secondary steps back to standby. |

### Failures to Inject

| Failure on site | Command | Bring it back |
|---|---|---|
| Runtime crash | `docker exec fx-primary pkill -KILL -f /fx-10/` | Start the primary again (step 4). |
| Power loss | `docker kill fx-primary` | `docker start fx-primary`, then step 4. |
| Frozen server | `docker pause fx-primary` | `docker unpause fx-primary` |
| Network cable unplugged | `docker network disconnect fxred fx-primary` | `docker network connect --ip 172.20.0.5 fxred fx-primary` |

⚠ A server takes part in a switch only after one minute has passed since it started, rejoined its partner or switched. A switch requested earlier is not carried out, and no message says so. If a failback seems to do nothing, wait for the minute and request it again.

---

## Troubleshooting

| Symptom | Likely Cause | Next Step |
|---|---|---|

| | | |
|---|---|---|
| Both nodes stay active, and nothing synchronizes. | The redundancy parameters were dropped at start, because IP1 and IP2 are not the container addresses. | Run `docker exec fx-secondary ps -eo args`: the `TServer.dll` line must carry `/redundancy /ip1:172.20.0.5 /ip2:172.20.0.4`. Correct IP1 and IP2 in the solution. |
| Both nodes report active for a few seconds after a freeze or a network loss. | Expected: the primary comes back while the secondary has taken over. | Wait. The secondary steps back to standby once the pair reconnects. |
| A commanded switch does nothing. | The one-minute wait after a start, rejoin or switch. | Request it again after the minute. |
| The solution does not start in the container. | The solution has a Windows target. | Set it to Multiplatform and copy it to both nodes again. |
| Property Watch or a client does not connect. | Main Port differs from 3101, or Auto Ports moved the port. | Set Main Port to 3101 and clear Auto Ports. |
| The runtime stops after 4 hours. | Evaluation mode. | Start it again, or license the containers from Solution Center. |

## Stop and Reset

- `docker compose stop` stops both nodes and keeps the solutions and databases in the volumes.
- `docker compose down -v` removes the containers and the volumes, so the next test starts from empty nodes.

## Related Pages

- Redundancy Reference: the redundancy model, settings and startup parameters.
- Redundancy in Docker Bridge Networks: addressing rules for each Docker network mode, including pairs across two hosts.
- Docker Deployment Guide: running the runtime in containers in production.

**In this section...**