

PR #54408 完整报告

vllm-project/vllm

[CI][ROCm] Avoid redundant image pulls during smoke validation

合并时间: 2026-09-01 10:48

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54408>

执行摘要

- 一句话: ROCm CI 冒烟校验移入 BuildKit, 免冷拉 13GB 镜像
- 推荐动作: 值得精读。这是“在构建系统内自验证、避免跨守护进程搬运镜像”的典型 CI 优化案例。重点看三点: smoke-test-image.sh 的双路径设计 (BuildKit 内直接执行 vs host docker 回退)、ci-bake-rocm.sh 的 fail-closed 编排 (局部输出目标不被已有镜像跳过逻辑短路)、以及测试中对 marker 跨构建复用的负例覆盖。对自建 CI 基础设施的团队有直接借鉴价值。

功能与动机

PR body 给出了明确的量化依据: AMD CI build 12440 的 image-build 重试耗时 13 分 54 秒, 其中 339 秒用于在 host Docker 中冷拉约 13 GB 镜像, 而实际 smoke 检查只占 4.6 秒; 持久化 agent 上同量级拉取只需 17-23 秒, 但 8 月 25 日起观察到的单任务 Kubernetes worker 从 0/41 层开始冷拉。作者用排除法定位根因: ROCm base 刷新 (#53182、#53712) 与版本敏感的 Rust 元数据 (#52593) 只是放大因素 (Rust 侧实测仅几十秒), 真正的转折点是 BuildKit 与 host Docker 使用分离存储——BuildKit 构建并推送完镜像图后, host Docker 还要完整下载第二遍, 因此本 PR 的目标是消除这次冗余传输。

实现拆解

1. 在 Dockerfile.rocm 中新增 BuildKit 内冒烟阶段: 新增 test_smoke 阶段从 test 镜像派生, 通过 ARG ROCM_SMOKE_ID 接收构建 ID, 并以 --network=none 运行重构后的 smoke-test-image.sh --inside, 完成目录结构、命令可用性、ROCm CLI 与 torch/vllm 导入检查后, 把 ROCM_SMOKE_ID 写入 /vllm-smoke-ok; 随后新增 export_test_smoke scratch 阶段, 仅从 test_smoke 复制该标记文件, 确保最终产出的只有几十字节文本, 而不是整张镜像。
2. 在 ci-rocm.hcl 中声明 smoke target 并接入构建组: 新增 smoke-test-rocm-ci target, 继承 _common-rocm 与 _ci-rocm 公共配置, target 指向 export_test_smoke, 复用 get_cache_from_rocm() 的缓存来源, 输出 type=local,dest=./build/rocm-smoke-export; 同时把它加入 test-rocm-ci-with-wheel group, 并在 _ci-rocm 的 args 中传入 ROCM_SMOKE_ID = BUILDKITE_BUILD_ID, 使 marker 与具体构建绑定。
3. 重构 smoke-test-image.sh 为双路径执行: 把原先内嵌在 docker run -ec '...' 里的检查脚本提取为 run_smoke_checks() 函数并支持 --inside 参数; 外层优先检查本地 marker 文件: 存在且 BUILDKITE_BUILD_ID 匹配则打印 verified inside BuildKit 并删除 marker; 存

在但不匹配则报错退出 (fail-closed) ; 不存在时回退到原 docker run --network=none 路径, 脚本经 stdin 传入容器, 并保留 VLLM_CI_SMOKE_IMAGE / IMAGE_TAG 显式 override 语义。

4. 在 ci-bake-rocm.sh 中补齐编排与校验: 新增 should_export_rocm_smoke 与 verify_rocm_smoke_export 两个函数; 修改 maybe_skip_existing_image, 让“局部输出目标” (wheel、smoke marker) 不被已有镜像跳过逻辑短路, 确保每次当前构建都产出 marker; main() 在 run_bake 前清理 ./build/rocm-smoke-export 防陈旧残留, 在 run_bake 后校验 marker 存在且 ID 匹配, 否则整条 CI 失败; 同时把 smoke-test-rocm-ci 纳入 csrc/rust 缓存使用与 Rust cache 注入的 target 列表。
5. 测试配套: 在 tests/tools/test_docker_build_metadata_args.py 中新增 5 个测试, 用 fake docker 脚本与临时目录模拟执行, 覆盖图契约断言 (Dockerfile/HCL 中出现 FROM test AS test_smoke、export_test_smoke、smoke-test-rocm-ci target 等关键串)、marker 命中时跳过 docker、跨构建 marker 拒绝 (返回码 1)、显式 override 走 docker 并传递 --inside、以及环境变量清理逻辑。

