Estrategia de Despliegue y Monitoreo Continuo para la Plataforma PortTrack

Alumno: César Herrera

Introducción y Contexto del Proyecto

PortTrack es una plataforma de navegación portuaria diseñada para monitorear y gestionar el flujo de embarcaciones en puertos comerciales, mejorando la eficiencia y seguridad mediante la gestión de inventarios, carga, personal y rutas en tiempo real.

Estrategia de Despliegue Continuo

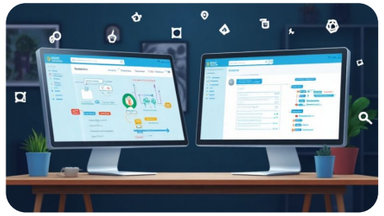
Selección del tipo de despliegue

Para la plataforma PortTrack, he seleccionado la estrategia Blue-Green como estrategia principal por las siguientes razones técnicas y operacionales:

- Zero Downtime Crítico: Las operaciones portuarias no pueden tolerar interrupciones. Un puerto comercial maneja barcos, carga y personal las 24 horas
- Rollback Instantáneo: Permite revertir inmediatamente a la versión estable ante cualquier problema
- Validación Completa: Posibilita pruebas exhaustivas en el ambiente verde antes del switch de tráfico
- Separación de Riesgos: Mantiene un ambiente de producción completamente funcional mientras se valida la nueva versión

```yaml
# Implementacion en Kubernetes
apiVersion: apps/v1
kind: Deployment
metadata:
  name: porttrack-blue
  labels:
    app: porttrack
    version: blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: porttrack
      version: blue
  template:
    metadata:
      labels:
        app: porttrack
        version: blue
    spec:
      containers:
        - name: porttrack
          image: porttrack:v1.0-blue
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: porttrack-service
spec:
  selector:
    app: porttrack
    version: blue  # Switch to green during deployment
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Estrategia de Despliegue Continuo



Herramientas CI/CD recomendadas

GitHub Actions como Orquestador Principal:

He seleccionado GitHub Actions como herramienta principal por:

- Integración Nativa: Perfecta integración con el repositorio de código fuente
- Costo-Efectividad: Modelo de pago por uso sin infraestructura propia que mantener
- Marketplace Extensivo: Más de 10,000 acciones predefinidas disponibles
- Facilidad de Configuración: Sintaxis YAML declarativa y versionada

AWS CodeDeploy para Despliegues en AWS:

- Complementamos con AWS CodeDeploy para:
- Despliegues Blue-Green Nativos: Soporte nativo para estrategias avanzadas
- Integración con Auto Scaling: Manejo automático de grupos de instancias
- Rollback Automático: Capacidades de rollback basadas en métricas

```yaml
# Pipeline propuesta
name: PortTrack Blue-Green Deployment

on:
  push:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run Tests
        run: |
          npm install
          npm run test:unit
          npm run test:integration

  security-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Container Security Scan
        uses: aquasecurity/trivy-action@master

  build-and-deploy:
    needs: [test, security-scan]
    runs-on: ubuntu-latest
    steps:
      - name: Build Docker Image
        run: |
          docker build -t porttrack:${{ github.sha }} .

      - name: Deploy to Green Environment
        run: |
          aws deploy create-deployment \
            --application-name porttrack \
            --deployment-group-name green-environment \
            --github-location repository=${{ github.repository }},commitId=${{ github.sha }}

      - name: Health Check Green Environment
        run: |
          ./scripts/health-check.sh green-environment

      - name: Switch Traffic to Green
        if: success()
        run: |
          ./scripts/switch-traffic.sh blue-to-green
```

Estrategia de Despliegue Continuo

Estrategias de rollback

Se Recomienda implementar rollback mediante versionado de contenedores Docker, permitiendo revertir rápidamente a versiones estables para mantener la operatividad sin interrupciones.

Rollback Automático:

- Monitoring Continuo: Métricas de salud durante los primeros 10 minutos post-despliegue
- Thresholds Automáticos: Error rate > 1%, latency > 500ms, o availability < 99.9%
- Rollback Instantáneo: Switch automático de vuelta al ambiente blue

Procedimiento de Rollback Manual:

```
# Rollback inmediato en caso de emergencia
kubectl patch service porttrack-service -p '{"spec":
{"selector":{"version":"blue"}}}'

