Vimic2 Architecture Paper

CONTENTS

22. 10. Future Work

Vimic2 Architecture Paper

Vimic2 Architecture Paper

Vimic2: CI/CD Orchestration for Ephemeral Build Environments

Wesley Robbins — STSGYM LLC — April 2026

---

1. Executive Summary

Vimic2 is an infrastructure add-on for Vimic that orchestrates ephemeral CI/CD build environments using lightweight virtual machines. Rather than relying on persistent, always-on build runners, Vimic2 provisions dedicated VMs on-demand for each pipeline, executes the workload, captures logs and artifacts, then tears down the entire environment. This approach delivers complete isolation between pipelines, near-instant VM creation via copy-on-write backing files, and clean resource reclamation after every run.

The system is intentionally platform-agnostic. It currently targets Libvirt/KVM hypervisors but is designed to work with Hyper-V on Windows and Apple Virtual Machine on macOS through a common hypervisor abstraction layer.

---

2. What is Vimic2?

Vimic2 is not an AI tool. It is a CI/CD infrastructure automation layer.

There are two related but distinct components in the ecosystem:

Vimic — A cross-platform desktop application (Go/Wails/Vue 3) that provides local VM provisioning and AI-powered infrastructure chat. It runs entirely on the user's machine, talking to a local KVM/QEMU hypervisor and a local Ollama instance. No cloud, no data leaves the device.

Vimic2 CI/CD Orchestration Add-On — A server-side system (Go) that extends Vimic's cluster management capabilities to orchestrate ephemeral build environments for GitLab and GitHub Actions pipelines. This is the focus of this paper.

Vimic2 the add-on lives within the broader **System-Work** project, which manages the full lifecycle of CI/CD infrastructure: VM provisioning, runner registration, network isolation, log collection, and cleanup.

---

3. Problem Statement

Traditional CI/CD runners suffer from two persistent problems:

Resource Inefficiency — Persistent runners sit idle between jobs, consuming CPU, memory, and disk even when not in use. Organizations pay for capacity they only partially utilize.

Environment Pollution — When runners are reused across many pipelines, accumulated dependencies, cached artifacts, and version conflicts cause flaky tests and inconsistent builds. A change in one project's dependencies can silently break another's.

The alternative—ephemeral runners in cloud VMs—solves these problems but introduces new ones: slow provisioning (booting a full VM takes minutes), high cost (cloud VMs billed by the hour), and operational complexity (managing credentials, networks, and cleanup at scale).

Vimic2 addresses these challenges by combining the economics of ephemeral infrastructure with the speed of container-based startup, using VM backing files to create isolated runners in seconds rather than minutes.

---

4. Architecture Overview

Vimic2 sits between the GitLab/GitHub API and a pool of Libvirt/KVM hypervisors. When a webhook fires indicating a new pipeline, Vimic2:

1. Creates an isolated network for the pipeline's VMs
2. Provisions runner VMs from pre-built backing file templates
3. Registers each runner with the appropriate CI platform
4. Monitors the pipeline execution while collecting logs
5. Tears down VMs, networks, and runner registrations when the pipeline completes

---

5. Pipeline Lifecycle

5.1 Trigger

Vimic2 receives pipeline events through webhook endpoints:

Platform	Endpoint	Events
GitLab	`/api/webhooks/gitlab`	

Platform	Endpoint	Events
		Pipeline Hook, Push Hook, Tag Push Hook
GitHub	`/api/webhooks/github`	Push Event, Release Event, Workflow Dispatch
Generic	`/api/webhooks/generic`	Custom triggers, manual execution

The webhook handler validates the payload, extracts pipeline metadata (repository, branch, commit SHA, trigger type), and creates a pipeline record in the state store.

5.2 VM Provisioning

Vimic2 maintains a library of **backing file templates** — pre-built VM images in QCOW2 format, one per runner role:

Template	Base OS	Pre-installed Tools
`base-ubuntu-24.04`	Ubuntu 24.04	git, curl, basic utils
`builder-go`	Ubuntu 24.04	Go 1.23, Make, GCC, Docker
`tester-node`	Ubuntu 24.04	Node 20, npm, Playwright, Chromium
`deployer-docker`	Ubuntu 24.04	Docker 27, kubectl, Helm, Terraform

When a pipeline starts, Vimic2 creates **overlay QCOW2 images** using copy-on-write:

```
        qemu-img create -f qcow2 -F qcow2 \
          -b /var/lib/vimic2/templates/builder-go.qcow2
          /var/lib/vimic2/overlays/pipeline-abc123-
runner-001.qcow2
```

This takes less than a second and consumes only the disk space needed for delta changes, not a full copy of the base image. A pipeline requiring three runners (builder, tester, deployer) provisions all three VMs in under 5 seconds total.