关键文件:

- .buildkite/scripts/rocm/smoke-test-image.sh (模块 构建脚本; 类别 other; 类型 core-logic; 符号 run_smoke_checks) : 核心执行逻辑所在: 抽出 run_smoke_checks() 函数, 支持 BuildKit --inside 与 host docker 两条路径, 并实现 marker 优先、fail-closed 校验、Docker 回退的完整流程。
- .buildkite/scripts/ci-bake-rocm.sh (模块 构建脚本; 类别 other; 类型 core-logic; 符号 should_export_rocm_smoke, verify_rocm_smoke_export) : 编排核心: 负责 smoke target 的预算跳过豁免、构建前清理、构建后 marker 的 fail-closed 校验, 以及 csrc/rust 缓存注入, 是本次改动正确性的关键保障。
- docker/ci-rocm.hcl (模块 镜像构建; 类别 infra; 类型 infrastructure; 符号 smoke-test-rocm-ci) : 声明 smoke-test-rocm-ci target 并把它接入 test-rocm-ci-with-wheel group, 同时通过 _ci-rocm 传入 ROCM_SMOKE_ID, 是 marker 机制与 BuildKit 输出通道的配置载体。
- docker/Dockerfile.rocm (模块 镜像构建; 类别 infra; 类型 infrastructure; 符号 test_smoke, export_test_smoke) : 新增 test_smoke 与 export_test_smoke 阶段, 从机制上把冒烟检查放进共享 BuildKit 图内, 并只通过 scratch 阶段导出几十字节的 marker。
- tests/tools/test_docker_build_metadata_args.py (模块 构建测试; 类别 test; 类型 test-coverage; 符号 test_rocm_ci_smoke_runs_in_shared_buildkit_graph, prepare_rocm_smoke_test, test_rocm_smoke_marker_avoids_host_image_pull, test_rocm_smoke_rejects_marker_from_another_build) : 用 fake docker 脚本与临时目录锁定了 5 个关键行为: 图契约、marker 命中跳过 docker、跨构建 marker 拒绝、显式 override 走 docker、以及环境清理, 是防止 CI 契约回归的主要保障。

关键符号: run_smoke_checks, should_export_rocm_smoke, verify_rocm_smoke_export, test_rocm_ci_smoke_runs_in_shared_buildkit_graph, prepare_rocm_smoke_test, test_rocm_smoke_marker_avoids_host_image_pull, test_rocm_smoke_rejects_marker_from_another_build, test_rocm_smoke_override_streams_current_checks_to_docker

关键源码片段

.buildkite/scripts/rocm/smoke-test-image.sh

核心执行逻辑所在：抽出 run_smoke_checks() 函数，支持 BuildKit --inside 与 host docker 两条路径，并实现 marker 优先、fail-closed 校验、Docker 回退的完整流程。

```
set -euo pipefail

# 冒烟检查本体：目录结构、关键命令、ROCm CLI 与 torch/vllm 导入。
# 两条路径复用同一函数：
# 1. BuildKit --inside: 由 Dockerfile 的 RUN 阶段直接调用；
# 2. host docker: 脚本经 stdin 传入容器执行（回退路径）。
run_smoke_checks() {
    local required_dir=""
    for required_dir in \
        /vllm-workspace \
        /vllm-workspace/tests \
        /vllm-workspace/src/vllm; do
        if [[ ! -d "${required_dir}" ]]; then
            echo "Missing directory: ${required_dir}" >&2
            return 1
        fi
    done
    if [[ ! -x /vllm-workspace/src/vllm/vllm-rs ]]; then
        echo "Missing executable: /vllm-workspace/src/vllm/vllm-rs" >&2
        return 1
    fi

    command -v python3
    command -v uv
    command -v pytest

    if ! command -v amd-smi >/dev/null 2>&1 \
        && ! command -v rocm_info >/dev/null 2>&1; then
        echo "No ROCm CLI found in image" >&2
        return 1
    fi

    PYTHONDONTWRITEBYTECODE=1 python3 - <<'PY'
import torch
import vllm

print(torch.__version__)
print(vllm.__version__)
PY
    echo "AMD image smoke OK"
}
```

