Enterprise Container, Serverless, and Kubernetes Security Governance on AWS
Author: Adedayo
Specialization: Cloud Security & Cloud-Native Security
Platform: Amazon Web Services (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

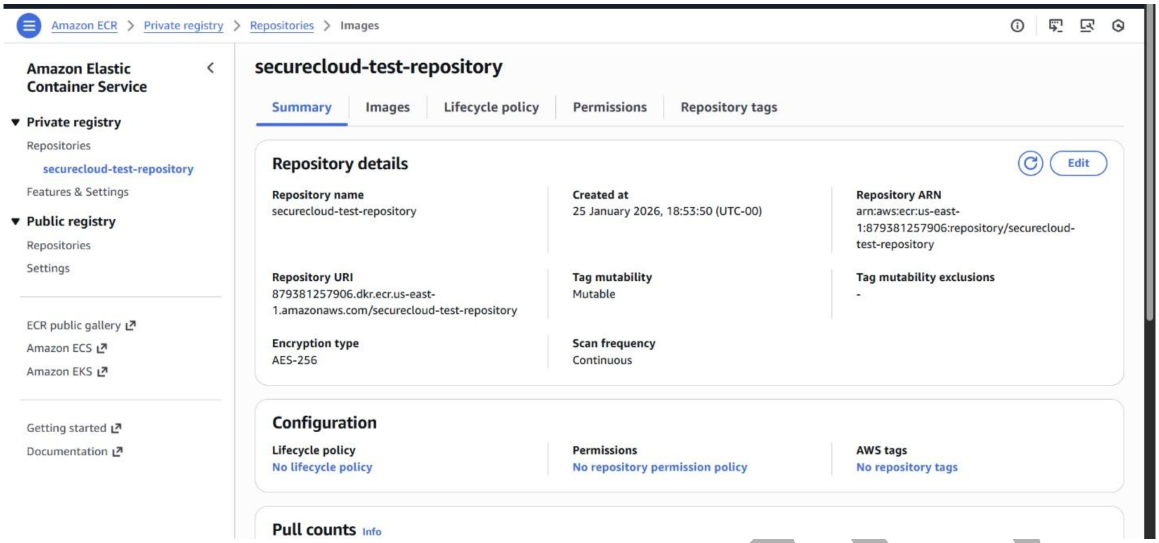
The primary objectives were to:

- 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.



Amazon ECR > Private registry > Repositories > Images

securecloud-test-repository

Repository details

Repository name securecloud-test-repository	Created at 25 January 2026, 18:53:50 (UTC-00)	Repository ARN arn:aws:ecr:us-east-1:879381257906:repository/securecloud-test-repository
Repository URI 879381257906.dkr.ecr.us-east-1.amazonaws.com/securecloud-test-repository	Tag mutability Mutable	Tag mutability exclusions -
Encryption type AES-256	Scan frequency Continuous	

Configuration

Lifecycle policy No lifecycle policy	Permissions No repository permission policy	AWS tags No repository tags

Pull counts Info

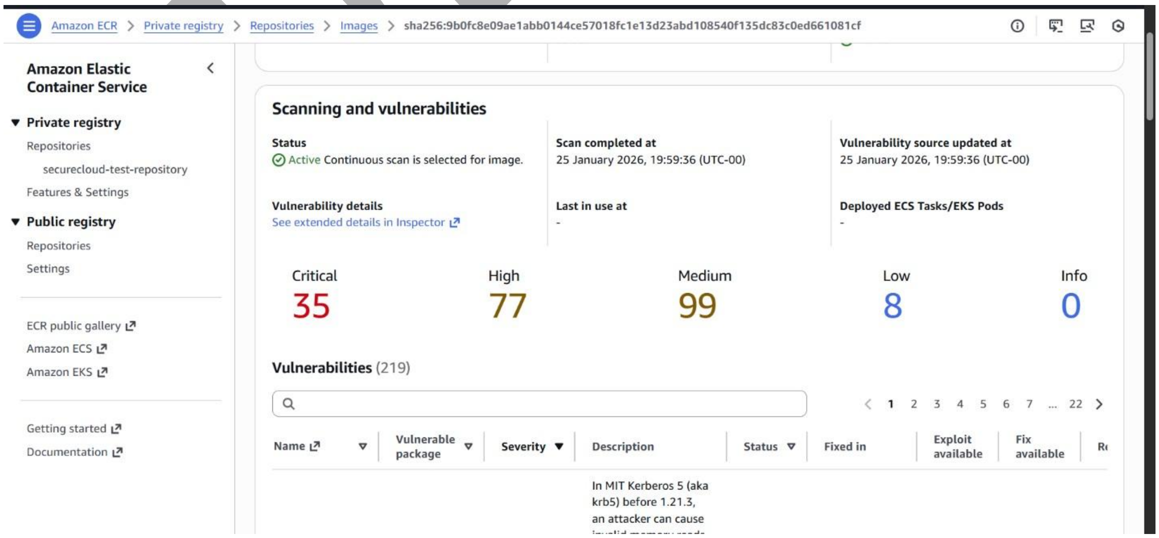
Image scanning on push was enabled and integrated with Amazon Inspector for continuous vulnerability assessments.

