

Эксплуатация: systemd и бэкапы

Таймеры вместо cron и бэкапы, которые проверяют себя сами

gegemonia.com

К cron у меня нет претензий идеологического характера: он работает и никуда не денется. Но на серверах, где уже есть systemd, таймеры дают три вещи бесплатно.

Состояние видно без чтения логов

systemctl list-timers показывает, когда задача запускалась и когда запустится в следующий раз. С cron для этого нужно держать в голове crontab -l и верить, что последний запуск вообще был.

```
$ systemctl list-timers --all | grep backup
Tue 2026-01-13 03:20:00 MSK 8h left Sun 2026-01-12 03:20:11 MSK 15h ago backup.timer backup.se
```

Пропущенный запуск не теряется

Если сервер был выключен в момент запуска, cron молча пропускает задачу. Таймер с Persistent=true догоняет её при загрузке — для ночных бэкапов и ротации это ровно то поведение, которое нужно.

Юнит — это файл, а не строка

```
# /etc/systemd/system/backup.service
[Unit]
Description=Nightly database backup
After=network-online.target postgresql.service

[Service]
Type=oneshot
User=backup
ExecStart=/usr/local/bin/backup.sh
Nice=10
IOSchedulingClass=idle
TimeoutStartSec=2h
```

```
# /etc/systemd/system/backup.timer
[Unit]
Description=Run backup nightly

[Timer]
OnCalendar=*-*-* 03:20:00
Persistent=true
RandomizedDelaySec=20m

[Install]
WantedBy=timers.target
```

RandomizedDelaySec — недооценённая строка. Двадцать минут разброса спасают от ситуации, когда десять серверов начинают бэкап в одну и ту же секунду и забивают канал.

Где cron всё ещё проще

Разовые задания на пользователя и всё, что живёт вне системы: @reboot-скрипты, задачи в контейнерах без systemd, машины, где systemd нет вообще. Переписывать их ради единообразия смысла нет.

Ещё одна деталь: таймер не запускает следующий экземпляр, пока работает предыдущий. Для бэкапа это защита от наложения, но если задача может зависнуть — ставьте TimeoutStartSec, иначе таймер тихо перестанет срабатывать навсегда.

Классическая история: три года бэкапы снимаются, места занимают прилично, а в день аварии выясняется, что последний архив битый с прошлой весны. Никто не заметил, потому что никто не восстанавливал.

Правило, к которому я пришёл: **каждый бэкап должен быть восстановлен автоматически, до того как его сочтут успешным.**

<figure>
</figure>

Схема

- Ночью снимается pg_basebackup или логическая копия.
- Сразу после — восстановление в отдельный каталог на той же машине (или на дешёвой резервной).
- Запуск PostgreSQL на нестандартном порту, три запроса на проверку.
- Если любой шаг упал — алерт, а не тишина.

```
#!/bin/bash
set -euo pipefail

DUMP=/var/backups/pg/latest.dump
pg_dump -Fc -Z6 -d app -f "$DUMP.new"
mv "$DUMP.new" "$DUMP"

TESTDB=verify_$(date +%s)
createdb "$TESTDB"
pg_restore -d "$TESTDB" --no-owner --exit-on-error "$DUMP"

COUNT=$(psql -tA -d "$TESTDB" -c "select count(*) from users")
if [ "$COUNT" -lt 1000 ]; then
    echo "verify failed: users=$COUNT" >&2
    exit 1
fi
dropdb "$TESTDB"
```

--exit-on-error в pg_restore — ключевой флаг. Без него битый дамп восстанавливается «частично», с кучей сообщений об ошибках и кодом возврата 0. Именно так бэкапы и умирают незаметно.

Что проверять, кроме «команда не упала»

- **Свежесть.** Дата внутри дампа должна быть вчерашней. Файл, который не обновлялся неделю, обычно означает, что упал таймер.
- **Объём.** Резкое падение размера на 90% — почти всегда признак того, что дамп снялся с пустой базы.
- **Наличие данных в ключевых таблицах.** Порог ≥ 1000 строк ловит случай «восстановили схему без данных».

Отдельная боль: права

Восстанавливать надо тем же пользователем, который будет читать базу после аварии, иначе половина грантов потеряется. Мы держим `pg_dumpall --globals-only` отдельным файлом и восстанавливаем его первым — это две строки, которые экономят час ручной работы в самый неподходящий момент.

И последнее: проверка должна идти **на другой машине**. Восстановление на том же диске, который утром отвалится, проверяет только вашу веру в SAN.