

00 Architecture

P1 Lane-VM Host

P2 Edge Worker

P3 GitHub App

P4 Programs

P5 Provisioning

→ Build Order

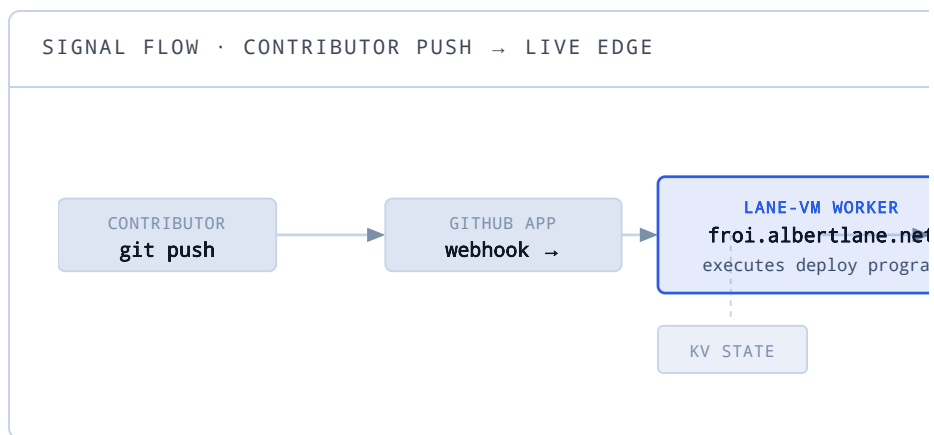
OWNER ACTIONS

5 items require provisioning before the Worker goes live.

Sovereign Froi Edge

Lane-VM as Infrastructure

A deployment pipeline where Lane-VM is the execution host and the GitHub App is an authenticated event emitter — not the infrastructure owner. GitHub webhooks. Everything after that is sovereign.



Design inversion. A conventional GitHub App owns the pipeline infrastructure secrets. Here, the App fires signed webhooks and provides installation tokens, executes a typed instruction program stored in the repo, and writes but never controls what happens at the edge.

P1

Lane-VM Deployment Host

TO BUILD

The Lane-VM interpreter from lane-mcp's Phase 15 lanevm module is the host. The ISA is unchanged — the host registers *sovereign deployment* functions, preserving the sandboxed typed-instruction property.

[workers/lanevm-deploy.ts](#)

Extracted from lane-mcp's `packages/modules/lanevm/src/` and extended specific to deployment. These are not new ISA instructions — they are

specific to deployment. These are not new IOA instructions – they are handlers the program can call by name, and the host controls exactly

COMPUTE OP	WHAT IT DOES
github_fetch	GET from api.github.com using App installation token. Returns artifact metadata or raw bytes.
r2_store	Write bytes to the bound R2 bucket at a given key path.
kv_write / kv_read	Read/write deployment state (current SHA, timestamps, PR preview keys).
sha256_verify	Verify an artifact's SHA-256 against an expected digest. Returns boolean in MEM.
webhook_verify	HMAC-SHA256 constant-time comparison of the GitHub webhook signature.

Per-call caps: 512 instructions (double lane-mcp's 256 – build operati
memory entries. SYS_CALL and IO_SINK remain unconditionally rejecte

```
WORKERS/LANEVM-DEPLOY.TS – HOST FUNCTION REGISTRATION

// Registered at Worker startup – these are the ONLY comput
const DEPLOY_HOST_FUNCTIONS = {
  webhook_verify: (secret: string, sig: string, body: strin
    hmacVerify(secret, sig, body),

  github_fetch: async (url: string, token: string) => {
    if (!url.startsWith('https://api.github.com/'))
      throw new Error('SSRF: host not allowed');
    return fetch(url, { headers: { Authorization: `Bearer $
  },

  r2_store: (env: Env, key: string, data: ArrayBuffer) =>
    env.FROI_ARTIFACTS.put(key, data),

  kv_write: (env: Env, key: string, value: string) =>
    env.FROI_DEPLOY_STATE.put(key, value),

  sha256_verify: async (data: ArrayBuffer, expected: string
    const digest = await crypto.subtle.digest('SHA-256', da
    return hex(digest) === expected;
  },
};
```

P2

Edge Worker Scaffold

TO BUILD

`workers/froi-edge-worker.ts` — the Cloudflare Worker deployed at `froi-edge-worker.com`. Receives GitHub App webhooks, loads the appropriate Lane-VM program, and executes it.

