Red Hat OpenShift

Liveness, Readiness et Startup Probes

Exemple avec Quarkus

Erik Lalancette
OpenShift TAM
elalance@redhat.com

Red Hat

The Health Probe pattern

- Kubernetes vérifie régulièrement l'état du processus du conteneur et le redémarre si des problèmes sont détectés. Cependant, par la pratique, nous savons que la vérification de l'état du processus ne suffit pas pour décider de l'intégrité d'une application. Dans de nombreux cas, une application se bloque, mais son processus est toujours opérationnel. Par exemple, une application Java peut lancer une OutOfMemoryError et toujours exécuter le processus JVM. Alternativement, une application peut se figer car elle s'exécute dans une boucle infinie, un blocage ou un battement (cache, tas, processus). Pour détecter ce type de situation, Kubernetes a besoin d'un moyen fiable de vérifier la santé des applications. Autrement dit, pas pour comprendre comment une application fonctionne en interne, mais une vérification qui indique si l'application fonctionne comme prévu et est capable de servir les consommateurs.

ref://https://github.com/k8spatterns/examples/

Red Hat

Contexte d’utilisation

- Liveness : Dans quelles circonstances est-il approprié de **redémarrer** le pod?
- Readiness : Dans quelles circonstances devons-nous retirer le pod de la liste des **service endpoints** afin qu'il ne réponde plus aux demandes?
- *Startup : Vérifie si l'application dans le conteneur **a démarré**.elle désactive les contrôles de liveness et readiness jusqu'à cela réussit. Très utile pour les conteneur lent à démarrer.

*Kube 1.18 beta+ /Openshift 4.5

Red Hat

Types de Probe

- The **HTTP probe** est le type le plus courant de sonde de liveness probe. La probe ping le chemin. S'il obtient une réponse HTTP dans la plage 200 ou 300, il marque l'application comme saine. Sinon, cela marque l'application comme malsaine.
- The **command probe** roule une commande à l'intérieur de votre conteneur. Si la commande retourne avec le code exit 0, la sonde marque le conteneur comme sain. Sinon, cela le marque comme malsain.
- The **TCP probe** tente une connexion TCP sur un port spécifié. Si la connexion réussit, le conteneur est considéré comme sain; si la connexion échoue, elle est considérée comme défectueuse. Les sondes TCP peuvent être utiles pour un service FTP et autre.

Configure Probes

- **initialDelaySeconds** : Nombre de secondes après le démarrage du conteneur avant que les sondes d'activité ou de disponibilité ne soient lancées. La valeur par défaut est 0 seconde. La valeur minimale est 0.

- **periodSeconds** : À quelle fréquence (en secondes) effectuer par la probe. Par défaut à 10 secondes. La valeur minimale est 1.

- **timeoutSeconds** : Nombre de secondes après lequel la probe expire. La valeur par défaut est 1 seconde. La valeur minimale est 1.

- **successThreshold** : Succès consécutifs minimum pour que la Probe soit considérée comme réussie après avoir échoué. La valeur par défaut est 1. Doit être 1 pour liveness. La valeur minimale est 1.

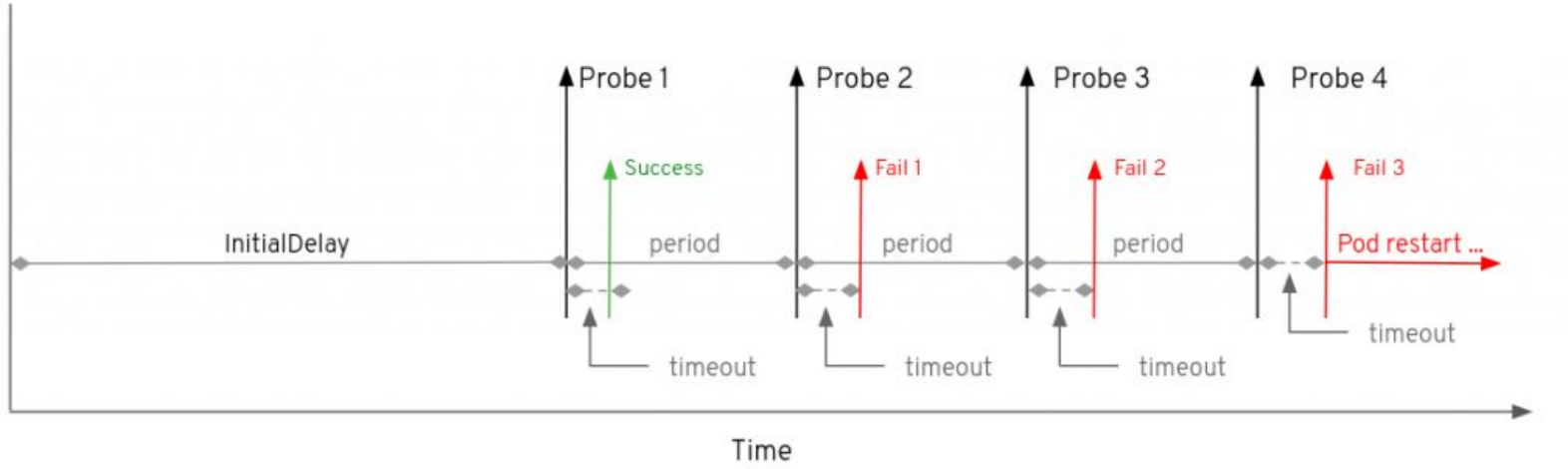
- **failureThreshold** : Lorsqu'une Probe échoue, Kubernetes essaiera des failureThresold avant d'abandonner. Abandonner en cas de probe de liveness signifie redémarrer le conteneur. En cas de probe de Readiness, le Pod sera marqué NotReady. La valeur par défaut est 3. La valeur minimale est 1.

