Decentralized Cloud Infrastructure Architecture on Hedera Hashgraph: A Technical and Economic Feasibility Study

1. Introduction: The Convergence of Hashgraph Consensus and Decentralized Physical Infrastructure Networks (DePIN)

The contemporary digital landscape is undergoing a paradigmatic shift from centralized cloud architectures—dominated by hyperscalers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure—toward Decentralized Physical Infrastructure Networks (DePIN). This transition is driven by the imperative for censorship resistance, verifiable provenance, and the democratization of computational resources. In this evolving ecosystem, Hedera Hashgraph has emerged not merely as a distributed ledger but as a high-throughput consensus layer capable of orchestrating complex service delivery networks. However, realizing a "Google Cloud-like" experience on Hedera requires a nuanced understanding of its architectural distinctions from traditional blockchains, specifically its directed acyclic graph (DAG) structure and the Gossip about Gossip protocol, which facilitate fair ordering and low-latency finality without the energy-intensive proof-of-work mechanisms.
This report provides an exhaustive technical blueprint for constructing a cloud infrastructure tool capable of deploying websites and decentralized applications (dApps) utilizing the Hedera network as the control plane. A central constraint of this architecture is a specific revenue model: a fixed 1 HBAR per month subscription with an automated revenue split. This economic requirement necessitates a rigorous analysis of Hedera’s fee schedules, smart contract gas costs, and transaction throughput limits to ensure commercial viability. Furthermore, the report integrates a comprehensive survey of the 2025 landscape of free decentralized cloud providers, offering a strategic pathway for bootstrapping infrastructure with minimal capital expenditure.
The proposed architecture does not attempt to store bulk data directly on the Hedera ledger, a practice that is economically inefficient and technically constrained by transaction size limits. Instead, it employs a hybrid model where Hedera functions as the immutable registry for identity, payments, and access control, while the heavy lifting of storage and compute is offloaded to specialized networks such as the InterPlanetary File System (IPFS), Arweave, Akash Network, and the Internet Computer (ICP). By leveraging Hedera’s Consensus Service (HCS) and Token Service (HTS), developers can build a robust "DeCloud Gateway" that offers the user experience of Web2 with the security guarantees of Web3.

2. Architectural Constraints and The Hedera Control Plane

To replicate the functionality of a centralized cloud provider, one must first deconstruct the monolithic cloud service model into its constituent parts: consensus, storage, and computation. Hedera excels at the first, providing a "Trust Layer" for the internet, but it is not designed to serve as a "Hard Drive Layer" or a "CPU Layer". Understanding these boundaries is critical for architectural success.

2.1 The Limitations of On-Chain Hosting via Hedera File Service (HFS)

A common misconception among developers transitioning to Hedera is the feasibility of hosting web assets directly on the ledger via the Hedera File Service (HFS). While HFS allows for the storage of data on all consensus nodes, it is engineered for small, high-value configuration files—such as address books or exchange rate tables—rather than static web content. The maximum size for a single Hedera transaction is capped at 6,144 bytes (6 KiB), which includes signatures and other transaction overhead. Although the HFS supports file appending to create larger logical files, the recommended chunk size is a mere 1,024 bytes (1 KiB).

From an economic perspective, hosting a standard modern website on HFS is prohibitive. The cost to append data to a file is approximately $0.05 USD per transaction. To host a modest 2 megabyte (MB) web application, a developer would need to execute approximately 2,000 append transactions. At $0.05 per transaction, the upload cost alone would reach $100 USD, a figure that renders the "1 HBAR per month" revenue model instantly insolvent given that 1 HBAR trades between $0.10 and $0.20 USD. Furthermore, this approach would bloat the ledger state, which contradicts the network's optimization for high-throughput transaction processing rather than bulk storage.

Consequently, the architecture must utilize Hedera strictly for the *logic* and *state management* of the cloud infrastructure. This involves managing the subscription status (Active/Inactive), processing the recurring 1 HBAR payments, executing the revenue split, and maintaining an immutable log of content versions using the Hedera Consensus Service (HCS). The actual website assets—HTML, CSS, JavaScript, and media—must reside on dedicated decentralized storage networks like IPFS or Arweave, which offer significantly lower costs per gigabyte and are designed for content addressing and retrieval.

