

Инфраструктура

- [Мониторинг и автоматизация серверов: infra](#)
- [Телеметрия использования: usage_telemetry](#)

Мониторинг и автоматизация серверов: infra

Для разработчиков. Состояние на 28.09.2026. Подробности, переменные ролей и команды — README репозитория `infra` и `files/heartbeat/README.md`. Как считается, кто и чем пользуется, — статья «Телеметрия использования: `usage_telemetry`».

Коротко

Репозиторий `infra` — единственный источник правды о серверах: инвентарь Ansible, роли базовой настройки, стек мониторинга, правила алертов, дашборды Grafana, статус-страница и служба окон обслуживания. Главное правило проекта: **изменение, сделанное мимо репозитория, не существует**. Накликанное на сервере или в веб-интерфейсе затрётся следующим прогоном роли.

Что	Где в репозитории	Снаружи
Prometheus, Alertmanager, blackbox_exporter, Pushgateway, Grafana	<code>roles/monitoring_server</code> , правила — <code>prometheus/rules/</code>	Grafana — <code>grafana.donula26.ru</code> (вход по логину); остальное наружу не смотрит
Агент метрик на каждом сервере	<code>roles/monitoring_agent</code>	—
Метрики PostgreSQL	<code>roles/postgres_exporter</code>	—
Статус-страница	<code>roles/status_page</code>	<code>status.donula26.ru</code>
Окна обслуживания (админка и MCP)	<code>roles/status_admin</code>	<code>status.donula26.ru/admin/</code> , <code>status.donula26.ru/mcp</code>
Heartbeat периодических задач	<code>files/heartbeat/heartbeat.py</code>	<code>push.donula26.ru</code>
GlitchTip (ошибки приложений)	роли нет; проба и реестр проектов — <code>docs/projects.md</code>	<code>monitoring.donula26.ru</code>
Бэкапы баз, фаервол, базовая настройка, аудит	<code>roles/db_backup</code> , <code>roles/firewall</code> , <code>roles/base</code> , <code>playbooks/audit.yml</code>	—
Коллектор телеметрии использования	<code>roles/telemetry_collector</code>	—

Метрики и алерты

- **Prometheus** опрашивает цели раз в 30 секунд и хранит данные 90 дней. Список серверов для опроса собирается из инвентаря (`inventory/hosts.yml`), поэтому новый сервер появляется в мониторинге только после прогона `monitoring_server`.
- **Сервер изнутри** — `node_exporter` с TLS и basic auth, плюс скрипты для `textfile collector`: состояние контейнеров (статус, число перезапусков, `healthcheck`, время старта), метрики хоста, возраст свежих бэкапов. `cAdvisor` не используется, поэтому CPU и памяти по отдельным контейнерам в метриках нет.
- **Снаружи** — `blackbox_exporter` проверяет публичные адреса из `monitoring_probe_targets` (`inventory/group_vars/all/main.yml`) и срок действия TLS-сертификатов. По умолчанию модуль `http_2xx`, для остальных случаев модуль задаётся явно. Пробы бьют в `/health`, где он есть.
- **Базы** — `postgres_exporter` у `amo_db` и у баз мобильного оператора: соединения, архивация WAL, отставание синхронизации.
- **Правила** — `prometheus/rules/*.yaml`: `hosts`, `containers`, `blackbox`, `backups`, `amo_db`, `umobile`, `contur_transcribe`, `meta` (сам мониторинг и `heartbeat`). В Grafana алерты не заводятся.

Alertmanager шлёт уведомления в Telegram через прокси (напрямую `api.telegram.org` недоступен), в два чата:

Алерт	Куда и когда
<code>severity: critical</code>	В критический чат сразу, с 08:00 до 23:00 по Москве. Если сработал ночью, придёт в 08:00, при условии что проблема ещё держится. Повтор раз в час.
<code>severity: warning</code> + <code>delivery: prompt</code>	В тот же чат, днём сразу, ночью придерживается до утра. Повтор раз в 6 часов. Метка для того, что не срочно, но нужно узнать в тот же день.
Остальные <code>warning</code> и <code>info</code>	Утренняя сводка: одно сообщение в окне 08:00–08:20.

Подавления: при `HostDown` молчит всё, что есть на этом хосте. При `DockerDown` и `DockerMetricsStale` молчат контейнерные алерты. При `AmoDbPostgresDown` молчат производные проверки `amo_db`. При `DiskUsageCritical` молчат предупреждения о том же диске. Истёкшие тишины Alertmanager хранит 30 дней: это журнал объявленных работ.

Дашборды Grafana лежат в папке «Инфраструктура»: «Инфраструктура: серверы», «Инфраструктура: контейнеры», «Доступность снаружи», «Периодические задачи», «`amo_db`», «Использование приложений». Их JSON генерируется скриптами `generate.py` и `generate_usage.py` в `roles/monitoring_server/files/dashboards/`.

Правка дашборда в браузере не сохранится: включено `allowUiUpdates: false`. Дашборды меняют в генераторе, потом прогоняют роль.

GlitchTip

GlitchTip на `monitoring.donula26.ru` собирает исключения приложений через `SENTRY_DSN`. Своя запись в нём есть у каждого нашего проекта. Одной переменной `SENTRY_DSN` мало: в коде нужны зависимость `sentry-sdk` и вызов инициализации. Проще всего взять `init_sentry()` из пакета `usage_telemetry`: `install_telemetry()` вызывает её сам.

- Об ошибке в чат приходит уведомление, когда одна и та же ошибка случилась пять раз за десять минут. Разовые ошибки видны только в интерфейсе GlitchTip.
- Uptime-мониторы в GlitchTip есть, но уведомления о доступности от них выключены с 18.09.2026. О том, что сервис недоступен, сообщает только Alertmanager, чтобы одно событие не приходило двумя сообщениями.
- Роли Ansible для GlitchTip нет: проекты и правила уведомлений заводятся в его интерфейсе. Сам GlitchTip проверяется внешней пробой. Эндпоинта здоровья у него нет, поэтому проба смотрит корень сайта.

Статус-страница

`status.donula26.ru` открыта без авторизации и сделана для сотрудников. Это не Uptime Kuma и не отдельный монитор: страницу генерирует скрипт `roles/status_page/files/generate.py`. Он запускается по таймеру раз в минуту, берёт из Prometheus ту же метрику `probe_success`, на которой построены алерты, и пишет статический HTML. Поэтому страница не может показывать не то, что пишут в чат алерты.

Список сервисов задаётся в `status_page_services`. Поля: `probe` — адрес пробы, как в `monitoring_probe_targets`; `title` — название, понятное сотруднику; `visible` — показывать ли на странице (инфраструктурные сервисы проверяются, но скрыты); `project` и `job` на странице не видны, по ним окно обслуживания глушит алерты. Внутренние имена, адреса и версии на страницу не попадают.

Состояние	Когда
Работает	проба успешна, и за сутки вне окон обслуживания сбоев не было
С перебоями	сейчас отвечает, но за сутки были сбои; или падает, но меньше двух минут подряд
Не работает	все пробы за последние 2 минуты неуспешны (<code>status_page_down_window</code>)
На обслуживании	сейчас идёт объявленное окно; перекрывает все остальные состояния
Нет данных	по пробе нет метрик, например недоступен Prometheus

У каждого сервиса есть полоса за сутки из 48 получасовых сегментов и процент доступности за сутки. Минуты внутри окон обслуживания в процент не входят.

Окна обслуживания

Любой деплой, обновление или фикс сервиса, у которого есть внешняя проба, идёт в окне обслуживания. Одна запись об окне делает три вещи:

- на статус-странице сервис становится синим, с бейджем «На обслуживании», а причина работ видна в подсказке к сегменту;
- его сбои во время окна не портят процент за сутки;
- в Alertmanager заводятся тишины на все имена сервиса: `instance` (проба), `project` (контейнеры) и `job` (отметки heartbeat). Алерты по сервису в чат не приходят.

Порядок работ: открыть окно, выкатить и проверить, **сразу** закрыть окно. Пока окно открыто, мониторинг сервиса слепой, поэтому лучше открыть короткое окно и продлить его, чем забыть длинное. Причину пишут человеческим языком, её видит любой сотрудник: «Обновление дашборда», а не «деплой v2.3.1».

Через админку

1. Открыть `status.donula26.ru/admin/`. Вход по именному логину: имя вошедшего записывается автором окна.
2. Выбрать «Сервис», заполнить «Причина», проверить «Начало» и «Конец». По умолчанию окно идёт от текущего момента на 30 минут.
3. Нажать «Объявить работы».
4. После работ нажать «Закончить сейчас» в строке окна. Конец окна переносится на текущий момент, тишины снимаются, запись остаётся в истории.