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.



Amazon ECR > Private registry > Repositories > Images > sha256:9b0fc8e09ae1abb0144ce57018fc1e13d23abd108540f135dc83c0ed661081cf

Scanning and vulnerabilities

Status Active Continuous scan is selected for image.	Scan completed at 25 January 2026, 19:59:36 (UTC-00)	Vulnerability source updated at 25 January 2026, 19:59:36 (UTC-00)
Vulnerability details See extended details in Inspector	Last in use at -	Deployed ECS Tasks/EKS Pods -

Critical	High	Medium	Low	Info
35	77	99	8	0

Vulnerabilities (219)

Name	Vulnerable package	Severity	Description	Status	Fixed in	Exploit available	Fix available	Re
			In MIT Kerberos 5 (aka krb5) before 1.21.3, an attacker can cause invalid memory reads					

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.

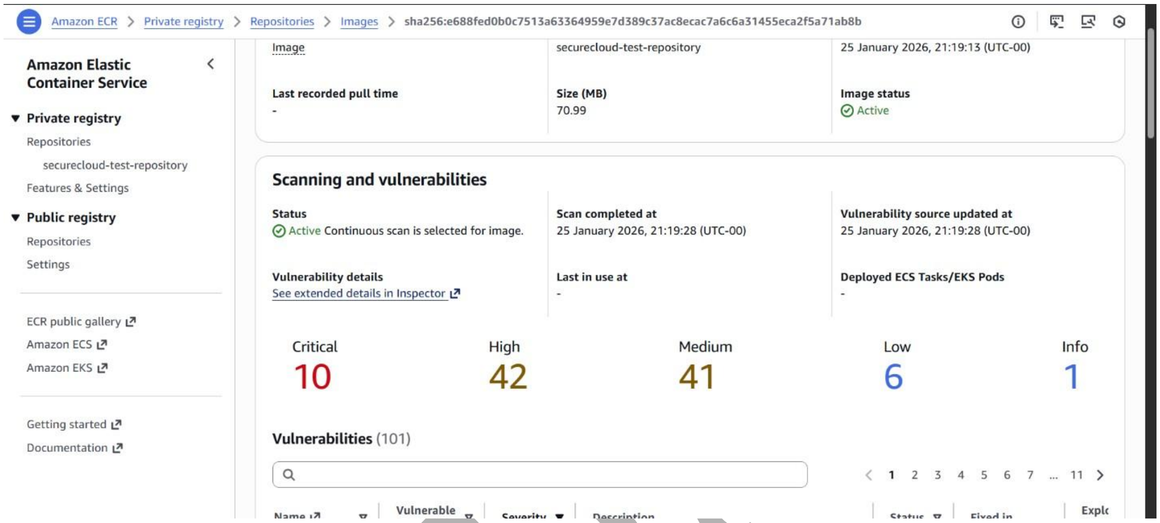

Amazon ECR > Private registry > Repositories > Images > sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b
Amazon Elastic Container Service
Private registry
Repositories
securecloud-test-repository
Features & Settings
Public registry
Repositories
Settings
ECR public gallery
Amazon ECS
Amazon EKS
Getting started
Documentation

Image: securecloud-test-repository — 25 January 2026, 21:19:13 (UTC-00)

Last recorded pull time: -

Size (MB): 70.99

Image status: Active

Scanning and vulnerabilities

Status: Active Continuous scan is selected for image.