# Verificación de rollback exitoso
kubectl get endpoints porttrack-service
```

Configuración de Entornos y Seguridad en Despliegues

Definición de entornos

DEV, STAGING, TEST y PRODUCCIÓN**: Para la plataforma PortTrack recomienda utilizar entornos diferenciados para un desarrollo controlado: DEV para desarrollo y pruebas iniciales, STAGING para validación previa a producción, TEST para pruebas automatizadas, y PRODUCCIÓN para operaciones en tiempo real con controles específicos.

Ambiente	Propósito	Configuración	Acceso
DEV	Desarrollo activo	1 replica, DB compartida	Desarrolladores
TEST	Pruebas automatizadas	2 replicas, DB dedicada	QA + CI/CD
STAGING	Validación pre-producción	3 replicas, mirror prod	Stakeholders
PROD	Operaciones en vivo	5+ replicas, HA setup	Usuarios finales

Configuración de Entornos y Seguridad en Despliegues

Gestión de credenciales y secretos con AWS Secrets Manager

Se recomienda emplear AWS Secrets Manager para almacenar y gestionar credenciales y secretos críticos, asegurando rotación automática, acceso basado en roles y facilitando auditorías para proteger información sensible de la plataforma.

Separación de Secretos por Ambiente:

- DEV: Credenciales de desarrollo sin cifrado especial
- TEST: Credenciales temporales con rotación semanal
- STAGING: Credenciales similares a producción pero separadas
- PROD: Credenciales altamente protegidas con rotación automática

Configuración de Entornos y Seguridad en Despliegues

Seguridad en pipeline

Escaneo de vulnerabilidades: Integrar escaneos automáticos en el pipeline CI/CD para detectar vulnerabilidades en código e imágenes, bloqueando despliegues críticos y asegurando la integridad y autenticidad antes del despliegue en producción.

Consideraciones de Seguridad en Pipeline

Security Gates:

- Static Code Analysis: integración SonarQube
- Dependency Scanning: Snyk para vulnerabilidades en dependencias
- Container Scanning: Trivy para escaneo de imágenes Docker
- Secret Detection: GitLeaks para prevenir commit de credenciales
- Infrastructure Security: tfsec para validación de IaC

Implementación de Monitoreo Continuo

Herramientas seleccionadas para monitoreo

Se eligieron Prometheus para la recolección y almacenamiento de métricas en tiempo real, y Grafana para la visualización mediante dashboards personalizables que supervisan la salud y rendimiento de la plataforma.

```yaml
# Prometheus configuration
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'porttrack-app'
    kubernetes_sd_configs:
    - role: pod
      namespaces:
        names: ['production', 'staging']
    relabel_configs:
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
      action: keep
      regex: true
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_port]
      target_label: __address__
      regex: (.+)
      replacement: $1:9090

  - job_name: 'kubernetes-nodes'
    kubernetes_sd_configs:
    - role: node
    relabel_configs:
    - action: labelmap
      regex: __meta_kubernetes_node_label_(.+)
```

Implementación de Monitoreo Continuo

Estrategia de manejo de logs y métricas

Se centralizan los logs y métricas con el stack ELK para detectar patrones anómalos en tiempo real, facilitando la trazabilidad y auditoría de eventos críticos en la plataforma portuaria.

```yaml
# Docker Compose configuration for ELK Stack
version: '3.8'
▷Run All Services
services:
  ▷Run Service
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.15.0
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms2g -Xmx2g"
      - xpack.security.enabled=true
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"

  ▷Run Service
  logstash:
    image: docker.elastic.co/logstash/logstash:8.15.0
    volumes:
      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf:ro
    ports:
      - "5044:5044"
    depends_on:
      - elasticsearch

  ▷Run Service
  kibana:
    image: docker.elastic.co/kibana/kibana:8.15.0
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch
```

Arquitectura de Logging