2.2 The Hybrid Aggregator Model: The DeCloud Gateway

The proposed solution functions as an orchestration layer, or a "DeCloud Gateway." This application serves as the interface between the user (who expects a simple drag-and-drop deployment experience) and the complex underlying decentralized protocols.

The workflow for this gateway involves distinct phases:

1. **Ingestion:** The user provides their website build folder via a Command Line Interface (CLI) or web dashboard.
2. **Storage Orchestration:** The tool uploads these files to a decentralized storage network (e.g., IPFS via Pinata or Fleek) to ensure persistence and retrieval availability.
3. **Consensus & Payment:** The tool executes a transaction on Hedera to pay the 1 HBAR subscription fee. Upon success, it logs the IPFS Content Identifier (CID) to a Hedera Consensus Service (HCS) topic or a Smart Contract, effectively "stamping" the deployment with a verifiable timestamp and proof of payment.
4. **Resolution:** A lightweight web server or edge worker queries the Hedera ledger to verify the subscription status. If valid, it resolves the IPFS CID associated with that user and serves the content to the visitor.

This model mirrors the architecture used by the Guardian project, which utilizes HCS and IPFS to create immutable audit trails for environmental assets. By decoupling storage from consensus, the system leverages the strengths of each network: Hedera for speed and low fixed fees, and IPFS/Arweave for bulk data retention.

3. Deep Dive: The Decentralized Storage and Compute Ecosystem

Before detailing the implementation of the revenue model, it is essential to analyze the specific providers that will constitute the "Data Plane" of this infrastructure. The 2025 ecosystem offers several robust options, many of which provide free tiers that can be leveraged to subsidize the initial operating costs of the DeCloud Gateway.

3.1 Hedera-Native and Integrated Solutions

While Hedera is the consensus layer, the ecosystem includes projects building physical infrastructure on top of it. **Neuron**, for instance, leverages Hedera to manage connectivity and sensor data for Internet of Things (IoT) devices, utilizing the trust layer to manage decentralized service delivery. While Neuron focuses on physical sensors, its architecture—using Hedera to manage decentralized resources—validates the proposed model. Similarly, the **Guardian** ecosystem demonstrates the viability of pairing HCS with IPFS for creating provenance chains, confirming that HCS topics are the ideal mechanism for logging "off-chain" storage events.
3.2 The Storage Layer: IPFS, Arweave, and Crust
The storage layer is responsible for holding the static assets of the websites.
InterPlanetary File System (IPFS) remains the industry standard for content-addressed storage. However, IPFS alone does not guarantee persistence; files must be "pinned" by a node to prevent garbage collection. Services like **Pinata** and **Fleek** provide this pinning infrastructure. Pinata, for example, offers extensive API support for managing IPFS content and includes dedicated gateways to accelerate content delivery, a critical feature for a consumer-facing cloud product.
Arweave offers a fundamentally different value proposition: permanent storage. Unlike IPFS, where payment guarantees storage for a contract period, Arweave uses a "blockweave" data structure and an endowment model to pay for storage in perpetuity with a single upfront fee. This is ideal for immutable deployments where the user wants the site to exist "forever" without recurring storage fees, although it conflicts slightly with a monthly subscription model unless the subscription covers the "gateway" access rather than the storage itself.
Crust Network, built on the Polkadot ecosystem, provides a decentralized storage market that is fully compatible with IPFS. It incentivizes nodes to store data using a Proof of Meaningful Work (MPoW) mechanism. Crust allows for the decentralized pinning of IPFS content, offering a redundant alternative to centralized pinning services like Pinata. It currently offers free storage trials via its "Crust Files" application to encourage developer adoption.

