# **M2 — Multi-hop, Policy Hardening, Multisig Ownership, Observability (≤300 steps)**

## **A. Branch, baseline, and environment**

1. Create branch feature/m2-multihop-policy-obs.
2. Tag current commit pre-m2-cut to freeze M1 state.
3. Ensure .env.example includes: MULTISIG\_OWNER, GUARDIAN, UNIV3\_ROUTER, UNIV3\_QUOTER, WETH, SEPOLIA\_RPC, ANVIL\_FORK\_URL.
4. Add deployments/sepolia.m2.template.json with empty addresses for new contracts (if any).
5. Add Make/PNPM script pnpm env:check to assert required envs exist before running deploy/tests.

## **B. Contract surface extensions (Solidity)**

### **B1. Multi-hop support in UniswapV3Adapter**

1. In contracts/src/venues/adapters/UniswapV3Adapter.sol, add **multi-hop** support via exactInput(bytes path, address recipient, uint256 amountIn, uint256 minOut).

Define a new calldata variant:  
  
 struct UniV3ExactInput {

bytes path; // tightly packed: tokenIn (20) | fee (3) | tokenMid (20) | fee (3) | tokenOut (20)

address recipient;

uint256 amountIn;

uint256 minOut;

uint256 deadline;

}

1. Expand adapter execute(bytes) to **branch**:  
   * If payload decodes as UniV3SwapParams → call exactInputSingle.
   * Else if decodes as UniV3ExactInput → call exactInput.
2. Add quoteExactInput(bytes path, uint256 amountIn) → uint256 using QuoterV2.quoteExactInput.
3. Add payload **discriminator**: first 4 bytes “type tag” approach:  
   * 0xAABBCC01 → single-hop,
   * 0xAABBCC02 → multi-hop.
4. Enforce deadline >= block.timestamp for both modes; revert with Errors.InvalidInput() otherwise.
5. Ensure approve(router, 0) then approve(router, amountIn); zero approval after swap if allowance remains.

### **B2. NATIVE/WETH handling (opt-in)**

Add contracts/src/common/Native.sol with helpers:  
  
 library Native { address constant NATIVE = address(0); }

1. In adapter, **for M2 leave native disabled by default**; add placeholder logic that reverts if path ends in Native.NATIVE. (Enable in M3.)
2. In BundleExecutor, assert tokenIn != address(0) and tokenOut != address(0) (ERC20 only for M2).

### **B3. Policy & registry hardening**

1. In PolicyGuards.sol, add **adapter-specific validation**:  
   * If Step.kind == UNIV3\_SWAP:  
     + Validate fee tier ∈ {500, 3000, 10000}.
     + For multi-hop path, validate path.length % 23 == 20 || 23\*k + 20 and disallow duplicate consecutive tokens.
2. Add setAllowedFee(uint24 fee, bool allowed) with mapping to override defaults if needed.
3. Add maxSteps guard configurable via owner, default 8; expose setMaxSteps(uint8).

In RouteRegistry, add version and metadata:  
  
 struct RouteMeta { address owner; bool disabled; uint64 version; bytes32 label; }

1. Add label parameter to registerRoute (e.g., bytes32 slug), emitted in RouteRegistered.
2. Add bumpVersion(routeId) only owner; auto-bump on updateRoute.
3. Add event RouteOwnershipTransferred(bytes32 routeId, address oldOwner, address newOwner); implement transferRouteOwnership(routeId, newOwner).

### **B4. Multisig ownership (contracts)**

1. For **M2 only**, keep contract owner fields pointing to EOA, but add post-deploy script step to transfer to MULTISIG\_OWNER. (Full timelock in M3.)
2. Add Ownable2Step pattern equivalents (only if using OZ v5; otherwise keep simple and document migration).

## **C. Executor upgrades (Solidity)**

1. In BundleExecutor.execute, iterate steps and **dynamically resolve** tokenIn/out from each step (ignore outer tokenIn/tokenOut arguments except for pre/post sanity).
2. Enforce **token chain continuity**: prev.tokenOut == next.tokenIn.
3. Add **per-step minOut** enforcement already in Step; continue to enforce minTotalOut at end.
4. Track gasUsed per step and accumulate in local var for event emission.

Emit enriched BundleExecuted:  
  
 event BundleExecuted(bytes32 routeId, address caller, uint256 amountIn, uint256 totalReceived, uint256 steps, uint256 totalGas, uint256 ts);

## **D. Foundry unit tests for new behavior**

1. Update/extend contracts/test/venues/UniswapV3Adapter.unit.t.sol:  
   * Decode discriminators, revert on unknown type tags.
   * Fee tier allow/deny behavior via guards.
2. Add contracts/test/policy/PolicyGuards.univ3.t.sol:  
   * setAllowedFee path.
   * Multi-hop path shape validation.
3. Add contracts/test/policy/RouteRegistry.versioning.t.sol:  
   * Version bump on update.
   * Ownership transfer emits event and restricts updates.
4. Add contracts/test/execution/Executor.chain-continuity.t.sol:  
   * Revert if token continuity broken.
5. Extend invariant test to ensure no **unbounded approvals** remain after execute for supported tokens.