Scan completed at: 25 January 2026, 21:19:28 (UTC-00)

Vulnerability source updated at: 25 January 2026, 21:19:28 (UTC-00)

Vulnerability details: See extended details in Inspector

Last in use at: -

Deployed ECS Tasks/EKS Pods: -

Critical	High	Medium	Low	Info
10	42	41	6	1

Vulnerabilities (101)

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

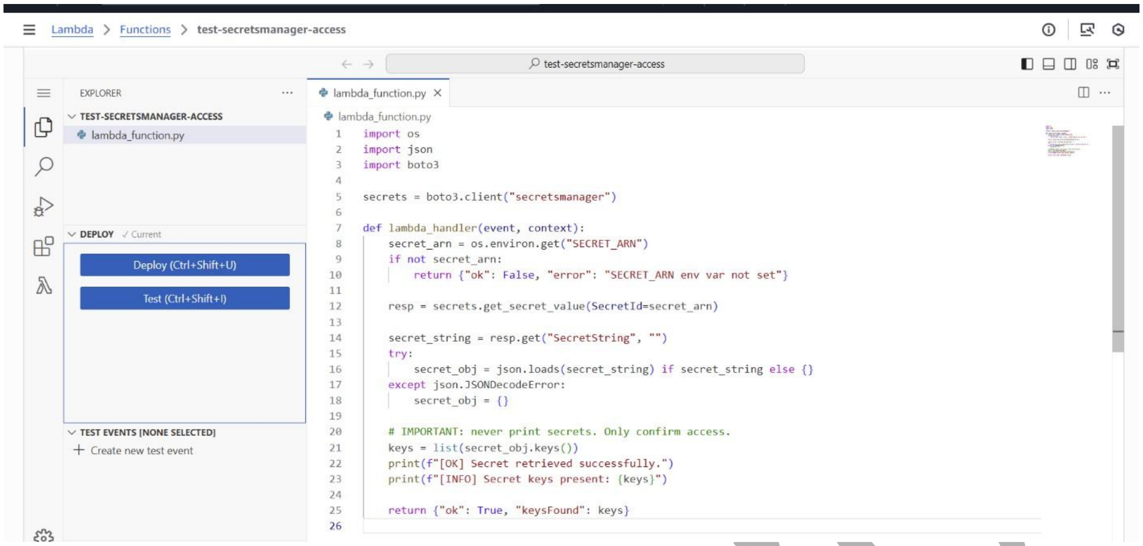
Sensitive credentials were migrated to AWS Secrets Manager.

Secrets were encrypted using KMS.

No plaintext secrets were stored in environment variables.

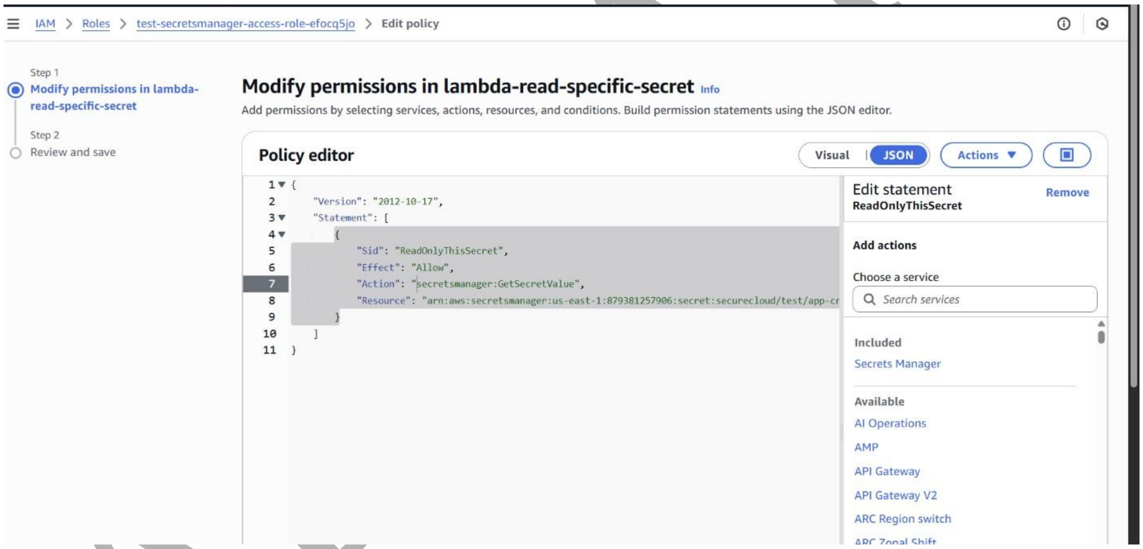
5.2 Secure Access Design

Lambda functions were configured with:


