Deployment Architecture

> **Source File:** generated-documents\technical-design\deployment-architecture.md
> **Generated:** 08/07/2025 at 09:44:02
> **Generated by:** Requirements Gathering Agent - PDF Converter

DeploymentArchitecture

Generated by adpa-enterprise-framework-automation v3.1.6
Category: technical-design
Generated: 2025-07-05T17:05:00.544Z
Description:

Deployment Architecture Document: Self-Charging Electric Vehicles (SCEV) Project

1. Deployment Overview:

This document outlines the deployment architecture for the Self-Charging Electric Vehicles (SCEV) project. The deployment strategy will be phased, aligning with project milestones (M1-M4) and incorporating a continuous integration/continuous delivery (CI/CD) pipeline. Initial deployments will focus on testing and validation of individual components and subsystems before progressing to full vehicle integration and eventual mass production. The architecture emphasizes modularity, allowing for independent updates and scaling of individual energy harvesting systems.

2. Infrastructure Architecture:

The infrastructure will consist of several interconnected layers:

- **Development Environment:** This environment will host development tools, version control (Git), and build systems. Cloud-based solutions like Azure DevOps or GitHub Actions will be leveraged for CI/CD.
- **Testing Environment:** This environment will mirror the production environment but with scaled-down resources. It will be used for unit, integration, and system testing of individual components and the integrated system. Virtual machines or containerized environments (Docker, Kubernetes) will be employed for flexibility and reproducibility.
- **Staging Environment:** A near-production environment for final testing before release. This will allow for thorough testing in a production-like setting.
- **Production Environment:** This environment will support the deployment of the final SCEV system, including data collection, remote diagnostics, and over-the-air (OTA) updates. Cloud-based services will be considered for data analytics and remote management.

3. Environment Setup:

- **Development:** Utilizing a Git repository (e.g., GitHub, GitLab) for code management, a CI/CD pipeline (e.g., Azure DevOps, GitHub Actions) for automated builds and testing, and a container registry (e.g., Docker Hub, Azure Container Registry) for storing containerized components. Development tools will include IDEs, simulation software (e.g., MATLAB/Simulink), and testing frameworks (e.g., pytest, Jest).
- **Testing & Staging:** Virtual machines or containerized environments running on a cloud provider (e.g., AWS, Azure, GCP) will replicate the hardware and software stack of the production environment. This ensures consistent testing across environments.

- **Production:** A robust and scalable infrastructure, potentially utilizing cloud services for data storage, processing, and analytics. On-premise infrastructure might be considered for specific aspects depending on security and regulatory requirements.

4. Deployment Process:

The deployment process will follow a CI/CD pipeline:

1. **Code Commit:** Developers commit code changes to the Git repository.
2. **Automated Build:** The CI/CD pipeline automatically builds the code and runs unit tests.
3. **Integration Tests:** Integration tests are executed to verify the interaction between different components.
4. **System Tests:** System tests are performed in the testing environment to validate the complete system functionality.
5. **Deployment to Staging:** The tested code is deployed to the staging environment for final validation.
6. **Manual Approval:** A manual approval step is required before deploying to production.
7. **Deployment to Production:** The code is deployed to the production environment.
8. **Monitoring:** Continuous monitoring of the production environment to detect and address any issues.

5. Configuration Management:

Configuration management will be handled through a combination of:

- **Version Control:** All configuration files will be stored in the Git repository.
- **Configuration as Code:** Infrastructure as Code (IaC) tools like Terraform or Ansible will be used to manage infrastructure configurations.
- **Environment Variables:** Sensitive information (API keys, passwords) will be stored as environment variables.

- **Secrets Management:** A dedicated secrets management service (e.g., Azure Key Vault, AWS Secrets Manager) will be used to securely store and manage sensitive data.

6. Scaling Strategy:

Scaling will be addressed at multiple levels:

- **Horizontal Scaling:** The cloud-based infrastructure allows for horizontal scaling of compute resources to handle increased demand.
- **Vertical Scaling:** Individual components (e.g., the AI-powered EMU) can be scaled vertically by increasing their processing power.
- **Modular Design:** The modular design of the SCEV system allows for independent scaling of individual energy harvesting systems.

7. Monitoring Setup:

A comprehensive monitoring system will be implemented to track the performance and health of the SCEV system:

- **System Metrics:** Monitoring of key system metrics (CPU usage, memory consumption, network traffic, energy generation, battery levels).
- **Log Aggregation:** Centralized log aggregation and analysis for troubleshooting and performance optimization.
- **Alerting:** Real-time alerts for critical events (e.g., system failures, low battery levels).
- **Dashboards:** Interactive dashboards for visualizing system performance and identifying potential issues.

8. Backup and Recovery:

- **Regular Backups:** Regular backups of all critical data (configuration files, sensor data, system logs) will be performed.
- **Backup Storage:** Backups will be stored in a secure, geographically redundant location.
- **Recovery Procedures:** Documented procedures for restoring the system from backups in case of failure.

9. Disaster Recovery:

A disaster recovery plan will be developed to ensure business continuity in case of a major outage:

- **Failover Mechanisms:** Implementation of failover mechanisms to ensure high availability.
- **Geographic Redundancy:** Data centers and infrastructure will be geographically distributed to minimize the impact of regional disasters.
- **Disaster Recovery Drills:** Regular disaster recovery drills will be conducted to test the effectiveness of the plan.

10. Maintenance Procedures:

- **Regular Updates:** Regular software updates and patches will be deployed to address security vulnerabilities and improve performance.
- **Maintenance Windows:** Scheduled maintenance windows will be implemented to minimize disruption to users.
- **Rollback Strategy:** A rollback strategy will be in place to revert to a previous stable version if necessary.

Dependencies: This deployment architecture relies on several key dependencies, including the cloud provider's infrastructure services, CI/CD tools, monitoring tools, and the specific hardware and software components of the SCEV system itself. These dependencies will be clearly documented and managed throughout the project lifecycle. Security considerations will be paramount, with appropriate access controls, encryption, and security auditing implemented at all stages.
