

PR #48750 完整报告

vllm-project/vllm

[CI] Build CPU image against torch nightly for TORCH_NIGHTLY runs

合并时间: 2026-09-01 08:25

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48750>

执行摘要

- 一句话: 修复 torch nightly lane CPU/arm64 镜像缺失, 标签与缓存对齐
- 推荐动作: 值得精读, 尤其适合负责 CI 镜像管理与 Docker 多阶段构建的工程师。三个设计决策值得借鉴: 一是用注册表实测 (200/200/404/404) 证明标签对齐问题而非臆测; 二是识别 COPY . . 恢复文件导致 wheel 硬 pin 的隐蔽链路; 三是用 CACHE_SUFFIX 隔离不同环境的 ECR 缓存。留意 PR 中明确遗留的 nightly 安装层缓存键问题, 建议后续跟踪。

功能与动机

PR body 明确指出: torch-nightly lane 的 CPU 和 arm64 测试步骤拉取的镜像标签无人构建, 每次 nightly 运行都以 manifest unknown 失败。注册表核对 build 85629 显示 -torch-nightly-cpu 与 -torch-nightly-arm64 均为 404, 而 CPU 构建脚本的存在性守卫检查的是 stable 标签 -cpu, 因此打印 Image found 并在 2 秒内退出, 构建实际什么都没做。

实现拆解

实现按以下步骤推进:

1. 修复 CPU 镜像构建脚本标签与存在性守卫 (`.buildkite/image_build/image_build_cpu.sh`): 在 `TORCH_NIGHTLY=1` 时输出 `$BUILDKITE_COMMIT-torch-nightly-cpu` 标签并与 `ci-infra get_image()` 的后缀规则对齐; 构建时追加 `--build-arg PYTORCH_NIGHTLY=1`。同时移除导致误报成功的死守卫 (它检查的是 stable 标签)。
2. 扩展 arm64 镜像构建脚本 (`.buildkite/image_build/image_build_arm64.sh`): 为 DGX Spark 测试步骤提供 `-torch-nightly-arm64` 标签, 保证 nightly lane 上 GPU arm64 作业也能拉到镜像。`image_build_cpu_arm64.sh` 被刻意保留不动——`ci-infra` 中 `use_cpu` 与 `use_arm64` 从单一 `step.device` 派生、互斥, `get_image(cpu=True, arm64=True)` 不可达, 且已有的 `-arm64-cpu` 标签只被 `release` 路径消费。
3. `Dockerfile.cpu` 新增 `PYTORCH_NIGHTLY` 构建路径: 先显式从 `/whl/nightly/cpu` 安装 `torch/torchaudio/torchvision/torchcodec` (`--pre`), 再用 `use_existing_torch.py --prefix` 剥离 `requirements` 中的 `release pin`, 最后从 nightly CPU index 安装其余依赖, 同时去掉 `--torch-backend cpu` (该参数只认 stable channel, 会静默拉回 release 版)。多阶段构建中 ARG 不跨 FROM, 因此在 `vllm-src`、`vllm-test-deps`、`vllm-dev` 各阶段重新声明了 `PYTORCH_NIGHTLY` 与 `PYTORCH_CPU_INDEX_BASE_URL`。

4. 解决 wheel 硬 pin 问题: `vllm-build` 阶段在 `COPY ..` 之前执行 `strip` 只会清理 `requirements/build/cpu.txt`, 随后 `COPY ..` 会恢复 pin 过的 `requirements/cpu.txt`; `setup.py` 将其读入 `install_requires` 导致 wheel 硬 pin `torch==2.11.0+cpu`, 在 nightly index 上无法解析。因此在 `vllm-src` 阶段末尾 (`COPY ..` 之后、派生 `vllm-build/vllm-dev` 之前) 对 nightly 再执行一次 `strip`, 使 wheel 不再携带 release pin。
5. 缓存标签隔离: `prepare_cache_tags` 引入 `CACHE_SUFFIX`, `TORCH_NIGHTLY=1` 时变为 `x86_cpu-nightly`, 所有缓存 ref (含 `CACHE_FROM_MAIN`) 都挂上后缀, 避免 nightly 与 stable 两个 lane 各自构建不同 torch 时共享同一 ECR ref 互相驱逐缓存。
6. `use_existing_torch.py` 补 `torchcodec` 前缀: nightly index 上的 `torchcodec` 为 `0.16.0.dev*+cpu`, 而 `requirements/test/cpu.txt` pin 了 `0.14.0+cpu` 且不在 nightly index 中, 必须一并剥离。
7. 验证与配套: 无新增测试文件; 验证手段为 `shellcheck/bash -n`、注册表四标签核对 (200/200/404/404)、以及合并后首轮 nightly run 的冷缓存构建观察。