Кнопка «Убрать» удаляет запись целиком, вместе с синим сегментом. Она нужна только для окон, заведённых по ошибке. В таблице видны окна за последние трое суток и все предстоящие.

Через MCP

Та же служба отвечает по MCP на `https://status.donula26.ru/mcp`. Имя сервера — `status-maintenance`, авторизация — заголовок `Authorization: Bearer <токен>`. Токен у каждого разработчика свой. Пример подключения в Claude Code:

```
claude mcp add --transport http status-maintenance https://status.donula26.ru/mcp \
--header "Authorization: Bearer $STATUS_MCP_TOKEN"
```

Инструмент	Параметры	Что делает
<code>list_maintenance</code>	<code>hours</code> (по умолчанию 72)	идушие, предстоящие и недавние окна с их <code>id</code>
<code>open_maintenance</code>	<code>service</code> , <code>minutes</code> (не больше 120), <code>reason</code> , необязательный <code>start</code> в ISO по Москве	открывает окно; по умолчанию оно начинается сейчас
<code>close_maintenance</code>	<code>id</code>	закрывает окно сейчас и снимает тишины

- В `service` передают название со статус-страницы или адрес пробы. Если название подходит нескольким сервисам (например, «МПП2» — и дашборду, и виджету), служба возвращает ошибку со списком подходящих, а не выбирает первый.
- Автором окна записывается «помощник <владелец токена>».
- Окно длиннее двух часов через MCP не открыть: такие работы объявляются вручную в админке.

Доступы задаются в Ansible Vault: `status_admin_web_users` (логины админки) и `status_admin_mcp_tokens` (токены MCP), затем прогоняется `playbooks/status_admin.yml`. Чтобы отозвать доступ у одного человека, его строку удаляют и снова прогоняют роль. Остальные доступы продолжают работать.

Heartbeat

Проблема: Celery Beat или APScheduler умирают, не роняя контейнер. Процесс жив, healthcheck зелёный, проба отвечает 200, а задача не запускалась трети сутки. Такое видит только сама задача, поэтому после каждого успешного прогона она отправляет отметку в Pushgateway.

- **Подключение.** Скопировать `files/heartbeat/heartbeat.py` в проект (зависимостей нет) и вызвать одну функцию: `install_celery(celery_app, job="...")`, `install_apscheduler(scheduler, job="...")` или `ping(job=..., task=..., max_age_seconds=...)` для разовой отметки. Сейчас так подключены шесть проектов.
- **Переменные:** `HEARTBEAT_URL` (<https://push.donula26.ru>; пустое значение выключает отправку — так работает локальная разработка), `HEARTBEAT_USER`, `HEARTBEAT_PASSWORD` (лежит в Vault), `HEARTBEAT_INSTANCE` (имя источника; для контейнеров его задают явно).
- **Порог** задача объявляет сама метрикой `heartbeat_max_age_seconds`. По умолчанию это полтора интервала расписания плюс пять минут, отдельные задачи переопределяют его через `max_age_overrides`. Правило `HeartbeatMissing` (critical) срабатывает, если отметки нет дольше порога.
- Отметку отправляет воркер после успешного выполнения. Упавшая задача отметку не обновляет, и молчание становится сигналом. Ошибки отправки пишутся в лог и

не роняют задачу.

Ansible: как выкатывать

- Запуск — с ноутбука разработчика по ssh, Ansible стоит в `.venv` внутри проекта (`pip install -r requirements.txt`). Секреты лежат в `inventory/group_vars/all/vault.yml` под Ansible Vault. Пароль от хранилища — в файле `.vault_pass`, в git он не попадает, копия есть в VaultWarden.
- Порядок любого прогона: `--check --diff`, затем `--limit` на одном хосте, затем на всех.
- Новый сервер: завести в `inventory/hosts.yml`, затем прогнать `base` → `firewall` → `monitoring_agent` → `monitoring_server`. Если фаервол настроить после агента, агент не создаст правило для своего порта.
- Службу окон обслуживания ставят до nginx: сначала `playbooks/status_admin.yml`, потом `playbooks/web_proxy.yml`.
- `playbooks/audit.yml` только читает: что запущено на серверах и что требует внимания. Результат попадает в `reports/`, этот каталог в git не коммитится.
- Работы на сервисе с пробой проводятся в окне обслуживания, см. выше.