Lambda > Functions > test-secretsmanager-access
test-secretsmanager-access
EXPLORER
TEST-SECRETSMANAGER-ACCESS
lambda_function.py
DEPLOY ✓ Current
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
+ Create new test event
lambda_function.py

```python
import os
import json
import boto3

secrets = boto3.client("secretsmanager")

def lambda_handler(event, context):
    secret_arn = os.environ.get("SECRET_ARN")
    if not secret_arn:
        return {"ok": False, "error": "SECRET_ARN env var not set"}

    resp = secrets.get_secret_value(SecretId=secret_arn)

    secret_string = resp.get("SecretString", "")
    try:
        secret_obj = json.loads(secret_string) if secret_string else {}
    except json.JSONDecodeError:
        secret_obj = {}

    # IMPORTANT: never print secrets. Only confirm access.
    keys = list(secret_obj.keys())
    print(f"[OK] Secret retrieved successfully.")
    print(f"[INFO] Secret keys present: {keys}")

    return {"ok": True, "keysFound": keys}

```

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



Modify permissions in lambda-read-specific-secret

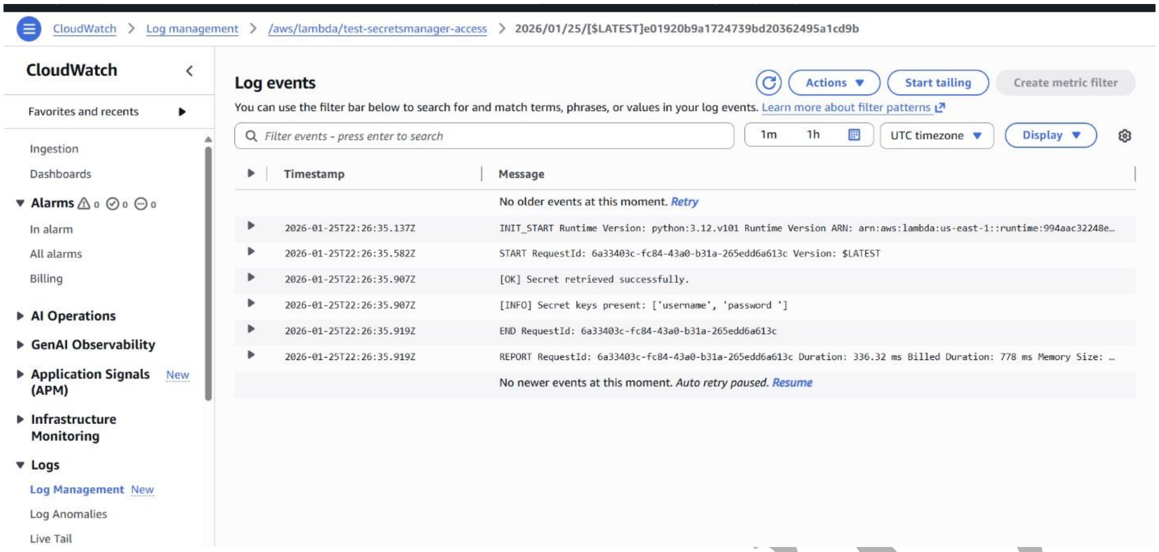
Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyThisSecret",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-east-1:879381257906:secret:securecloud/test/app-cr
        }
    ]
}
```

5.3 Validation

Secrets retrieval was tested successfully.



You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                   Message
                            No older events at this moment. Retry
2026-01-25T22:26:35.137Z    INIT_START Runtime Version: python:3.12.v101 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:994aac32248e…
2026-01-25T22:26:35.582Z    START RequestId: 6a33403c-fc84-43a0-b31a-265edd6a613c Version: $LATEST
2026-01-25T22:26:35.907Z    [OK] Secret retrieved successfully.
2026-01-25T22:26:35.907Z    [INFO] Secret keys present: ['username', 'password ']
2026-01-25T22:26:35.919Z    END RequestId: 6a33403c-fc84-43a0-b31a-265edd6a613c
2026-01-25T22:26:35.919Z    REPORT RequestId: 6a33403c-fc84-43a0-b31a-265edd6a613c Duration: 336.32 ms Billed Duration: 778 ms Memory Size: …
                            No newer events at this moment. Auto retry paused. Resume
```

Permission removal resulted in expected failures.

This confirmed strict access control.