```
Containers → Filebeat → Logstash → Elasticsearch → Kibana
                          ↓
                  Log Processing & Enrichment
```

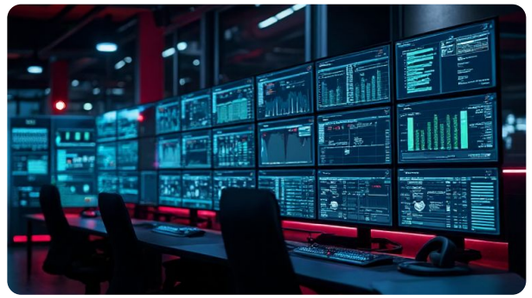
Implementación de Monitoreo Continuo

Estrategia de manejo de logs y métricas

```
# Logstash configuration for PortTrack

input {
  beats {
    port => 5044
  }
}

filter {
  if [kubernetes][container][name] == "porttrack" {
    grok {
      match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:level} %{GREEDYDATA:message}" }
    }

    if [level] == "ERROR" {
      mutate {
        add_tag => ["alert"]
      }
    }
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "porttrack-logs-%{+YYYY.MM.dd}"
  }
}
```

Categoría	Métrica	Threshold	Acción
Aplicación	Response Time	> 500ms	Alert
Aplicación	Error Rate	> 1%	Alert + Rollback
Infraestructura	CPU Usage	> 80%	Scale Up
Infraestructura	Memory Usage	> 85%	Scale Up
Base de Datos	Connection Pool	> 90%	Alert
Red	Latency	> 100ms	Investigation

Implementación de Monitoreo Continuo



Configuración de alertas con Alertmanager

```yaml
# Alert manager configuration for PortTrack
groups:
- name: porttrack.rules
  rules:
  - alert: HighErrorRate
    expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.01
    for: 2m
    labels:
      severity: critical
    annotations:
      summary: "High error rate detected"
      description: "Error rate is above 1% for more than 2 minutes"

  - alert: HighLatency
    expr: histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m])) > 0.5
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High latency detected"
      description: "95th percentile latency is above 500ms"

  - alert: PodCrashLooping
    expr: rate(kube_pod_container_status_restarts_total[5m]) > 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Pod is crash looping"
      description: "Pod {{ $labels.pod }} is restarting frequently"
```

Prometheus Alertmanager gestiona alertas basadas en métricas críticas, enviando notificaciones automáticas para una respuesta rápida ante anomalías, complementado con dashboards claros en Grafana.

```json
# Grafana configuration for PortTrack Operations Dashboard
{
  "dashboard": {
    "title": "PortTrack Operations Dashboard",
    "panels": [
      {
        "title": "Request Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(http_requests_total[5m])",
            "legendFormat": "{{method}} {{status}}"
          }
        ]
      },
      {
        "title": "Response Time P95",
        "type": "stat",
        "targets": [
          {
            "expr": "histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))"
          }
        ]
      },
      {
        "title": "Active Ships",
        "type": "stat",
        "targets": [
          {
            "expr": "porttrack_active_ships_total"
          }
        ]
      }
    ]
  }
}
```

Automatización y ChatOps

Integración de ChatOps con Slack y Hubot

La integración de ChatOps con Slack y Hubot mejora la coordinación y respuesta rápida en PortTrack, enviando notificaciones automáticas sobre despliegues y eventos críticos en canales comunes para facilitar la colaboración en tiempo real.

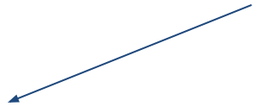
Flujos de trabajo automatizados

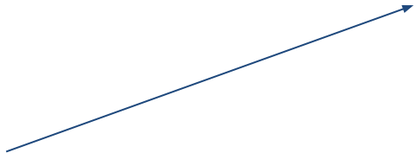
Despliegue y rollback: Se implementan flujos automatizados para gestionar despliegues continuos y rollback mediante comandos en ChatOps, reduciendo tiempos de respuesta, asegurando continuidad operativa y minimizando la intervención manual.

Automatización y ChatOps

```javascript
// scripts/porttrack-ops.js
module.exports = function(robot) {

  // Deployment commands
  robot.respond(/deploy (.*) to (.*)/i, function(res) {
    const version = res.match[1];
    const environment = res.match[2];

    if (['dev', 'staging', 'prod'].includes(environment)) {
      res.send(`🚀 Deploying ${version} to ${environment}...`);

      // Trigger GitHub Actions workflow
      triggerDeployment(version, environment)
        .then(() => {
          res.send(`✅ Deployment to ${environment} initiated successfully`);
        })
        .catch((err) => {
          res.send(`❌ Deployment failed: ${err.message}`);
        });
    } else {
      res.send(`❌ Invalid environment. Use: dev, staging, prod`);
    }
  });