关键文件:

- `docker/Dockerfile.cpu` (模块 镜像构建; 类别 infra; 类型 infrastructure; 符号 `PYTORCH_NIGHTLY`, `PYTORCH_CPU_INDEX_BASE_URL`, `vllm-src`, `vllm-test-deps`): 核心改动文件: 新增 `PYTORCH_NIGHTLY` 构建路径, 涉及 nightly torch 安装、release pin 剥离、多阶段 ARG 重声明以及 wheel 硬 pin 修复, 是本次变更的技术主体。
- `.buildkite/image_build/image_build_cpu.sh` (模块 CI 脚本; 类别 infra; 类型 core-logic; 符号 `prepare_cache_tags`, `clean_docker_tag`): CPU 镜像构建入口: 负责 nightly 标签生成、`PYTORCH_NIGHTLY=1` 透传以及 ECR 缓存标签隔离, 是修复 manifest unknown 与缓存驱逐的关键脚本。
- `.buildkite/image_build/image_build_arm64.sh` (模块 CI 脚本; 类别 infra; 类型 core-logic; 符号 `IMAGE`): arm64 (DGX Spark) 镜像构建入口: 与 CPU 脚本对齐, 为 nightly lane 产出 `-torch-nightly-arm64` 标签, 补上 GPU arm64 的覆盖盲区。
- `use_existing_torch.py` (模块 依赖清洗; 类别 source; 类型 core-logic; 符号 `TORCH_LIB_PREFIXES`, `main`): 依赖 pin 剥离工具: 为 nightly 构建新增 `torchcodec` 前缀, 确保 `torchcodec` release pin (`0.14.0+cpu`) 不会在 nightly index 上导致解析失败。

关键符号: `prepare_cache_tags`, `main` (`use_existing_torch.py`)

关键源码片段

`docker/Dockerfile.cpu`

核心改动文件: 新增 `PYTORCH_NIGHTLY` 构建路径, 涉及 nightly torch 安装、release pin 剥离、多阶段 ARG 重声明以及 wheel 硬 pin 修复, 是本次变更的技术主体。

```
# PYTORCH_NIGHTLY=1 时走 nightly 分支: 先显式安装 torch nightly 全家桶,
# 再用 use_existing_torch.py --prefix 剥离 requirements 里的 release pin,
# 最后从 nightly index 安装其余依赖。不能使用 --torch-backend cpu,
# 它只认 stable channel, 会静默把依赖拉回 release 版本。
ARG PYTORCH_NIGHTLY=0
ARG PYTORCH_CPU_INDEX_BASE_URL=https://download.pytorch.org/whl
```

```

RUN --mount=type=cache,target=/root/.cache/uv \
  uv pip install --upgrade pip && \
  if [ "${PYTORCH_NIGHTLY}" = "1" ]; then \
    echo "Installing torch nightly (CPU)..." \
    && uv pip install torch torchaudio torchvision torchcodec --pre \
    --index-url ${PYTORCH_CPU_INDEX_BASE_URL}/nightly/cpu \
    && python3 use_existing_torch.py --prefix \
    && uv pip install -r requirements/cpu.txt \
    --extra-index-url ${PYTORCH_CPU_INDEX_BASE_URL}/nightly/cpu; \
  else \
    # release 路径保持原样，完全不受 nightly 改动影响
    uv pip install -r requirements/cpu.txt --torch-backend cpu; \
  fi

```

```

# vllm-src 阶段末尾: COPY . . 会把 pin 过的 requirements/cpu.txt 恢复回源码树,
# setup.py 又把它读进 install_requires, 导致 wheel 硬 pin torch==2.11.0+cpu,
# 在 nightly index 上无法解析。这里对 nightly 再执行一次 strip, 使 wheel
# 依赖不再携带 release pin, 由 base 镜像里已装的 nightly torch 满足依赖。
FROM base-arch AS vllm-src
ARG PYTORCH_NIGHTLY=0
ARG PYTORCH_CPU_INDEX_BASE_URL=https://download.pytorch.org/whl
COPY . .
RUN if [ "${PYTORCH_NIGHTLY}" = "1" ]; then \
  python3 /workspace/use_existing_torch.py --prefix; \
fi

```

[.buildkite/image_build/image_build_cpu.sh](#)

