Container, Serverless, and Kubernetes Security Governance on AWS .

1. Problem Statement.

A cloud-native application platform leveraging containers, Kubernetes, and serverless services lacked unified security governance. Misconfigured container permissions and overly permissive execution roles introduced risk. A structured governance framework was required to enforce workload security and runtime controls.

I implemented this framework to ensure that container images, Kubernetes workloads, and Lambda functions operate securely, follow least-privilege principles, and prevent vulnerability exploitation and secrets exposure.

2. Objectives

The primary objectives were to:

- Secure container images before deployment
- Detect and remediate vulnerabilities in ECR
- Enforce least privilege for Lambda execution roles
- Prevent secrets leakage
- Enforce Kubernetes workload security standards
- Implement RBAC for container workloads
- Validate security controls through testing

3. Container Image Security (Amazon ECR)

3.1 Vulnerability Scanning

A private Amazon ECR repository was created to store container images.

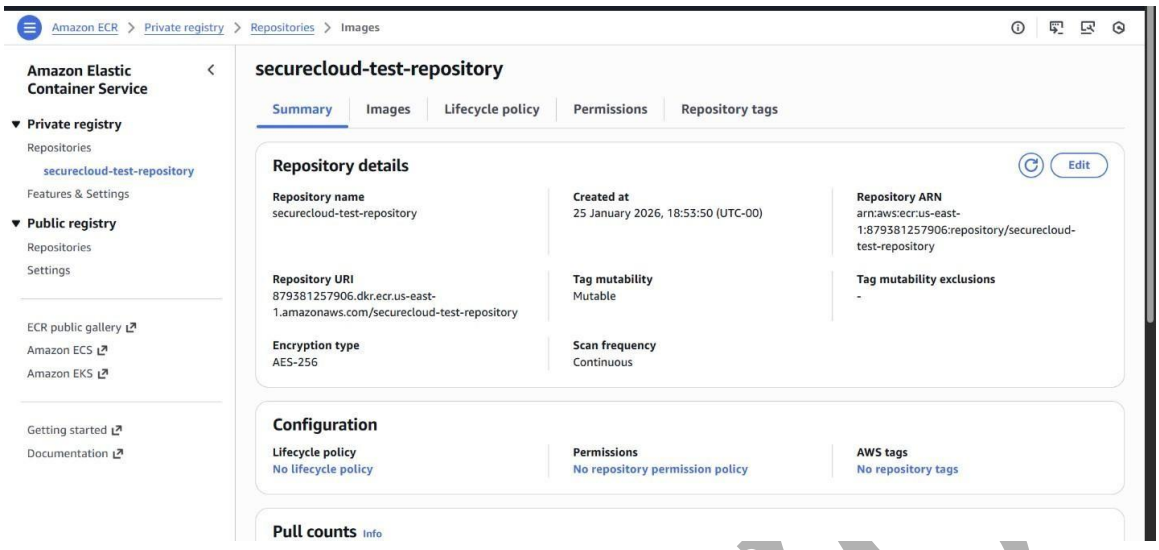


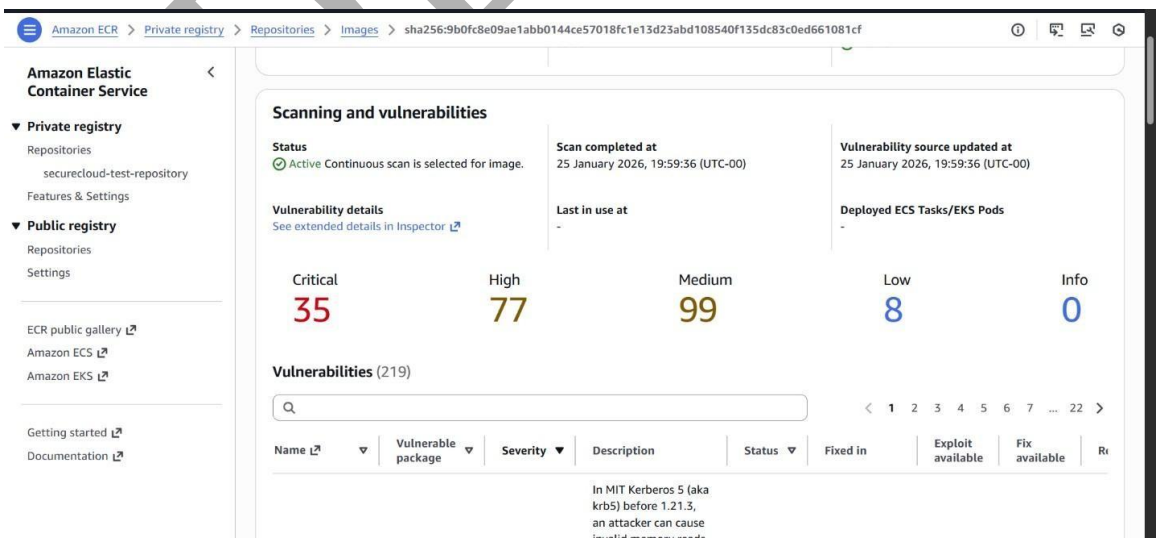
Image scanning on push was enabled and integrated with Amazon Inspector for continuous vulnerability assessments.