6. Kubernetes Workload Security (Amazon EKS)

6.1 Namespace Isolation

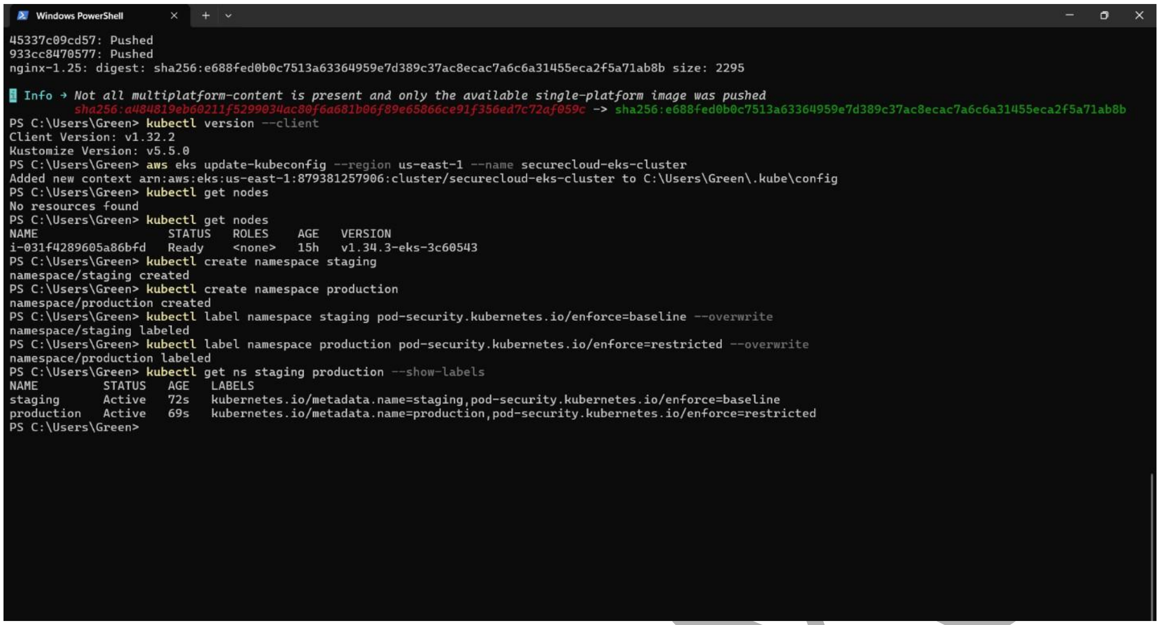
Separate namespaces were created for staging and production environments.

