Shiraga

vibe

Technical Overview and Roadmap

Table of contents

Summary

vibe is an all-in-one, Foundry-powered development platform that redefines how decentralized applications are built, tested, deployed and verified. By combining instant, consistent and stable forks, seamless and consistent contract deployment and verification, reproducible snapshots and replays all through user-friendly yet powerful configuration, vibe removes the friction that has held back developers for years.

It is designed to be instantly approachable for JavaScript and TypeScript developers while powerful enough for professional auditors and enterprise teams. With a modular plugin system for unmatched extensibility and a roadmap toward hosted infrastructure, courses, certifications and enterprise licensing, vibe is positioned not only as the most advanced development toolkit but also as the default entrypoint to decentralized application development.

In short: vibe delivers speed, reproducibility, and accessibility at every stage of the dApp lifecycle, paving the way for safer, more scalable, and more widely adopted decentralized applications.

The problem

Despite years of evolution, the dApp development workflow remains fragmented, inefficient, and unnecessarily complex. Developers are forced to piece together multiple tools and frameworks just to cover the basics: local development, forking and testing against live networks, auditing, deployment, and cross-chain interoperability.

This fragmentation creates several major problems:

- **Context switching & setup overhead**
  Developers spend significant time configuring environments, stitching together tools, and resolving incompatibilities instead of building.

- **Lack of standardization**
  Every team ends up reinventing the wheel with custom scripts and processes, which are brittle and hard to scale.

- **Barriers to entry**
  New developers face a steep learning curve and often abandon projects due to setup friction.

- **Cross-chain complexity**
  With the rise of multi- and omni-chain apps, the pain multiplies: each chain requires separate deployments, monitoring, and bridging integrations.

- **Auditing bottlenecks**
  Security analysis is scattered across different tools and formats, making audits slow, costly, and error-prone.

The result is a developer experience that feels stuck in the past with high friction, wasted effort and limited accessibility, while the ecosystem requires greater efficiency, stability, and safety to unlock its next wave of growth.

The solution

vibe removes the friction and fragmentation of today's dApp development workflow by unifying everything developers need into a single, coherent toolchain and platform. From local development to auditing, from cross-chain testing to deployment, vibe streamlines the process into one seamless experience, built for speed, safety, and scalability.

These are the pain points directly and uniquely addressed by vibe:

- **Unified workflow and zero setup overhead**
  Instead of juggling tools and configs, developers spin up entire environments in minutes using a single configuration file. Forks, snapshots, and transaction replays are orchestrated out-of-the-box.

- **Standardization by design**
  Vibe establishes consistent workflows across teams and projects, replacing brittle custom scripts with reliable, well-typed APIs and plugin-based extensibility.

- **Lowering barriers to entry**
  With TypeScript-first design, strong typing, and intuitive commands, new developers can get started quickly, reducing setup friction and flattening the learning curve.

- **First-class cross-/omni-chain support**
  Vibe abstracts the complexity of multi-chain deployments and messaging. Developers choose whether data syncs to one chain or many, with vibe managing orchestration and verification behind the scenes.

- **Built-in auditing and analysis**
  Security tools are integrated directly into the workflow, providing consistent formats, automated checks, and reproducible environments for faster, safer audits.

The result is a next-generation developer experience: fast, consistent, and accessible to veterans and newcomers alike, while enabling the ecosystem to scale securely towards an omni-chain future.

Features

vibe-cli

Setup

- Install with `npm install -g vibe-cli` or `pnpm / yarn / bun add -g vibe-cli`.
- Zero assumptions. Will detect if Foundry is installed upon use and will ask the user if vibe should install Foundry for them.
- Supports both Windows and Linux natively (Powershell included).

Initialization

- Run `vibe init [name]` to initialize vibe in a Git project with starting template and contracts, Foundry-specific folders, a `.vibe` file, a vibe folder for data on forks, snapshots etc. and `vibe.config.ts` (or .js if tsconfig is not detected).
- Ensures that `forge-std` is added as a module and runs `forge install`.

