PR #24278 完整报告

sgl-project/sglang

[SMG][CI] Add K8s integration tests + wire into pr-test-rust

合并时间：2026-05-05 01:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24278

执行摘要

- 一句话：为 SMG 添加端到端 K8s 集成测试并接入 CI
- 推荐动作：值得精读。本 PR 展示了如何为网关组件构建端到端集成测试，尤其是 fake worker 的设计、基于 UID 的驱逐验证以及 CI 轻量构建策略，对类似组件的测试设计有参考价值。

功能与动机

确保 SMG 网关在 K8s 环境中的可靠性，尤其是服务发现和 PD 模式下的滚动更新场景。PR body 提到："Adds an end-to-end K8s integration test suite for SMG ... exercising the K8s watcher, the reconciliation loop, and the UID-based eviction path used during hostNetwork rollouts."

实现拆解

1. 测试套件与基础结构：创建 sgl-model-gateway/e2e_test/k8s_integration/conftest.py 提供与 kind 集群交互的核心辅助函数（_kubectl、_wait_for_pod_ready、_poll_until 等），定义可安全重试的传输层异常集合 _TRANSIENT_ERRORS，以及 reconciliation 等待时间常量。同时新增 fake_worker.py 实现一个轻量 HTTP 服务器，模拟 SGLang worker 的 /health、/v1/models、/server_info 等端点。
2. 服务发现与 reconciliation 测试：在 test_reconciliation.py 中实现 TestWatcherDiscovery 测试类，通过程序化地部署和删除 Pod，验证 K8s watcher 能否正确发现新 Pod、worker 注册与注销是否及时、Prometheus 指标是否准确，以及多次 reconciliation 周期内系统保持稳定。
3. PD 类型转换测试：在 test_pd_type_change.py 中专门模拟 hostNetwork 场景下的 Prefill -> Decode 滚动更新：当旧 prefill Pod 被删除、新 decode Pod 以相同 IP 不同 UID 启动时，网关必须基于 UID 驱逐旧 worker 并注册新 worker。测试通过检查 /workers 端点的 worker_type 字段确保转换正确。
4. 部署清单与集群设置：在 manifests/ 下提供 namespace、RBAC、基础网关和 PD 模式网关的 YAML 文件；setup.sh 自动化 kind 集群创建、镜像构建 / 加载、基础清单应用，并支持 SKIP_DOCKER_BUILD=1 环境变量以在 CI 中复用预构建镜像。
5. CI 集成：在 .github/workflows/pr-test-rust.yml 中新增 k8s-integration 作业，使用 ubuntu-22.04 运行器，通过 docker/build-push-action 并使用 GHA 缓存构建测试镜像，

运行完整测试套件，并加入 finish.needs 作为 PR 合并门禁。同时，在 sgl-model-gateway/e2e_test/conftest.py 中调整了导入逻辑，避免在缺少 sglang_router 依赖时破坏测试收集。

