# NodeJs Application Deployment Report

## Summary of Security Risks Identified

1. **Container Vulnerabilities**: Base images may contain known vulnerabilities.
2. **Privilege Escalation**: Running containers as root can lead to host compromise.
3. **Secrets Exposure**: Hardcoded secrets in code or images.
4. **Code Vulnerabilities**: Unpatched dependencies or insecure coding practices.
5. **Infrastructure Misconfigurations**: Weak IAM policies or exposed services.
6. **Runtime Attacks**: Lack of runtime protections against exploits.

## What Was Implemented and Why

### 1. Docker Hardening

- **Minimal Base Image**: Used Alpine Linux to reduce attack surface.
- **Multi-stage Builds**: Separated build and runtime stages to exclude build tools from final image.
- **Non-root User**: Created and used a non-privileged user to prevent privilege escalation.
- **Health Checks**: Added health checks for container monitoring.
- **Scanning**: Integrated Trivy for vulnerability scanning in CI/CD.

### 2. CI/CD Pipeline Security

- **Merged CI/CD Workflow**: Combined continuous integration and deployment into a single GitHub Actions workflow for streamlined automation.
- **Triggers**: Pipeline runs on pull requests to `main` and pushes to `main` and `feature/**` branches.
- **Static Code Analysis**: Used Semgrep to detect code vulnerabilities.
- **Container Scanning**: Trivy scans Dockerfile and built images for vulnerabilities.
- **Security Gates**: Pipeline fails on critical issues, preventing insecure deployments.
- **Artifact Upload**: Security scan results uploaded for review.
- **Deployment**: Automated deployment to AWS ECS with ECR, ensuring infrastructure is provisioned before image push.

### 3. Secrets Management

- **Environment Variables**: Secrets loaded from environment variables, not hardcoded.
- **Env File**: Provided .env for safe secret configuration.
- **Docker Compose**: Uses environment variables for MongoDB credentials.

### 4. Infrastructure Hardening

- **IaC with Terraform**: Defined infrastructure as code for AWS ECS deployment.
- **Least-Privilege IAM**: Created specific roles with minimal permissions.
- **Security Groups**: Restricted ingress/egress rules.
- **Logging**: Enabled CloudWatch logging for monitoring.

### 5. Runtime Security

- **Seccomp Profile**: Basic seccomp profile to restrict system calls.

## Suggestions for Production-Grade Hardening

1. **Advanced Scanning**: Integrate Snyk or other commercial scanners.
2. **Secrets Manager**: Use AWS Secrets Manager or HashiCorp Vault for production.
3. **Network Security**: Implement VPC, subnets, and NACLs properly.
4. **Monitoring**: Add comprehensive logging and alerting with tools like ELK stack.
5. **Compliance**: Regular audits and compliance checks (e.g., CIS benchmarks).
6. **Zero Trust**: Implement service mesh like Istio for microsegmentation.
7. **Backup and Recovery**: Implement automated backups and disaster recovery plans.
8. **Rate Limiting**: Add rate limiting to prevent DDoS attacks.
9. **Encryption**: Ensure all data in transit and at rest is encrypted.
10. **Regular Updates**: Keep dependencies and base images updated.

This implementation provides a solid foundation for NodeJs Application deployment, following industry best practices.