Configuration

- A single, robust and fully typed configuration file per project:
  - **privateKey:** the private key used to deploy contracts and run scripts.
  - **paths:** `{ src, out, scripts, deployed, vibe }`
  - **chains:** a map of chain keys to `ExtendedChain` (viem compatible) allowing you to specify scripts to run after deployment.
  - **forks:** an object with options specific to forking per chain (funding amounts per token and what scripts to run when the fork starts, or override the scripts that run after deployment). Referenced with `fork:chain`.
  - **wallets:** an object with wallet addresses per (forked) chain - for funding (on forks) and saving snapshots.
  - **tokens:** an object with (viem) tokens (or their addresses) per chain, categorized by type (eg. ERC20).
  - **contracts:** an object consisting of contract declarations with typed constructor arguments which can be referenced by name/key in **compile** and **deploy** for convenience.
  - **scripts:** an object consisting of script declarations with arguments that can be referenced by name/key in **onFork** and **onDeploy** for convenience.
  - **tests:** an object containing tests to run per chain.
  - **compile:** an array with contract declarations and/or references to **contracts**.
  - **deploy:** an object with chain names as keys and values of arrays, each holding contract declarations and/or references to **contracts** by name/key.
  - **fork:** `{ privateKey, deploy }` overrides for forking.
  - **optimizer:** `{ enabled, runs }` optimizer settings for compilation.
  - **via_ir:** whether or not to use intermediate representation for compilation.
  - **verbosity:** the verbosity level of the log messages from tests.
  - **plugins:** an array of imported vibe plugins.

Plugins will be able to extend this configuration with their own options, including types.

.vibe

- A file containing chain data and deployment information along with ABIs on each chain and fork, to be used internally and by other parts of the application/project.

Compilation

- With `vibe compile` all contracts specified under **compile** in the config file will be compiled using `forge build`, honoring the configured compilation options.
- Passes any command arguments and flags to `forge build`.

Type generation

- vibe is able to generate TypeScript types based on a project's smart-contracts and scripts with `vibe tsc`.
- This enables typed constructor arguments for smart-contracts, and arguments for running scripts, inside of the config file.
- The command runs automatically after running `vibe compile`.

Deployment

- Deploy your project's smart-contracts with `vibe deploy <chain>`.
- Deployment is sequential. If a constructor argument calls for the deployment address of a previously deployed smart-contract, you can reference it by name and the deployment address will automatically be resolved.
- Emits artifacts and metadata for verification and replays.
- Adds deployment information to the `.vibe` file.

Runnings scripts

- Use `vibe run <chain> <script> [args]` to run a Solidity script defined in the config file in your project.
- Arguments in scripts can be accessed with e.g. `vm.envUint("ARG0");`
- You can get a deployment address with `vm.envAddress("Contract Name");`
- To get the configured private key in script use `vm.envUint("PRIVATE_KEY");`

Testing

- Using `vibe test <chain>` or `vibe check <chain>` will run the tests specified under that chain in the config file.

Running RPC methods

- `vibe curl <chain> <method> [args]` will run the RPC method with the provided arguments on the specified chain.

Verification

- Instant and consistent Etherscan (or Etherscan equivalent) verification of all deployed smart-contracts on a chain with `vibe verify <chain>`.

Forking

- Fork a chain with Anvil according to the config using `vibe fork <chain> [-d]`.
- If the `--deploy (-d)` flag is used, configured smart-contracts are deployed on this forked chain once it starts.

Funding wallets on a fork

- With `vibe fund <chain>` all wallets specified in this chain's fork options inside the config will be funded by altering the token's storage directly.
- Use `vibe fund <chain> <address>` to directly fund a specific address.

Saving and restoring snapshots

- Save a snapshot of the state of wallets and smart-contracts for the current block on any chain by using the `vibe save <chain>` command, saved by default in the `vibe/snapshots` folder.
- The token balance of each token specified in the config file is saved per wallet, along with their gas (ETH) balance.
- Running `vibe restore <chain> <block>` on a forked chain restores the state of the configured wallets and smart-contracts on that chain to that of the snapshot.

Recording and replaying transactions

- Record transactions happening on any chain with `vibe watch <chain>`.
- Transactions for each smart-contract specified in the config on that chain are saved with headers and hashes by default by block in the `vibe/transactions` folder.
- Replay the transactions during a block with `vibe replay <chain> <block>`.
- You can replay the transactions related to a specific smart-contract on a chain with the `vibe replay <chain> <block> <contract>` command.
- Add the `--assert (-a)` flag to assert if the resulting hash is identical to the recorded hash, enabling consistent reproducibility.