关键文件：

- sgl-model-gateway/e2e_test/k8s_integration/test_reconciliation.py（模块 服务发现测试；类别 test；类型 test-coverage；符号 _get_metrics, _get_worker_urls, _parse_metric_value, _deploy_worker_pod）：核心测试文件，覆盖服务发现、worker 注册 / 注销、reconciliation 循环和 Prometheus 指标验证。
- sgl-model-gateway/e2e_test/k8s_integration/conftest.py（模块 测试配置；类别 test；类型 test-coverage；符号 pytest_configure, _kubectl, _kubectl_json, _wait_for_pod_ready）：所有测试的基座，提供 kubectl 封装、等待函数、端口转发管理、轮询工具和可重试错误分类。
- sgl-model-gateway/e2e_test/k8s_integration/test_pd_type_change.py（模块 PD 类型测试；类别 test；类型 test-coverage；符号 _get_workers_by_type, _deploy_pd_worker, _safe_delete_pod, pd_gateway）：专门测试 PD 模式下 hostNetwork 滚动更新的 UID 驱逐路径，验证 worker_type 转换正确。
- sgl-model-gateway/e2e_test/k8s_integration/fake_worker.py（模块 Fake 工作节点；类别 test；类型 test-coverage；符号 FakeWorkerHandler, do_GET, log_message）：轻量级 HTTP 服务器模拟真实 worker 端点，支撑服务发现测试。
- sgl-model-gateway/e2e_test/k8s_integration/setup.sh（模块 集群设置；类别 test；类型 test-coverage）：自动化 kind 集群创建、镜像构建 / 加载、基础清单应用，支持 CI 预构建优化。
- .github/workflows/pr-test-rust.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：将 K8s 集成测试接入 CI，新增 k8s-integration 作业并作为合并门禁。
- sgl-model-gateway/e2e_test/k8s_integration/Dockerfile.gateway（模块 构建配置；类别 test；类型 test-coverage）：轻量级 Dockerfile 仅构建 Rust 二进制（~5 分钟），避免全量 maturin 构建。
- sgl-model-gateway/e2e_test/k8s_integration/manifests/gateway-pd.yaml（模块 PD 网关部署；类别 test；类型 test-coverage）：PD 模式网关的 Deployment 和 Service，供类型转换测试使用。
- sgl-model-gateway/e2e_test/k8s_integration/manifests/gateway.yaml（模块 基础网关部署；类别 test；类型 test-coverage）：基础模式网关的 Deployment 和 Service，供 reconciliation 测试使用。
- sgl-model-gateway/e2e_test/conftest.py（模块 测试配置；类别 test；类型 test-coverage）：修改导入逻辑，避免在缺少 sglang_router 依赖时破坏测试收集。

关键符号：_kubectl, _kubectl_json, _wait_for_pod_ready, _wait_for_deployment_ready, _get_gateway_url, _get_metrics_url, _wait_for_port, _poll_until, _get_workers, _get_worker_count, _cleanup_port_forward, _get_metrics, _get_worker_urls, _parse_metric_value, _deploy_worker_pod, _delete_worker_pod, _safe_delete_worker_pod, _wait_for_pod_gone, _get_workers_by_type,

_deploy_pd_worker, _safe_delete_pod, pd_gateway, FakeWorkerHandler.do_GET,FakeWorkerHandler.log_message, TestWatcherDiscovery, TestPDRolloutTypeChange

关键源码片段

sgl-model-gateway/e2e_test/k8s_integration/conftest.py

所有测试的基座，提供 kubectl 封装、等待函数、端口转发管理、轮询工具和可重试错误分类。

```
# 只重试传输层异常（连接级别），不重试 HTTP 状态码错误。
# httpx.HTTPStatusError (4xx/5xx) 故意排除，
# 网关返回 5xx 应被视为回归而暴露，不应被静默吞掉。
_TRANSIENT_ERRORS = (
    httpx.TransportError,
    httpx.TimeoutException,
    ConnectionError,
    OSError,
)

def _poll_until(
    predicate: Callable[[], bool],
    description: str,
    timeout: int,
    interval: float = 5,
) -> bool:
    """轮询直到 predicate 返回 True，超时时引发 TimeoutError。"""
    deadline = time.time() + timeout
    while time.time() < deadline:
        try:
            if predicate():
                return True
        except _TRANSIENT_ERRORS:
            logger.debug("Transient error while polling %s, retrying", description)
        time.sleep(interval)
    raise TimeoutError(
        f"Timed out after {timeout}s waiting for: {description}"
    )
```