5.3 Network Isolation

Each pipeline gets its own isolated network segment using Open vSwitch (OVS) with VXLAN encapsulation:

- **VXLAN VNI**: Derived from pipeline ID hash (e.g., `1000 + hash(pipeline_id) % 1000`)
- **Subnet**: `10.{vni}.0.0/24` — unique per pipeline
- **DNS**: Internal resolver for `*.pipeline.local`
- **NAT Gateway**: Outbound internet access through the hypervisor host

Security groups enforce role-based traffic rules:

Pipelines cannot reach each other's VMs. All inter-runner communication is scoped to the pipeline's isolated network.

5.4 Runner Registration

For each VM, Vimic2 SSHs into the fresh instance and registers a runner with the CI platform:

GitLab:

```
gitlab-runner register \
  --url https://gitlab.example.com \
  --registration-token $TOKEN \
  --executor shell \
  --description "runner-$VM_ID" \
  --tag "builder,$PIPELINE_ID" \
  --run-untagged=false
```

GitHub Actions:

```
./config.sh --url https://github.com/$OWNER/$RE
  --token $RUNNER_TOKEN \
  --labels "builder,$PIPELINE_ID" \
  --work /work
./run.sh
```

Tags include both the role (`builder`, `tester`, `deployer`) and the pipeline ID, ensuring the correct runners pick up the correct jobs and nothing else.

5.5 Build and Test Execution

Once registered, the CI platform's scheduler dispatches jobs to the Vimic2-managed runners. The pipeline executes normally — Vimic2 does not modify or intercept the CI configuration. Standard `.gitlab-ci.yml` or GitHub Actions workflow files drive the actual build.

During execution, each runner streams logs to Vimic2's log aggregator:

```
{
  "pipeline_id": "pl-abc123",
  "runner_id": "runner-001",
  "stage": "build",
  "job": "compile",
  "timestamp": "2026-04-04T14:00:00Z",
  "log": "go build -o bin/service ./cmd/server'
  "metadata": {
    "cpu_usage": 85,
    "memory_usage": 4096,
    "duration_seconds": 120
  }
}
```

Logs are written to Elasticsearch for search and to S3/MinIO for archival. Pipeline metadata (status, duration, artifacts) is stored in PostgreSQL.

5.6 Cleanup

When the pipeline reaches a terminal state (success, failure, or cancelled), Vimic2 executes a cleanup sequence:

1. **Unregister runners** — Remove each runner from GitLab/GitHub so it stops accepting new jobs
2. **Stop services** — Halt Docker, GitLab Runner, and other running services inside each VM
3. **Archive logs** — Bundle `/var/log` from each runner, upload to S3, tag with pipeline ID
4. **Destroy VMs** — `virsh destroy` then `virsh undefine` each runner VM

5. **Delete overlays** — Remove the delta QCOW2 files (base image untouched)
6. **Tear down network** — Delete the OVS bridge and VXLAN interface for the pipeline
7. **Update state** — Mark pipeline as `cleaned` in PostgreSQL with final status and artifact URLs

The entire cleanup takes under 30 seconds. No VM state persists between pipelines.

---

6. Component Details

6.1 Webhook Handler

The webhook handler exposes three endpoints and validates signatures for each platform:

- GitLab: HMAC-SHA256 using `X-Gitlab-Token`
- GitHub: HMAC-SHA256 using `X-Hub-Signature-256`
- Generic: Static token in `Authorization` header

After validation, payloads are normalized into a common `PipelineRequest` struct and enqueued to Redis for asynchronous processing.

6.2 Pipeline Coordinator

The coordinator owns the state machine for each pipeline:

It orchestrates calls to the VM Pool Manager, Runner Orchestrator, and Network Manager, handling partial failures by rolling back any completed steps before reporting the error.

