PARTIE 5 : Sécurité, SRE et Chaos Engineering

Durée : 45 minutes

Objectifs

Dans cette cinquième et dernière partie, vous allez implémenter les pratiques SRE avancées : Policy as Code avec Kyverno, analyse de sécurité runtime avec Falco, CI/CD sécurisé avec signature d'images, calcul d'Error Budget, et tests de résilience avec Chaos Engineering.

Compétences évaluées : - Policy as Code (Kyverno) - Runtime Security (Falco) - CI/CD sécurisé avec signature d'images (Cosign) - SLO/SLI et Error Budget - Chaos Engineering (Litmus)

Travail à Réaliser

Tâche 5.1 : Policy as Code avec Kyverno (8 points)

Installation de Kyverno

```
# 1. Installation via Helm
helm repo add kyverno https://kyverno.github.io/kyverno/
helm repo update

helm install kyverno kyverno/kyverno \
  --namespace kyverno \
  --create-namespace \
  --set replicaCount=1

# 2. Vérifier l'installation
kubectl get pods -n kyverno
kubectl get crd | grep kyverno
```

Policy 1 : Interdire le Tag "latest"

Fichier : policies/kyverno/disallow-latest-tag.yaml

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-latest-tag
```

```yaml
  annotations:
    policies.kyverno.io/title: Disallow Latest Tag
    policies.kyverno.io/category: Best Practices
    policies.kyverno.io/severity: medium
    policies.kyverno.io/description: >-
      L'utilisation du tag 'latest' est interdite car elle empêche
      le versionning et rend les rollbacks difficiles.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
  - name: require-image-tag
    match:
      any:
      - resources:
          kinds:
          - Pod
          namespaces:
          - cloudshop-prod
    validate:
      message: "L'utilisation du tag 'latest' est interdite. Utilisez un tag versionné (ex:
v1.0.0)."
      pattern:
        spec:
          containers:
          - image: "!*:latest"
```

Policy 2 : Exiger Requests et Limits

Fichier : policies/kyverno/require-resources.yaml

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-resources
  annotations:
    policies.kyverno.io/title: Require Resources
    policies.kyverno.io/category: Best Practices
    policies.kyverno.io/severity: high
    policies.kyverno.io/description: >-
      Tous les conteneurs doivent avoir des requests et limits CPU/Memory
      pour garantir la stabilité du cluster.
spec:
  validationFailureAction: Enforce
```

```
  background: true
  rules:
  - name: check-cpu-memory-requests
    match:
      any:
      - resources:
          kinds:
          - Pod
          namespaces:
          - cloudshop-prod
    validate:
      message: "Les conteneurs doivent avoir des requests et limits CPU/Memory définis."
      pattern:
        spec:
          containers:
          - resources:
              requests:
                memory: "?*"
                cpu: "?*"
              limits:
                memory: "?*"
                cpu: "?*"
```

Policy 3 : Interdire Privileged

Fichier : policies/kyverno/disallow-privileged.yaml

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged
  annotations:
    policies.kyverno.io/title: Disallow Privileged Containers
    policies.kyverno.io/category: Security
    policies.kyverno.io/severity: critical
    policies.kyverno.io/description: >-
      Les conteneurs privileged ont accès à toutes les ressources du host
      et représentent un risque de sécurité majeur.
spec:
  validationFailureAction: Enforce
  background: true
  rules:
  - name: check-privileged
    match:
```

```yaml
    any:
    - resources:
        kinds:
        - Pod
        namespaces:
        - cloudshop-prod
  validate:
    message: "Les conteneurs privilégiés sont interdits."
    pattern:
      spec:
        containers:
        - securityContext:
            privileged: false
```

Déploiement et Test

```
# 1. Appliquer les policies
kubectl apply -f policies/kyverno/

# 2. Vérifier les policies
kubectl get clusterpolicy

# 3. Tester la policy disallow-latest-tag
kubectl run test-latest --image=nginx:latest -n cloudshop-prod
# Attendu : Error from server: admission webhook denied the request

# 4. Tester avec un tag valide
kubectl run test-versioned --image=nginx:1.25 -n cloudshop-prod
# Attendu : pod/test-versioned created

# 5. Nettoyer
kubectl delete pod test-versioned -n cloudshop-prod
```

Tâche 5.2 : Runtime Security avec Falco (5 points)

Installation de Falco

