Smart Contract Deployment & Integration

Session Summary — March 3, 2026
Author: Lam Lai. Produced with AI assistance

This report documents the session in which the simulated anchoring layer was replaced with a real smart contract deployed on Polygon Mainnet. The goal was to verify that the full flow — from a human approver clicking a button to a permanent on-chain record — was technically achievable with the project's existing infrastructure.

The test was conducted on Hostinger, the current available development hosting environment, because it was the fastest way to prove the integration without setting up a dedicated server. The Node.js backend, the PHP layer, and the admin interface were all already running there from earlier prototype work. This made it the practical choice for a proof-of-concept test — not the final production environment.

The report covers the architecture, the deployment steps, the problems encountered and how they were solved, and the three live Polygon Mainnet transactions that confirm the system works.

Overview

This session covered the full journey from deploying a Solidity smart contract on Polygon Mainnet to integrating it with an existing **`PHP/Node.js`** web application. The goal was to replace simulated blockchain anchoring with real on-chain transactions.

Final Architecture

The completed system works as follows:

- User clicks **Approve & Anchor** in p1-admin.html
- **`p1-api.php`** generates a **`SHA-256`** dataHash of the submitted entries
- PHP calls **`node.example.com/anchor`** via HTTP POST
- Node.js (Express + ethers.js) signs and sends a real transaction to Polygon
- Smart contract stores submissionId, dataHash, approver, and timestamp on-chain
- Real txHash and blockNumber are returned and saved to **`p1-db_anchors.json`**

Steps Completed

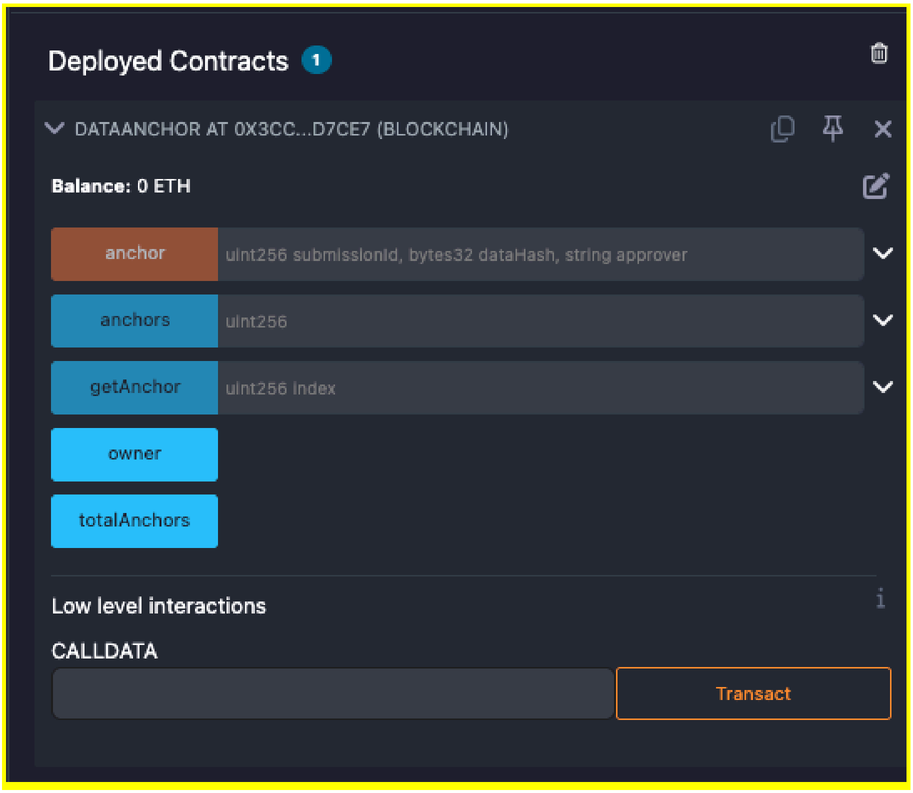
1. Smart Contract (**`DataAnchor.sol`**)

- Written in Solidity, compiled in Remix IDE
- Stores: submissionId, dataHash (bytes32), approver, anchoredAt
- Emits DataAnchored event with indexed fields for fast lookup
- onlyOwner modifier restricts who can anchor
- Deployed to Polygon Mainnet via Remix + MetaMask
- Contract address: **`0x3CCF479641622ea91927fc4266242cF2068D7Ce7`**