Images are automatically scanned for CVEs and categorized by severity.

3.2 Vulnerability Testing

A test image (nginx:1.18) was pushed to ECR and scanned.

Multiple vulnerabilities were detected and documented.

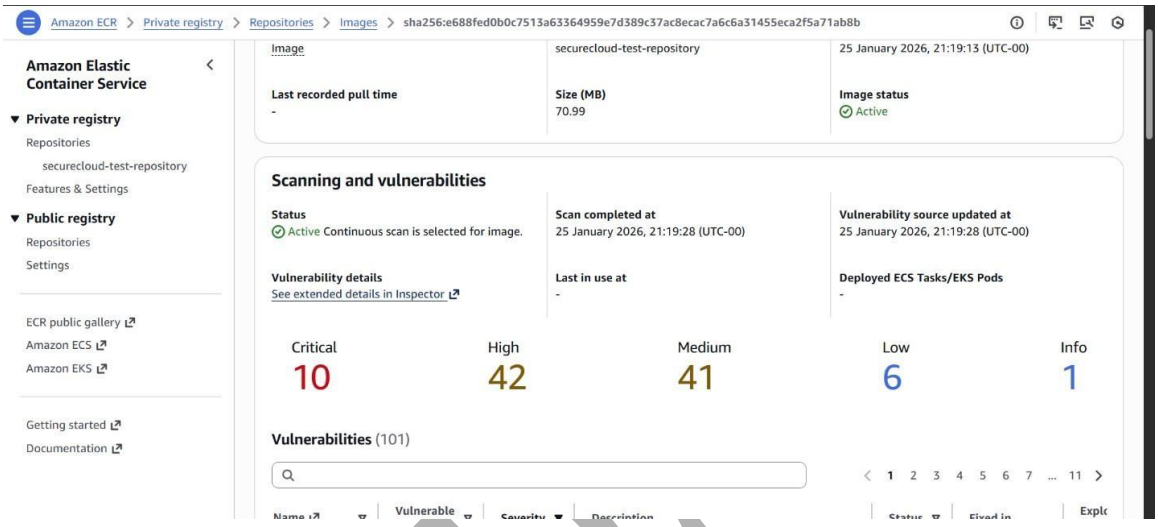


3.3 Remediation Process

To remediate vulnerabilities:

- Base images were upgraded (nginx:1.25)
- Containers were rebuilt
- Updated images were rescanned

Scan results confirmed reduced risk levels.



3.4 Secure Image Lifecycle

Only compliant images are approved for deployment.

Insecure images are blocked until remediation is completed.

4. Serverless Security (AWS Lambda)

4.1 Execution Role Review

Lambda execution roles were reviewed for:

- Wildcard permissions
- Overly broad managed policies
- Unnecessary service access

4.2 Least Privilege Enforcement

Broad permissions were removed and replaced with scoped policies.

Roles were limited to required services, actions, and resources.

Baseline logging permissions were retained.

4.3 Validation Testing

Functions were tested after permission reduction.

CloudWatch logs confirmed successful execution.

AccessDenied errors were used to validate permission boundaries.