```
# 1. Installation via Helm
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update

helm install falco falcosecurity/falco \
  --namespace falco \
```

```
  --create-namespace \
  --set falco.grpc.enabled=true \
  --set falco.grpcOutput.enabled=true

# 2. Vérifier l'installation
kubectl get pods -n falco
kubectl logs -l app.kubernetes.io/name=falco -n falco --tail=20
```

Configuration des Règles Falco

Fichier : policies/falco/custom-rules.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: falco-custom-rules
  namespace: falco
data:
  custom-rules.yaml: |
    - rule: Suspicious Shell in Container
      desc: Détecte l'exécution de shell dans un conteneur
      condition: >
        spawned_process and
        container and
        proc.name in (bash, sh, zsh) and
        container.image.repository in (cloudshop/frontend, cloudshop/api-gateway)
      output: >
        Shell exécuté dans un conteneur CloudShop
        (user=%user.name container=%container.name image=%container.image.repository
        command=%proc.cmdline)
      priority: WARNING
      tags: [container, shell, cloudshop]

    - rule: Write Below Root
      desc: Détecte une écriture dans un répertoire système
      condition: >
        open_write and
        container and
        fd.name startswith /root and
        not proc.name in (dpkg, rpm, yum)
      output: >
        Écriture détectée dans /root
        (user=%user.name container=%container.name file=%fd.name
command=%proc.cmdline)
```

```
    priority: ERROR
    tags: [filesystem, container]

  - rule: Sensitive File Access
    desc: Détecte l'accès à des fichiers sensibles
    condition: >
      open_read and
      container and
      fd.name in (/etc/shadow, /etc/passwd, /etc/sudoers)
    output: >
      Accès à un fichier sensible détecté
      (user=%user.name container=%container.name file=%fd.name
command=%proc.cmdline)
    priority: CRITICAL
    tags: [filesystem, security]

  - rule: Unexpected Network Connection
    desc: Détecte une connexion réseau inattendue
    condition: >
      outbound and
      container and
      fd.sport != 80 and fd.sport != 443 and fd.sport != 8080 and
      not proc.name in (curl, wget)
    output: >
      Connexion réseau inattendue depuis un conteneur
      (container=%container.name connection=%fd.name command=%proc.cmdline)
    priority: WARNING
    tags: [network, container]

# Appliquer la configuration
kubectl apply -f policies/falco/custom-rules.yaml

# Redémarrer Falco pour charger les règles
kubectl rollout restart daemonset/falco -n falco
```

Test des Règles Falco

```
# 1. Exec dans un pod (doit déclencher "Suspicious Shell in Container")
kubectl exec -it deployment/frontend -n cloudshop-prod -- /bin/sh

# 2. Voir les alertes Falco
kubectl logs -l app.kubernetes.io/name=falco -n falco --tail=50 | grep WARNING

# 3. Tester accès fichier sensible
```

```
kubectl exec -it deployment/api-gateway -n cloudshop-prod -- cat /etc/shadow 2>/dev/null
# Doit déclencher "Sensitive File Access"

# 4. Voir les alertes
kubectl logs -l app.kubernetes.io/name=falco -n falco | grep CRITICAL
```

Tâche 5.3 : CI/CD Sécurisé avec Cosign (8 points)

Installation de Cosign

```
# macOS
brew install cosign

# Linux
wget https://github.com/sigstore/cosign/releases/latest/download/cosign-linux-amd64
sudo mv cosign-linux-amd64 /usr/local/bin/cosign
sudo chmod +x /usr/local/bin/cosign
```

Génération des Clés de Signature

```
# Générer une paire de clés
cosign generate-key-pair

# Crée deux fichiers :
# - cosign.key (clé privée - à garder secrète)
# - cosign.pub (clé publique - à partager)

# Créer un Secret Kubernetes avec la clé privée
kubectl create secret generic cosign-key \
  --from-file=cosign.key=cosign.key \
  -n cloudshop-prod
```