3.3 The Compute Layer: Akash, Flux, and ICP

For dynamic applications requiring backend logic (e.g., databases, API servers), simple file storage is insufficient. The infrastructure must integrate decentralized compute providers.
Akash Network operates as a decentralized marketplace for cloud computing, utilizing a

reverse auction mechanism where providers bid to host Docker containers. This model results in costs that are roughly 80% lower than centralized hyperscalers. Akash is particularly suitable for hosting the backend APIs of the DeCloud Gateway. While it does not offer a permanent free tier, it frequently provides trial credits (e.g., $10 USD) to new developers, allowing for risk-free experimentation.

RunOnFlux (Flux) provides a similar decentralized compute network but emphasizes high availability and redundancy. Flux nodes are geographically distributed, and the network offers an AWS-like experience with Docker support. Pricing comparisons indicate that Flux can be significantly cheaper than Google Cloud or Azure; for instance, a configuration with 8GB RAM and 100GB storage might cost ~$2.29/month on Flux compared to over $170 on Google Cloud. Flux also utilizes a distinct pricing model that includes unlimited bandwidth, a major advantage over the egress fees charged by centralized providers.

The Internet Computer (ICP) represents a more integrated approach, where the entire application stack—frontend, backend, and data—is hosted on-chain in "canisters". ICP uses a "Reverse Gas" model where developers convert ICP tokens into "Cycles" to pay for computation, stabilizing the cost of hosting regardless of token volatility. The network offers "Cycles Faucets" providing $20–$100 worth of free cycles to verify developers, making it an attractive option for bootstrapping the compute component of the architecture.

4. Technical Implementation: The 1 HBAR Revenue Split Model

The core economic requirement of the user's query is a subscription model charging 1 HBAR per month, with the revenue automatically split (e.g., between the infrastructure provider and the application developer). Implementing this on Hedera requires a careful balancing of technical capability against transaction costs.

4.1 The Economic Constraints of the 1 HBAR Model

As of early 2025, the price of 1 HBAR fluctuates between $0.10 and $0.20 USD. This implies a gross monthly revenue of approximately $0.15 per user. This tight margin necessitates extreme efficiency in the transaction logic.

Executing a split payment via a **Smart Contract** on Hedera involves gas fees. While Hedera's smart contract service (HSCS) is EVM-compatible, it imposes specific costs. A basic Contract Call costs approximately $0.05, and there are surcharges for HBAR transfers initiated from within a smart contract. If the subscription payment logic consumes $0.05 to $0.10 in gas fees, the "1 HBAR" model becomes economically unviable, as fees would consume 30% to 60% of the revenue.

To mitigate this, the architecture should prioritize **Hedera Token Service (HTS)** native transfers or **Atomic Swaps** over heavy smart contract execution. Native HTS transfers have a fixed fee of $0.0001 USD, orders of magnitude cheaper than smart contract operations.

4.2 Implementation Strategy: Batching vs. Atomic Splits

There are two primary methods to implement the revenue split:

Method A: Atomic Transaction (The "Real-Time" Split) Hedera allows for atomic batch transactions where multiple transfers are executed simultaneously. If any part fails, the entire

transaction reverts.

- **Mechanism:** The user signs a TransferTransaction that sends 0.9 HBAR to the Provider Account and 0.1 HBAR to the Developer Account.
- **Pros:** Trustless, immediate settlement.
- **Cons:** Requires the user's wallet to sign a complex transaction with multiple outputs.
- **Code Structure:** Using the Hedera JavaScript SDK, a TransferTransaction can be constructed with multiple addHbarTransfer calls. The sum of all transfers must equal zero (inputs equal outputs).

Method B: Smart Contract with Logic (The "Trustless" Manager) A Solidity contract accepts the 1 HBAR, verifies the amount, and programmatically sends the split.

- **Pros:** Encapsulates logic (e.g., updating a subscription expiry date state variable).
- **Cons:** Higher gas fees.
- **Optimization:** To reduce costs, the contract should avoid complex storage operations. Using "Weibars" (10^-18 HBAR) for precision is necessary when interacting with EVM tools, as defined in HIP-410.

Method C: Native Transfer with Memo (The "Economical" Approach) The user sends 1 HBAR to a central treasury with a specific Memo (e.g., SUBSCRIBE: <DID>). A backend service monitors the Mirror Node for these transactions and updates the database. The revenue split is handled via a daily batch transfer (Scheduled Transaction) from the treasury.

