Absolutely—here's how you'd design an **MCP server purpose-built for testing blockchain networks**. I'll map the capabilities (main types → sub-types → concrete tools/resources), and show how they plug into the common Web3 testing stack (Foundry/Anvil, Echidna, Mythril, Tenderly, etc.), with references.

MCP server for blockchain testing

A) Core MCP capability types (what the server exposes)

1. **Tools (actions)**

- Invokable operations: run tests, spin up/fork a node, simulate tx, fuzz, symbolically analyze bytecode, etc. Tools are named, JSON-schema'd functions per MCP spec. (MCP Protocol)

2. **Resources (context)**

- Readable artifacts: compiled ABIs/bytecode, test reports, traces, gas profiles, coverage, fork metadata, chain snapshots. (MCP resources are listable/fetchable by clients.) (MCP Protocol)

3. **Prompts (templates)**

- Reusable test recipes & runbooks (e.g., "invariant suite for AMMs", "simulate governance proposal flow"), discoverable via prompts/list. (MCP Protocol)

> Note: MCP itself rides on JSON-RPC 2.0 messaging and supports transports like stdio/HTTP/WS; you'll implement whichever fits your deployment. (Medium)

B) Sub-systems & feature buckets (with concrete Web3 integrations)

1) **Local & forked chain control**

- **anvil.start** — start local EVM (optionally fork mainnet/testnet), configure block/time, chainId. (Foundry's Anvil supports forking & dev-node features.) (GitHub)
- **anvil.setState** — cheatcode-like helpers for tests: set balance, prank msg.sender, warp time/block. (Cheatcodes underpin state control in Foundry tests.) (Abstract Docs)
- **anvil.stop/reset** — tear down or reset to snapshot.

Resources: `/chains/current`, `/forks/:id/meta` (rpcUrl, blockNumber).

2) **Test orchestration (unit/property/invariant)**

- **forge.test** — run Foundry unit & fuzz tests, return JUnit/JSON. (Foundry v1.0 emphasizes improved invariant testing.) (Paradigm)
- **forge.invariant** — run invariant suites with sequences of random calls. (Invariant testing definition & behavior.) (getfoundry.sh)
- **echidna.run** — property-based fuzzing against user predicates. (Echidna is the ToB fuzzer for EVM.) (GitHub)

Resources: `/reports/test/latest`, `/coverage/summary`, `/invariants/findings`.

3) **Transaction simulation & debugging**

- **tenderly.simulateTx** — remote simulation with mainnet data, “virtual testnets”, re-simulate/trace. (Simulation API, forks, gas profiler, debugger.) (GitHub)
- **trace.get** — get call traces, storage diffs, revert reasons (via Tenderly or local node debugging). (GitHub)
- **gas.profile** — function-level gas breakdowns for a tx/suite. (docs.tenderly.co)

Resources: `/tx/:hash/simulation`, `/tx/:hash/trace`, `/tx/:hash/gasProfile`.

4) **Static & symbolic analysis**

- **mythril.analyze** — run Mythril on bytecode/solc-output; return detected vulns (reentrancy, tx-ordering, etc.). (Mythril: symbolic execution/SMT/taint.) (GitHub)
- **ruleset.apply** — optional Semgrep-style static checks on Solidity/Foundry tests (if you add a rule engine).

Resources: `/analysis/mythril/:runId`, `/findings/vulns`.

5) **Coverage, quality gates & metrics**

- **coverage.collect** — pull statement/branch/function coverage from Foundry run (or integrate a coverage reporter) and compute gates (e.g., ≥90%).
- **quality.gate** — enforce thresholds (coverage, 0 criticals from Mythril/Echidna invariants).

Resources: `/quality/summary`, `/metrics/test`, `/metrics/security`.

6) **Artifact & schema registry**

- **build.compile** — produce ABIs, bytecode, metadata; publish to resources.
- **artifact.get** — retrieve ABI/bin for contract X@version.
- **schema.list** — list tool input/output JSON Schemas for agent validation (MCP tools include schema’d IO). (MCP Protocol)

Resources: `/artifacts/:contract/abi`, `/artifacts/:contract/bytecode`.