Pipeline GitHub Actions avec Signature

Fichier : .github/workflows/docker-ci.yml

```
name: Docker Build, Sign and Push

on:
  push:
    branches: [ main ]
    paths:
    - 'src/**'
  pull_request:
```

```yaml
    branches: [ main ]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  build-and-sign:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
      id-token: write

    strategy:
      matrix:
        service: [frontend, api-gateway, auth-service, products-api, orders-api]

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}
        tags: |
          type=ref,event=branch
          type=sha,prefix={{branch}}-
          type=semver,pattern={{version}}

    - name: Build and push Docker image
```

```
      id: build
      uses: docker/build-push-action@v5
      with:
        context: ./src/${{ matrix.service }}
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max

    - name: Install Cosign
      uses: sigstore/cosign-installer@v3

    - name: Sign the container image
      env:
        COSIGN_KEY: ${{ secrets.COSIGN_PRIVATE_KEY }}
        COSIGN_PASSWORD: ${{ secrets.COSIGN_PASSWORD }}
      run: |
        IMAGE_TAG="${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}@$
{{ steps.build.outputs.digest }}"
        echo "Signing image: $IMAGE_TAG"
        echo "$COSIGN_KEY" > cosign.key
        cosign sign --key cosign.key --yes "$IMAGE_TAG"
        rm cosign.key

    - name: Generate SBOM with Syft
      run: |
        curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s --
-b /usr/local/bin
        syft ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}:$
{{ steps.meta.outputs.version }} \
          -o spdx-json > sbom.json

    - name: Attach SBOM to image
      env:
        COSIGN_KEY: ${{ secrets.COSIGN_PRIVATE_KEY }}
        COSIGN_PASSWORD: ${{ secrets.COSIGN_PASSWORD }}
      run: |
        IMAGE_TAG="${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}@$
{{ steps.build.outputs.digest }}"
        echo "$COSIGN_KEY" > cosign.key
        cosign attest --predicate sbom.json --key cosign.key --yes "$IMAGE_TAG"
        rm cosign.key
```

```
    - name: Scan image with Trivy
      uses: aquasecurity/trivy-action@master
      with:
        image-ref: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}:$
{{ steps.meta.outputs.version }}
        format: 'sarif'
        output: 'trivy-results.sarif'

    - name: Upload Trivy results to GitHub Security
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: 'trivy-results.sarif'
```

Policy Kyverno pour Vérifier les Signatures

Fichier : policies/kyverno/verify-image-signature.yaml

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: verify-image-signature
  annotations:
    policies.kyverno.io/title: Verify Image Signature
    policies.kyverno.io/category: Security
    policies.kyverno.io/severity: critical
    policies.kyverno.io/description: >-
      Vérifie que toutes les images sont signées avec Cosign.
spec:
  validationFailureAction: Enforce
  background: false
  webhookTimeoutSeconds: 30
  rules:
  - name: check-signature
    match:
      any:
      - resources:
          kinds:
          - Pod
          namespaces:
          - cloudshop-prod
    verifyImages:
    - imageReferences:
      - "ghcr.io/votre-org/cloudshop/*"
```

```
    attestors:
    - count: 1
      entries:
      - keys:
          publicKeys: |-
            -----BEGIN PUBLIC KEY-----
            # Contenu de cosign.pub
            -----END PUBLIC KEY-----
```

Tâche 5.4 : Error Budget et SLO (4 points)

Calcul de l'Error Budget

SLO : 99.9% de disponibilité sur 30 jours

Formule :

Error Budget = (1 - SLO) * Période
Error Budget = (1 - 0.999) * 30 jours * 24h * 60min
Error Budget = 0.001 * 43,200 minutes
Error Budget = 43.2 minutes par mois

Queries PromQL pour Error Budget

Fichier : monitoring/error-budget/queries.txt

```
# 1. Availability actuelle (30 jours)
(
  sum(rate(http_requests_total{status!~"5..",namespace="cloudshop-prod"}[30d]))
  /
  sum(rate(http_requests_total{namespace="cloudshop-prod"}[30d]))
) * 100

# 2. Error Budget consommé (en %)
(
  1 - (
    sum(rate(http_requests_total{status!~"5..",namespace="cloudshop-prod"}[30d]))
    /
    sum(rate(http_requests_total{namespace="cloudshop-prod"}[30d]))
  )
) / (1 - 0.999) * 100

# 3. Error Budget restant (en minutes)
```

```
43.2 * (
  1 - (
    (1 - (
      sum(rate(http_requests_total{status!~"5..",namespace="cloudshop-prod"}[30d]))
      /
      sum(rate(http_requests_total{namespace="cloudshop-prod"}[30d]))
    )) / (1 - 0.999)
  )
)

