Hack-a-thon UNster #2

Latest version of this document **hack-a-thon_2.md**

https://gist.github.com/Frajder/1b89d48ee964c80e7af616cd5645e365#file-hack-a-thon_2-md

Application Overview and Purpose

This prototype application provides **secure access management** and **role-based permissions** using **NEAR Protocol wallets** and **NFTs**. It aims to authenticate users via their **whitelisted wallet addresses** and issue API keys as **bearer tokens** for authorized operations. Leveraging a **serverless architecture on AWS** (Lambda, API Gateway, and S3) and **NFT-based roles** for **Role-Based Access Control (RBAC)**, the system ensures that only authorized users can perform specific operations. Future iterations will migrate the whitelist and access control mechanisms to a **blockchain-based solution** for decentralized access.

By integrating this NEAR Protocol wallet and NFT-based system, the application will:

- Ensure **secure access management** with **wallet-based authentication**.
- Implement a **scalable, serverless infrastructure** using **AWS services**.
- Automate **role-based permissions** via **NFTs**.
- Facilitate seamless **user management** through **open workspaces** and **teams**.

Application Architecture

1. **Development Environment & CI/CD**
    - Local development and CI pipelines will include modules for **NEAR Protocol wallet integration** and **NFT minting** logic.
    - **GitHub Actions** will trigger **security scans and tests** to validate wallet-based authentication and NFT logic using tools like **eslint**.
2. **AWS Infrastructure**
    - **API Gateway** and **Lambda** will manage **wallet authentication**, **NFT verification**, and **API key issuance**.
    - **S3** stores the `allowed_wallets.json` for whitelist management and **JavaScript modules** for client-side functionality.
    - **CloudFront** ensures efficient delivery of **JavaScript modules** and public resources.

3. **Security and Monitoring**
    - **CloudWatch** and **X-Ray** will monitor API requests and log potential access issues.
    - **IAM roles** will enforce fine-grained access control to ensure only authorized users interact with system components.
4. **NFTs as Dynamic Roles**
    - **Minted NFTs** define dynamic roles that can be updated or revoked as needed.
    - **Admins** manage users, teams, and workspaces by assigning specific NFTs to new users, granting them access to appropriate functions.

Process Flow with NFT and Wallet Authorization Integration

1. **User Authorization via NEAR Protocol Wallets**
    - A JSON file named `allowed_wallets.json` stored in **Amazon S3** contains whitelisted wallet addresses.
    - Users **authenticate by signing a message with their NEAR wallet**. This signature is verified in a **Lambda function** to confirm the wallet address is on the whitelist.
    - Upon successful verification, the user receives an **API key** (issued as a bearer token) included in **request headers** to access other API endpoints.
2. **Role Management Using NFTs**
    - Roles are represented by **NFTs** minted by users upon successful whitelisting.
    - The three roles are:
        a. **Admin:** Can manage permissions and users but cannot access or modify data.
        b. **Researcher:** Can view data but is restricted from performing administrative tasks.
        c. **Volunteer:** Limited to **submitting questionnaires** and accessing specific data.
    - **Admins** assign new users to **open workspaces and teams**, and users mint their **role-specific NFT** to gain access to appropriate API operations.
3. **API Key and Role-Based Access Control (RBAC) Flow**
    - Once whitelisted, a user connects their **NEAR wallet** to **mint a role-specific NFT**.
    - The **NFT acts as proof of role**, defining the user's permissions.
    - **Lambda functions** verify NFT ownership and validate **API requests**. Based on the user's role, the API Gateway grants or denies access to requested resources.
        - **Example operations:**
            - **PUT / POST:** Upload or modify files.
            - **GET:** Retrieve files or data.
            - **DELETE:** Remove files or data.

        - **Role Validation:** Sensitive endpoints may require additional **wallet signature verification**.
4. **Development Workflow and Deployment Pipeline**
    - **JavaScript-based microservices** handle core logic like **signature verification, role mapping, and NFT management**.
    - **Docker containers** running on **Amazon ECS with Fargate** manage scalable microservices for complex tasks.
    - **JavaScript modules** deployed via **S3** and **CloudFront** handle client-side operations like NFT minting.
    - **API Gateway** controls access to Lambda functions, issuing bearer tokens based on wallet verification and NFT ownership.