sgl-model-gateway/e2e_test/k8s_integration/fake_worker.py

轻量级 HTTP 服务器模拟真实 worker 端点，支撑服务发现测试。

```
"""Minimal fake worker that mimics an SGLang worker for integration testing."""
import json
from http.server import BaseHTTPRequestHandler, HTTPServer

PORT = 8000

class FakeWorkerHandler(BaseHTTPRequestHandler):
    def do_GET(self):
```

```
        # 健康检查端点： always return OK
        if self.path == "/health":
            self.send_response(200)
            self.send_header("Content-Type", "text/plain")
            self.end_headers()
            self.wfile.write(b"OK")
        # 模型列表端点： 返回固定的 fake-model 信息
        elif self.path == "/v1/models":
            body = json.dumps({"object": "list", "data": [{"id": "fake-model", "object": "model",
            "created": 0, "owned_by": "sglang"}]})
            self.send_response(200)
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            self.wfile.write(body.encode())
        # 服务器信息端点： 返回 fake-model 的元信息
        elif self.path in ("/server_info", "/get_server_info"):
            body = json.dumps({"model_path": "fake-model", "version": "0.0.0-test", "tp_size": 1, "dp_
            size": 1})
            self.send_response(200)
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            self.wfile.write(body.encode())
        # 模型信息端点： 标记为生成模型
        elif self.path in ("/model_info", "/get_model_info"):
            body = json.dumps({"model_path": "fake-model", "is_generation": True})
            self.send_response(200)
            self.send_header("Content-Type", "application/json")
            self.end_headers()
            self.wfile.write(body.encode())
        else: # 未匹配路径返回 404
            self.send_response(404)
            self.end_headers()

    def log_message(self, format, *args):
        pass # 静默日志

if __name__ == "__main__":
    server = HTTPServer(("0.0.0.0", PORT), FakeWorkerHandler)
    print(f"Fake worker listening on port {PORT}", flush=True)
    server.serve_forever()
```

评论区精华

- Rust 版本无效：gemini-code-assist[bot] 指出 Dockerfile.gateway 中 FROM rust:1.90 的 1.90 尚未发布（当前稳定版约 1.85），导致构建失败，建议改为 rust:1.85 或通用标签如 rust:bookworm。
- 未使用函数：gemini-code-assist[bot] 发现 conftest.py 中的 _kubectl_json 和 test_reconciliation.py 中的 _get_worker_urls 定义但未被调用，建议移除以保持代码整洁。

- Pod 清理逻辑可以简化：gemini-code-assist[bot] 认为 _cleanup_pods 函数可以使用 kubectl delete pods -l app=fake-worker --ignore-not-found 替代列举和循环，并避免固定 time.sleep(5)。
  - Dockerfile 中 Rust 版本无效 (correctness)：未在后续提交中明确看到修复，但 PR 已合并，推测已修正或标记为待处理。
  - 未使用函数 _kubectl_json 和 _get_worker_urls (style)：未发现移除这些函数的后续提交，建议仍未处理。
  - 简化 Pod 清理逻辑 (design)：未明确解决。

风险与影响

- 风险：
  1. Rust 版本错误：Dockerfile.gateway 中 rust:1.90 镜像标签不存在，可能导致测试镜像构建失败，阻塞 CI。
  2. 外部运行时依赖：测试依赖 kind 集群和 kubectl，要求运行器支持 Docker daemon；当前配置为 ubuntu-22.04 满足条件，但未来环境变更可能导致兼容性问题。
  3. 测试耗时较长：reconciliation 测试需等待完整的 60 秒 tick 加上 30 秒缓冲，约 90 秒的轮询等待在 CI 中可能引发超时或 flaky 失败。
  4. 未使用函数遗留：_kubectl_json 和 _get_worker_urls 未使用但保留，增加维护负担和代码歧义。 - 影响：用户：无直接影响，仅 CI 流程变化。系统：新增 K8s 集成测试作业，首次构建镜像约 5-10 分钟，后续利用 GHA 缓存可缩短；总 CI 时间增加约 5 分钟。团队：为 SMG 核心的 K8s 服务发现和 PD 模式提供自动化回归防护，提升对 hostNetwork 滚动更新场景的信心。可维护性：Dockerfile.gateway 分离了测试与生产构建，减少构建时间；但需确保版本号同步更新。 - 风险标记：Rust 版本无效，依赖外部运行时环境，reconciliation 测试等待较长，存在未使用函数

关联脉络

- PR #24189 [SMG] Add K8s integration tests for service discovery and reconciliation: 本 PR 替代了 #24189，基于相同内容但将分支移至 sgl-project/sglang 以使用组织级 CI 自托管运行器和 secret。