# 4. Burn Rate (1h)
(
  sum(rate(http_requests_total{status=~"5..",namespace="cloudshop-prod"}[1h]))
  /
  sum(rate(http_requests_total{namespace="cloudshop-prod"}[1h]))
) / (1 - 0.999)
```

Dashboard Grafana Error Budget

Créer un dashboard avec 4 panels :

1. **Gauge** : SLO Actuel (99.9% = target)
2. **Gauge** : Error Budget Restant (%)
3. **Graph** : Burn Rate sur 24h
4. **Stat** : Temps de downtime autorisé restant (en minutes)

Tâche 5.5 : Chaos Engineering avec Litmus (5 points)

Installation de Litmus

```
# 1. Installation
kubectl apply -f https://litmuschaos.github.io/litmus/litmus-operator-v3.0.0.yaml

# 2. Vérifier
kubectl get pods -n litmus

# 3. Installer les ChaosExperiments
kubectl apply -f
https://hub.litmuschaos.io/api/chaos/3.0.0?file=charts/generic/experiments.yaml -n
cloudshop-prod
```

Chaos Experiment : Pod Delete

Fichier : chaos/experiments/pod-delete.yaml

```yaml
apiVersion: litmuschaos.io/v1alpha1
kind: ChaosEngine
metadata:
  name: frontend-chaos
  namespace: cloudshop-prod
spec:
  appinfo:
    appns: cloudshop-prod
    applabel: 'app=frontend'
    appkind: deployment
  engineState: 'active'
  chaosServiceAccount: litmus-admin
  experiments:
  - name: pod-delete
    spec:
      components:
        env:
        - name: TOTAL_CHAOS_DURATION
          value: '30'
        - name: CHAOS_INTERVAL
          value: '10'
        - name: FORCE
          value: 'false'
        - name: PODS_AFFECTED_PERC
          value: '50'
      probe:
      - name: check-frontend-availability
        type: httpProbe
        httpProbe/inputs:
          url: http://frontend.cloudshop-prod.svc.cluster.local
          insecureSkipVerify: true
          method:
            get:
              criteria: ==
              responseCode: "200"
        mode: Continuous
        runProperties:
          probeTimeout: 5
          interval: 2
          retry: 1
```

RBAC pour Litmus

Fichier : chaos/rbac/litmus-admin.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: litmus-admin
  namespace: cloudshop-prod
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: litmus-admin
  namespace: cloudshop-prod
rules:
- apiGroups: [""]
  resources: ["pods", "services", "events"]
  verbs: ["get", "list", "patch", "delete", "deletecollection"]
- apiGroups: ["apps"]
  resources: ["deployments", "statefulsets", "replicasets"]
  verbs: ["get", "list", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: litmus-admin
  namespace: cloudshop-prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: litmus-admin
subjects:
- kind: ServiceAccount
  name: litmus-admin
  namespace: cloudshop-prod
```

Exécution du Chaos Test

```
# 1. Créer le RBAC
kubectl apply -f chaos/rbac/litmus-admin.yaml

# 2. Lancer le Chaos Experiment
kubectl apply -f chaos/experiments/pod-delete.yaml
```

```
# 3. Observer l'exécution
kubectl get chaosengine -n cloudshop-prod
kubectl describe chaosengine frontend-chaos -n cloudshop-prod

# 4. Voir les pods supprimés et recréés
watch kubectl get pods -n cloudshop-prod

# 5. Vérifier le résultat
kubectl get chaosresult -n cloudshop-prod
kubectl describe chaosresult frontend-chaos-pod-delete -n cloudshop-prod

# 6. Vérifier que l'application a survécu
curl -H "Host: shop.local" http://localhost
# Attendu : 200 OK
```

Validation

Checklist Complète

```
# 1. Kyverno policies actives
kubectl get clusterpolicy
# Doit lister : disallow-latest-tag, require-resources, disallow-privileged

# 2. Test policy
kubectl run test --image=nginx:latest -n cloudshop-prod
# Attendu : denied

# 3. Falco détecte les activités suspectes
kubectl logs -l app.kubernetes.io/name=falco -n falco | grep WARNING

# 4. Images signées (si CI/CD configuré)
cosign verify --key cosign.pub ghcr.io/org/cloudshop/frontend:tag

# 5. Error Budget calculé dans Grafana
# Dashboard Error Budget affiche les métriques

# 6. Chaos Experiment réussi
kubectl get chaosresult -n cloudshop-prod
# Pass : true
```

Ressources

- Kyverno Documentation
- Falco Rules
- Cosign Documentation
- Litmus Chaos Experiments
- SRE Workbook - Implementing SLOs

Félicitations ! Vous avez terminé le TD final Docker & Kubernetes.