eks

¡Por supuesto! A continuación, te proporcionaré una **guía completa** y bien estructurada sobre cómo implementar una pasarela de pagos en **EKS con Fargate**, cumpliendo con los requisitos de **PCI DSS**. A lo largo de esta guía, abordaré desde los fundamentos de Kubernetes y EKS hasta configuraciones avanzadas de seguridad, control de recursos y escalabilidad.

Manual Extendido: Implementación de una Pasarela de Pagos en EKS con Fargate Cumpliendo PCI DSS

Introducción

Este manual está diseñado para guiarte en la implementación de una **pasarela de pagos** en **AWS EKS con Fargate**, mientras cumples con los exigentes requisitos de **PCI DSS** (Payment Card Industry Data Security Standard) y sigues las **mejores prácticas de AWS**. Veremos cómo construir una infraestructura que sea:

- **Segura**: Usando cifrado TLS 1.2+, grupos de seguridad estrictos y otras herramientas de protección de AWS.
- **Escalable**: Configurando Kubernetes para escalar automáticamente con **Horizontal Pod Autoscaler (HPA)**.
- **Optimizada en costos**: Implementando control de recursos para evitar gastos innecesarios.

Sección 1: Comprendiendo AWS EKS y Fargate

1.1 ¿Qué es AWS EKS?

Amazon Elastic Kubernetes Service (EKS) es un servicio gestionado que permite ejecutar aplicaciones basadas en contenedores utilizando Kubernetes sin necesidad de gestionar los nodos directamente. **Kubernetes** es una plataforma que permite la automatización de la implementación, la escalabilidad y la operación de aplicaciones en contenedores.

1.2 ¿Qué es AWS Fargate?

Fargate es un servicio de AWS que permite ejecutar contenedores sin tener que gestionar servidores o infraestructura subyacente. Cuando usas **Fargate** con **EKS**, los pods de Kubernetes se ejecutan en un entorno "serverless", donde no necesitas gestionar ni aprovisionar nodos.

1.3 Configuración de EKS con Fargate

Cuando configuras un clúster de EKS, puedes optar por ejecutar tus pods en **nodos EC2** o en **Fargate**. En este caso, usarás Fargate para tu pasarela de pagos.

Perfil de Fargate en CloudFormation

Aquí tienes un ejemplo de configuración para un perfil de Fargate que se usará en el namespace `mi-app`:

```yaml
Resources:
  EKSCluster:
    Type: AWS::EKS::Cluster
    Properties:
      Name: MyPaymentCluster
      RoleArn: !GetAtt EKSClusterRole.Arn
      ResourcesVpcConfig:
        SubnetIds:
          - !Ref PublicSubnet1
          - !Ref PrivateSubnet1
          - !Ref PrivateSubnet2
      Version: "1.21"

  FargateProfile:
    Type: AWS::EKS::FargateProfile
    Properties:
      ClusterName: !Ref EKSCluster
      FargateProfileName: payments-fargate-profile
      PodExecutionRoleArn: !GetAtt PodExecutionRole.Arn
      Subnets:
        - !Ref PrivateSubnet1
        - !Ref PrivateSubnet2
      Selectors:
        - Namespace: mi-app
```

Este perfil asegura que cualquier pod creado en el namespace `mi-app` sea ejecutado en **Fargate**.

Sección 2: Fundamentos de Kubernetes

2.1 Namespaces y Pods

Un **namespace** en Kubernetes permite **aislar recursos** dentro del clúster. Por ejemplo, puedes crear un namespace llamado `mi-app` para todos los recursos relacionados con tu pasarela de pagos. Un **pod** es la unidad básica de Kubernetes que ejecuta uno o varios contenedores. Para gestionar pods de forma eficiente y escalable, se usan **deployments**.

