PR #2303 完整报告

radixark/miles

feat(ci): add authorized Neon SQL workflow

合并时间：2026-08-12 05:05

原文链接：http://prhub.com.cn/radixark/miles/pull/2303

执行摘要

- 一句话：新增授权 Neon SQL 工作流，写权限用户可安全执行任意 SQL
- 推荐动作：值得精读，尤其适合关注“受权临时数据库访问”“避免 secret 暴露”设计的工程师。重点看三点：双重 actor 权限校验（工作流与客户端各自独立执行）、CMS 加密结果交付与 artifact 自清理、执行器对 SQL 的 byte-for-byte 透传与结果截断语义。鉴于该 PR 没有公开 review 讨论，正式用于生产前建议再补充一轮针对权限与故障路径的走查。

功能与动机

PR body 明确说明动机："Run arbitrary SQL against Miles Neon through a write-gated workflow without exposing credentials"，即让有写权限的调用者通过已存在的 Neon secret 执行 SQL，同时避免任何用户接触数据库连接串。该能力用于查询或修复托管的历史指标数据（如 docs/ci/03-metric-history-gate.md 中提到的 hosted metric history），并强调 "Users never handle NEON_DATABASE_URL"。

实现拆解

1. 本地发起端.claude/skills/neon-access/scripts/run_neon_workflow.py：新增 main、pack_sql、_create_recipient、decrypt_result、validate_result、_delete_artifact。脚本读取 SQL 文件，先做 UTF-8 校验再 gzip（mtime=0）+base64 打包并计算 SHA-256；本地用 openssl 生成一次性 RSA 3072 密钥与一日期证书；通过 gh api 核对当前用户为 write/admin 后 dispatch neon-access.yml；随后 watch 运行、下载加密 artifact、本地解密并按 request_id/run_id/actor/sql_sha256 校验身份，最后删除远端 artifact。
2. 工作流授权层.github/workflows/neon-access.yml：新增 workflow_dispatch 入口，authorize job 先要求运行在 default branch，再对 github.actor 与 github.triggering_actor 双 actor 逐一调用 collaborators permission API，强制 write/admin；execute job 使用固定 SHA 的 checkout 做 sparse checkout，只检出执行器脚本，安装 psycopg[binary]==3.3.4，在 NEON_DATABASE_URL 环境下执行 SQL，结果用 openssl cms -aes256 加密后上传 artifact，continue-on-error 保证失败路径也返回加密结果，最后失败时显式 exit 1。
3. 远端执行器.github/workflows/scripts/neon_access_job.py：新增 decode_sql、encode_value、_base_result、execute_request、main。执行器从环境变量取参，REQUEST_ID 必须是规范 UUID；execute_request 使用 autocommit=True、prepare_threshold=None 一次性提交整段 SQL（不拆分、不改写），对每个结果集按列名

+ 行值编码，bytes/float/ 嵌套结构转为可 JSON 序列化的结构，并累计行字节数实现 truncated 上限；数据库异常不抛出，打包为 status=error 与 sqlstate 返回。

4. 技能与文档.claude/skills/neon-access/SKILL.md、agents/openai.yaml：定义 $neon-access 技能用法，明确边界：不暴露 secret、不分类 SQL、autocommit 语义、结果 4 MiB 上限、不支持 psql 反斜杠命令；docs/ci/03-metric-history-gate.md 增加“Inspect hosted history”小节，指引仓库作者用 $neon-access 查询或修复托管指标历史，保持运行时 store 的 DML-only 边界。
5. 测试与 CI 注册tests/ci/test/test_neon_access.py：新增 6 个离线测试，用 load_module 直接加载工作流脚本与客户端脚本，构造 FakeDriver/FakeCursor/FakeResult 验证 SQL 字节级透传、结果编码与截断、数据库错误打包、SQL 传输往返、CMS 加解密与身份校验、以及 workflow 边界（secret 仅出现一次、无 GITHUB_STEP_SUMMARY、retention-days=1 等）；最后一个提交把测试注册到 stage-a-cpu，修复 CI 收集失败问题。

关键文件：

- .claude/skills/neon-access/scripts/run_neon_workflow.py（模块 客户端；类别 source；类型 core-logic；符号 main, pack_sql, _create_recipient, decrypt_result）：本地发起端核心：SQL 打包、一次性密钥生成、workflow dispatch、结果解密与身份校验、artifact 清理，是整套流程的客户端主路径。
- .github/workflows/scripts/neon_access_job.py（模块 执行器；类别 infra；类型 core-logic；符号 main, decode_sql, encode_value, _base_result）：远端执行器核心：解码 SQL、调用 psycopg 执行任意语句、编码结果行、限制返回字节数、打包数据库错误，是 SQL 语义与防泄露的关键实现。
- .github/workflows/neon-access.yml（模块 工作流；类别 infra；类型 infrastructure）：工作流入口：定义 workflow_dispatch 输入、default-branch 约束、双 actor 权限检查、执行 job 与加密 artifact 上传，是整个授权链路的安全边界。
- tests/ci/test/test_neon_access.py（模块 测试；类别 test；类型 test-coverage；符号 load_module, FakeResult, FakeCursor, FakeConnection）：离线契约测试：用 FakeDriver/FakeCursor 验证 SQL 透传、结果编码与截断、错误打包、CMS 加解密与 workflow 边界断言，并注册到 stage-a-cpu 保证 CI 收集。
- .claude/skills/neon-access/SKILL.md（模块 技能文档；类别 docs；类型 documentation）：定义 agent 技能的使用边界与安全约束：不暴露 secret、不分类 SQL、autocommit 语义、结果上限、不支持 psql 反斜杠命令。
- .claude/skills/neon-access/agents/openai.yaml（模块 技能配置；类别 config；类型 configuration）：技能对外接口声明，供 agent 发现与调用 neon-access 技能。
- docs/ci/03-metric-history-gate.md（模块 文档；类别 docs；类型 documentation）：为 metric-history-gate 补充“通过 neon-access 查询 / 修复托管历史数据”的运维指引，保持运行时 store 的 DML-only 边界。

关键符号：run_neon_workflow.main, run_neon_workflow.pack_sql, run_neon_workflow.validate_result, run_neon_workflow._delete_artifact, neon_access_job.main, neon_access_job.execute_request, neon_access_job.decode_sql, neon_access_job.encode_value

关键源码片段

.claude/skills/neon-access/scripts/run_neon_workflow.py

本地发起端核心：SQL 打包、一次性密钥生成、workflow dispatch、结果解密与身份校验、artifact 清理，是整套流程的客户端主路径。

```
#!/usr/bin/env python3
# 核心常量：仓库、工作流文件名、GitHub API 版本
REPOSITORY = "radixark/miles"
WORKFLOW = "neon-access.yml"
API_VERSION = "2026-03-10"
MAX_DISPATCH_CHARACTERS = 65_535
DEFAULT_MAX_RESULT_BYTES = 4 * 1024 * 1024


def pack_sql(sql_bytes):
    # 先验证 UTF-8 编码，保证 SQL 字节能无丢失地传送到执行端
    sql_bytes.decode("utf-8")
    # gzip 固定 mtime=0，保证相同输入产生相同压缩结果，便于可复现调试
    compressed = gzip.compress(sql_bytes, mtime=0)
    # 同时返回 base64 载荷与 SHA-256 摘要，供执行端回传校验
    return base64.b64encode(compressed).decode("ascii"), hashlib.sha256(sql_bytes).hexdigest()


def validate_result(result, *, request_id, run_id, actor, sql_sha256):
    # 结果身份绑定：五项都必须与本地发起时的值一致，防止结果被调包
    expected = {
        "schema_version": 1,
        "request_id": request_id,
        "run_id": str(run_id),
        "actor": actor,
        "sql_sha256": sql_sha256,
    }
    for key, value in expected.items():
        if result.get(key) != value:
            raise RuntimeError(f"result identity mismatch for {key}")
    # 状态白名单：只接受 ok 或 error，拒绝任何未知状态
    if result.get("status") not in {"ok", "error"}:
        raise RuntimeError("result has an invalid status")


def _delete_artifact(run_id, artifact_name):
    # 列出本次运行的所有 artifact，要求重名项必须唯一，避免误删
    response = _gh_api(f"repos/{REPOSITORY}/actions/runs/{run_id}/artifacts")
    matches = [item for item in response["artifacts"] if item["name"] == artifact_name]
    if len(matches) != 1:
        raise RuntimeError(f"expected one artifact named {artifact_name}, found {len(matches)}")
    # 通过 artifact id 删除，确保加密结果不长期留在仓库可见区
    _gh_api(
```

```
        f"repos/{REPOSITORY}/actions/artifacts/{matches[0]['id']}",
        method="DELETE",
    )
```

.github/workflows/scripts/neon_access_job.py

远端执行器核心：解码 SQL、调用 psycopg 执行任意语句、编码结果行、限制返回字节数、打包数据库错误，是 SQL 语义与防泄露的关键实现。

```
def execute_request(driver, *, dsn, sql_text, request_id, run_id, actor, reason, max_result_bytes)
:
    # 构造基础结果结构，其中 sql_sha256 用于和客户端侧比对，确认执行的就是原始 SQL
    result = _base_result(
        request_id=request_id,
        run_id=run_id,
        actor=actor,
        reason=reason,
        sql_text=sql_text,
    )
    returned_bytes = 0
    try:
        # autocommit=True 表示不隐式包事务；prepare_threshold=None 关闭服务端预编译
        with driver.connect(dsn, autocommit=True, prepare_threshold=None) as connection:
            with connection.cursor() as cursor:
                # 整段 SQL 一次性传给 PostgreSQL，不拆分、不改写，支持多语句
                cursor.execute(sql_text, prepare=False)
                for current in cursor.results():
                    # 列名取驱动描述信息；无描述（如 DDL 语句）则列为空
                    columns = (
                        [column.name for column in current.description]
                        if current.description is not None
                        else []
                    )
                    result_set = {
                        "command_status": current.statusmessage,
                        "columns": columns,
                        "rows": [],
                    }
                    if current.description is not None:
                        for row in current:
                            if result["truncated"]:
                                continue
                            encoded_row = [encode_value(value) for value in row]
                            # 按 JSON 序列化后的字节数计算返回上限，达到上限即标记 truncated
                            并丢弃后续行
                            row_bytes = len(
                                json.dumps(
                                    encoded_row,
                                    ensure_ascii=False,
                                    separators=(",", ":"),
```

```
                        allow_nan=False,
                    ).encode("utf-8")
                )
                if returned_bytes + row_bytes > max_result_bytes:
                    result["truncated"] = True
                    continue
                result_set["rows"].append(encoded_row)
                returned_bytes += row_bytes
            result["results"].append(result_set)
    except driver.Error as error:
        # 数据库异常不抛出，而是打包进结果，便于调用方展示错误与部分执行效果
        result["status"] = "error"
        result["error"] = {
            "type": type(error).__name__,
            "sqlstate": getattr(error, "sqlstate", None),
            "message": str(error),
        }
    return result
```

评论区精华

该 PR 未产生任何公开 review 评论或讨论线程，仅由 yushengsu-thu 直接 approve。PR body 作者自列了三个 review focus：.github/workflows/neon-access.yml 的权限执行、.github/scripts/neon_access_job.py 的 SQL 执行语义、run_neon_workflow.py 的结果校验与 artifact 清理，但仓库内没有留下公开交锋。从 6 个提交演进可见作者自行完成了关键设计收敛：去掉重复文档与 public summary 输出；把执行器移入既有 workflow 所有权路径以避免改动 CODEOWNERS；取消二次确认（显式数据库请求即视为授权）；以及为 tests/ci 补注册 stage-a-cpu 修复 CPU 收集失败。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 任意 SQL 执行能力（高危）：neon_access_job.py 不分类、不限制语句类型，DDL/DML/ 事务控制均可执行，且 autocommit=True 意味着报错前语句已经提交，无法回滚；一旦权限校验被绕过，可造成数据破坏。
  - 权限检查单点依赖：客户端与工作流都依赖 repos/{repo}/collaborators/{actor}/permission API 的实时结果，GH_TOKEN 的权限范围如果小于仓库级 write 判定，可能产生误判；工作流对 github.actor 与 github.triggering_actor 都做了检查，但未验证 token 实际 scopes。
  - 结果保密依赖本机密钥：run_neon_workflow.py 的一次性私钥保存在本地临时目录（chmod 0o700），artifact 保留 1 天；若发起方机器被攻破，加密结果可被解密；workflow 侧 RECIPIENT_CERT_BASE64 会出现在 workflow inputs 中，存在被审计日志记录的可能。

    - 测试覆盖缺口：6 个离线测试均基于 FakeDriver，未覆盖真实 psycopg 行为、openssl cms 在不同发行版的差异、以及 workflow 中 shell 权限检查脚本本身的正确性（测试只是字符串断言）。
    - 对 GitHub API 版本的依赖：客户端固定 X-GitHub-Api-Version: 2026-03-10，较新版本若在旧 GitHub Enterprise 或代理环境不可用，dispatch 会失败；MAX_DISPATCH_CHARACTERS=65535 对大 SQL 有限制。
- 影响：
    - 用户影响：拥有 write/admin 权限的仓库贡献者和 agent 可通过 $neon-access 技能直接查询或运维 Neon 数据库，无需接触 NEON_DATABASE_URL；无权限用户会被明确拒绝。普通读者只能看到文档说明。
    - 系统影响：仓库新增一个 workflow_dispatch CI 入口、一个执行器脚本和一个技能目录，CI 会增加少量 runner 消耗与 artifact 存储（保留 1 天）。NEON_DATABASE_URL 的暴露面没有扩大，仍只存在于 workflow secret。
    - 团队影响：metric-history-gate 的托管数据查询 / 修复从“带外人工操作”变成“受控自助通道”；需要维护一次性密钥生成、CMS 加密、artifact 清理链路，后续若出现真实数据库事故，该通道会是最直接的追溯入口。
    - 风险标记：任意 SQL 执行能力，权限检查单点依赖，autocommit 无回滚，离线测试未覆盖真实驱动，依赖新 GitHub API 版本

关联脉络

- PR #2363 ci(docker): run image builds on docker-build runners: 同为 CI 基础设施演进，调整 GitHub Actions 工作流与 runner 策略，本 PR 新增的 neon-access 工作流属于同一 CI 扩展方向。
- PR #2279 Run the launch script snapshot tests by hand instead of in CI: 同为 CI 测试策略调整，本 PR 将离线测试注册到 stage-a-cpu，延续了规范 CI 测试收集的治理趋势。
- PR #2380 docs: use NVIDIA's "NeMo Gym" spelling instead of "NeMo-Gym": 同为文档维护类变更，本 PR 同步修改 docs/ci 页面，与持续改进文档体系的脉络一致。