На что наступают

- **У сервиса три имени, тишина нужна на каждое.** Если у сервиса в `status_page_services` не заполнены `project` и `job`, окно заглушит только пробу, а контейнеры и heartbeat продолжают писать в чат. Лишнее имя тоже вредно: чужой `project` заглушит соседний сервис. Заполняют только то, что сверено с метками в Prometheus.
- **Сервиса нет в пробах — окно открыть не на что.** Сначала сервис заводят в `monitoring_probe_targets` и `status_page_services`, только потом объявляют для него работы.
- **Окно, заведённое задним числом,** красит историю, но тишины не создаёт: Alertmanager не принимает тишины в прошлом.
- **Обслуживание не прячет соседнюю аварию.** Сегмент на полосе синее, только если все неудачные минуты получаса попали внутрь окна.
- **Heartbeat после деплоя молчит до первого прогона.** Правило отсчитывает возраст от более позднего из двух событий: последней отметки или последнего старта контейнеров проекта. Это работает, только если `job` отметок совпадает с именем compose-проекта. Проект с другим именем заалертит сразу после деплоя.
- **Переехал проект — поправьте `HEARTBEAT_INSTANCE`** и удалите старую группу из Pushgateway. То же с задачей, убранной из расписания: Pushgateway помнит отметки бесконечно, и старая группа будет держать алерт.

- **Порог heartbeat считается при старте процесса.** Новое расписание подхватится только после перезапуска контейнера.
- **Лимит docker-логов не действует сразу.** Роль `base` пишет `daemon.json`, но нужен полный перезапуск docker (в окне обслуживания) и пересоздание контейнеров.
- **Docker публикует порты в обход ufw.** Роль `firewall` закрывает этот обход. Порт контейнера, который должен смотреть наружу, открывают маршрутным правилом через `firewall_route_rules`. В файлах правил ufw пишите комментарии только латиницей, иначе ufw падает.
- **Ansible ставят только в .venv.** Фильтр `password_hash` не работает (`passlib` несовместим с `bcrypt 5`), поэтому bcrypt-хеши хранятся в Vault готовыми.
- **--check на чистой машине падает** на первой же задаче `apt`: нет `python3-apt`. Первый прогон на пустом сервере делается сразу боевым.
- **С ноутбука ssh может прийти не на тот сервер:** локальный прокси подменяет адрес. Признаки — «Permission denied» с рабочим ключом и предупреждение о смене ключа хоста. Сервер проверяют по `hostname`, а не по тому, что подключение прошло.
- **Хост вне инвентаря не мониторится.** Инвентарь пополняется по мере того, как доступы переносятся в VaultWarden.

Телеметрия

использования:

usage_telemetry

Для разработчиков. Состояние на 28.09.2026. Подробности и переменные — README репозитория `usage_telemetry`. Выкатка коллектора и дашборд лежат в репозитории `infra`, про него — статья «Мониторинг и автоматизация серверов: `infra`».

Коротко

`usage_telemetry` отвечает на вопрос, какими приложениями и какими их разделами пользуются живые люди. Цель — отличить приложение, которое висит мёртвым грузом, от приложения, которым пользуются редко, но по делу, и выключать только первое. Проект состоит из двух частей:

- **пакет** `usage-telemetry`: встраивается в приложение одной строкой и считает обращения;
- **коллектор**: принимает события, складывает их в PostgreSQL и раз в час пересчитывает суточные своды.

Результат смотрят в Grafana, на дашборде «Использование приложений».

приложение (middleware) —пачками по HTTP—> коллектор —> PostgreSQL —> Grafana

Как подключить к проекту

1. Добавить пакет в `requirements.txt` (или в `pyproject.toml`), закрепив тег:

```
usage-telemetry @
https://github.com/Buharestskie/usage_telemetry/archive/refs/tags/v0.2.0.tar.gz
```

Для отправки ошибок в GlitchTip нужен ещё `sentry-sdk` (extra `sentry`).

2. Повесить сбор на приложение FastAPI или Starlette:

```
from usage_telemetry import install_telemetry

app = FastAPI()
install_telemetry(app)
```

Если задан `SENTRY_DSN`, `install_telemetry` заодно инициализирует `GlitchTip`. Если в проекте SDK уже поднят своим кодом, вызывайте `install_telemetry(app, sentry=False)`.