Crear un Namespace

Para crear un namespace en Kubernetes, usa el siguiente comando:

```
kubectl create namespace mi-app
```

Ejemplo de Deployment

Este es un ejemplo de **deployment** para tu pasarela de pagos:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payments-api
  namespace: mi-app
spec:
  replicas: 3  # Número de pods
  selector:
    matchLabels:
      app: payments-api
  template:
    metadata:
      labels:
        app: payments-api
    spec:
      containers:
      - name: payments-container
        image: your-payment-app-image:latest
        ports:
        - containerPort: 8080
```

Sección 3: Control de Recursos y Escalabilidad

3.1 ResourceQuota y LimitRange

ResourceQuota y **LimitRange** son dos mecanismos que Kubernetes utiliza para controlar los recursos de CPU y memoria utilizados por los pods en un namespace.

- **ResourceQuota**: Define límites totales de CPU y memoria en un namespace.
- **LimitRange**: Define límites de CPU y memoria por cada pod o contenedor en el namespace.

Ejemplo de ResourceQuota

El siguiente ejemplo establece una cuota máxima de recursos para el namespace `mi-app`:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: fargate-quota
  namespace: mi-app
spec:
  hard:
    requests.cpu: "4"
    requests.memory: "8Gi"
    limits.cpu: "8"
    limits.memory: "16Gi"
```

Ejemplo de LimitRange

Este ejemplo establece los límites por pod:

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: fargate-limits
  namespace: mi-app
spec:
  limits:
    - default:
        cpu: "500m"
        memory: "512Mi"
      defaultRequest:
        cpu: "200m"
        memory: "256Mi"
      type: Container
```

3.2 Horizontal Pod Autoscaler (HPA)

El **HPA** ajusta automáticamente el número de réplicas de pods en función de la demanda de recursos como CPU o memoria. Esto permite que la aplicación escale automáticamente según las necesidades.

Ejemplo de Configuración de HPA

Aquí te muestro cómo configurar el HPA para tu pasarela de pagos:

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: payments-hpa
  namespace: mi-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: payments-api
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

Sección 4: Seguridad y Cumplimiento PCI DSS

4.1 Cifrado de Datos en Tránsito: TLS 1.2+

Para cumplir con PCI DSS, es obligatorio que toda la comunicación entre los usuarios y la pasarela de pagos esté cifrada utilizando **TLS 1.2 o superior**.

Configuración de TLS con ALB en Kubernetes

Aquí te muestro cómo configurar un **ALB** para usar TLS 1.2:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: payments-service
  namespace: mi-app
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:region:account-
id:certificate/certificate-id  # Certificado SSL
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 443  # HTTPS
      targetPort: 8080
  selector:
    app: payments-api
```

4.2 Cifrado de Datos en Reposo

Los datos almacenados, como las transacciones de pago, deben estar cifrados. AWS proporciona **KMS (Key Management Service)** para cifrar todos los datos en reposo.

```yaml
S3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: payments-data
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: aws:kms  # Cifrado con AWS KMS
```

4.3 Monitoreo y Auditoría con CloudTrail

AWS CloudTrail monitorea y registra todas las acciones realizadas en tu cuenta de AWS. Esto es obligatorio para cumplir con PCI DSS.

```
aws cloudtrail create-trail --name MyTrail --s3-bucket-name MyLogsBucket
aws cloudtrail start-logging --name MyTrail
```

4.4 Protección con AWS WAF

Para proteger tu aplicación de ataques como inyección SQL o XSS, debes implementar un **Web Application Firewall (WAF)**. Aquí te muestro cómo configurarlo:

```
WAFWebACL:
  Type: AWS::WAFv2::WebACL
  Properties:
    DefaultAction:
      Allow: {}
    Scope: REGIONAL
    Rules:
      - Name: SQLInjectionRule
        Priority: 1
        Statement:
          SqliMatchStatement:
            FieldToMatch:
              UriPath: {}
            TextTransformations:
              - Priority: 0
                Type: NONE
        Action:
          Block: {}
    VisibilityConfig:
      SampledRequests

Enabled: true
      CloudWatchMetricsEnabled: true
      MetricName: WebACL
```

Conclusión

Este manual cubre los aspectos más importantes de la implementación de una **pasarela de pagos en AWS EKS con Fargate**, garantizando el cumplimiento con **PCI DSS** y optimizando los recursos para una solución escalable y segura. Siguiendo las configuraciones recomendadas, tu infraestructura estará protegida y lista para manejar pagos de manera eficiente y conforme a los estándares de seguridad más exigentes.

Este enfoque modular te permitirá ajustar los recursos, la seguridad y las capacidades de escalado según las necesidades de tu aplicación de pagos. ¡Buena suerte en la implementación de tu pasarela de pagos!