CPU 镜像构建入口：负责 nightly 标签生成、PYTORCH_NIGHTLY=1 透传以及 ECR 缓存标签隔离，是修复 manifest unknown 与缓存驱逐的关键脚本。

```

# TORCH_NIGHTLY=1 时按 ci-infra 的后缀规则输出 nightly 标签：
# get_image() 会在 -cpu 前插入 -torch-nightly，测试步骤拉取的就是
# <commit>-torch-nightly-cpu；release 路径保持 <commit>-cpu 不变。
PYTORCH_NIGHTLY_ARGS=()
if [[ "${TORCH_NIGHTLY:-0}" == "1" ]]; then
  IMAGE="$REGISTRY/$REPO:$BUILDKITE_COMMIT-torch-nightly-cpu"
  PYTORCH_NIGHTLY_ARGS=(--build-arg PYTORCH_NIGHTLY=1)
else
  IMAGE="$REGISTRY/$REPO:$BUILDKITE_COMMIT-cpu"
fi

```

```

# 缓存隔离：nightly 与 stable lane 解析的是不同 torch，若共享同一 ECR
# 缓存 ref，两个 lane 每次运行都会把对方构建出的缓存层逐出。
# CACHE_SUFFIX 在 nightly 模式下变为 x86_cpu-nightly，所有缓存 ref
# （含 CACHE_FROM_MAIN）都挂上该后缀；回退到 stable 主缓存同样会
# 引入 stable torch 层，这正是独立标签要避免的污染。
prepare_cache_tags() {
  TEST_CACHE_ECR="936637512419.dkr.ecr.us-east-1.amazonaws.com/vllm-ci-test-cache"
  MAIN_CACHE_ECR="936637512419.dkr.ecr.us-east-1.amazonaws.com/vllm-ci-postmerge-

```

```

cache"
CACHE_SUFFIX="x86_cpu"
if [[ "${TORCH_NIGHTLY:-0}" == "1" ]]; then
    CACHE_SUFFIX="x86_cpu-nightly"
fi
# 非 PR (main 分支) 与 PR 分支分别使用 latest-${CACHE_SUFFIX} 与
# pr-<N>-${CACHE_SUFFIX}, 保证两条 lane 互不驱逐
if [[ "${BUILDKITE_PULL_REQUEST:-false}" == "false" ]]; then
    if [[ "${BUILDKITE_BRANCH:-}" == "main" ]]; then
        cache="${MAIN_CACHE_ECR}:latest-${CACHE_SUFFIX}"
    else
        clean_branch=$(clean_docker_tag "${BUILDKITE_BRANCH:-unknown}")
        cache="${TEST_CACHE_ECR}:${clean_branch}-${CACHE_SUFFIX}"
    fi
    CACHE_TO="$cache"; CACHE_FROM="$cache"; CACHE_FROM_BASE_BRANCH="$cache"
else
    CACHE_TO="${TEST_CACHE_ECR}:pr-${BUILDKITE_PULL_REQUEST}-${CACHE_SUFFIX}"
    CACHE_FROM="${TEST_CACHE_ECR}:pr-${BUILDKITE_PULL_REQUEST}-${CACHE_SUFFIX}"
    if [[ "${BUILDKITE_PULL_REQUEST_BASE_BRANCH:-main}" == "main" ]]; then
        CACHE_FROM_BASE_BRANCH="${MAIN_CACHE_ECR}:latest-${CACHE_SUFFIX}"
    else
        clean_base=$(clean_docker_tag "${BUILDKITE_PULL_REQUEST_BASE_BRANCH}")
        CACHE_FROM_BASE_BRANCH="${TEST_CACHE_ECR}:${clean_base}-${CACHE_SUFFIX}"
    fi
fi
CACHE_FROM_MAIN="${MAIN_CACHE_ECR}:latest-${CACHE_SUFFIX}"
}

```

use_existing_torch.py

依赖 pin 剥离工具：为 nightly 构建新增 torchcodec 前缀，确保 torchcodec release pin (0.14.0+cpu) 不会在 nightly index 上导致解析失败。

```

# 只剥离目标库的前缀匹配，避免误删其他依赖。
# 新增 torchcodec: nightly 安装包含 torchcodec (0.16.0.dev*+cpu) ,
# 而 requirements/test/cpu.txt pin 的是 torchcodec==0.14.0+cpu,
# 该 release pin 在 nightly CPU index 上不存在，必须一并剥离。
TORCH_LIB_PREFIXES = (
    # requirements/*.txt/in
    "torch=",
    "torchvision=",
    "torchaudio=",
    "torchcodec=",
    # pyproject.toml
    "'torch ='",
    "'torchvision ='",
    "'torchaudio ='",
    "'torchcodec ='",
)