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

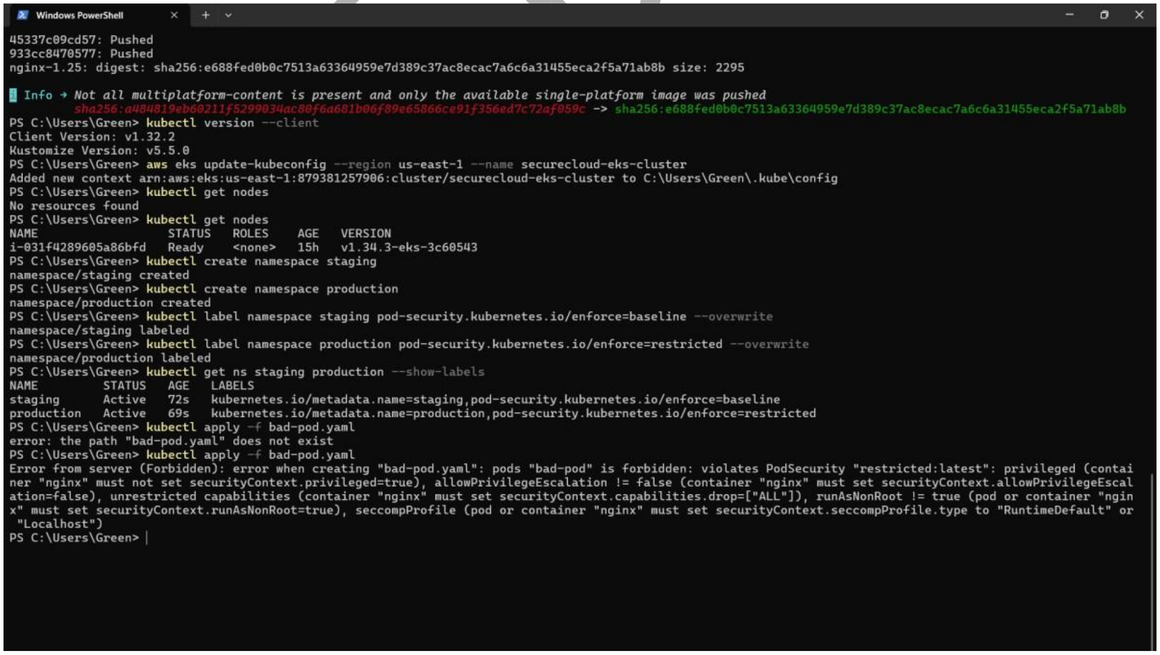


```
45337c09cd57: Pushed
933cc8470577: Pushed
nginx-1.25: digest: sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b size: 2295

i Info → Not all multiplatform-content is present and only the available single-platform image was pushed
        sha256:a484819eb60211f5299034ac80f6a681b06f89e65866ce91f356ed7c72af059c -> sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b
PS C:\Users\Green> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
PS C:\Users\Green> aws eks update-kubeconfig --region us-east-1 --name securecloud-eks-cluster
Added new context arn:aws:eks:us-east-1:879381257906:cluster/securecloud-eks-cluster to C:\Users\Green\.kube\config
PS C:\Users\Green> kubectl get nodes
No resources found
PS C:\Users\Green> kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
i-031f4289605a86bfd   Ready    <none>   15h   v1.34.3-eks-3c60543
PS C:\Users\Green> kubectl create namespace staging
namespace/staging created
PS C:\Users\Green> kubectl create namespace production
namespace/production created
PS C:\Users\Green> kubectl label namespace staging pod-security.kubernetes.io/enforce=baseline --overwrite
namespace/staging labeled
PS C:\Users\Green> kubectl label namespace production pod-security.kubernetes.io/enforce=restricted --overwrite
namespace/production labeled
PS C:\Users\Green> kubectl get ns staging production --show-labels
NAME         STATUS   AGE   LABELS
staging      Active   72s   kubernetes.io/metadata.name=staging,pod-security.kubernetes.io/enforce=baseline
production   Active   69s   kubernetes.io/metadata.name=production,pod-security.kubernetes.io/enforce=restricted
PS C:\Users\Green>
```

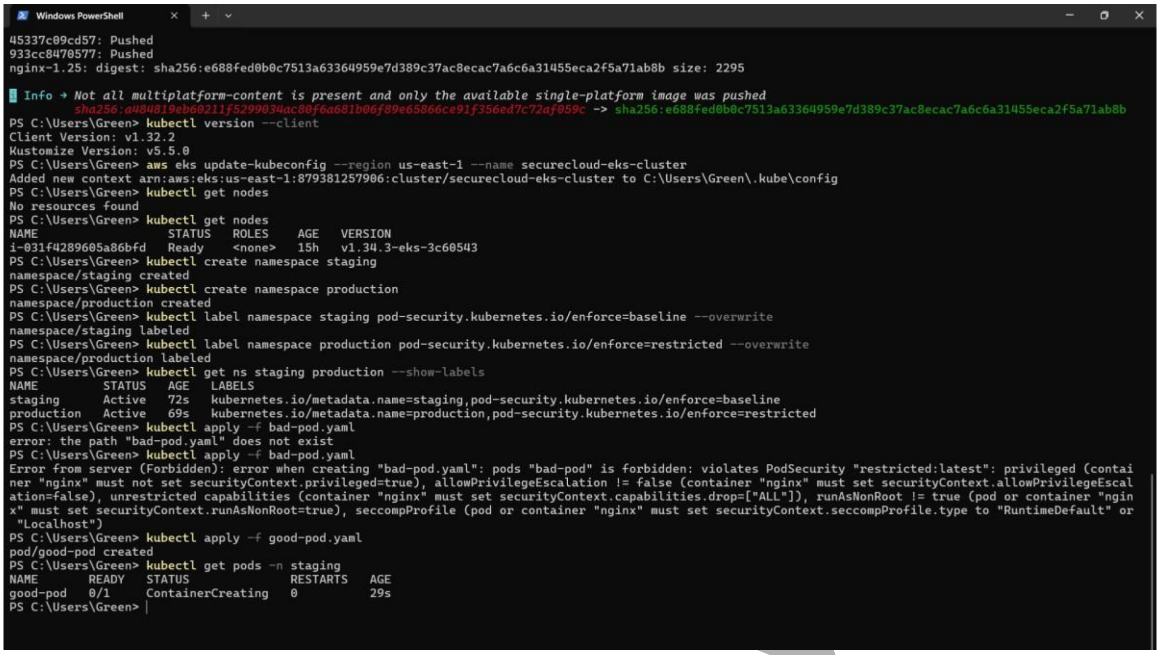
6.3 Policy Validation

Privileged containers were blocked in production.