## **E. Integration tests (Anvil fork or local stack)**

1. Create contracts/test/integration/Univ3.multiHop.t.sol:  
   * Build path DAI → WETH → USDC.
   * Use QuoterV2.quoteExactInput to compute expected out; set minOut = expectedOut \* (1 - slippage).
   * Execute via adapter with UniV3ExactInput.
2. Extend BundleExecutor.univ3Route.t.sol to multi-hop:  
   * Register route with 1 step (UNIV3-multi) or 2 single-hop steps (for comparison).
3. Add **gas snapshot** for multi-hop route; set generous cap (for regression alarms).

## **F. TS payload/validation parity**

### **F1. Encoders/Decoders**

In packages/core-exec/src/encode.ts, add:  
  
 export const TAG\_UNIV3\_SINGLE = "0xaabbcc01";

export const TAG\_UNIV3\_MULTI = "0xaabbcc02";

export function encodeUniV3Path(hops: {token: `0x${string}`, fee?: number}[]): `0x${string}` {

// hops: [tokenIn, {fee, tokenMid}, {fee, tokenOut}] → 20 | 3 | 20 | 3 | 20

}

export function encodeUniV3ExactInput(args: {

path: `0x${string}`; recipient: `0x${string}`;

amountIn: bigint; minOut: bigint; deadline: bigint;

}): `0x${string}` { /\* abi.encode(TAG + args) \*/ }

1. Add decoders decodeUniV3Path and guards checking structure (length & alignment).

### **F2. Step builders**

1. In packages/adapters-evm/src/UniswapV3AdapterClient.ts:  
   * Add buildMultiHop({hops, amountIn, minOut, recipient, deadline}) → Step:  
     + Compute path = encodeUniV3Path(hops).
     + payload = encodeUniV3ExactInput({path, ...}).
2. Add helper assertTokenContinuity(steps) used by RouteBuilder.

### **F3. Quoting**

1. Add quoteMultiHop(publicClient, quoter, path, amountIn) → bigint using QuoterV2.
2. Implement aggregate route quote:  
   * If step.kind is UNIV3\_MULTI → single quoter call.
   * If multiple single hops → chain the amounts.

### **F4. Route builder & ID parity**

1. Update computeRouteId encoder to include **version** and **label** if present to match Solidity changes.
2. Add unit test to compare TS/Solidity routeId for a fixed route.

## **G. CLI expansions**

Add subcommand route build-multihop:  
  
 route build-multihop \

--adapter 0x... \

--hops '["0xTokenIn",500,"0xMid",3000,"0xTokenOut"]' \

--amount-in 1000000000000000000 \

--min-out 0 \

--recipient 0xYOU \

--deadline <unix> \

--salt 0x... \

--label swap-dai-weth-usdc

1. Add route quote-multihop --hops ... --amount-in ... --network ... returning expectedOut.
2. Update route register to pass label and ensure version bump behavior is visible in output.
3. Add route validate --file route.json, printing:  
   * step continuity (OK/FAIL),
   * allowed adapters (OK/FAIL),
   * fee tiers (OK/FAIL),
   * maxSteps.
4. Update execute to accept --per-step-bps <e.g., 50> to compute each step’s minAmountOut from quotes.
5. Add --json output for quote and execute so agents can parse deterministically.
6. Add CLI error surface for policy violations with friendly hints.

## **H. Config updates**

Extend config/venues.json:  
  
 {

"sepolia": {

"uniswapV3": {

"router": "0x...",

"quoter": "0x...",

"weth": "0x..."

}

}

}

Extend config/policies.json:  
  
 {

"allowedAdapters": ["0xUniv3Adapter","0xNoopAdapter"],

"allowedFees": [500,3000,10000],

"maxSteps": 8,

"slippageDefaultBps": 50

}

1. Write scripts/dev/sync-policies.ts to read on-chain allowedAdapter/allowedFees/maxSteps and mirror to config/policies.json.

## **I. Deployment scripts (Foundry)**

1. Update script/Deploy.s.sol to **reuse** existing addresses for unchanged contracts; only redeploy adapter if needed; log version.
2. Update script/Configure.s.sol:  
   * Set allowed fees based on config/policies.json.
   * setMaxSteps.
3. Add script/OwnershipTransfer.s.sol:  
   * Transfer PolicyGuards, RouteRegistry, SettlementVault owner to env:MULTISIG\_OWNER.
4. Add script/RouteLabelMigrate.s.sol to back-fill label for existing routes (if any) by re-registering or setting label where available.