6.3 VM Template Manager

Templates are stored as read-only QCOW2 files in `/var/lib/vimic2/templates/`. Each template has a JSON manifest describing:

```
        {
          "id": "builder-go",
          "name": "Go Builder",
```

```
            "base_image": "base-ubuntu-24.04.qcow2",
            "packages": ["golang-go", "make", "gcc",
"docker.io"],
            "runner_install_url": "https://packages.gitla
install/repositories/runner/gitlab-runner"
        }
```

Templates are built once using `virt-customize` and can be refreshed periodically to apply security patches.

6.4 VM Pool Manager

The pool manager tracks available VMs and implements an acquire/release interface:

```
        func (m *VMPoolManager) AcquireVM(poolID string
timeout time.Duration) (*VM, error)
        func (m *VMPoolManager) ReleaseVM(vmID string)
```

VMs transition through states: `creating` → `running` → `idle` → `busy` → `stopping` → `stopped`. The pool maintains minimum and maximum sizes per pool type. When demand exceeds `min_size`, new VMs are created. When demand drops below `min_size`, excess VMs are terminated.

6.5 Runner Orchestrator

The orchestrator handles runner registration and unregistration via SSH. It maintains a database of active runner tokens so it can cleanly unregister runners during cleanup even if the VM is still reachable.

6.6 Log Collector

Log collection runs as a sidecar process on each runner VM, tailing service logs and forwarding them over a TLS connection to the central Elasticsearch cluster. On shutdown, it performs a final flush to ensure no logs are lost.

---

7. Storage Architecture

Data Type	Storage	Retention
VM templates	`/var/lib/vimic2/templates/` (local filesystem)	Until manually refreshed
VM overlays	`/var/lib/vimic2/overlays/` (local filesystem)	Deleted after pipeline cleanup
Pipeline metadata	PostgreSQL	90 days
Build logs	Elasticsearch + S3	30 days in ES, 1 year in S3
Build artifacts	S3/MinIO	30 days

---

8. Deployment Considerations

Vimic2 is designed to run alongside Vimic's cluster management components on one or more dedicated hypervisor hosts. The typical deployment on a Linux host requires:

- **Libvirt + KVM**: For VM lifecycle management
- **Open vSwitch**: For network isolation between pipeline environments
- **Redis**: For the job queue (can be co-located)
- **PostgreSQL**: For pipeline state (can use an existing instance)
- **Elasticsearch + S3**: For log aggregation (can use existing infrastructure)

The controller component (webhook handler, coordinator, pool manager) runs as a Go service and can be deployed as a Docker container or as a systemd service. Runner VMs are ephemeral and do not run any Vimic2 software beyond the log collector sidecar.

---

9. Comparison with Alternatives

Feature	Vimic2	GitLab.com Runners	GitHub Actions	Jenkins (static agents)
VM per pipeline	Yes	No (containers)	No (containers)	No

Feature	Vimic2	GitLab.com Runners	GitHub Actions	Jenkins (static agents)
Isolated network	Yes	No	No	No
Copy-on-write provisioning	Yes	N/A	N/A	No
Provisioning time	< 5 sec	N/A	N/A	Minutes
Automatic cleanup	Yes	Yes	Yes	Manual
Multi-platform CI	GitLab + GitHub	GitLab only	GitHub only	Any
Runs on-premises	Yes	No (SaaS)	No (SaaS)	Yes

Vimic2 occupies a specific niche: teams that need **full VM isolation** (not containers) for their CI workloads, want to run **on-premises** without cloud dependency, and need **sub-minute provisioning** rather than waiting for cloud VM boot times.

---

10. Future Work

Near-term development priorities:

- **Multi-host pooling**: Distribute runner VMs across multiple hypervisor hosts for capacity and resilience
- **Spot/preemptible support**: Allow pools to include lower-cost VMs that may be reclaimed, with graceful job migration
- **Artifact cache acceleration**: Share build caches between pipelines using a distributed filesystem to speed up incremental builds
- **Windows and macOS hypervisors**: Implement the Hyper-V and AppleVMM backends for the hypervisor abstraction layer

---

Document version 1.0 — April 2026 Part of the STSGYM P1 Implementation Deliverable 4.3