5. Secrets Management and Leakage Prevention

5.1 Secure Secrets Storage

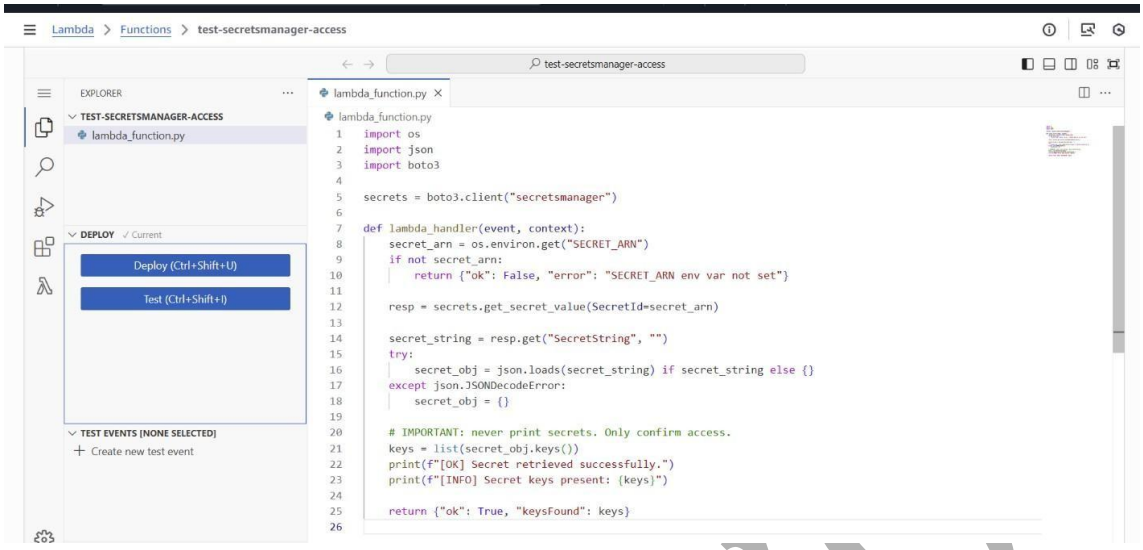
Sensitive credentials were migrated to AWS Secrets Manager.

Secrets were encrypted using KMS.

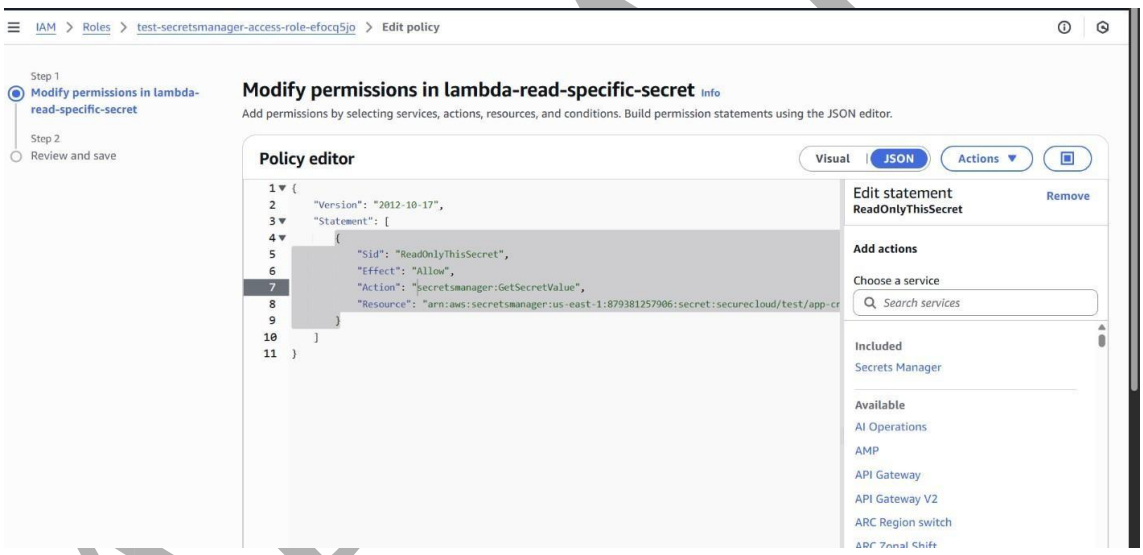
No plaintext secrets were stored in environment variables.

5.2 Secure Access Design

Lambda functions were configured with:

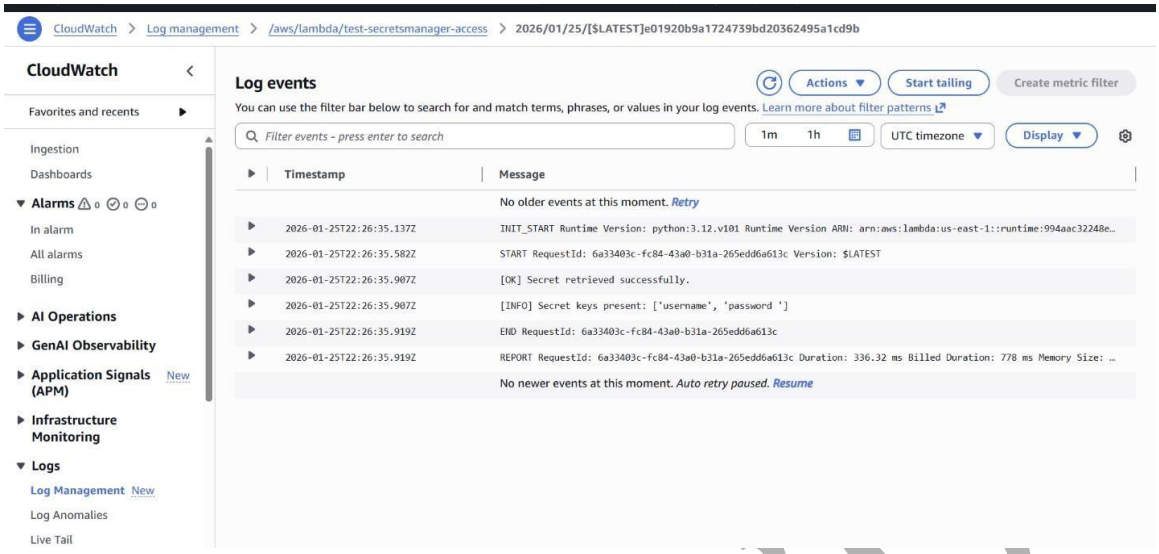


- secretsmanager:GetSecretValue permission
- Access scoped to specific secret ARNs



5.3 Validation

Secrets retrieval was tested successfully.



Permission removal resulted in expected failures.

This confirmed strict access control.

6. Kubernetes Workload Security (Amazon EKS)

6.1 Namespace Isolation

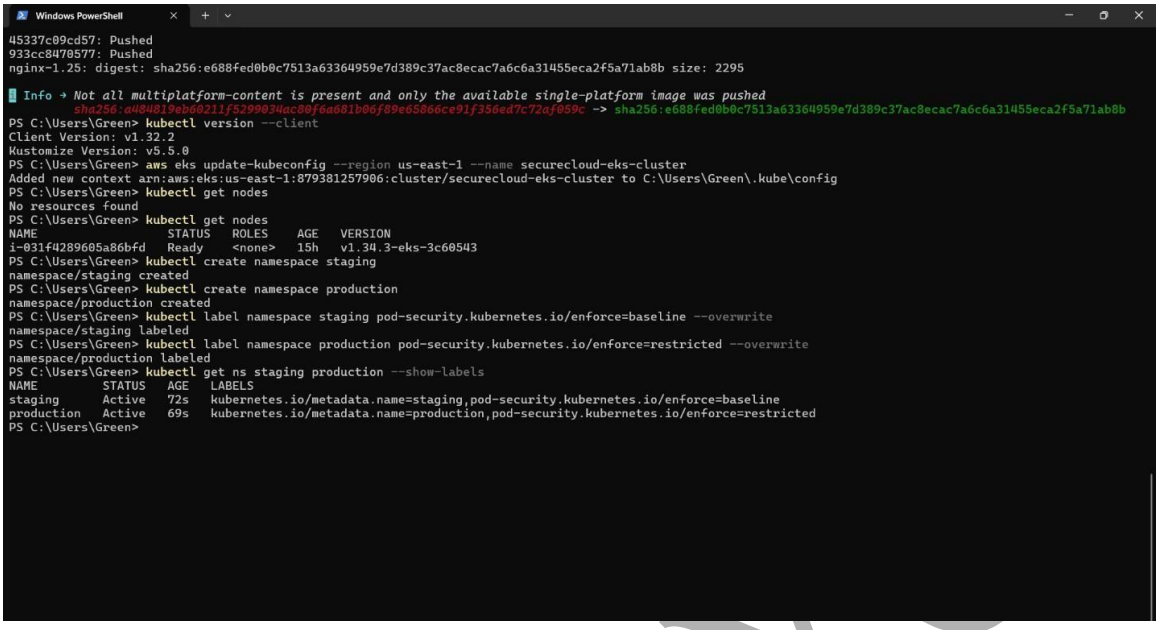
Separate namespaces were created for staging and production environments.

This prevents cross-environment access and supports tiered controls.

