Helioq Alpha Smart Contract: Features and Requirements

1. NodeX Server Management

1. **NodeX Server Registration**
   - **Purpose**: Allows the admin to register NodeX servers.
   - **Requirements**:
     - Each server must have a unique server ID.
     - Servers are linked to the user's wallet address.
   - **Features**:
     - Emit a `ServerRegistered` event after successful registration.
2. **Server Deregistration**
   - **Purpose**: Allows the admin to deactivate servers that fail to meet performance standards or are offline for extended periods.
   - **Features**:
     - Deactivate server by server ID.
     - Emit a `ServerDeregistered` event for transparency.
3. **Server Reassignment**
   - **Purpose**: Enables transferring ownership of NodeX servers between users.
   - **Requirements**:
     - Admin must approve reassignment.
     - Record new wallet address for the server.
   - **Features**:
     - Emit a `ServerReassigned` event for each successful transfer.

4. **Multiple Servers per Wallet**
    - **Purpose**: Users can register and manage multiple NodeX servers linked to a single wallet address.
    - **Features**:
        - Aggregate performance metrics and rewards for all servers linked to the wallet.

---

2. Performance Tracking and Rewards

1. **Performance Metrics Submission**
    - **Purpose**: Admin or backend system submits performance data for each NodeX server.
    - **Metrics**:
        - Uptime (%)
        - Tasks Completed
        - Resource Utilization (CPU, memory, bandwidth)
    - **Features**:
        - Emit a `MetricsUpdated` event.
2. **Grace Period for New Servers**
    - **Purpose**: Allows new servers to stabilize before being subject to penalties or regular reward calculations.
    - **Features**:
        - Define a grace period (e.g., 7 days) for new servers.
        - Adjust point calculations during the grace period.
        - Emit a `GracePeriodStarted` and `GracePeriodEnded` event.
3. **Penalty System for Poor Performance**
    - **Purpose**: Penalizes servers with extended downtime or failing to meet minimum performance thresholds.

- **Features**:
  - Reduce performance points if uptime or tasks fall below defined thresholds (e.g., <50% uptime).
  - Emit a `PenaltyApplied` event for transparency.

4. **Reward Allocation**
   - **Purpose**: Distribute SOL rewards based on server performance points.
   - **Features**:
     - Update pending rewards after each metrics submission.
     - Emit a `RewardsAllocated` event.
5. **Reward Claiming**
   - **Purpose**: Allow users to claim their pending SOL rewards after a cooldown period.
   - **Features**:
     - Enforce a 7-day cooldown before rewards can be claimed.
     - Users can claim rewards for specific servers or all servers linked to their wallet.
     - Emit a `RewardsClaimed` event.

---

3. Admin Controls

1. **Reward Pool Management**
   - **Purpose**: Manage the SOL reward pool for distribution.
   - **Features**:
     - Admin can deposit SOL into the contract.
     - Emit a `RewardsDeposited` event.
     - Emit a `LowRewardPool` alert if balance drops below a defined threshold.
2. **Reclaim Unclaimed Rewards**
   - **Purpose**: Admin can reclaim unclaimed SOL rewards after 1 year.

- **Features**:
  - Transfer unclaimed SOL back to admin wallet.
  - Emit a `RewardsReclaimed` event.

3. **Emergency Pause Mechanism**
   - **Purpose**: Temporarily suspend critical functions in case of emergencies.
   - **Features**:
     - Pause operations such as reward claiming, server registration, or metrics submission.
     - Emit an `EmergencyPaused` and `EmergencyResumed` event.

---

4. Data and Transparency

1. **Data Retrieval**
   - **Purpose**: Allow users and admins to query data from the contract.
   - **Features**:
     - View server metrics (uptime, tasks, resources).
     - Query pending rewards for a wallet or server.
     - Aggregate metrics and rewards for multiple servers linked to a wallet.
2. **Audit Trail for Metrics and Rewards**
   - **Purpose**: Maintain transparency for performance data and reward calculations.
   - **Features**:
     - Emit detailed logs for performance submissions and reward updates.

---

5. Security and Access Control

1. **Access Control**
   - Admin-only functions:
     - Register and deregister servers.

    - Submit performance metrics.
    - Deposit and reclaim SOL rewards.
  - Public functions:
    - Query data.
    - Claim rewards.

2. **Reentrancy Protection**
   - Use `nonReentrant` modifier for functions involving SOL transfers to prevent reentrancy attacks.
3. **Input Validation**
   - Validate inputs (e.g., serverID, metrics) to prevent incorrect data submissions.

---

6. Events

1. **Server Events**
   - `ServerRegistered(string serverID, address indexed walletAddress)`
   - `ServerDeregistered(string serverID)`
   - `ServerReassigned(string serverID, address indexed newWalletAddress)`
2. **Metrics and Rewards Events**
   - `MetricsUpdated(string serverID, uint256 points)`
   - `GracePeriodStarted(string serverID)`
   - `GracePeriodEnded(string serverID)`
   - `PenaltyApplied(string serverID, uint256 penaltyAmount)`
   - `RewardsAllocated(string serverID, uint256 rewardAmount)`
   - `RewardsClaimed(address indexed walletAddress, uint256 rewardAmount)`
3. **Admin and Pool Events**
   - `RewardsDeposited(uint256 amount)`

- `LowRewardPool(uint256 balance)`
- `RewardsReclaimed(uint256 amount)`
- `EmergencyPaused()`
- `EmergencyResumed()`

Development Notes

1. **Framework**:
   - Use **Rust** and the **Anchor Framework** for Solana smart contract development.
2. **Testing**:
   - Write unit tests for all functions and edge cases (e.g., reward calculations, claim cooldown).
3. **Deployment**:
   - Deploy to **Solana Devnet** for initial testing, then transition to **Mainnet Beta**.