Compliant workloads were permitted in staging.



```
45337c09cd57: Pushed
933cc8470577: Pushed
nginx-1.25: digest: sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b size: 2295

i Info → Not all multiplatform-content is present and only the available single-platform image was pushed
        sha256:a484819eb60211f5299034ac80f6a681b06f89e65866ce91f356ed7c72af059c -> sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b
PS C:\Users\Green> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
PS C:\Users\Green> aws eks update-kubeconfig --region us-east-1 --name securecloud-eks-cluster
Added new context arn:aws:eks:us-east-1:879381257906:cluster/securecloud-eks-cluster to C:\Users\Green\.kube\config
PS C:\Users\Green> kubectl get nodes
No resources found
PS C:\Users\Green> kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
i-031f4289605a86bfd   Ready    <none>   15h   v1.34.3-eks-3c60543
PS C:\Users\Green> kubectl create namespace staging
namespace/staging created
PS C:\Users\Green> kubectl create namespace production
namespace/production created
PS C:\Users\Green> kubectl label namespace staging pod-security.kubernetes.io/enforce=baseline --overwrite
namespace/staging labeled
PS C:\Users\Green> kubectl label namespace production pod-security.kubernetes.io/enforce=restricted --overwrite
namespace/production labeled
PS C:\Users\Green> kubectl get ns staging production --show-labels
NAME         STATUS   AGE   LABELS
staging      Active   72s   kubernetes.io/metadata.name=staging,pod-security.kubernetes.io/enforce=baseline
production   Active   69s   kubernetes.io/metadata.name=production,pod-security.kubernetes.io/enforce=restricted
PS C:\Users\Green> kubectl apply -f bad-pod.yaml
error: the path "bad-pod.yaml" does not exist
PS C:\Users\Green> kubectl apply -f bad-pod.yaml
Error from server (Forbidden): error when creating "bad-pod.yaml": pods "bad-pod" is forbidden: violates PodSecurity "restricted:latest": privileged (contai
ner "nginx" must not set securityContext.privileged=true), allowPrivilegeEscalation != false (container "nginx" must set securityContext.allowPrivilegeEscal
ation=false), unrestricted capabilities (container "nginx" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "ngin
x" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container "nginx" must set securityContext.seccompProfile.type to "RuntimeDefault" or
 "Localhost")
PS C:\Users\Green> |
```



```
45337c09cd57: Pushed
933cc8470577: Pushed
nginx-1.25: digest: sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b size: 2295

i Info → Not all multiplatform-content is present and only the available single-platform image was pushed
         sha256:a484819eb60211f5299034ac80f6a681b06f89e65866ce91f356ed7c72af059c -> sha256:e688fed0b0c7513a63364959e7d389c37ac8ecac7a6c6a31455eca2f5a71ab8b
PS C:\Users\Green> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
PS C:\Users\Green> aws eks update-kubeconfig --region us-east-1 --name securecloud-eks-cluster
Added new context arn:aws:eks:us-east-1:879381257906:cluster/securecloud-eks-cluster to C:\Users\Green\.kube\config
PS C:\Users\Green> kubectl get nodes
No resources found
PS C:\Users\Green> kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
i-031f4289605a86bfd   Ready    <none>   15h   v1.34.3-eks-3c60543
PS C:\Users\Green> kubectl create namespace staging
namespace/staging created
PS C:\Users\Green> kubectl create namespace production
namespace/production created
PS C:\Users\Green> kubectl label namespace staging pod-security.kubernetes.io/enforce=baseline --overwrite
namespace/staging labeled
PS C:\Users\Green> kubectl label namespace production pod-security.kubernetes.io/enforce=restricted --overwrite
namespace/production labeled
PS C:\Users\Green> kubectl get ns staging production --show-labels
NAME         STATUS   AGE   LABELS
staging      Active   72s   kubernetes.io/metadata.name=staging,pod-security.kubernetes.io/enforce=baseline
production   Active   69s   kubernetes.io/metadata.name=production,pod-security.kubernetes.io/enforce=restricted
PS C:\Users\Green> kubectl apply -f bad-pod.yaml
error: the path "bad-pod.yaml" does not exist
PS C:\Users\Green> kubectl apply -f bad-pod.yaml
Error from server (Forbidden): error when creating "bad-pod.yaml": pods "bad-pod" is forbidden: violates PodSecurity "restricted:latest": privileged (contai
ner "nginx" must not set securityContext.privileged=true), allowPrivilegeEscalation != false (container "nginx" must set securityContext.allowPrivilegeEscal
ation=false), unrestricted capabilities (container "nginx" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "ngin
x" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container "nginx" must set securityContext.seccompProfile.type to "RuntimeDefault" or
 "Localhost")
PS C:\Users\Green> kubectl apply -f good-pod.yaml
pod/good-pod created
PS C:\Users\Green> kubectl get pods -n staging
NAME       READY   STATUS              RESTARTS   AGE
good-pod   0/1     ContainerCreating   0          29s
PS C:\Users\Green> |
```