`DataAnchor.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract DataAnchor {
    struct AnchorRecord {
        bytes32 dataHash;
        uint256 anchoredAt;
        string  approver;
        uint256 submissionId;
    }
    mapping(uint256 => AnchorRecord) public anchors;
    uint256 public totalAnchors;
    address public owner;
    event DataAnchored(
        uint256 indexed submissionId,
        bytes32 indexed dataHash,
        uint256 anchoredAt,
        string  approver
    );
    modifier onlyOwner() {
        require(msg.sender == owner, "Not authorized");
        _;
    }
    constructor() {
        owner = msg.sender;
    }
    function anchor(
        uint256 submissionId,
        bytes32 dataHash,
        string calldata approver
    ) external onlyOwner {
        anchors[totalAnchors] = AnchorRecord({
            dataHash:     dataHash,
            anchoredAt:   block.timestamp,
            approver:     approver,
            submissionId: submissionId
        });
        totalAnchors++;
        emit DataAnchored(submissionId, dataHash, block.timestamp, approver);
    }
    function getAnchor(uint256 index) external view returns (AnchorRecord memory)
{
        return anchors[index];
    }
}
```

Transaction tested directly at Remix:
`https://polygonscan.com/tx/0xd3950f9943e137e03b55353e137162000efaba3b5e7ff8c623c7af3191c45f74`



2. **`Node.js`** Backend (node.example.com)

- Express app deployed on Hostinger via GitHub import
- Added ethers.js and dotenv packages via SSH
- New /anchor endpoint connects to Polygon using ethers.JsonRpcProvider
- Uses wallet private key from .env to sign transactions automatically
- .env file stored on server only — never pushed to GitHub

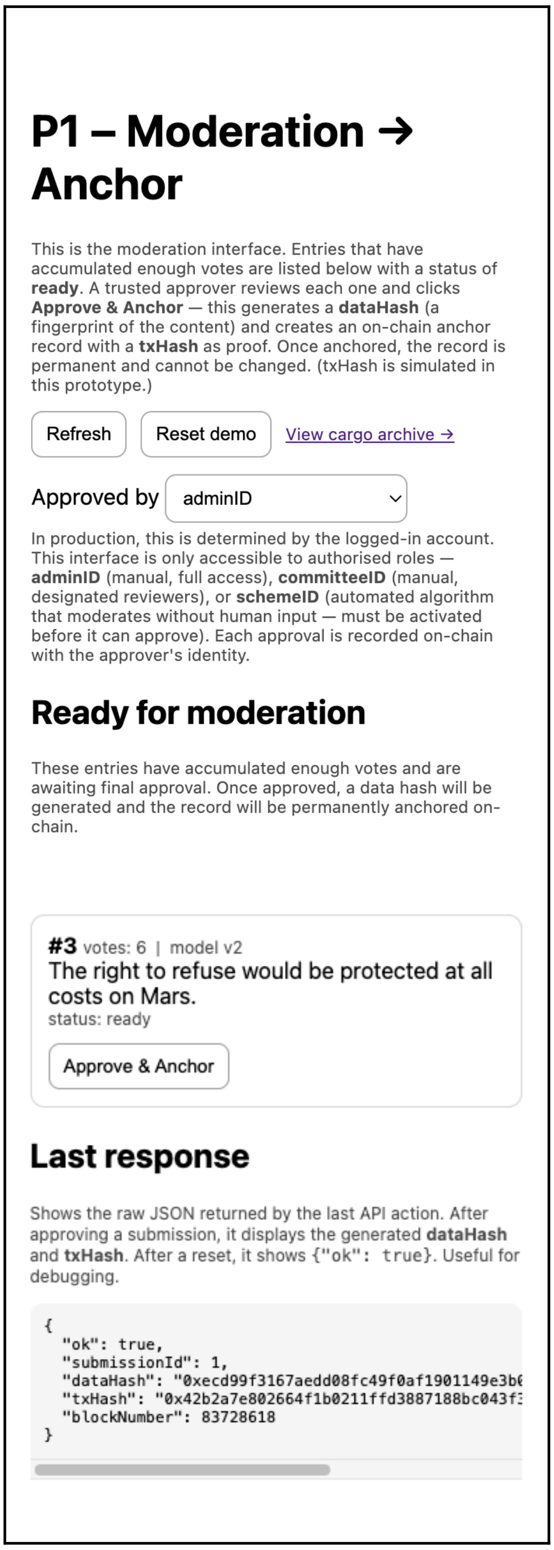
Transaction tested using Terminal:

https://polygonscan.com/tx/0x0c246a50c775b6f14ef6cc2edc7e04040df3d6cfb295a28907c3243a16a20c82#eventlog

3. PHP Integration (**`p1-api.php`**)

- Added call_node_anchor() function using PHP cURL
- Replaced all fake/simulated txHash generation with real blockchain calls
- Real txHash and blockNumber now stored in **`p1-db_anchors.json`**

Transaction tested through **`p1-admin.html`**: https://polygonscan.com/tx/0x42b2a7e802664f1b0211ffd3887188bc043f3d86a5ae060d06cba974fa9c3781



P1 – Moderation → Anchor

This is the moderation interface. Entries that have accumulated enough votes are listed below with a status of **ready**. A trusted approver reviews each one and clicks **Approve & Anchor** — this generates a **dataHash** (a fingerprint of the content) and creates an on-chain anchor record with a **txHash** as proof. Once anchored, the record is permanent and cannot be changed. (txHash is simulated in this prototype.)

Approved by adminID

In production, this is determined by the logged-in account. This interface is only accessible to authorised roles — **adminID** (manual, full access), **committeeID** (manual, designated reviewers), or **schemeID** (automated algorithm that moderates without human input — must be activated before it can approve). Each approval is recorded on-chain with the approver's identity.

Ready for moderation

These entries have accumulated enough votes and are awaiting final approval. Once approved, a data hash will be generated and the record will be permanently anchored on-chain.

#3 votes: 6 | model v2
The right to refuse would be protected at all costs on Mars.
status: ready

Last response

Shows the raw JSON returned by the last API action. After approving a submission, it displays the generated **dataHash** and **txHash**. After a reset, it shows `{"ok": true}`. Useful for debugging.