- **Pros:** Lowest fee ($0.0001 per user).
- **Cons:** Introduces a centralized processing step (the backend listener), although the record on the ledger is immutable.

Given the user's request for a "Cloud Infrastructure" tool, **Method B (Smart Contract)** is the most robust solution as it allows on-chain verification of subscription status by third-party nodes, but **Method A (Atomic Transfer)** is the most economically efficient for the 1 HBAR price point.

4.3 Step-by-Step Build Guide

The following section details the construction of the DeCloud Gateway using the Hedera JavaScript SDK and a Solidity smart contract optimized for the 1 HBAR model.

Phase 1: Environment Configuration

The development environment requires Node.js and the Hedera SDK.

1. **Initialize the Project:**
   ```
   mkdir hedera-decloud-tool
   cd hedera-decloud-tool
   npm init -y
   npm install --save @hashgraph/sdk dotenv @pinata/sdk
   ```

2. **Configure Credentials:** Create a .env file containing the operator ID and private key, which can be obtained from the Hedera Portal for testnet access.
   ```
   OPERATOR_ID=0.0.xxxx
   OPERATOR_KEY=302e0201...
   NETWORK=testnet
   PINATA_API_KEY=...
   PINATA_SECRET_KEY=...
   ```

3. **Initialize the Client:** Create index.js to establish the connection to the Hedera network.

```
const { Client, AccountId, PrivateKey } =
require("@hashgraph/sdk");
require("dotenv").config();

if (!process.env.OPERATOR_ID ||!process.env.OPERATOR_KEY) {
    throw new Error("Must set OPERATOR_ID and OPERATOR_KEY
in.env");
}

const operatorId = AccountId.fromString(process.env.OPERATOR_ID);
const operatorKey =
PrivateKey.fromString(process.env.OPERATOR_KEY);
const client = Client.forTestnet().setOperator(operatorId,
operatorKey);
```

Phase 2: The Revenue Split Smart Contract

To enforce the 1 HBAR logic trustlessly, we deploy a Solidity contract. This contract acts as the "Subscription Manager."

Solidity Code (RevenueSplitter.sol):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract CloudSubscription {
    address public provider;
    address public treasury;
    uint256 public constant SUBSCRIPTION_COST = 100000000; // 1 HBAR
in tinybar
    uint256 public constant DURATION = 30 days;

    // Mapping from user address to subscription expiry timestamp
    mapping(address => uint256) public subscriptionExpiry;

    event SubscriptionRenewed(address indexed user, uint256
newExpiry);
    event RevenueSplit(uint256 providerShare, uint256 treasuryShare);

    constructor(address _provider, address _treasury) {
        provider = _provider;
        treasury = _treasury;
    }

    // Function to handle payment and split
    function paySubscription() external payable {
        // 1. Verify Payment Amount
        require(msg.value == SUBSCRIPTION_COST, "Must send exactly 1
```

```
HBAR");

        // 2. Calculate Split (90% Provider, 10% Treasury)
        uint256 providerShare = (msg.value * 90) / 100;
        uint256 treasuryShare = msg.value - providerShare;

        // 3. Execute Transfers
        // Note: Using call to save gas and avoid reentrancy limits of
transfer()
        (bool sentProvider, ) = provider.call{value:
providerShare}("");
        require(sentProvider, "Failed to send HBAR to provider");

        (bool sentTreasury, ) = treasury.call{value:
treasuryShare}("");
        require(sentTreasury, "Failed to send HBAR to treasury");

        // 4. Update Subscription State
        if (subscriptionExpiry[msg.sender] > block.timestamp) {
            subscriptionExpiry[msg.sender] += DURATION;
        } else {
            subscriptionExpiry[msg.sender] = block.timestamp +
DURATION;
        }

        emit RevenueSplit(providerShare, treasuryShare);
        emit SubscriptionRenewed(msg.sender,
subscriptionExpiry[msg.sender]);
    }

    // Read-only function to check status
    function isSubscribed(address user) external view returns (bool) {
        return subscriptionExpiry[user] > block.timestamp;
    }
}
```

Technical Note: The use of call instead of transfer is recommended to prevent issues with gas stipends, especially if the receiving accounts are contracts themselves. However, on Hedera, these calls incur system contract fees. To minimize these, ensure the receiving addresses are simple accounts (EOAs).

Deploying the Contract: Use the ContractCreateFlow from the SDK to deploy the bytecode.

