

Infernet Protocol - ARCHITECTURE.md

Overview

This document outlines the planned technical architecture and implementation details for the Infernet Protocol, supporting CLI servers, desktop applications, and native mobile apps.

Supported Platforms

- **CLI / Server:** Node.js using vanilla JavaScript.
 - **Desktop:** Electron-based application using Svelte for UI.
 - **Mobile (Native):** React Native with Expo.dev for cross-platform mobile apps.
-

Core Technologies

- **Communication:** WebSockets for real-time, bidirectional communication.
 - **Networking:** Decentralized discovery and peer-to-peer job distribution via DHT (Kademlia implementation).
 - **Containerization:** Docker-based task execution for sandboxing and security.
 - **Data Transport:** Large payloads (models, input data) distributed via IPFS.
 - **Styling:** Vanilla CSS across desktop and web interfaces for simplicity and consistency.
-

CLI (Server Nodes)

- Built on Node.js with vanilla JavaScript.
- Headless daemon process running:
 - Persistent DHT connections for discovery.
 - WebSocket server for receiving inference tasks.
 - Task execution engine using Docker.
 - Result verification and submission.
 - Configuration file-based setup (JSON or YAML).

Desktop (Electron + Svelte)

- Electron wrapper for cross-platform desktop app (Windows, Mac, Linux).
- UI built with Svelte:
 - Node dashboard for monitoring jobs, GPU usage, reputation.
 - Aggregator interface for splitting and tracking jobs.
 - Client submission form.

- Embedded WebSocket client to communicate with the network.
- IPFS client integration for uploading large models and datasets.
- Local storage for logs and history.

Mobile (React Native with Expo)

- React Native app (expo.dev-based) for both iOS and Android.
 - Features:
 - Real-time job tracking for clients.
 - Push notifications for job completion and status changes.
 - Ability to submit small inference jobs (low-volume interfaces).
 - Display provider reputation and availability.
 - Communication via secure WebSocket clients.
-

Distributed Hash Table (DHT) Implementation

- **Protocol:** Kademlia-based DHT.
 - Peer discovery:
 - GPU providers register nodes with hardware specs.
 - Aggregators search for best-fit nodes by reputation and load.
 - Dynamic updates for provider uptime and status.
 - Gossip-based protocol for reputation dissemination.
-

Task Workflow Summary

1. **Discovery:** Aggregator queries DHT for available providers.
 2. **Job Dispatch:** Aggregator distributes inference job shards via WebSocket.
 3. **Execution:** Provider executes job in a Docker container.
 4. **Result Submission:** Provider sends result hashes and data via WebSocket.
 5. **Verification:** Aggregator performs redundant checks or validator sampling.
 6. **Payment Settlement:** Upon verification, payments released via micro-payment systems.
-

Security

- Container isolation (Docker).
- End-to-end encryption for job payloads.
- Validator sampling for result trust.
- Potential use of secure enclaves (SGX/SEV) in future iterations.

Logging and Monitoring

- CLI/Server: JSON-based structured logs.
 - Desktop: Visual logs with exportable summaries.
 - Mobile: Lightweight notifications and log snapshots.
-

Next Steps

- Define schemas for discovery messages and job specifications.
- Define Docker execution templates.
- Create WebSocket communication spec (message types and formats).
- Finalize DHT node join/leave protocols and health checks.

Contact

For technical contributions or questions: protocol@infern.net