```
{
  "ok": true,
  "submissionId": 1,
  "dataHash": "0xecd99f3167aedd08fc49f0af1901149e3b
  "txHash": "0x42b2a7e802664f1b0211ffd3887188bc043f
  "blockNumber": 83728618
}
```

Problems Encountered

Problem	**Solution**
Testnet faucets rejected wallet (insufficient mainnet activity)	Skipped testnet, deployed directly to Polygon Mainnet (cheap)
MetaMask not showing in Remix environment dropdown	Installed MetaMask browser extension, reopened Remix
SSH login Permission Denied	Reset SSH password in hPanel, retried
npm not found via SSH	Used full path: **`/opt/alt/alt-nodejs20/root/usr/bin/npm`**
Public RPC endpoints (polygon-rpc.com, ankr) required API keys	Created free Alchemy account, used Alchemy RPC URL
node.example.com calling old app without .env	Found Hostinger's managed process (lsnode) was separate from our manual one. Created .env in correct folder, restarted manually.
No Node.js section in hPanel to manage env vars or restart	Used SSH to manage .env and process directly. Manual restart needed for now.

Key Files & Resources

- Smart contract: **`DataAnchor.sol`** — deployed on Polygon Mainnet
- Contract address: **`0x3CCF479641622ea91927fc4266242cF2068D7Ce7`**
- Node.js app: **`~/domains/example.com/nodejs/node/app.js`**
- .env file: **`~/domains/example.com/nodejs/node/.env`** (PRIVATE_KEY, CONTRACT_ADDRESS, RPC_URL)
- GitHub repo: express-test (app.js, package.json, .env.example)
- PHP file: **`p1-api.php`** — updated to call **`node.example.com/anchor`**

Pending

- Find proper way to manage Node.js process on Hostinger (auto-restart on reboot)
- Find environment variables section in hPanel to avoid SSH dependency
- Keep GitHub repo in sync with server changes
- Consider adding transaction error handling and retry logic in PHP