Enhanced RBAC Architecture Using NEAR Protocol Wallets and NFTs

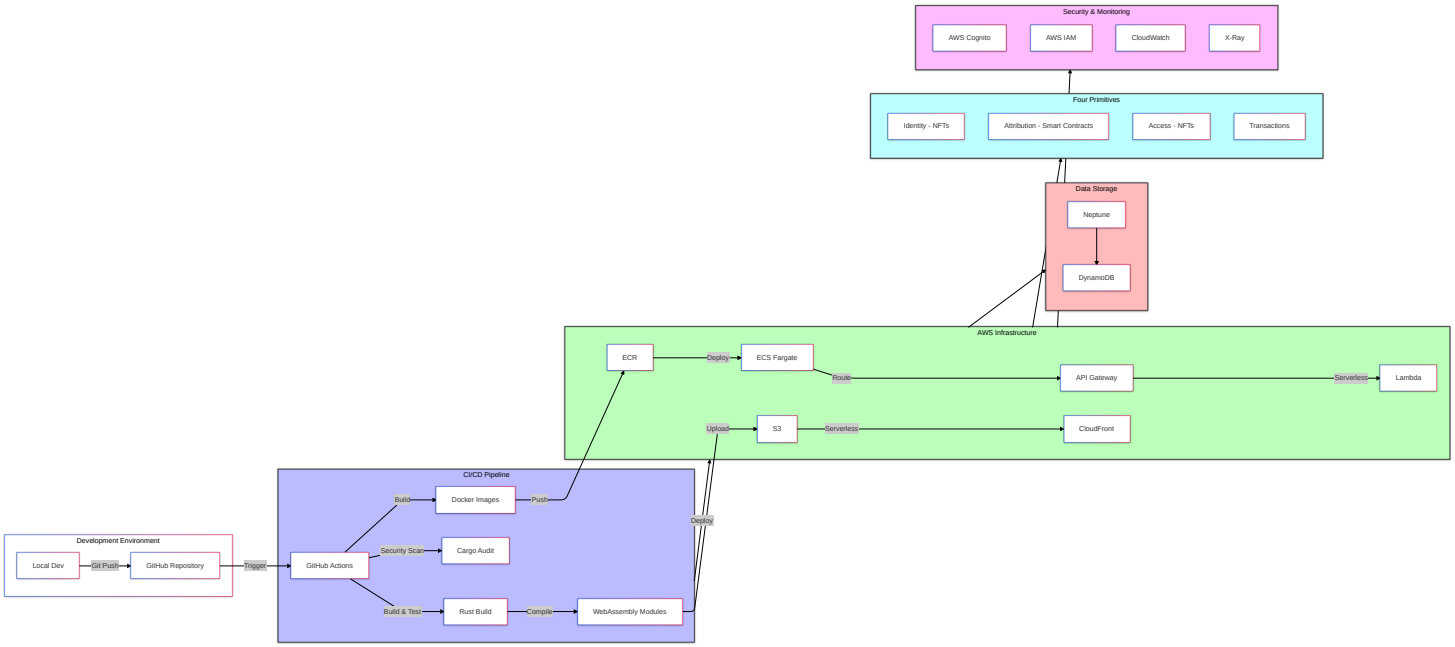
- **Whitelist Management**
    - Admins manage the whitelist by **adding wallet addresses** to `allowed_wallets.json` in **S3**.
    - Only whitelisted addresses can **mint NFTs**, providing access to specific roles within the application.
- **Minting NFTs for Role Assignment**
    - Users mint NFTs via a **frontend interface** by connecting their **NEAR wallet**.
    - The **NFT defines the user's role** (Admin, Researcher, or Volunteer), granting permissions accordingly.
- **Token Verification and Permissions Mapping**
    - API keys are generated and issued only to verified users.
    - A **JavaScript-based Lambda function** verifies wallet signatures and **checks for NFT ownership** to map roles to permissions.
    - **Permissions are defined in the codebase**, and sensitive API endpoints may require additional **wallet signature validation** for secure access.

