Investigate a failing Linux application with systemd

Identify the service and preserve its evidence

A systemd restart policy can recover an application process after some failures. It cannot keep a powered-off VPS running, guarantee recovery from memory pressure or fix an invalid configuration. Start with the failing service and its first useful error. The examples use `myapp.service`; replace that name with the confirmed unit, not a guessed application label.

```
systemctl status myapp.service --no-pager -l
systemctl cat myapp.service
sudo journalctl -u myapp.service --since "1 hour ago" --no-pager
```

Record the failure time, exit result and application message. Unit files and logs can contain tokens or connection strings, so redact them before sharing. If the journal is not persistent, older boot evidence may be unavailable; a missing entry does not prove that nothing happened.

Separate configuration, dependencies and capacity

Check the executable path, working directory, service user and application configuration against the unit. A missing environment file or inaccessible directory often causes an immediate exit. Verify required databases and local ports independently rather than repeatedly restarting the whole VPS.

Review available disk and memory. A process killed by the kernel needs a capacity investigation, not an endless restart loop. If the whole guest unexpectedly restarted, collect boot and kernel evidence separately. Avoid changing ownership recursively across system directories or replacing a

vendor unit with an unrelated internet example.

Use a small override when a restart policy is appropriate

Back up the existing configuration, confirm a maintenance window and check how the application saves data during shutdown. For a long-running service that should retry after failure, open an override with:

```
sudo systemctl edit myapp.service
```

An example bounded policy is:

```
[Unit]
StartLimitIntervalSec=300
StartLimitBurst=5

[Service]
Restart=on-failure
RestartSec=10s
```

This allows a limited number of starts within the specified interval. It does not make five attempts a universal recommendation. Choose limits for the actual workload and keep failure alerts enabled. A deliberate `systemctl stop` does not trigger automatic restart through this policy.

Apply, test and keep a rollback

After saving, reload unit definitions with `sudo systemctl daemon-reload`. Inspect the resulting configuration before a planned `sudo systemctl restart myapp.service`. Confirm status, recent logs and a real application request. Do not test recovery by killing a production database or game process while it is writing.

If start limiting is reached, investigate the cause first. Once repaired, `sudo systemctl reset-failed myapp.service` clears the failed state and rate counter so a controlled start can proceed. It does not repair the application.

To undo this change, restore or remove only the override you added, keeping any pre-existing overrides. Reload definitions and perform a planned restart. Enabling a service at boot is a

separate decision from configuring restart behaviour. Keep an external check of application availability so that a service marked active is not mistaken for a working website or game.

For service access, see VPS access and console recovery. When contacting us, follow the support-ticket checklist.

Further reading

- systemd service restart behaviour
- systemd unit start limits
- Journal inspection

Revision #2
Created 2026-11-08 21:45:13 UTC by CloudExa Hosting Admin
Updated 2026-11-08 21:45:13 UTC by CloudExa Hosting Admin