3. Сообщить, кто пришёл. В зависимости авторизации, где сессия или токен уже разобраны:

```
from usage_telemetry import mark_user

mark_user(request, user_id=session.amocrm_user_id, auth="session",
session_id=session_id)
```

Значения `auth`: `session` — человек в браузере; `jwt` — виджет amoCRM с токеном от `widget_auth`; `telegram` — бот или Mini App; `basic` — админки; `api_key` — вызов между сервисами; `none` — без авторизации. Без `mark_user` вид входа угадывается по заголовкам, но идентификатора пользователя не будет.

4. Для бота на aiogram 3:

```
from usage_telemetry import install_bot_telemetry

install_bot_telemetry(dispatcher)
```

5. Добавить переменные в compose проекта и выкатить в окне обслуживания.

Переменная	Обязательна	Что означает
<code>TELEMETRY_SERVICE</code>	да	машинный код приложения. Должен совпадать с <code>project</code> этого сервиса в <code>status_page_services (infra)</code>
<code>TELEMETRY_URL</code>	да	адрес коллектора. Приложения на одном сервере с коллектором используют <code>http://host.docker.internal:8130</code> , с других серверов — <code>https://donula26.ru/telemetry</code>
<code>TELEMETRY_ENABLED</code>	нет	<code>0</code> — выключает сбор целиком
<code>TELEMETRY_QUEUE_SIZE</code> , <code>TELEMETRY_BATCH_SIZE</code> , <code>TELEMETRY_FLUSH_SECONDS</code> , <code>TELEMETRY_TIMEOUT</code>	нет	очередь (5000), размер пачки (200), интервал отправки неполной пачки (5 с), таймаут запроса (3 с)
<code>SENTRY_DSN</code>	нет	проект в <code>GlitchTip</code> ; без него ошибки не отправляются

Если не задан `TELEMETRY_SERVICE` или `TELEMETRY_URL`, пакет пишет предупреждение в лог и ничего не собирает. Приложение при этом работает как обычно.

Что собирается

Одно событие на одно обращение к приложению:

Поле	Что в нём
<code>at</code> , <code>service</code> , <code>source</code>	время, код приложения, источник: <code>http</code> или <code>bot</code>
<code>method</code> , <code>path</code>	метод и шаблон пути из маршрута (<code>/api/deals/{id}</code> , а не <code>/api/deals/451</code>). Если маршрута нет, путь нормализуется: числа заменяются на <code>{id}</code> , UUID — на <code>{uuid}</code> , длинные непрозрачные сегменты — на <code>{token}</code>
<code>status</code> , <code>duration_ms</code>	код ответа и длительность
<code>auth</code> , <code>user_id</code> , <code>session_id</code>	чем запрос аутентифицирован и чей идентификатор из уже существующей авторизации: <code>amocrm_user_id</code> , id из JWT, <code>telegram_id</code>
<code>user_agent</code>	первые 200 символов
<code>client_ip</code>	доходит только до коллектора. Там из него, user-agent и общей соли считается отпечаток посетителя, а сам адрес не сохраняется

- **Не пишется никогда:** строка запроса, тело, referer, имена и почта. В строке запроса ходят разовые токены, а хранилище не должно превращаться в базу утёкших секретов.
- **Отсекается на входе:** `OPTIONS`, `/health`, `/healthz`, `/metrics`, `/ping`, `/favicon.ico`, `/robots.txt`, пути под `/static/`, `/assets/`, `/_next/`, `/_app/` и файлы статики по расширению. Свои исключения добавляются параметрами `skip_paths` и `skip_prefixes`.
- **В ботах** считаются только входящие от людей: сообщения (команда с короткой меткой вроде `/start beeline` или тип вложения) и нажатия кнопок. Данные кнопки сводятся к шаблону: `service:1842` превращается в `service:{id}`. Рассылки не считаются, прочтения Telegram ботам не отдаёт.

Как устроено внутри

- **Приложение ничего не ждёт.** Middleware кладёт событие в очередь в памяти, фоновая задача отправляет пакки в `POST /events`. Если коллектор недоступен, пачка выбрасывается без повторов. Если очередь переполнилась, она сбрасывается

целиком. При остановке приложения остаток отправляется.