7) **Network catalog & chain data**

- **net.list** — enumerate supported chains, RPCs, faucets; indicate rate limits.
- **net.block/tx** — fetch canonical block/tx receipts for asserting determinism across runs.

Resources: `/chains`, `/rpc/:chainId`, `/blocks/:n`.

8) **Prompts (test recipes & runbooks)**

- **prompts.list** — “ERC-20 test battery”, “DEX swap invariants”, “governance proposal flow”, “oracle drift checks”.
- **prompts.render** — return a tailored plan (steps → tools to call + parameters).

Resources: `/prompts/:name/template`.

9) **Security & governance**

- API keys/roles controlling destructive tools (e.g., state mutation vs. read-only).
- Audit logs of tool invocations; RBAC per method. (Security concerns & governance are highlighted in MCP integrations.) (IETF)

C) Example tool catalog (what an agent would "see")

Tool	Purpose	Inputs (examples)	Output
`anvil.start`	Start local forked chain	`{forkUrl, blockNumber?, chainId?}`	`{rpcUrl, forkId}`
`forge.test`	Run unit/fuzz tests	`{pattern?, fuzzRuns?, reporters?}`	`{passed, failed, coverage?}`
`forge.invariant`	Run invariants	`{suite, maxRuns?, depth?}`	`{violations: [...], seeds}`
`echidna.run`	Property fuzzing	`{config, targetContracts[]}`	`{counterexamples[], stats}`
`tenderly.simulateTx`	Mainnet-data sim	`{chainId, from, to, data, value?, stateOverrides?}`	`{status, trace, gasProfile}`
`mythril.analyze`	Symbolic analysis	`{bytecode | sources, solcVersion?}`	`{findings[], severityCounts}`
`coverage.collect`	Gather coverage	`{runId}`	`{statements, branches, functions}`
`quality.gate`	Enforce gates	`{coverage>=, zeroCriticals}`	`{pass, details}`

(Foundry invariants & cheatcodes; Echidna fuzzing; Mythril symbolic; Tenderly simulation/profiler.) (getfoundry.sh)

D) Minimal resource set (read-only "files")

- `/artifacts/*` (ABI/bin/metadata)
- `/reports/test/*` (junit.json, failures)
- `/analysis/*` (Mythril/Echidna outputs)
- `/tx/*` (trace, gas profile)
- `/forks/*` (snapshot info)

E) Ops features you'll want

- **Streaming & progress** for long-running fuzz/symbolic jobs (MCP supports notifications/streaming patterns depending on transport). (Medium)
- **Caching** of compiled artifacts and mainnet snapshots for speed.
- **Versioned schemas** so agents can adapt to tool updates.
- **Audit telemetry**: duration, flakiness, seed/repro strings, block anchors.

F) Example workflows your agents can automate

1. **AMM invariant battery**
    - `anvil.start(fork mainnet)` → `forge.invariant(amm_suite)` → if violation: fetch `/tx/*/trace` + `gas.profile`. (Invariant background.) (getfoundry.sh)
2. **Pre-deploy safety gate**
    - `build.compile` → `mythril.analyze` (no criticals) → `echidna.run` (no counterexamples) → `forge.test` (≥90% coverage) → `quality.gate`. (GitHub)
3. **What-if governance simulation**
    - `tenderly.simulateTx` for proposal execution bundle, review call trace/gas profile & storage diffs. (GitHub)

G) Implementation notes (Rust-friendly)

- Wrap CLIs/SDKs: Foundry (forge/cast/anvil), Echidna, Mythril; Tenderly Simulation API. (Docs detail forks/simulation/gas profiler.) (GitHub)
- Expose each as MCP tools with JSON-schema'd inputs/outputs so your LLM agents can confidently call them. (MCP tool spec.) (MCP Protocol)
- Start with stdio/WebSocket transport and add HTTP if you want remote agents. (MCP messaging is JSON-RPC 2.0.) (Medium)

If you want, I can generate a **ready-to-drop tools.json (schemas)** plus a **Rust Axum MCP server skeleton** that wraps `forge`, `anvil`, `mythril`, `echidna`, and the Tenderly API—so your agents can start running invariant/fuzz/sim pipelines with one endpoint.