Red Hat

Exemple de Liveness

L'exemple ci-dessous illustre la probe en action à travers une chronologie. La première probe réussit, mais les deuxième, troisième et quatrième échouent. En supposant un réglage par défaut de 3 pour le failureThreshold le pod redémarrera après l'échec de la quatrième probe

failureThreshold : 3

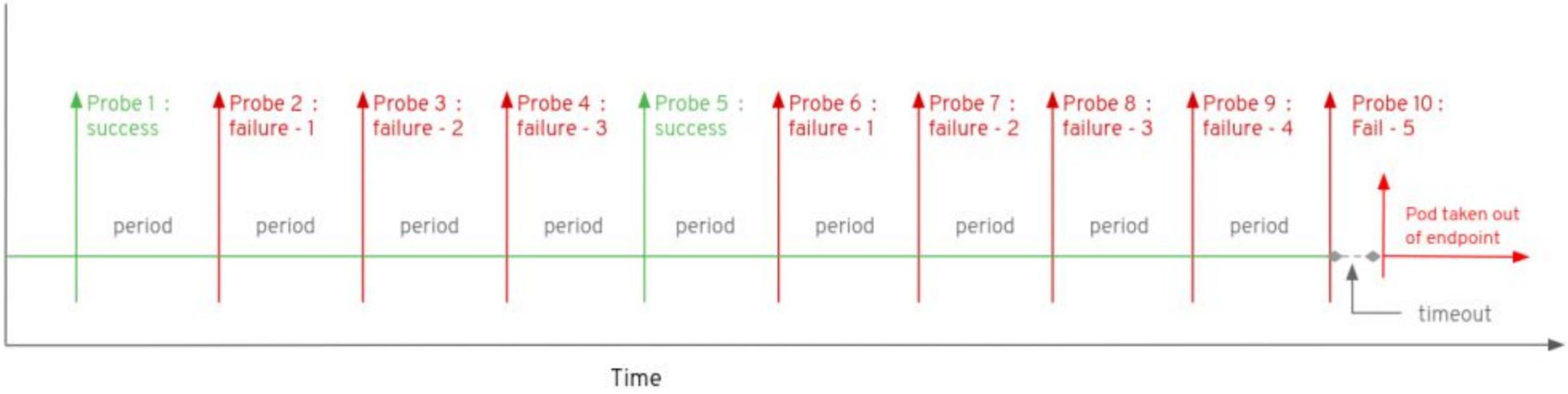


Red Hat

Exemple de Readiness

Dans le diagramme ci-dessous, le pod subit trois échecs consécutifs pour répondre à la probe, suivis d'une réponse réussie (probe 5). Cette réponse réussie réinitialise le compteur en cas d'échecs de sorte qu'une autre séquence de cinq échecs se produit (probe 6 à 10) avant le retrait du pod de la liste endpoints à la probe 10.

failureThreshold : 5 successThreshold : 1



Red Hat

Exemple YAML

tcpSocket

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

httpGet

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

command

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30;
rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

Red Hat

Qu'est-ce que Quarkus ?

