# MERN API Deployment Architecture on OpenStack with GitLab CE CI/CD and Monitoring

## Overview

This document outlines the architectural design for deploying a MERN (MongoDB, Express.js, React, Node.js) API monolithic application to a Virtual Machine (VM) hosted on OpenStack, which is itself hosted on 3 Bare Metal servers. The deployment process is automated using a GitLab Community Edition (CE) Continuous Integration and Continuous Deployment (CI/CD) pipeline, targeting two environments: NON-PRD and PRD. Additionally, the architecture includes end-to-end monitoring using Prometheus and Grafana for both the MERN API and the GitLab CE infrastructure.

### Components

#### Bare Metal Servers

**Bare Metal 1:** Hosts the OpenStack environment for the NON-PRD environment.  
**Bare Metal 2 & 3:** Host the OpenStack environment for the PRD environment, with redundancy for high availability.

#### OpenStack VMs

**NON-PRD VM:**- Deployed on Bare Metal 1.  
- Runs the MERN API application.  
- Includes MongoDB, Express.js, React, and Node.js components.  
**PRD VMs:**- Two VMs for redundancy and load balancing.  
- Deployed on Bare Metal 2 & 3.  
- Each VM runs the MERN API application with the same components as NON-PRD.

#### GitLab Community Edition

**Source Code Management:** Stores the MERN API codebase.  
**CI/CD Pipeline:** Automates build, test, and deployment processes.  
**Deployment:**- Deploys to NON-PRD VM on Bare Metal 1.  
- Deploys to PRD VMs on Bare Metal 2 & 3.

#### Monitoring and Visualization

**Prometheus:**- Collects metrics from the MERN API applications and GitLab.  
- Deployed on a separate monitoring VM or integrated within the existing VMs.  
**Grafana:**- Visualizes metrics collected by Prometheus.  
- Provides dashboards for both NON-PRD and PRD environments.

### Data Flow and Monitoring

**MERN API:**- Handles requests and interacts with MongoDB for data storage.  
- Monitored by Prometheus for performance metrics.  
**GitLab:**- Monitored by Prometheus for CI/CD pipeline performance and health.  
**Grafana:**- Connects to Prometheus to display real-time metrics and historical data.

### Network and Connectivity

Ensure secure and efficient network connections between all components. Use load balancers for PRD VMs to distribute traffic evenly.

### Deployment Process

**1. Code Commit:** Developer commits code changes to the GitLab repository.  
**2. CI/CD Pipeline:** GitLab CE pipeline is triggered, automating build, test, and deployment processes.  
**3. Deployment to NON-PRD:** The pipeline deploys the application to the NON-PRD VM on Bare Metal 1.  
**4. Deployment to PRD:** The pipeline deploys the application to the PRD VMs on Bare Metal 2 & 3.  
**5. Monitoring:** Prometheus collects metrics from the MERN API and GitLab, and Grafana visualizes these metrics.

### Benefits

- Automated deployment to multiple environments using GitLab CE CI/CD pipeline.  
- End-to-end monitoring and visualization of the MERN API and GitLab infrastructure.  
- High availability and redundancy in the PRD environment.

### Conclusion

This architecture provides a robust and scalable solution for deploying a MERN API application on OpenStack, leveraging the power of GitLab CE for automation and Prometheus and Grafana for comprehensive monitoring and visualization.

Implementing security in a CI/CD pipeline for a MERN (MongoDB, Express.js, React, Node.js) API monolithic application involves several best practices to ensure that the application is secure throughout its development, testing, and deployment stages. Here are detailed suggestions for enhancing security:

**1. Secure Code Practices**

* **Code Reviews**: Implement mandatory code reviews to catch potential security vulnerabilities early in the development process.
* **Static Code Analysis**: Use tools like ESLint, SonarQube, or Snyk to automatically scan your codebase for security vulnerabilities and coding standard violations.
* **Dependency Management**: Regularly update dependencies and use tools like npm audit to identify and fix vulnerabilities in third-party libraries.

**2. CI/CD Pipeline Security**

* **Access Control**: Limit access to the CI/CD pipeline to only those who need it. Use role-based access control (RBAC) to manage permissions.
* **Secrets Management**: Use secure methods to manage secrets and environment variables, such as HashiCorp Vault, AWS Secrets Manager, or GitLab's built-in secret management.
* **Pipeline Isolation**: Run CI/CD jobs in isolated environments (e.g., containers) to prevent cross-job contamination.
* **Artifact Integrity**: Use checksums or digital signatures to verify the integrity of build artifacts before deployment.

**3. Environment Security**

* **Environment Segregation**: Separate development, testing, and production environments to minimize the risk of accidental data exposure or unauthorized access.
* **Network Security**: Implement network segmentation and use firewalls to control traffic between different parts of your infrastructure.
* **Secure Configuration**: Ensure that all environments are configured securely, following best practices for MongoDB, Node.js, and other components.

**4. Application Security**

* **Input Validation**: Implement strong input validation to prevent common attacks such as SQL injection and cross-site scripting (XSS).
* **Authentication and Authorization**: Use robust authentication mechanisms (e.g., OAuth, JWT) and ensure proper authorization checks are in place.
* **HTTPS Everywhere**: Use HTTPS to encrypt data in transit between clients and servers.

**5. Monitoring and Logging**

* **Security Monitoring**: Implement monitoring tools to detect and alert on suspicious activities. Tools like Prometheus and Grafana can be extended with security-focused plugins.
* **Log Management**: Centralize logs and use tools like ELK Stack (Elasticsearch, Logstash, Kibana) to analyze logs for security incidents.

**6. Regular Security Audits and Testing**

* **Penetration Testing**: Conduct regular penetration tests to identify and address vulnerabilities.
* **Vulnerability Scanning**: Use automated tools to regularly scan your application and infrastructure for vulnerabilities.
* **Security Audits**: Perform regular security audits to ensure compliance with security policies and standards.

**7. Incident Response Plan**

* **Incident Response**: Develop and maintain an incident response plan to quickly address and mitigate security incidents.
* **Backup and Recovery**: Implement regular backups and test recovery procedures to ensure data integrity and availability in case of a security breach.

By integrating these best practices into your CI/CD pipeline and overall development process, you can significantly enhance the security posture of your MERN API monolithic application.