GPU Cost Optimizer & Recommender

#hackathon #Y2025 #hiring #campus

1. Domain / Themes

1. Cloud Infrastructure Optimization
2. Application Development (Full-stack Web)
3. AI/ML-assisted Decision Support (basic AI/ML usage)
4. Cost & Resource Efficiency in Cloud Computing

2. Background / Context

AceCloud offers a diverse portfolio of GPU-enabled cloud instances (A100, L40s, L4, RTX A6000, A30, A2, etc.) with varying CPU, RAM, storage, and pricing options across multiple regions. Customers often find it challenging to select the optimal GPU instance that balances their workload requirements and budget constraints.

AceCloud provides a public pricing API with detailed GPU instance specifications and costs. Internally, AceCloud maintains a knowledge base mapping GPU types to workload suitability. This challenge invites participants to build a cloud-native web application that helps customers input their workload needs and receive tailored GPU instance recommendations, including cost breakdowns and contextual explanations. The solution should be developer-friendly, scalable, and user-centric.

3. Objectives

The solution must:

1. Provide a user-friendly interface to capture workload characteristics (e.g., type of AI/ML model, dataset size, training vs inference, budget).
2. Fetch real-time GPU instance pricing and specs from AceCloud's public API.
3. Implement recommendation logic (rule-based or simple heuristics) that maps workload inputs to appropriate GPU instance types with CPU, RAM, and storage sizing.
4. Present multiple instance options with detailed cost comparisons (hourly, monthly, spot vs. on-demand).
5. Provide clear, contextual explanations for each recommendation referencing AceCloud's GPU knowledge base.
6. (**Optional/Advanced Feature**) Integrate an AI-powered assistant (using OpenAI/OpenRouter or RAG) to answer natural language queries about GPU selection and cost optimization. *(This is an advanced stretch goal and not required for baseline submission.)*

7. (**Optional/Advanced Feature**) Support streaming responses for recommendations and explanations. *(This feature enhances user experience but is considered advanced and optional.)*
8. Containerize the full-stack application for easy deployment.
9. Include automated tests and comprehensive API documentation.

4. Functional Requirements

#	Scenario	Expected Behavior
FR-1	Workload Input Form	User inputs workload details: model type, dataset size, training/inference, budget, preferred region, etc.
FR-2	GPU Pricing Fetch	Backend fetches current GPU instance pricing/specs from AceCloud's public API.
FR-3	Recommendation Engine	Based on workload inputs and knowledge base, returns suitable GPU instance(s) with sizing and cost details.
FR-4	Cost Comparison View	Displays side-by-side comparison of recommended instances, including spot vs on-demand pricing.
FR-5	Explanation Panel	Shows contextual reasons for each recommendation (e.g., "A100 preferred for large batch training due to VRAM").
FR-6	(Optional/Advanced Feature) AI Assistant	Conversational UI for natural language queries about GPU selection and cost trade-offs.
FR-7	(Optional/Advanced Feature) Streaming Response	Recommendations and explanations should stream to the frontend as they are generated, improving user experience.
FR-8	Request Button Action	If a GPU instance is unavailable or has zero/missing pricing, show a "Request" button. When clicked, display a simple success message such as "Your request has been submitted. We will notify you when this GPU becomes available." No backend integration required for this button in the hackathon scope.

5. Technical Requirements & Constraints

1. Technology Stack: Frontend (React or Next.js), Backend (Node.js with NestJS/Express or Python Flask/FastAPI).
2. API Integration: Use the provided AceCloud GPU pricing public API endpoint (example below).
3. Containerization: Dockerize frontend and backend with clear build and run instructions.
4. Testing: Implement unit and integration tests for backend APIs; frontend tests optional but encouraged.
5. Documentation: Provide basic API documentation (simple markdown overview acceptable), setup instructions, and brief user guide. *(Full Swagger/OpenAPI documentation is optional and considered advanced.)*

6. Performance: Recommendations should be generated within 2 seconds of input submission.
7. Security: Secure API endpoints with HTTPS (self-signed or mock TLS acceptable).
8. Data: Use synthetic or public datasets for workload examples if needed.
9. AI Tools & LLM Usage:
    1. Participants may leverage AI-powered code assistants such as GitHub Copilot, Cursor, or any advanced AI coding tools to accelerate development.
    2. For AI-powered recommendation explanations or conversational features, teams may integrate any public or commercial LLMs including but not limited to OpenAI (GPT-4), Anthropic Claude, Google Gemini, OpenRouter, Grok, or similar.
    3. Usage of these models should comply with their respective usage policies and must not expose proprietary AceCloud data.
10. Streaming Responses: The system should support streaming partial recommendation or explanation results to the frontend to improve user experience.
11. The provided PDF knowledge base must be processed and queried only in the backend (e.g., using LangChain, ChromaDB, or similar).
12. No raw document content may be exposed to frontend or end-users; only answers generated via retrieval/LLM are allowed.