--inside: 供 Dockerfile 的 test_smoke 阶段调用，检查后立即退出

```

if [[ "${1:-}" == "--inside" ]]; then
    run_smoke_checks
    exit
fi
if (($#)); then
    echo "Usage: $0 [--inside]" >&2
    exit 2
fi

# 外层 (Buildkite host) : 优先消费 BuildKit 图内产出的 marker,
# 校验 BUILDKITE_BUILD_ID 匹配才算通过, 彻底避免拉取多 GB 镜像
smoke_marker="./build/rocm-smoke-export/vllm-smoke-ok"
expected_smoke_id="${BUILDKITE_BUILD_ID:-local}"
if [[ -f "${smoke_marker}" \
    && ( -z "${VLLM_CI_SMOKE_IMAGE:-}" \
        || "${VLLM_CI_SMOKE_IMAGE}" == "${IMAGE_TAG:-}" ) ]]; then
    actual_smoke_id="$(< "${smoke_marker}")"
    # marker 属于其他构建时直接失败 (fail-closed), 不接受陈旧产物
    if [[ "${actual_smoke_id}" != "${expected_smoke_id}" ]]; then
        echo "ROCm smoke marker belongs to ${actual_smoke_id}, not ${expected_smoke_id}" >&2
        exit 1
    fi
    echo "AMD image smoke OK (verified inside BuildKit)"
    rm -f -- "${smoke_marker}"
    rmdir -- "$(dirname "${smoke_marker}")" 2>/dev/null || true
    exit
fi

# 回退路径: marker 缺失或显式指定 VLLM_CI_SMOKE_IMAGE 时,
# 走 docker run --network=none, 脚本经 stdin 传入容器执行
image_ref="${VLLM_CI_SMOKE_IMAGE:-${IMAGE_TAG:-rocm/vllm-ci:${BUILDKITE_COMMIT:-?BUILDKITE_COMMIT is required}}}"
docker run --rm -i --network=none --entrypoint /bin/bash "${image_ref}" \
    -s -- --inside < "${BASH_SOURCE[0]}"

```

docker/ci-rocm.hcl

声明 smoke-test-rocm-ci target 并把它接入 test-rocm-ci-with-wheel group, 同时通过 _ci-rocm 传入 ROCM_SMOKE_ID, 是 marker 机制与 BuildKit 输出通道的配置载体。

```

# 新冒烟 target: 不推送镜像, 只向本地 ./build/rocm-smoke-export 导出标记。
# cache-from 与 test 镜像一致, 因此分层在 BuildKit 图内已存在, 不会二次下载。
target "smoke-test-rocm-ci" {
    inherits = ["_common-rocm", "_ci-rocm"]
    target = "export_test_smoke"
    cache-from = get_cache_from_rocm()
    output = ["type=local,dest=./build/rocm-smoke-export"]
}

```

完整构建组: 冒烟校验作为 test 镜像的图内子阶段参与同一构建

```
# 这里把 smoke-test-rocm-ci 追加到原有 wheel 导出链,
# 保证冒烟与镜像推送使用同一次 BuildKit 构建产物。
group "test-rocm-ci-with-wheel" {
  targets = [
    "rust-rocm-ci",
    "csrc-rocm-ci",
    "test-rocm-ci",
    "smoke-test-rocm-ci",
    "export-wheel-rocm",
  ]
}
```

docker/Dockerfile.rocm

新增 test_smoke 与 export_test_smoke 阶段，从机制上把冒烟检查放进共享 BuildKit 图内，并只通过 scratch 阶段导出几十字节的 marker。

```
# 冒烟检查阶段：在共享 BuildKit 图内直接复用 test 镜像的分层，
# 避免把约 13 GB 镜像重新拉进 host Docker 守护进程。
FROM test AS test_smoke
ARG ROCM_SMOKE_ID
# --network=none 强制检查不依赖网络，保证冒烟是纯结构验证；
# 检查通过后把构建级 ID 写入标记文件，供后续 Bake 编排校验。
RUN --network=none \
    bash /vllm-workspace/.buildkite/scripts/rocm/smoke-test-image.sh --inside \
    && printf '%s\n' "${ROCM_SMOKE_ID:-local}" > /vllm-smoke-ok

# export 阶段从 scratch 起步，只复制标记文件，
# 输出的实际产物是几十字节文本而非整张镜像。
FROM scratch AS export_test_smoke
COPY --from=test_smoke /vllm-smoke-ok /
```