Plugin support

Extend vibe's capabilities further with plugins, which can hook into typed lifetime functions, supplied with a context object unique to each action:

- **onCompile:** called after smart-contracts have been compiled.
- **onBeforeCompile:** called right before smart-contracts are compiled.
- **onDeploy:** called after smart-contracts have been deployed.
- **onBeforeDeploy:** called right before smart-contracts are deployed.
- **onVerify:** called after smart-contracts have been verified.
- **onBeforeVerify:** called before smart-contracts are verified.
- **onFork:** called when a fork has been started.
- **onBeforeFork:** called right before a chain is forked.
- **onTxRecorded:** called when a transaction is recorded while watching a chain.
- **onReplay:** called after a transaction is replayed.
- **onBeforeReplay:** called right before a transaction is replayed.
- **onSaveSnapshot:** called right before a snapshot is saved to disk.
- **onLoadSnapshot:** called right after a snapshot is loaded, but before it is applied.

vibe-core

- Contains helper functions for reading and writing to the `.vibe` file.
- Supports reading from either a `.vibe` file in the project's root, or from the root of another GitHub repository by passing a repo url (and optionally a GitHub token if the repository is private), this allows support for both monorepos, and individual repos for segmented projects.
- Streamlines interaction with deployed smart-contracts in both the front-end and back-end of the application by providing dynamically typed helper functions.

Plugins on release

The launch of vibe is planned to include the following plugins:

- **vibe-plugin-gas-reporter**
  Integrates gas usage reporting directly into the workflow, giving developers instant insights into contract efficiency.

- **vibe-plugin-slither**
  Runs static analysis with Slither on every compile/deploy, surfacing vulnerabilities early in the development cycle.

- **vibe-plugin-mythril**
  Automated symbolic execution analysis via Mythril for deeper vulnerability detection.

- **vibe-plugin-tenderly**
  Seamlessly connects projects to Tenderly for real-time debugging, simulation, and monitoring.

Example projects

A trio of example projects will be made to showcase and further expound on vibe:

- **vibe-example-minimal**
  A barebones starter template showcasing the simplest possible vibe setup.

- **vibe-example-contracts**
  A multi-contract project with dependencies, deployment, and verification flows.

- **vibe-example-svelte**
  A SvelteKit dApp project, emphasizing type generation and contract interaction.

Landing page

This page will showcase as well as explain vibe in order to onboard new developers. The landing page will be hosted at https://getvibe.sh (the domain has been acquired).

Monetization

For vibe I've thought out three potential monetization strategies targeting different levels of individuals and teams to maximize adoption while creating strong recurring and enterprise-grade revenue streams as teams scale. This ensures accessibility for newcomers, sustainability for professionals, and long-term growth potential for the ecosystem as a whole.

Education and certification

In order to introduce new developers into the ecosystem and enable experienced devs and auditors to gain credentials to showcase professional competency, I plan on creating a trio of courses to help individuals familiarize themselves with, and master vibe:

- One of which will be a free introductory course targeted towards everyone which broadly explains what vibe does differently and how to make use of it in order to speed up and simplify the dApp workflow.
- A premium developer course/certification program will go into more detail on how to get the most out of vibe as a dApp developer.
- Another premium course/certification program, targeted towards auditors, will cover the entire auditing process using vibe with practical examples.

vibe cloud / hosted infrastructure

Fully managed cloud environments, enabling developers to bypass complex local setup and instantly spin up reproducible environments in the cloud.

Could support advanced features like a collaborative dashboard and would be able to support tiered subscriptions as a revenue model.

Enterprise licensing

Private, hybrid, or custom deployments of vibe for large organizations with regulatory, compliance, or security needs, offering custom integrations/plugins, audit/compliance reporting and premium enterprise support with SLA.

This opens up another revenue stream in the form of annual licensing contracts.

Together, these strategies form a cohesive growth engine: education brings new developers into the ecosystem, cloud infrastructure scales their workflows as they progress and enterprise licensing anchors long-term adoption at the institutional level. This layered approach not only maximizes accessibility and sustainability but also ensures vibe grows with its users all the way from first-time developers to professional teams and enterprises.

Roadmap

Completed

- A previous, experimental version of **vibe-cli** has been publicly released on my personal GitHub account: https://github.com/GreenWojak/vibe-cli.
- The to-be-released **vibe-cli** and **vibe-core** are in active development, being partially feature-complete.
- Every feature and component of vibe has been mapped out in detail.
- The **vibe.config** file and **.vibe** file are fully designed and partially implemented.
- Initialization and compilation have been fully implemented.

In active development

- Type generation for smart-contracts and scripts in config.
- Forking and fork orchestration.
- Deploying smart-contracts.
- Funding wallets on forks.
- Running scripts on specific chains.
- Running tests.
- **vibe-core**.

Pending

- Saving and loading snapshots.
- Recording and replaying transactions.
- Verification.
- The plugins and example projects.
- The landing page and its design.

At this stage, vibe is already partially usable, with a clear roadmap to production-ready release. Support at this point accelerates the final stretch of development and enables faster adoption across the ecosystem.

Timeline

Phase 1

vibe-cli and vibe-core (3 - 4 weeks)

Fully functional, production ready vibe-cli and vibe-core packages.