Route table

POST	<code>/webhook</code> GitHub App webhook receiver. Verifies HMAC-SHA256 signature (push/pull_request), loads the corresponding Lane-VM program, and executes it.
GET	<code>/</code> Setup page. Current deployment status, APK download link, and build history from KV.
GET	<code>/download/:artifact</code> Serve artifact directly from R2. <code>/download/latest.apk</code> returns the latest branch APK. No redirect — raw bytes from the R2 binding.
GET	<code>/status</code> JSON deployment state: current SHA, build timestamp, and last successful build read from KV.
GET	<code>/preview/:pr</code> PR preview artifact. Serves the build artifact associated with the PR number. Key: <code>preview:{pr}:apk</code> in R2.
GET	<code>/health</code> Liveness probe. Returns KV/R2 binding state and last successful build.

Webhook execution flow

POST /WEBHOOK — EXECUTION PATH
1. Read <code>X-Hub-Signature-256</code> header 2. Execute <code>webhook_verify(GITHUB_APP_SECRET, sig, rawBody)</code> → reject 401 immediately if mismatch

3. Parse event type from X-GitHub-Event header
 - push → load programs/deploy-main.lvm.json
 - pull_request → load programs/deploy-preview.lvm.json
 - * other events → 204 No Content (acknowledged, ignored)
4. Execute the Lane-VM program with the event payload as in
5. Return 200 + execution summary, or 500 + Lane-VM error

P3

GitHub App – Input Surface Only

OWNER ACTION

The GitHub App is registered once. After that, it fires webhooks and is request. It does not own, trigger, or control any deployment infrastruc

[github-app/manifest.json](#)

Stored in the Froi-Browser repo. Used by the one-time registration scri
lives here — not in GitHub's UI.

GITHUB-APP/MANIFEST.JSON

```
{
  "name": "Froi Sovereign Edge",
  "url": "https://froilane.net",
  "hook_attributes": {
    "url": "https://froilane.net/webhook",
    "active": true
  },
  "redirect_url": "https://froilane.net/github-app/s",
  "default_permissions": {
    "contents": "read",
    "pull_requests": "read",
    "checks": "write"
  },
  "default_events": ["push", "pull_request"],
  "public": false
}
```

[github-app/register.ts](#)

One-time setup script. POSTs the manifest to GitHub's App creation A
private key, outputs the Wrangler secret commands to run. Never stor
the output straight to your shell.

OWNER ACTION — ONE TIME

```
npx ts-node github-app/register.ts
# Outputs:
wrangler secret put GITHUB_APP_ID          --config wrangler.fr
wrangler secret put GITHUB_APP_SECRET      --config wrangler.fr
wrangler secret put GITHUB_APP_PRIVATE_KEY --config wrangler.fr
```

P4

Deployment Programs

TO BUILD

The pipeline itself is a Lane-VM program stored in the repo as a `.lvm`. deployments work means making a PR that changes this file — the same review flow as any other code change.