- **Коллектор** — FastAPI с PostgreSQL. Сырые события хранятся в таблице `events` 90 дней. Раз в час коллектор пересчитывает своды за последние трое суток и чистит устаревшее сырьё. Сводьы хранятся бессрочно:
 - `daily_users` — человеко-дни: пара «день + человек». В этот свод попадают только обращения с `auth`, `session`, `jwt`, `telegram` или `basic` и с известным `user_id`. Вызовы между сервисами сюда не входят;
 - `daily_paths` — разделы по дням: число обращений, людей, посетителей по отпечатку, ошибок 5xx и 95-й перцентиль длительности.
- **Справочник приложений** (таблица `services`) собирается из `status_page_services` и `telemetry_extra_services` в инвентаре `infra`. Коллектор читает его только при старте. По отметке первого события видно, раскатан ли сбор в приложение.
- **Ручки коллектора:** `POST /events` (ответ `{"accepted": N}`), `GET /health`, `GET /stats` — сколько накоплено и когда был последний свод. Коллектор слушает только внутренний интерфейс сервера. С других серверов события идут через путь `/telemetry/` на donula26.ru, куда пускают только адреса из списка в роли `web_proxy`.

Где посмотреть

Grafana, grafana.donula26.ru → папка «Инфраструктура» → дашборд «Использование приложений». Период по умолчанию — 30 дней.

Панель	Что показывает
Приложений под наблюдением	сколько приложений в справочнике
Сбор не раскатан	приложения, от которых не пришло ни одного события. Пока здесь не ноль, выводы о неиспользуемых делать рано
Без единого обращения за период	сбор работает, а обращений не было — кандидаты на выключение
Приложения по живости	главная таблица: обращения, люди, посетители (оценка), человеко-дни, последний заход и замечание: «сбор не подключён», «никто не заходил», «люди не опознаются» или «только оценка по отпечатку»
Человеко-дни по приложениям	график по дням
Разделы и данные	разделы выбранного приложения (переменная «Приложение»)
Кто пользуется	идентификаторы пользователей выбранного приложения, без имён: нужны, чтобы предупредить людей перед выключением
Медленное и ошибающееся	разделы с ошибками или с 95-м перцентилем больше 3 секунд

Дашборд генерируется скриптом `roles/monitoring_server/files/dashboards/generate_usage.py` в `infra`. В браузере он не редактируется.

Выкатка

- **Коллектор** выкатывается из репозитория `infra`: `ansible-playbook` `playbooks/telemetry_collector.yml`. Роль забирает код из ветки `main` этого репозитория, выпускает самоподписанный сертификат для базы, один раз создаёт соль отпечатка, раскладывает справочник приложений и поднимает compose. Пароль базы берётся из Vault (`telemetry_collector_db_password`), без него роль останавливается. База входит в ежедневные бэкапы.
- **Пакет** приезжает в приложения тегом в зависимостях. Новая версия пакета — это новый тег здесь и правка строки в каждом проекте.
- **Проверки:** `python tests/test_telemetry.py` проверяет пакет без базы и сети. `COLLECTOR_TEST_DSN=... python collector/tests/test_collector.py` проверяет коллектор и требует настоящий PostgreSQL: тест работает в отдельной схеме и удаляет её за собой.

На что наступают

- `TELEMETRY_SERVICE` не совпал с `project` в `status_page_services`. События пишутся, но приложение не попадает в список ожидаемых и на дашборде в таблице живости не видно. Приложениям без пробы на статус-странице код прописывают в `telemetry_extra_services`.
- **После правки справочника в инвентаре** коллектор нужно перезапустить. Роль `telemetry_collector` делает это сама.
- **События теряются осознанно.** Коллектор лежал — в данных дыра. Приложение, которое молчит, и приложение, которым не пользуются, на сводах выглядят одинаково. Различают их по колонке «Замечание».
- **Отпечаток — оценка, а не счёт людей.** Сотрудники из офиса сидят за одним адресом, и различает их только user-agent: одинаковые браузеры схлопнутся в одного посетителя.
- **Приложение на другом сервере шлёт в пустоту,** если адреса этого сервера нет в списке для `/telemetry/` в роли `web_proxy`: nginx отвечает 403, пакет молча выбрасывает события.
- **Без привязки к завершению приложения** последняя пачка теряется при каждом перезапуске. `install_telemetry` привязывается к `lifespan`, а если не получилось, пишет предупреждение в лог. Его не игнорируйте.
- **Не у каждого бота есть Dispatcher.** У односторонних ботов, которые только рассылают, входящих от людей нет, и мерить там нечего.