**Project Title: GPU Clustering and Multi-Tenant Utilization Tool**

**Project Goals**

* **Develop a tool that can effectively cluster GPUs and manage their utilization across multiple tenants.**
* **Ensure the tool is efficient, scalable, secure, and user-friendly.**

**Project Scope**

* **Core** **Features**: GPU discovery, clustering, tenant management, workload distribution, monitoring, and reporting.
* **Optional** **Features**: Security, scalability, integration, and fault tolerance.

**Core Requirements**

1. **GPU Discovery and Inventory:**
   * **Automatic detection**: The tool should be able to automatically discover GPUs in a network or cluster environment.
   * **Comprehensive inventory**: It should provide detailed information about each GPU, including its model, specifications, and current utilization.
2. **Clustering Algorithm:**
   * **Efficiency**: The clustering algorithm should be efficient in grouping GPUs based on factors like performance, compatibility, and workload requirements.
   * **Flexibility**: It should be customizable to accommodate different clustering strategies and priorities.
3. **Tenant** **Management**:
   * **Tenant** **creation**: The tool should allow for the creation and management of multiple tenants or users.
   * **Resource** **allocation**: It should provide mechanisms for allocating GPU resources to different tenants based on their needs and priorities.
4. **Workload** **Distribution**:
   * **Intelligent** **assignment**: The tool should be able to intelligently distribute workloads across clusters of GPUs to optimize performance and utilization.
   * **Priority** **management**: It should allow for the prioritization of different workloads based on their importance or deadlines.
5. **Monitoring and Reporting:**
   * **Real**-**time** **monitoring**: The tool should provide real-time monitoring of GPU utilization, workload distribution, and system performance.
   * **Detailed** **reporting**: It should generate comprehensive reports on GPU usage, tenant activity, and workload performance.

**Additional Requirements**

* **Security**: Implement robust security measures to protect sensitive data and prevent unauthorized access.
* **Scalability**: Design the tool to handle large-scale GPU clusters and a growing number of tenants.
* **Integration**: Enable integration with other systems or tools, such as cloud platforms or workload management systems.
* **Fault** **tolerance**: Implement mechanisms for detecting and recovering from hardware failures or software errors.

**Examples of Use Cases**

* Cloud Computing: A cloud service provider can use the tool to efficiently manage and allocate GPU resources to multiple customers.
* Scientific Research: Researchers can use the tool to optimize the execution of computationally intensive simulations and experiments.
* Machine Learning: Machine learning practitioners can use the tool to train and deploy large-scale models on clusters of GPUs.
* Deep Learning: Deep learning researchers can leverage the tool to accelerate the development and training of neural networks.
* Gaming: Gaming platforms can use the tool to manage GPU resources for game servers and ensure optimal performance.

**Deliverables**

1. **GPU** **Clustering** **System**: A system that combines multiple GPU servers and workstations into a unified pool.
2. **User** **Access** **Control**: A mechanism to assign GPU resources based on user access levels.
3. **Interactive** **Notebook** **Interface**: A user-friendly interface similar to Jupyter Notebooks or Google Colab.
4. **Resource** **Management** **Dashboard**: A dashboard to monitor GPU availability, utilization, and user access.
5. **Scalability** **and** **Load** **Balancing**: Strategies to handle multiple users and ensure optimal resource allocation.

**Technical Deliverables**

* Minimum Viable Product: A functional demonstration of the tool's core capabilities, including GPU discovery, clustering, tenant management, and workload distribution.
* Source Code: The complete source code for the tool, including any libraries or dependencies.
* Documentation: Comprehensive documentation covering the tool's architecture, design decisions, usage instructions, and API reference.
* Test Cases: A suite of test cases to verify the tool's functionality, performance, and reliability.
* Performance Benchmarks: Results from performance benchmarks to demonstrate the tool's efficiency and scalability.
* Security Assessment: A security assessment report outlining any vulnerabilities or risks identified and the measures taken to mitigate them.
* User Interface (UI) Mockups: Visual representations of the tool's user interface, demonstrating the intended look and feel.