```

评论区精华

khluu 的 PR review agent 提出两条实质问题，atalman 全部确认并修复：

1. Body/diff 不一致与 arm64 CPU 覆盖：body 声称修改了 `image_build_cpu_arm64.sh`，实际 diff 改的是 `image_build_arm64.sh`（GPU arm64 构建）。后者确实匹配 ci-infra 对 DGX Spark 的 `get_image(arm64=True)` 期望，是合法修复，但 body 描述陈旧；且若 nightly lane 存在 arm64 CPU 作业，`-torch-nightly-cpu-arm64` 仍无人构建。结论：`use_cpu` 与 `use_arm64` 互斥、`get_image(cpu=True, arm64=True)` 不可达，nightly lane 不存在 arm64 CPU 作业，遂重写 body 并保留 `image_build_cpu_arm64.sh` 不动。
 2. 缓存标签冲突：`prepare_cache_tags` 未修改时，nightly 运行会把 `--cache-to` 推到与常规 CI 相同的 `pr-<N>-x86_cpu` ref，两个 lane 基于不同 torch 构建，会轮流把对方缓存逐出。结论：用 `CACHE_SUFFIX` 在 nightly 模式加 `-nightly` 后缀，包括 `CACHE_FROM_MAIN`，避免回退到 stable 主分支缓存造成污染。
- PR body 与 diff 不一致：arm64 CPU 覆盖问题 (design): atalman 承认 body 陈旧并重写；依据 ci-infra `_get_step_plugin` 中 `use_cpu` 与 `use_arm64` 互斥、`get_image(cpu=True, arm64=True)` 不可达，确认 nightly lane 无 arm64 CPU 作业，`image_build_cpu_arm64.sh` 刻意不动，且已存在的 `-arm64-cpu` 标签仅被 release 路径消费。
 - nightly 与 stable lane 共享 ECR 缓存 ref (design): atalman 承认遗漏并修复：引入 `CACHE_SUFFIX`，`TORCH_NIGHTLY=1` 时为 `x86_cpu-nightly`，所有缓存 ref（含 `CACHE_FROM_MAIN`）都挂上后缀，避免回退到 stable 缓存造成 torch 层污染；同时顺带移除了误报成功的存在性守卫。

风险与影响

- 风险：
 1. 首次 nightly 运行是冷缓存构建：`latest-x86_cpu-nightly` 不存在，合并后首轮 nightly 的 CPU 构建需从头编译，耗时会显著增加，PR body 已声明。
 2. nightly torch 安装层缓存键不随日期变化：安装 nightly torch 的 RUN 层键不包含日期，warm BuildKit/registry 缓存可能复用旧 nightly，与官方 torch nightly 语义不符。PR 作者明确将其列为未处理项。
 3. 多阶段 ARG 重复声明易遗漏：`Dockerfile.cpu` 中 `PYTORCH_NIGHTLY/PYTORCH_CPU_INDEX_BASE_URL` 需在 `vllm-src`、`vllm-test-deps`、`vllm-dev` 三个阶段重复声明，后续新增 FROM 阶段若忘记声明会静默退回 release 路径。
 4. 影响面隔离良好：所有功能改动都受 `TORCH_NIGHTLY/PYTORCH_NIGHTLY` 门控，release 路径（`--torch-backend cpu` 分支）完全不变；`image_build_arm64.sh` 的改动只影响 nightly 模式，release 镜像产物名不受影响。- 影响：对用户与生产代码无影响——本次改动全部位于 CI 与 Docker 构建基础设施：修复 torch-nightly lane 每次运行 8 个 CPU 作业与 2 个 DGX Spark 作业的系统性 manifest unknown 失败，使 nightly 测试真正基于 nightly torch 运行而非静默跳过构建。对团队而言，nightly lane 从此具备与 GPU 对等的 CPU/arm64 覆盖，torch nightly 兼容性问题可以在合并前被发现；同

时 ECR 缓存 ref 的隔离避免了 stable 与 nightly lane 互相污染。 - 风险标记: nightly 镜像首次构建为冷缓存, nightly torch 安装层缺少按日缓存键, 多阶段 ARG 重复声明易遗漏, 仅影响 TORCH_NIGHTLY 路径

关联脉络

- PR #54549 [CI] Mark 1-GPU L4 test steps with device: l4 for EKS migration: 同为 CI 基础设施演进: Buildkite 测试步骤标签与设备映射约定, 与本 PR 的 step.device 派生逻辑 (use_cpu/use_arm64) 属于同一套 CI 配置体系。
- PR #54365 [CI] Exclude kv_transfer changes from broad spec-decode/kernels/multimodal triggers: 同为 .buildkite 下的 CI 触发与镜像标签维护工作, 反映 vLLM CI 配置持续收敛的趋势。
- PR #53669 [XPU] [CI] Add retry for v1/sample in Intel GPU CI: 同为针对特定硬件 lane 的 CI 稳定性修复, 与本 PR 解决 nightly lane 镜像缺失问题属于同一维护脉络。