[programs/deploy-main.lvm.json](#)

MAIN BRANCH DEPLOYMENT PROGRAM

```
{
  "description": "Froi main-branch deployment pipeline",
  "max_instructions": 512,
  "instructions": [
    // Step 1 — Verify the webhook is genuinely from GitHub
    { "type": "COMPUTE", "op": "webhook_verify",
      "operands": [ "$.secret", "$.sig", $.body ] },
    { "type": "MEM_WRITE", "address": 0, "value": "verified" }

    // Step 2 — Fetch the build artifact from GitHub using
    { "type": "COMPUTE", "op": "github_fetch",
      "operands": [ $.artifacts_url, $.installation_token ] },
    { "type": "MEM_WRITE", "address": 1, "value": $.result }

    // Step 3 — Verify artifact integrity
    { "type": "COMPUTE", "op": "sha256_verify",
      "operands": [ "mem:1", $.expected_sha ] },

    // Step 4 — Store artifact in R2
    { "type": "COMPUTE", "op": "r2_store",
      "operands": [ "froii/latest.apk", "mem:1" ] },
    { "type": "COMPUTE", "op": "r2_store",
      "operands": [ "froii/sha256.txt", $.expected_sha ] },

    // Step 5 — Update deployment state in KV
    { "type": "COMPUTE", "op": "kv_write",
      "operands": [ "deploy_main.sha", $.commit_sha ] }
```

```

    "operands": ["deploy:main:ts", "$.timestamp"] },
    { "type": "COMPUTE", "op": "kv_write",
      "operands": ["deploy:main:ref", "$.ref"] }
  ]
}

```

programs/deploy-preview.lvm.json follows the same shape but writes {pr_number}:apk in R2 and records PR metadata in KV.

The program is the audit trail. Every deployment decision is encoded in a \repo. If a deployment behaves unexpectedly, you diff the program. If some the pipeline works, it's a PR like any other — reviewable, revertible, traceabl

P5

Wrangler Config + Provisioning

OWNER ACTIONS ·

wrangler.froi-edge.toml

WRANGLER.FROI-EDGE.TOML

```

name = "froie-edge"
main = "workers/froi-edge-worker.ts"
compatibility_date = "2024-09-23"

routes = [{ pattern = "froie.albertlane.net/*", zone_name =

[[kv_namespaces]]
binding = "FROI_DEPLOY_STATE"
id      = "REPLACE_WITH_KV_ID"

[[r2_buckets]]
binding    = "FROI_ARTIFACTS"
bucket_name = "froie-artifacts"

```

Owner actions — in order

1 · CREATE KV NAMESPACE

```

wrangler kv:namespace create FROI_DEPLOY_STATE --config wrangl
# Copy the returned id into wrangler.froi-edge.toml

```

2 · CREATE R2 BUCKET

```
wrangler r2 bucket create froi-artifacts --config wrangler.froi
```

3 · REGISTER GITHUB APP (P3 – ONE-TIME)

```
npx ts-node github-app/register.ts  
# Pipe output to wrangler secret put commands
```

4 · DEPLOY THE WORKER

```
wrangler deploy --config wrangler.froi-edge.toml
```

5 · DNS – CNAME AT YOUR REGISTRAR / CLOUDFLARE DASHBOARD

```
froi.albertlane.net CNAME froi-edge.{your-workers-subdomain}
```

The same CLOUDFLARE_API_TOKEN that gates every other Worker in the
All five steps can run in a single session once the token is provisioned

→ Implementation Order

#	FILE	DEPENDS ON
1	workers/lanevm-deploy.ts	lane-mcp Phase 15 ISA (reference c
2	workers/froi-edge-worker.ts	lanevm-deploy.ts
3	programs/deploy-main.lvm.json	lanevm-deploy.ts host ops defined
4	programs/deploy-preview.lvm.json	deploy-main.lvm.json pattern
5	github-app/manifest.json	None
6	github-app/register.ts	manifest.json
7	wrangler.froi-edge.toml	None
8	KV + R2 provisioning	wrangler.froi-edge.toml, CLOUDFLARE_API_TOKEN

9	GitHub App registration	Worker live at froil.albertlane.net/we
10	APK build workflow	ACT-009 resolved or local cargo apk

APK build note. Steps 1–9 can be built and deployed before any APK exist honest status page until the first artifact lands. The build itself — cargo al android-bridge target — needs either ACT-009 resolved (GitHub Actions run upload manually as the first artifact. Either way, the pipeline is live before t nothing to serve yet.