## **J. Observability: Events → Telemetry**

1. Add **indexer** in packages/core-exec/src/telemetry/indexer.ts:  
   * Subscribes to RouteRegistered, RouteUpdated, RouteDisabled, BundleExecuted, AdapterExecuted.
   * Writes JSON lines to ./.out/telemetry/events.ndjson (M2 local).
2. Add **metrics sink** packages/core-exec/src/telemetry/metrics.ts:  
   * Export counters: bundles\_executed\_total, adapter\_gas\_used{kind}, route\_versions\_total.
   * Implement a simple HTTP /metrics server (Prometheus text format) guarded by TELEMETRY\_BIND=:9464 (dev only).
3. Wire CLI to emit **hooks**:  
   * After register → call onRouteRegistered.
   * After execute → call onBundleExecuted.
4. Add pnpm dev:metrics script to run the metrics server.

## **K. Security hardening (scope M2)**

1. Add slither job (if available) or mythril basic scan in CI as a **non-blocking** informational step.
2. Write “approval hygiene” Foundry test ensuring no non-zero leftover approvals after execute for ERC20s used in tests.
3. Add revert reasons for all guards (extend Errors.sol):  
   * error InvalidFee(uint24 fee)
   * error InvalidPath()
   * error MaxStepsExceeded(uint256 provided, uint256 allowed)
4. Ensure adapters **do not transfer** unexpected tokens to recipient (stick to router behavior).
5. Verify guard coverage: add unit tests for each new error in PolicyGuards.

## **L. Gas & performance baselines**

1. Add forge snapshot target for multi-hop route (2 hops).
2. Introduce GAS\_LIMIT\_JSON with per-test caps; assert within +15% of baseline to catch regressions.
3. Record gas figures in docs/dev/gas.md (single-hop vs multi-hop comparisons).

## **M. TS tests (Vitest) for multihop & parity**

1. packages/adapters-evm/test/univ3.client.multihop.spec.ts:  
   * Encode path → decode path roundtrip.
   * Compare quoteMultiHop vs on-chain quoter (fork).
2. packages/core-exec/test/routeBuilder.validation.spec.ts:  
   * Broken continuity → fail.
   * Disallowed fee → fail.
   * Allowed path length/shape → pass.
3. packages/cli/test/route.multihop.cli.spec.ts:  
   * Build → validate → quote → execute (dry-run on mocked provider).

## **N. End-to-end scripts (TS)**

1. Create packages/core-exec/src/run/sepoliaMultihop.ts:  
   * Load config (DAI/WETH/USDC sample).
   * Build multi-hop route with fee tiers from config.
   * Quote with quoter; compute minOut = quote \* (1 - slippageBps/10000).
   * Register (label “dai-weth-usdc”).
   * Execute; log BundleExecuted receipt; assert balances.
2. Add "demo:multihop": "tsx packages/core-exec/src/run/sepoliaMultihop.ts" to root package.json.

## **O. Docs (developer & ops)**

1. Update docs/content/uc4-multiswap-permissions.mdx:  
   * Add multi-hop diagrams (path encoding).
   * Document fee allowlist and maxSteps guard.
2. Add docs/runbooks/m2-ops.md:  
   * How to transfer ownership to multisig.
   * How to set allowed fees and adapters.
   * How to pause/unpause.
   * How to roll back (disable route, pause executor).
3. Update README Quickstart with multi-hop example and CLI commands.
4. Add docs/dev/telemetry.md showing how to run local indexer and view /metrics.

## **P. CI pipeline extensions**

1. Add job ts-fork-tests:  
   * Start Anvil fork (background).
   * Run Vitest suite that hits the live quoter (marked @fork).
2. Add contracts-integration job executing Foundry multi-hop tests with --fork-url.
3. Add artifact upload of events.ndjson from telemetry job so teammates can inspect logs.
4. Ensure jobs run in intended order: unit → ts unit → solidity unit → integration → fork tests (non-blocking).

## **Q. Staging deployment & smoke**

1. Deploy adapter/executor updates to **staging Sepolia** using Deploy.s.sol.
2. Run OwnershipTransfer.s.sol to set owners to MULTISIG\_OWNER.
3. Configure guards (Configure.s.sol) with allowed fees and maxSteps.
4. Seed staging tokens (DAI/WETH/USDC test mints or faucets).
5. Run demo:multihop pointed at staging RPC; record routeId and tx hash.
6. Verify metrics:  
   * bundles\_executed\_total increments.
   * adapter\_gas\_used{UNIV3\_SWAP} non-zero.
7. Verify events.ndjson contains BundleExecuted with non-zero totalGas.