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Description** | **Success Criteria** |
| **1. System Architecture Design** | **Design the overall architecture for the GPU clustering system, including user access control and resource management components.** | **Completed system architecture diagram and technical specifications.** |
| **2. GPU Cluster Setup** | **Set up the GPU cluster environment, including network configuration and resource discovery.** | **Successfully clustered multiple GPUs and verified resource availability.** |
| **3. User Access Control Implementation** | **Implement the user access control system, integrating with existing university authentication mechanisms.** | **Successfully tested user access control based on different roles and permissions.** |
| **4. Interactive Notebook Interface Development** | **Develop the interactive notebook interface, providing a similar experience to Jupyter Notebooks or Google Colab.** | **Completed prototype of the notebook interface with basic functionality.** |
| **5. Resource Management Dashboard** | **Create a dashboard to visualize GPU availability, utilization, and user access in real-time.** | **Completed dashboard with key metrics and visualizations.** |
| **6. Testing and Optimization** | **Conduct thorough testing to identify and address any issues, optimize performance, and ensure scalability.** | **Successful completion of testing and optimization, meeting performance and scalability requirements.** |

**Tools:-**

1. Docker – Containerization tool to package applications.

2. NVIDIA Docker – GPU-enabled Docker for leveraging GPU resources.

3. NVIDIA DCGM (Data Center GPU Manager)– For GPU monitoring and management.

4. Slurm or HTCondor – Workload managers for scheduling jobs.

5. Google Cloud/AWS/Azure GPU Services – Cloud platforms offering GPU resources.

6. Grafana + Prometheus – Tools for real-time monitoring and visualization.

7. NVIDIA-SMI (System Management Interface)– Utility to monitor GPU metrics.

8. Telegraf, InfluxDB – Tools for data collection and storage of monitoring data.

9. HAProxy or NGINX – Load balancing tools to manage resource allocation.

10. OAuth 2.0 – Authentication tool for secure access control.

11. LDAP or Active Directory Integration – For managing user roles and permissions.

**Frameworks:**

1. Kubernetes (K8s)– Orchestration framework for managing containers and workloads.

2. KubeFlow – Kubernetes-based framework for machine learning workflows with GPU support.

3. TensorFlow/Keras with GPU Support – Deep learning frameworks to run on GPUs.

4. PyTorch with Distributed Training– Machine learning framework for distributed GPU training.

5. Jupyter Notebooks– Interactive notebook framework for code execution.

6. Flask/Django– Web frameworks for building custom interfaces.

7. Apache Kafka – Messaging framework for handling high-throughput messaging.

**Tool Requirements for GPU Clustering and Multi-Tenant Utilization Tool**

**Hardware:**

* **GPU Servers: Multiple GPU servers with compatible hardware and drivers.**
* **Network Infrastructure: A high-speed network to connect the GPU servers and provide reliable communication.**
* **Storage: Sufficient storage capacity to store user data, application code, and system logs.**

**Software:**

* **Container Orchestration: Kubernetes or Docker Swarm for managing and deploying containers.**
* **GPU Management: NVIDIA Container Toolkit or ROCm for managing GPU resources within containers.**
* **Programming Languages: Python, C++, or other languages suitable for developing the tool's components.**
* **Web Framework: A web framework like Flask, Django, or React for building the user interface.**
* **Database: A database like PostgreSQL or MySQL for storing user data, resource information, and usage statistics.**
* **Monitoring Tools: Prometheus and Grafana for monitoring GPU utilization, system performance, and user activity.**
* **Security Tools: Firewall, intrusion detection systems, and encryption tools to protect the system and user data.**

**project\_directory/**

**├── requirements.txt**

**├── Dockerfile**

**├── docker-compose.yml**

**├── app/**

**│ ├── \_\_init\_\_.py**

**│ ├── views.py**

**│ ├── models.py**

**│ └── ...**

**├── config/**

**│ ├── \_\_init\_\_.py**

**│ └── settings.py**

**└── ...**