Plugins (2 - 3 weeks)

Build the initial suite of plugins (gas-reporter, Slither, Mythril, Tenderly).

Example projects (1 - 2 weeks)

Ready the three example projects (minimal, multi-contract, SvelteKit dApp) for release.

Phase 2

Courses / certification (2 - 3 weeks)

Structured courses: intro (free), premium developer and auditor certification programs.

Landing page (1 - 2 weeks)

Design and launch a polished landing page explaining vibe's value proposition, features, and getting-started flow.

Auditing efforts / promotion (4 - 8 weeks)

Perform real audits and shadow audits using vibe, publishing results to highlight workflow speed and reproducibility. Run promotional campaigns (content, social media, community engagement) to build credibility and attract early adopters.

Phase 3

Cloud infrastructure (3 - 4 months)

Develop managed cloud environments for one-click reproducible dev/test setups. This opens a SaaS revenue stream, provides collaborative dashboards, and removes local setup friction for teams.

Enterprise licensing (3 - 4 months)

Offer private/hybrid deployments with custom plugins, audit/compliance features, and enterprise support contracts. Positions vibe as the enterprise standard for large organizations with regulatory or security requirements.

The future (?)

Focus on omni-chain development and advanced messaging capabilities, positioning vibe as the backbone for next-generation cross-chain dApps.

Use cases and examples

Developers will be able to count on vibe at every stage of their journey, from writing their first smart contract to powering enterprise-grade workflows. These examples highlight the versatility and strength of vibe in real-world scenarios.

1. First smart-contract in minutes (beginner)
A new developer installs vibe, runs `vibe init hello-world`, edits one file, and deploys their first Solidity contract to Base with a single command. The contract is instantly verified and reproducible, turning what once took hours of setup into a simple, approachable learning experience.

2. Learning by forking (beginner to intermediate)
Students and hackathon participants can fork Ethereum mainnet with one command, replay real Uniswap trades, and inspect state changes safely. This transforms intimidating production dApps into interactive, hands-on learning opportunities, accelerating education and lowering barriers to entry.

3. Reliable cross-chain deployments (intermediate)
A small hackathon team wants to launch the same NFT contract on both Base and Polygon. With vibe, they configure once and deploy everywhere in one step, complete with verification and metadata. What normally takes days of manual work is reduced to minutes, with far fewer opportunities for error.

4. Expert-level auditing (professional)
An auditor uses vibe's snapshot and replay features to create reproducible attack scenarios. Static analysis plugins like Slither and Mythril run automatically on every compile or deploy, surfacing vulnerabilities early. This eliminates hours of environment setup and makes vulnerabilities easier to reproduce, document, and share with clients.

5. Enterprise-Grade Cloud Environments (advanced)
A startup or large organization spins up managed, reproducible dev/test environments in the cloud using vibe. Teams gain collaborative dashboards, versioned snapshots, and compliance-ready reporting. This not only reduces local setup friction, but also positions vibe as an enterprise standard for scalable, secure development workflows.

Together, these use cases show vibe's full spectrum of value:

- Onboarding new developers.
- Educating through live forking and replay.
- Empowering builders with cross-chain workflows.
- Enabling professionals with audit-ready tooling.
- Scaling enterprises with managed infrastructure.

Ecosystem impact

vibe aims to be the ultimate and go-to dApp development toolchain and platform. By unifying fragmented workflows into one reproducible system, vibe unlocks efficiency, security, and accessibility across the entire development and auditing pipeline.

Lowering the barrier to entry

Developers can go from zero to a verified deployment in minutes. What once required deep Solidity knowledge and hours of setup is now accessible to students, hackathon participants, and new builders, dramatically widening the funnel of talent entering the ecosystem.

Raising the baseline for security

With audit tooling, reproducible snapshots, and transaction replays built in, security checks become a default part of the workflow. Every developer, not just experts, gains access to professional-grade practices, raising the standard of safety ecosystem-wide.

Accelerating innovation cycles

Fork orchestration and instant deployment pipelines compress iteration timelines from weeks to hours. This speed of testing and deployment enables more experiments, more ideas validated, and ultimately more successful applications launched.

Enabling the omni-chain future

vibe abstracts away the complexity of cross-chain development, letting teams deploy and sync across chains with minimal overhead. This capability is not just convenient, it is a prerequisite for the next generation of omni-chain applications.

Strengthening the professional pipeline

Through certification and reproducible standards, vibe creates a training and verification pathway for developers and auditors. This builds a reliable talent pool in decentralized development, equipped to face the challenges and complexities of the next decade.

In short: vibe is the foundation for faster, safer, and more accessible decentralized development, and adoption will accelerate the entire ecosystem’s next wave of growth.