```
const { ContractCreateFlow, ContractFunctionParameters } =
require("@hashgraph/sdk");
const fs = require("fs");

//... inside an async function
const bytecode =
fs.readFileSync("RevenueSplitter_sol_CloudSubscription.bin");
```

```
const createContractTx = new ContractCreateFlow()
   .setBytecode(bytecode)
   .setGas(1_000_000) // Estimate gas requirements
   .setConstructorParameters(
        new ContractFunctionParameters()
           .addAddress(providerAddress) // Solidity address format
           .addAddress(treasuryAddress)
    );

const txResponse = await createContractTx.execute(client);
const receipt = await txResponse.getReceipt(client);
console.log(`Contract Deployed at: ${receipt.contractId}`);
```

Phase 3: Content Orchestration (The "Hosting")

The tool must upload the user's website to IPFS and then link that content to the user's subscription on Hedera.

Step 3A: IPFS Upload via Pinata Using the pinata-sdk, we upload the build directory. This step ensures the content is available on the public IPFS network.

```
const pinataSDK = require('@pinata/sdk');
const pinata = pinataSDK(process.env.PINATA_API_KEY,
process.env.PINATA_SECRET_KEY);

async function uploadSite(sourcePath) {
    const options = {
        pinataMetadata: { name: 'DeCloud-Deployment' },
        pinataOptions: { cidVersion: 0 }
    };
    const result = await pinata.pinFromFS(sourcePath, options);
    return result.IpfsHash; // Returns the CID
}
```

Step 3B: Linking CID to Subscription (HCS Logging) Once the file is uploaded and the subscription paid, the tool must create a verifiable link between the user and their new site version. We use the Hedera Consensus Service (HCS) for this, as it provides a time-ordered, immutable log at a fraction of the cost of smart contract storage ($0.0001 vs potentially dollars for large state storage).

```
const { TopicMessageSubmitTransaction } = require("@hashgraph/sdk");

async function
logDe[span_18](start_span)[span_18](end_span)ployment(topicId, cid,
userAccountId) {
    // Create the message payload
    const message = JSON.stringify({
        user: userAccountId,
        cid: cid,
```

```
        timestamp: Date.now(),
        action: "DEPLOY"
    });

    // Submit to HCS
    const transaction = await new TopicMessageSubmitTransaction()
       .setTopicId(topicId)
       .setMessage(message)
       .execute(client);