评论区精华

该 PR 没有实质性的 review 讨论：claude[bot] 仅提示本仓库配置了手动 review（未触发实质评审），shen-shanshan 直接批准；PR 上唯一的互动是作者执行 `/ci run` 触发 Buildkite CI。设计决策全部沉淀在 PR body：作者明确排除了 #53182、#53712、#52593 作为突发回归的主因，把转折点定位到 BuildKit 与 host Docker 分离存储；同时声明实现、测试与 PR 草稿由 OpenAI Codex 辅助，并要求提交者逐行复核。这种“量化排除 + 根因收敛”的分析方式本身值得参考。

- 暂无高价值评论线程

风险与影响

- 风险：
 1. fail-closed 双保险依赖 marker 一致性：ci-bake-rocm.sh 的 `verify_rocm_smoke_export` 要求 marker 存在且 BUILDKITE_BUILD_ID 匹配，若 smoke-test-rocm-ci 因构建图异常未产出 marker，整条 bake 失败。这是设计意图，但

未来调整 target 列表时需同步更新测试锁定契约。

2. BUILDKITE_BUILD_ID 降级为 local: 脚本与 bake 校验在非 Buildkite 环境将 expected_smoke_id 设为 local, 本地手动执行 run_bake 时可能出现 marker 误判, 本地调试 CI 脚本时需注意。
3. BuildKit 网络限制: test_smoke 阶段以 --network=none 运行, 未来若 smoke 检查需要网络, 只能走 Docker 回退路径。
4. 验证路径语义变化: 即使持久化 agent 上 host Docker 有层缓存, 只要 marker 存在且 ID 匹配就会跳过 docker run; 该路径依赖“BuildKit 构建出的 test 镜像与推送镜像一致”的前提 (同图构建可保证), 但显式 VLLM_CI_SMOKE_IMAGE override 场景下走的是 docker 路径, 语义不同。
5. marker 清理依赖工作区状态: rm -rf ./build/rocm-smoke-export 只在本次 bake 前执行, 若前一次构建的 marker 残留而本次未运行 smoke target, 校验会因 ID 不匹配失败, 错误信息中已包含 build ID 以便排查。- 影响: 对 ROCm CI 而言, 冒烟验证环节预计从分钟级 (最差 339 秒拉取) 降到秒级, 消除每任务 worker 上约 13 GB 的第二次镜像图下载, 显著降低带宽与磁盘压力; 对开发者而言, ROCm 相关 PR 的构建与失败重试时间明显缩短。对 vLLM 运行时零影响, 推送的 test 镜像内容与镜像契约完全不变。对团队而言, 新增了一组必须维护的 CI 契约 (Dockerfile 阶段、Bake target、脚本 marker), 但 5 个 fake-docker 测试已锁定核心行为, 后续修改构建链需同步维护这些测试。- 风险标记: CI 流程架构调整, fail-closed 校验, 依赖 BuildKit 图内构建, BUILDKITE_BUILD_ID 降级路径

关联脉络

- PR #46904 Pull-based smoke 路径 (PR body 提及的设计起源): PR body 明确说明当前被优化的 host Docker 拉取式 smoke 验证起源于该 PR, 本 PR 是它在新 Kubernetes worker 场景下的性能修正。
- PR #48646 等效镜像复用扩展 (PR body 提及): 引入等效镜像复用机制, 本 PR 将其收益扩展到 smoke 验证环节, 避免重复传输。
- PR #53182 ROCm base 镜像刷新 (PR body 提及): ROCm base 刷新使镜像内容变化, 在冷缓存 worker 上放大了拉取开销, 是问题的促成因素之一。
- PR #53712 ROCm base 镜像刷新 (PR body 提及): 与 53182 类似, 贡献了镜像内容变化; 作者明确评估它并非首次变慢的主因。
- PR #52593 版本敏感 Rust 构建元数据 (PR body 提及): 作者通过量化评估 (仅几十秒级影响) 排除了它作为突发回归主因的可能。