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.

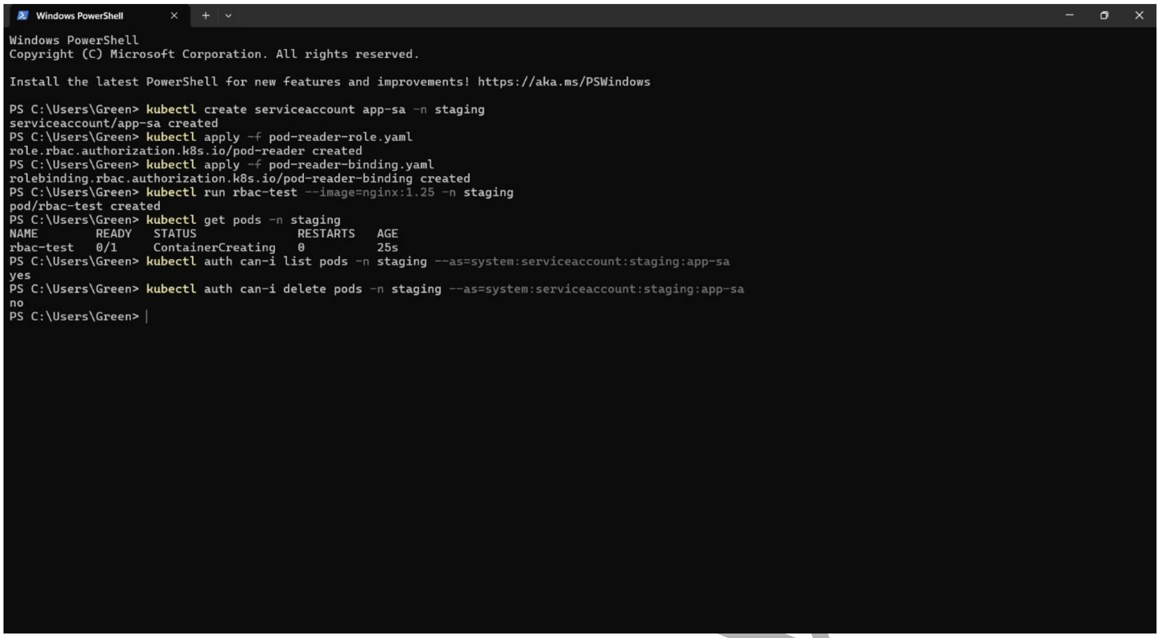
7.2 Role Binding

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.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Green> kubectl create serviceaccount app-sa -n staging
serviceaccount/app-sa created
PS C:\Users\Green> kubectl apply -f pod-reader-role.yaml
role.rbac.authorization.k8s.io/pod-reader created
PS C:\Users\Green> kubectl apply -f pod-reader-binding.yaml
rolebinding.rbac.authorization.k8s.io/pod-reader-binding created
PS C:\Users\Green> kubectl run rbac-test --image=nginx:1.25 -n staging
pod/rbac-test created
PS C:\Users\Green> kubectl get pods -n staging
NAME        READY   STATUS              RESTARTS   AGE
rbac-test   0/1     ContainerCreating   0          25s
PS C:\Users\Green> kubectl auth can-i list pods -n staging --as=system:serviceaccount:staging:app-sa
yes
PS C:\Users\Green> kubectl auth can-i delete pods -n staging --as=system:serviceaccount:staging:app-sa
no
PS C:\Users\Green> |
```

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.