## **R. Acceptance tests (sign-off)**

1. Create acceptance/m2.md with 10 objective checks:  
   * Multi-hop quote parity ±0.5% vs chain quoter (on fork).
   * Step continuity enforced.
   * Fee allowlist enforced.
   * Max steps enforced.
   * Pausing blocks execute.
   * Ownership transferred to multisig.
   * No residual approvals.
   * Gas budget within baseline +15%.
   * CLI --json outputs machine-readable artifacts.
   * Telemetry events captured.
2. Run acceptance checklist on staging and attach artifacts (tx hashes, logs, screenshots).

## **S. Backwards compatibility & migrations**

1. Document in docs/dev/migrations/m2.md:  
   * Routes registered in M1 remain valid (version default = 0, label empty).
   * New registration path requires label; old clients can pass 0x0 label via wrapper (provided).
2. Add CLI fallback: if label not provided, set 0x00..00 and warn.

## **T. Final security & review gates (M2 scope)**

1. Run slither and review major findings; mark mitigated or consciously accepted with justification.
2. Manual review: approvals, token transfers, deadline checks.
3. Peer sign-off on PolicyGuards rules and default fee tiers.
4. Confirm all custom errors are surfaced to CLI with mapped messages.

## **U. Release packaging**

1. Bump package versions: adapters-evm, core-exec, cli to 0.2.0-m2.
2. Generate ABIs and publish to packages/\*/dist/abi.
3. Create release notes m2-release-notes.md summarizing features, configs, and known limits (no native unwrap).
4. Tag repo m2-multihop-policy-obs.
5. Merge feature/m2-multihop-policy-obs → main after all CI green.

## **V. Copy-pasteable deltas (key snippets)**

### **V1. Adapter payload tags (Solidity)**

bytes4 constant TAG\_UNIV3\_SINGLE = 0xaabbcc01;

bytes4 constant TAG\_UNIV3\_MULTI = 0xaabbcc02;

### **V2. Multi-hop execute (Solidity — inside adapter)**

function execute(bytes calldata payload) external returns (Types.AdapterResult memory r) {

uint256 g0 = gasleft();

bytes4 tag;

assembly { tag := calldataload(payload.offset) } // read first 4 bytes

if (tag == TAG\_UNIV3\_SINGLE) {

UniV3SwapParams memory sp = abi.decode(payload[4:], (UniV3SwapParams));

// ... approve & exactInputSingle ...

return Types.AdapterResult(true, sp.amountIn, amountOut, g0 - gasleft(), "");

} else if (tag == TAG\_UNIV3\_MULTI) {

UniV3ExactInput memory mi = abi.decode(payload[4:], (UniV3ExactInput));

// ... approve & exactInput using mi.path ...

return Types.AdapterResult(true, mi.amountIn, amountOut, g0 - gasleft(), "");

}

revert Errors.InvalidInput();

}

### **V3. TS path encoder (concise)**

export function encodeUniV3Path(hops: (`0x${string}` | number)[]): `0x${string}` {

// hops like ["0xTokenIn", 500, "0xMid", 3000, "0xTokenOut"]

const bytes: number[] = [];

for (let i = 0; i < hops.length; i++) {

const v = hops[i];

if (typeof v === "string") { // token

bytes.push(...Buffer.from(v.slice(2).padStart(40, "0"), "hex"));

} else { // fee uint24

const b = Buffer.alloc(3); b.writeUIntBE(v, 0, 3); bytes.push(...b);

}

}

return ("0x" + Buffer.from(bytes).toString("hex")) as `0x${string}`;

}

### **V4. CLI command wiring (example)**

routeCmd.command("build-multihop")

.requiredOption("--adapter <addr>")

.requiredOption("--hops <json>") // '["0xA",500,"0xB",3000,"0xC"]'

.requiredOption("--amount-in <wei>")

.requiredOption("--min-out <wei>")

.requiredOption("--recipient <addr>")

.requiredOption("--deadline <ts>")

.requiredOption("--salt <bytes32>")

.option("--label <bytes32>")

.action(async (opts) => { /\* build step, route, print JSON \*/ });

## **W. Done-ness criteria (what “exhaustive” means here)**

1. All new Solidity units & integrations passing locally and in CI (fork tests included).
2. TS multi-hop quoting parity validated on fork within ±0.5%.
3. CLI can build/validate/register/quote/execute multi-hop route end-to-end on Sepolia fork and staging.
4. Policy guards configurable and enforced (allowed fees/maxSteps).
5. Ownership transferred to multisig; pause/unpause verified.
6. Telemetry (events & metrics) observable locally with sample outputs attached to the PR.
7. Release notes prepared; known limitations documented.