6.2 Pod Security Standards Enforcement

Pod Security Standards were enforced using namespace labels:

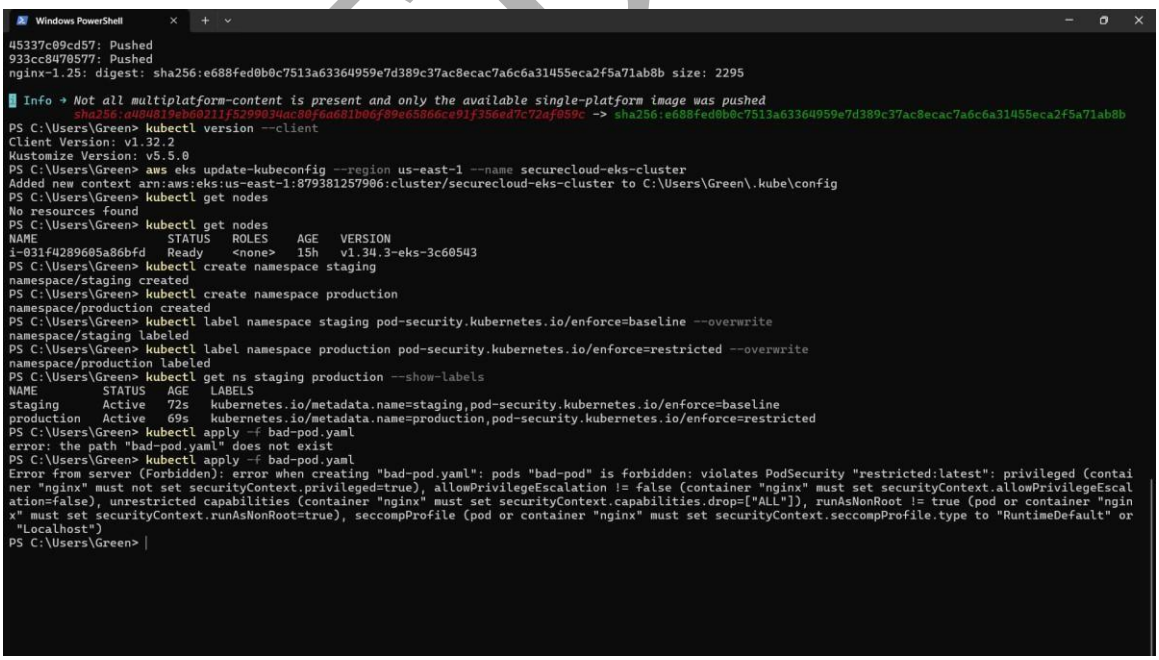
- Production: Restricted policy
- Staging: Baseline policy

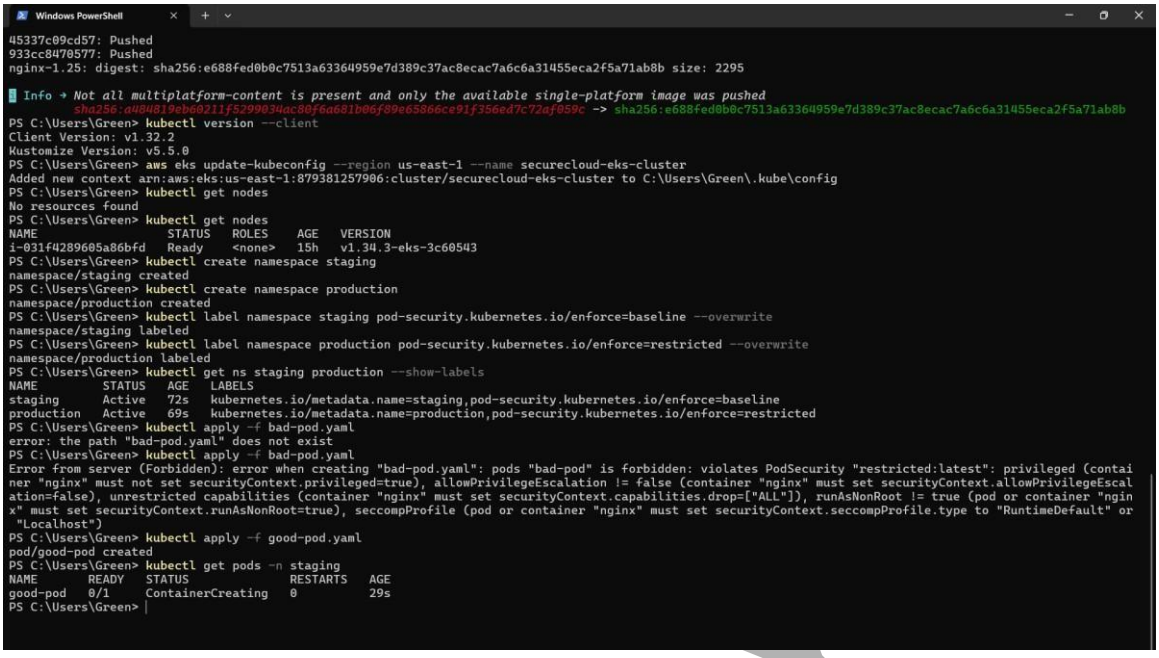


6.3 Policy Validation

Privileged containers were blocked in production.