  // Status checks
  robot.respond(/status (.*)/i, function(res) {
    const environment = res.match[1];

    getEnvironmentStatus(environment)
      .then((status) => {
        res.send(formatStatusMessage(status));
      });
  });

  // Rollback command
  robot.respond(/rollback (.*)/i, function(res) {
    const environment = res.match[1];

    if (environment === 'prod') {
      res.send(`🔄 Initiating rollback for production...`);
      executeRollback(environment)
        .then(() => {
          res.send(`✅ Production rollback completed successfully`);
        });
    }
  });
};
```

Slack + Hubot Implementation





Slack Webhook Integration

```yaml
# .github/workflows/notify-deployment.yml
name: Deployment Notifications

on:
  workflow_run:
    workflows: ["PortTrack Blue-Green Deployment"]
    types: [completed]

jobs:
  notify:
    runs-on: ubuntu-latest
    steps:
      - name: Notify Success
        if: ${{ github.event.workflow_run.conclusion == 'success' }}
        uses: 8398a7/action-slack@v3
        with:
          status: success
          channel: '#porttrack-deployments'
          webhook_url: ${{ secrets.SLACK_WEBHOOK_URL }}
          custom_payload: |
            {
              "text": "🚀 PortTrack Deployment Successful",
              "attachments": [
                {
                  "color": "good",
                  "fields": [
                    {
                      "title": "Environment",
                      "value": "${{ github.event.workflow_run.head_branch }}",
                      "short": true
                    },
                    {
                      "title": "Commit",
                      "value": "${{ github.event.workflow_run.head_sha }}",
                      "short": true
                    }
                  ]
                }
              ]
            }

      - name: Notify Failure
        if: ${{ github.event.workflow_run.conclusion == 'failure' }}
        uses: 8398a7/action-slack@v3
        with:
          status: failure
          channel: '#porttrack-alerts'
          webhook_url: ${{ secrets.SLACK_WEBHOOK_URL }}
```

Arquitectura Propuesta para Despliegue y Monitoreo

Resumen de Infraestructura

La arquitectura de PortTrack inicia con los usuarios que acceden a través de un Gateway API, el cual gestiona la seguridad y el acceso. Las solicitudes pasan por sistemas de autenticación como Auth0 o AWS Cognito, y son procesadas por microservicios que manejan datos en bases SQL y NoSQL. La observabilidad se garantiza con Prometheus, Grafana y ELK, mientras Kubernetes orquesta los despliegues y CI/CD automatiza las actualizaciones.

Beneficios Clave de la Arquitectura

La arquitectura propuesta ofrece escalabilidad mediante Kubernetes para manejar altos volúmenes de datos y solicitudes, resiliencia con monitoreo continuo que permite detección y respuesta rápida a fallos, seguridad robusta con autenticación centralizada, automatización eficiente de despliegues con CI/CD, y observabilidad integral para análisis y mejora continua.

Beneficios y Consideraciones Finales

Mejora en la estabilidad y disponibilidad de la plataforma

La integración de Kubernetes, Prometheus y Grafana asegura una arquitectura escalable y monitoreo en tiempo real, permitiendo actualizaciones continuas sin afectar operaciones y facilitando la recuperación rápida ante fallos.

Reducción de riesgos en despliegues y actualizaciones

La diferenciación de entornos y el uso de AWS Secrets Manager junto con CI/CD y ChatOps garantizan seguridad, protección de datos y respuestas inmediatas ante incidencias, minimizando impactos en la operación portuaria.

Conclusiones y Recomendaciones

Automatización y Resiliencia Garantizadas

La estrategia de despliegue y monitoreo continuo en PortTrack asegura automatización avanzada y resiliencia mediante pipelines CI/CD robustos, monitoreo proactivo y comunicación eficiente entre equipos.

Actualización Continua de Herramientas DevOps

Mantener actualizadas las herramientas DevOps es esencial para la seguridad y eficacia, permitiendo corregir vulnerabilidades, mejorar compatibilidad y fortalecer la postura de seguridad del sistema.

Gracias

Contact: Cesarherr321@gmail.com