Development Plan

1. **Development Environment**
    - Set up local development environments with Docker for consistency.
    - Use **GitFlow** branching strategy (main, develop, feature, release, hotfix branches).
    - Implement pre-commit hooks for code linting and formatting using **eslint** and **prettier**.
2. **Continuous Integration (CI) with GitHub Actions**

- Trigger CI pipeline on pull requests and pushes to develop/main branches.
- Steps in CI pipeline:
  - **Code linting and style checks** using eslint and prettier.
  - **Unit tests** with Jest.
  - **Integration tests** using Supertest or similar tools.
  - **Security scans** (e.g., npm audit for dependency checks).
  - **Build JavaScript modules**.
  - **Build Docker images** for JavaScript-based microservices.
  - **Push Docker images** to Amazon Elastic Container Registry (ECR).

3. **AWS Microservices Architecture**
   - Use **Amazon ECS (Elastic Container Service)** with Fargate for running JavaScript-based microservices.
   - Implement **API Gateway** for managing and securing APIs, with JavaScript-based Lambda functions where applicable.
   - Deploy JavaScript modules to **Amazon S3** for client-side execution.



4. **Testing Strategy**
   - **Unit Testing:** Implement with Jest.
   - **Integration Testing:** Use Supertest and Jest.
   - **Load Testing:** Employ tools like Artillery for performance testing.
   - **Security Testing:** Integrate npm audit and other JavaScript-compatible security tools.
5. **Documentation**
   - Use tools like **JSDoc** to generate documentation for JavaScript modules.
   - Implement automated documentation generation from code comments.

- Maintain a comprehensive wiki for development and operational procedures, focusing on JavaScript best practices.

Punch-List Tasks for the Hack-a-thon #2

- **Wallet Integration**
  - Integrate NEAR Protocol wallet authentication in the frontend.
  - Implement message signing and verification with NEAR wallets in Lambda functions.
- **NFT Minting Functionality**
  - Develop smart contracts on NEAR Protocol for NFT roles.
  - Create frontend interfaces for users to mint role-specific NFTs.
- **Whitelist Management**
  - Set up `allowed_wallets.json` in S3 and implement admin functionality to add/remove wallets.
  - Implement Lambda functions to check wallet addresses against the whitelist.
- **API Key Issuance**
  - Develop Lambda functions to issue API keys (bearer tokens) upon successful wallet verification.
  - Ensure tokens are securely stored and managed.
- **RBAC Implementation**
  - Define roles and permissions in the codebase.
  - Implement middleware in Lambda functions to enforce role-based access control based on NFT ownership.
- **Frontend Development**
  - Build user interfaces for authentication, NFT minting, and accessing protected resources.
  - Ensure responsive design and user-friendly experience.
- **Testing and QA**
  - Write unit tests for all new functions and modules.
  - Perform integration testing to ensure all components work together seamlessly.
- **Security Measures**
  - Implement security best practices for handling wallets, signatures, and tokens.
  - Conduct security audits using tools like npm audit and fix any vulnerabilities.
- **Documentation**
  - Document all APIs, modules, and functions developed during the Hackathon.
  - Update the project wiki with setup instructions and developer guidelines.
- **Deployment Pipeline**
  - Set up CI/CD pipelines in GitHub Actions for automated testing and deployment.
  - Configure AWS resources (Lambda, API Gateway, S3, ECS) for deployment.

- **Monitoring and Logging**
  - Set up **CloudWatch logs** for Lambda functions and API Gateway.
  - Implement alerts for critical issues or failures.
- **Team Collaboration**
  - Coordinate tasks among team members to ensure efficient progress.
  - Hold daily stand-ups to track progress and address blockers.

Conclusion

By focusing on integrating NEAR Protocol and using JavaScript, this development plan outlines a logical approach to building a secure, scalable application with role-based access control using NFTs. The punch-list tasks provide a clear roadmap for what needs to be accomplished during the Hackathon to advance the project effectively.