Compliant workloads were permitted in staging.





This confirmed enforcement effectiveness.

7. Kubernetes Role-Based Access Control (RBAC)

7.1 RBAC Design

A dedicated service account (app-sa) was created.

Namespace-scoped roles were defined with read-only permissions.

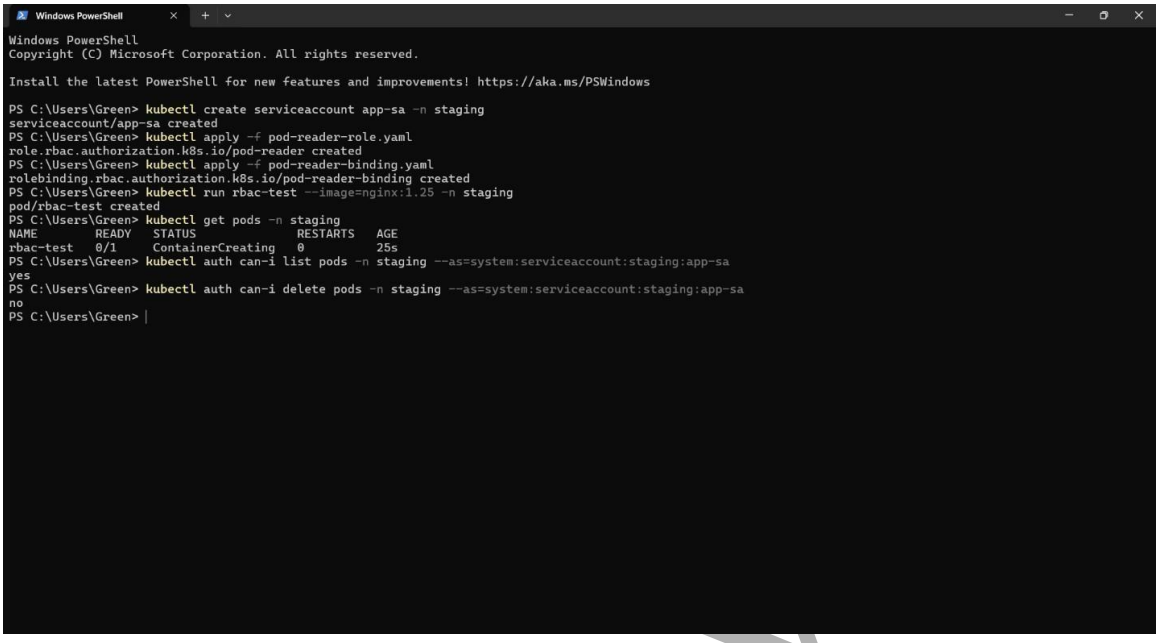
7.2 Role Binding

RoleBindings were configured to associate roles with service accounts.

7.3 Authorization Testing

The service account could list pods.

Access to delete pods and secrets was denied.



This validated least-privilege enforcement.

8. Governance and Operational Controls

All container, Lambda, and Kubernetes security controls were documented.

Standard operating procedures were established for:

- Image remediation
- Permission reviews
- Secrets rotation
- Policy updates

9. Testing and Validation Framework

Security controls were validated through:

- ECR vulnerability remediation testing
- Pod Security enforcement testing
- RBAC authorization testing
- Lambda permission testing
- Secrets access validation

All controls operated as intended.

10. Outcomes and Impact

This implementation delivered:

- Reduced container vulnerability exposure
- Secure serverless execution roles
- Protected secrets management
- Enforced Kubernetes security standards
- Strong workload isolation
- Improved cloud-native security posture

11. Conclusion

I designed and implemented an enterprise-grade security governance framework for containerized, serverless, and Kubernetes workloads on AWS.

Through integrated vulnerability management, identity controls, secrets protection, and runtime enforcement, this solution ensures secure and resilient cloud-native operations.