    const receipt = await transaction.getReceipt(client);
    console.log(`Deployment logged. Status: ${receipt.status}`);
}
```

This logging action creates a public audit trail. Any observer can replay the topic messages to find the current IPFS CID for a specific user.

Phase 4: The Gateway Resolution Logic

The final piece is the gateway that serves the site. This is a web server (e.g., Nginx with a Node.js middleware) that intercepts requests.

1. **Incoming Request:** A user visits alice.decloud.com.
2. **Lookup:** The server queries a Hedera Mirror Node to find the latest HCS message from Alice's account ID on the designated Topic.
3. **Verification:** The server checks the RevenueSplitter smart contract: isSubscribed(AliceAddress).
4. **Service:**
    - If isSubscribed returns true: The server fetches the content from IPFS using the CID found in the HCS log and streams it to the browser.
    - If false: The server redirects the visitor to a payment page ("Subscription Expired").

This architecture ensures that only active subscribers consume the gateway's bandwidth, preserving the economic viability of the service.

5. Operationalizing Free Tiers: Bootstrapping the Infrastructure

To minimize the operational costs of the "DeCloud Gateway" (specifically the IPFS pinning and potential backend compute), the architecture can be bootstrapped using the free tiers of existing decentralized providers. This strategy allows the 1 HBAR revenue to be almost pure profit (minus Hedera network fees) during the early growth phase.

5.1 Comparative Analysis of Free Tier Providers (2025)

A rigorous analysis of the market reveals several providers offering substantial free resources that can be integrated into the backend of the DeCloud Gateway.

Provider	Storage Technology	Free Tier Limits (2025)	Bandwidth Limits	Compute/Build Limits	Suitability for Backend
Fleek	IPFS / Arweave	~3GB Storage	~50GB/month	250 Build Minutes	**High.** Excellent CLI & API for automated deployments.
4EVERLAND	IPFS / Arweave	6GB Storage	100GB/month	250 Build Minutes	**High.** S3-compatible API makes integration trivial.
Spheron	IPFS / Filecoin	Trial Period	Varies	Trial Only	**Medium.** Good for compute, but moves to pay-as-you-go quickly.
Akash	Docker (K8s)	$10 Credit (Trial)	N/A (Provider dependent)	N/A	**High.** Best for hosting the Gateway API/Database.
ICP	On-chain Canister	Cycles Faucet ($20-$100)	N/A	N/A	**Medium.** Requires rewriting logic for Canisters; complex but powerful.
Crust	IPFS (MPoW)	Trial via Crust Files	Unlimited (P2P)	N/A	**Medium.** Good for redundancy; utilizes Web3Auth gateway.

5.2 Strategic Utilization of Providers

Primary Storage: 4EVERLAND 4EVERLAND offers the most generous free tier among the "S3-compatible" pinning services, with 6GB of storage and 100GB of bandwidth per month. The S3 compatibility allows the DeCloud Gateway to use standard libraries (like aws-sdk in JavaScript) to upload user files, simply by changing the endpoint configuration to endpoint.4everland.co. This creates a seamless integration path where the tool acts as a reseller of 4EVERLAND's free tier.

Secondary Storage / Redundancy: Fleek Fleek is the industry standard for IPFS hosting. Its free tier includes 3GB of storage and 50GB of bandwidth. The DeCloud Gateway can be architected to failover to Fleek if 4EVERLAND limits are reached, or to use Fleek for premium users. Fleek also offers "Dank," a service for cycle management on ICP, which could be leveraged if the architecture expands to the Internet Computer.

Backend Compute: Akash Network The Gateway logic (the Node.js server checking subscriptions and routing requests) must run somewhere. **Akash Network** is the optimal decentralized choice. While it typically operates on a reverse-auction model using AKT tokens,

new deployments can utilize the ~$10 free trial credits available via the Akash Console. Furthermore, even after the trial expires, hosting a simple container on Akash is significantly cheaper than AWS or DigitalOcean, often costing less than $5/month, which can be easily covered by the aggregated 1 HBAR subscriptions of just 30-40 users.
Database & Logic: Internet Computer (ICP) For a truly decentralized backend that avoids the complexity of managing Docker containers, the Gateway logic could be deployed as a **Canister** on the Internet Computer. The Cycles Faucet provides enough resources to run a typical canister for a substantial period. This removes the need for a centralized server entirely, as the Canister can serve HTTP requests directly.

6. Detailed Cost & Revenue Analysis

The commercial viability of the 1 HBAR model rests on the margin between the revenue (1 HBAR) and the aggregate costs of Hedera network fees and underlying infrastructure.

6.1 Financial Modeling per User

Scenario: A user subscribes for 1 month, uploads a 5MB website, and updates it 4 times a month. **Revenue:** 1 HBAR (~$0.15 USD).
Expenses:

1. **Subscription Payment Transaction:**
    - *Native Transfer:* $0.0001 USD. (Paid by user or provider).
    - *Smart Contract Call (Method B):* ~$0.05 USD.
    - *Impact:* Using a Smart Contract eats ~33% of the revenue. Using Native Transfer eats <0.1%.
2. **HCS Logging (4 updates):**
    - *Cost:* 4 * $0.0001 = $0.0004 USD.
3. **Storage (IPFS):**
    - *Cost:* $0.00 (Utilizing 4EVERLAND/Fleek Free Tier).
4. **Revenue Split (Transfer to Developer):**
    - *Cost:* $0.0001 (If batched daily for all users).

Net Margin Analysis:

- **With Smart Contract:** Revenue ($0.15) - Gas ($0.05) - HCS ($0.0004) = **$0.0996 (66% Margin)**.
- **With Native Transfer:** Revenue ($0.15) - Transfer ($0.0001) - HCS ($0.0004) = **$0.1495 (99% Margin)**.

Conclusion: While the Smart Contract approach is more "trustless," the Native Transfer model (where the user sends 1 HBAR with a Memo, and the provider batch-transfers the split) is far more economically scalable for a micropayment model.

6.2 Future Outlook: Pricing Adjustments and HIP-991

Developers must account for the upcoming pricing change for the Hedera Consensus Service. In **January 2026**, the cost for ConsensusSubmitMessage is scheduled to increase from $0.0001 to $0.0008 USD. While this is an 8x increase, the absolute cost remains low (less than a tenth of a penny). However, for high-volume applications logging millions of messages, this must be factored into long-term revenue modeling.

Additionally, **HIP-991** proposes "Permissionless Revenue-Generating Topic IDs". This future feature will allow Topic creators to enforce a fee (in HBAR or HTS tokens) for submitting messages to a topic. This would allow the DeCloud Gateway to monetize the *logging* action itself directly at the protocol level, simplifying the revenue model further by embedding the fee into the HCS transaction rather than a separate subscription payment.

7. Strategic Recommendations and Conclusion

7.1 Second-Order Insights: The Reseller Opportunity

The analysis suggests that building raw physical infrastructure from scratch on Hedera is not the most efficient path. Instead, the most potent opportunity lies in the **Service Aggregator** model. Users face high friction in acquiring specific tokens like AKT (Akash) or managing Arweave wallets. By accepting **1 HBAR**—a widely accessible token—and handling the complexity of the underlying storage and compute networks in the background, the DeCloud Gateway solves a User Experience (UX) problem rather than just an infrastructure problem.

7.2 Final Recommendations

1. **Adopt the Hybrid Architecture:** Do not attempt to store website files on Hedera. Use IPFS for storage and Hedera HCS for the "Proof of Publication" and version control.
2. **Optimize for Native Transfers:** To maximize the margin of the 1 HBAR model, prefer native HTS transfers or Atomic Swaps over complex Smart Contract interactions, or ensure the Smart Contract logic is extremely lightweight to minimize gas.
3. **Leverage Free Tiers for Bootstrapping:** Utilize the free tiers of 4EVERLAND and Fleek to handle the storage load initially. This keeps the variable cost per user at effectively zero ($0.00) for storage, allowing the 1 HBAR revenue to cover development and HCS fees.
4. **Monitor Policy Changes:** Prepare for the HCS pricing increase in 2026 by optimizing message payloads or batching updates to reduce message frequency.

By following this blueprint, a developer can successfully deploy a decentralized, "Google Cloud-like" infrastructure tool on Hedera that is not only technically sound but also economically viable within the constraints of a 1 HBAR revenue model.

Works cited

1. Hedera: Hello future, https://hedera.com/ 2. How to Build a Decentralized App on Hedera | Beginner to Pro Guide - YouTube, https://www.youtube.com/watch?v=hlzblTgyudI 3. Cost of storage : r/Hedera - Reddit, https://www.reddit.com/r/Hedera/comments/rr5bny/cost_of_storage/ 4. HCS Message Size Limit · Issue #1145 · hiero-ledger/hiero-consensus-node - GitHub, https://github.com/hashgraph/hedera-services/issues/1145 5. Web3 Dapp Hosting: Components, Preferences, and Best Practices, https://tatum.io/blog/web3-dapp-hosting-a-guide 6. Submit a message - Hedera Docs, https://docs.hedera.com/hedera/sdks-and-apis/sdks/consensus-service/submit-a-message 7. Transactions - Hedera Docs, https://docs.hedera.com/hedera/sdks-and-apis/sdks/transactions 8. Fees - Hedera Docs, https://docs.hedera.com/hedera/networks/mainnet/fees 9. Convert Hedera (HBAR) to US Dollars (USD) - Revolut, https://www.revolut.com/crypto/price/hbar/usd/?amount=1 10. HBAR/USD: Convert Hedera to

United States Dollar - Crypto.com, https://crypto.com/en/converter/hbar/usd 11. Hedera Consensus Service, https://hedera.com/consensus-service 12. Arweave - A community-driven ecosystem, https://arweave.org/ 13. IPFS Vs Arweave Permaweb - Medium, https://medium.com/coinmonks/ipfs-vs-arweave-permaweb-a482dc717150 14. IPFS Pinning - Scaleway Labs, https://labs.scaleway.com/en/ipfs-pinning/ 15. Compare IPFS Cluster vs. Pinata in 2025 - Slashdot, https://slashdot.org/software/comparison/IPFS-Cluster-vs-Pinata/ 16. Guardian - IPFS & Hedera, https://blog.ipfs.tech/2022-11-10-guardian-ipfs-and-hedera/ 17. Neuron | Hedera, https://hedera.com/users/neuron 18. How To Run Your Own IPFS Gateway - Pinata, https://pinata.cloud/blog/how-to-run-your-own-ipfs-gateway/ 19. Understanding Crust Network: The DePin Cloud Storage Project Difficulty: Intermediate, https://www.gate.com/learn/articles/understanding-crust-network-the-depin-cloud-storage-project-difficulty-intermediate/3634 20. Crust Files Premium Memberships Give Away For OKC Users, https://crustnetwork.medium.com/crust-files-premium-memberships-give-away-for-okc-users-b0528aa5f02e 21. Akash Network - Decentralized Compute Marketplace, https://akash.network/ 22. FluxCloud: Decentralized Cloud for Scalable Deployments, https://runonflux.com/fluxcloud/ 23. Introducing: Credit Card Payments in Akash Console, https://akash.network/blog/introducing-credit-card-payments-in-akash-console/ 24. Cost to host a website. Fee model. : r/dfinity - Reddit, https://www.reddit.com/r/dfinity/comments/ofytz9/cost_to_host_a_website_fee_model/ 25. Cycles Faucet: Free Cycles to Build on the Internet Computer | by DFINITY - Medium, https://medium.com/dfinity/cycles-faucet-free-cycles-to-build-on-the-internet-computer-789166a95140 26. Gas and Fees - Hedera Docs, https://docs.hedera.com/hedera/core-concepts/smart-contracts/gas-and-fees 27. Predictable fees. Made easy. - Hedera, https://hedera.com/fees 28. Transfer tokens - Hedera Docs, https://docs.hedera.com/hedera/sdks-and-apis/sdks/token-service/transfer-tokens 29. HIP-410 - Wrapping Ethereum Transaction Bytes in a Hedera Transaction · hiero-ledger hiero-improvement-proposals · Discussion #411 - GitHub, https://github.com/hashgraph/hedera-improvement-proposal/discussions/411 30. Coding with Cooper - Get started with the cryptocurrency API & HBAR - YouTube, https://www.youtube.com/watch?v=hqQWdcPhTDk 31. Pricing Model | 4EVERLAND Documents, https://docs.4everland.org/get-started/billing-and-pricing/pricing-model 32. Simple, Flexible Pricing - 4EVERLAND - A Cloud Computing Platform of WEB 3.0, https://www.4everland.org/price/ 33. Plans and Pricing - Fleek | Build on the New Internet, https://fleek.co/pricing/ 34. Tokens & cycles | Internet Computer, https://internetcomputer.org/docs/building-apps/getting-started/tokens-and-cycles 35. Price Update to ConsensusSubmitMessage in Consensus Service… - Hedera, https://hedera.com/blog/price-update-to-consensussubmitmessage-in-consensus-service-january-2026 36. Introducing HIP-991: Permissionless Revenue-Generating Topic… | Hedera, https://hedera.com/blog/introducing-hip-991-permissionless-revenue-generating-topic-ids-for-topic-operators