6. Deliverables

1. Working Demo: Hosted web app or recorded video demonstrating all functional scenarios.
2. Source Repository: Public Git repo with modular, well-documented code, Docker/Helm deployment artifacts.
3. Documentation (Required baseline):
    1. Architecture overview
    2. Simple API endpoint descriptions (markdown or README)
    3. Basic user guide and setup instructions
4. Documentation (Optional/Advanced)
    1. Detailed Swagger/OpenAPI documentation
    2. Dialog-flow diagrams (only if AI assistant is implemented).
5. Test Cases: Demonstrate backend test runs; frontend test runs if implemented.
6. Optional: Slide deck (≤10 slides) summarizing design, challenges, and learnings.

7. Evaluation Criteria

Weight	Criterion	Details
30%	Architectural Soundness	Modular, scalable design adhering to cloud-native principles and fault tolerance.
25%	Innovation Quotient	Novel use of AceCloud’s GPU catalog and AI/ML for recommendations and explanations.
20%	Operational Efficiency	Cost-effective resource recommendations and efficient API usage.
15%	Code Quality	Readability, maintainability, test coverage, and security best practices.
5%	Testing & Documentation	Completeness and clarity of tests and documentation.
5%	UI/UX Quality	Intuitive, consistent, and user-friendly interface design.

8. Example AceCloud GPU Pricing API Response (Partial)

https://customer.acecloudhosting.com/api/v1/pricing?is_gpu=true&resource=instances

```
{
 "data": [
   {
     "country": "india",
     "resource_class": "a100",
     "vcpus": 16,
     "ram": 96,
     "price_per_hour": 3.42,
     "price_per_month": 1563,
     "price_per_spot": 2.394,
     "gpu_description": "1x A100-80GB",
     "region": "mumbai"
   },
   {
     "country": "india",
     "resource_class": "a30",
     "vcpus": 8,
     "ram": 32,
     "price_per_hour": 0.9,
     "price_per_month": 525,
     "price_per_spot": 0.63,
     "gpu_description": "1x A30-24GB",
     "region": "mumbai"
   }
   // More instances...
 ]
}
```

9. Resources Provided

1. AceCloud GPU pricing public API endpoint.
2. Internal GPU knowledge base document (PDF, backend-only access; for RAG/AI assistant use, not to be exposed to frontend or users).
3. Teams may also create a supplementary knowledge base from public internet sources using GPU names from the pricing API.

10. Timeline (4–6 Hours Suggested)

1. Hour 0–1: Environment setup, API exploration, basic frontend form and backend API integration.
2. Hour 1–3: Implement recommendation logic, cost comparison UI, and explanations panel.
3. Hour 3–4: (Optional/Advanced Task) Add AI assistant/chatbot, or focus on refining recommendation accuracy/UI.
4. Hour 4–5: Testing, documentation, and demo preparation.
5. Hour 5–6: Final polish, bug fixes, and presentation.

11. Non-Functional Requirements (NFRs)

1. Availability Filtering: Display only GPU instances that are currently available and have valid (non-zero) pricing.
2. Request Handling: If a GPU is unavailable or its price is zero/missing, show a “Request” button.
3. Request Button Behavior: On clicking the “Request” button, display a dummy success message (e.g., via modal or toast). No backend processing is required.
4. Frontend Testing: Implement minimal frontend tests (using Cypress, React Testing Library, etc.) for key flows such as workload input, recommendations display, cost comparison, and the “Request” button. Frontend tests are optional but encouraged.
5. Data Integrity: Validate all API responses; do not display invalid or incomplete data.
6. Streaming Response (Optional): Recommendations and explanations may stream dynamically to the frontend as they are generated. (Not mandatory but enhances UX.)
7. Code Quality: Code must be clean, modular, and well-documented, following standard best practices.
8. Performance: All user actions (e.g., recommendations) should respond within 2 seconds.
9. Security: Use HTTPS/TLS for all communications; do not expose sensitive data.
10. Professional UI: Maintain a clean, intuitive, and consistent user experience—avoid unnecessary creativity or non-standard controls.

11. Error Handling: Show clear, user-friendly error messages for API failures or missing data.
12. Testing & Documentation: Include automated tests for critical backend flows and clear API documentation.