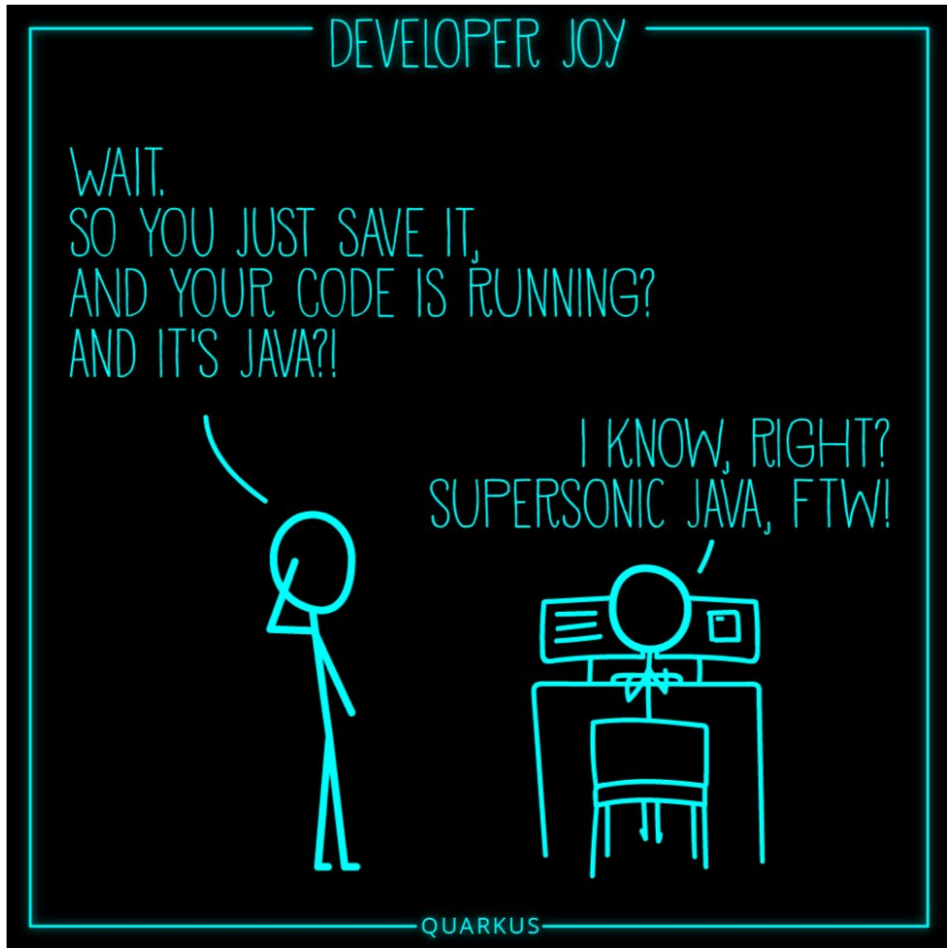
Quarkus est un framework full-stack Java natif pour Kubernetes, conçu pour les machines virtuelles Java (JVM) et la compilation native, qui permet d'optimiser Java spécifiquement pour les conteneurs afin d'en faire une plateforme efficace pour les environnements sans serveur, cloud et Kubernetes.

Par rapport aux applications Java traditionnelles, les applications créées avec Quarkus consomment 10 fois moins de mémoire et démarrent plus rapidement (jusqu'à 300 fois plus vite), ce qui permet de réduire grandement le coût des ressources cloud. 6 à 10)

Red Hat

Demo

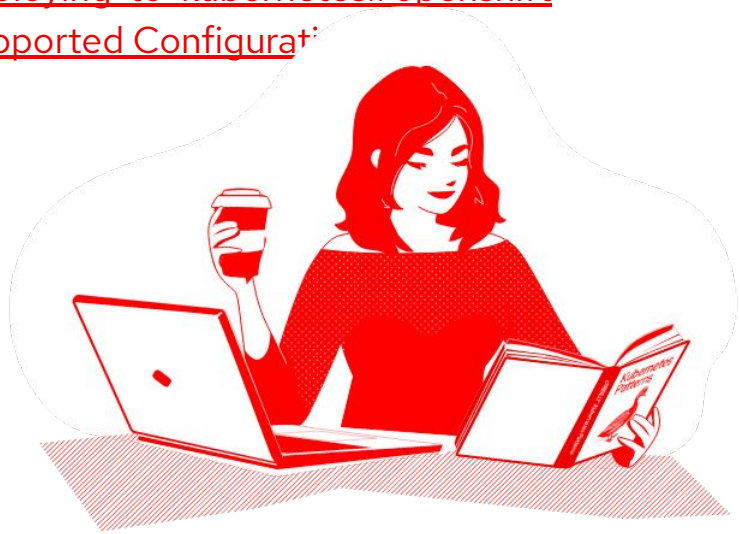
- Quarkus smallrye-health microprofile health



Red Hat

Plus de Resources

- Understanding health checks
  - https://docs.openshift.com/container-platform/4.5/applications/application-health.html
- Configure Liveness, Readiness and Startup Probes
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/
- Kubernetes Patterns: Reusable elements for designing cloud-native applications E-BOOK
  - https://www.redhat.com/en/engage/kubernetes-containers-architecture-s-201910240918
  - https://github.com/k8spatterns/examples/
- Quarkus
  - https://redhat-developer-demos.github.io/quarkus-tutorial/quarkus-tutorial/setup.html
  - https://quarkus.io/guides/deploying-to-kubernetes#openshift
  - Red Hat build of Quarkus Supported Configurat



Red Hat

Merci

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat