富士通 Java ブログ

# MicroProfile Config/Health と Kubernetes

2022 年 2 月 4 日 初版
廣石 真也

## MicroProfile Config/Health が Kubernetes と親和性が高い理由

MicroProfile はコンテナ環境での利用に適していると言われますが、どういった部分が適しているのでしょうか。デファクトスタンダードなコンテナ環境となった Kubernetes 上で、Launcher などの MicroProfile 規約に準拠した実装を利用した場合に、どのようなメリットがあるかご紹介します。

### Liveness と Readiness

Kubernetes にはコンテナの状態を監視する「Liveness Probe」と「Readiness Probe」と呼ばれる活性プローブがあります。
Liveness Probe は、コンテナのヘルスチェックに失敗（例えば、アプリケーションが起動しているのにも関わらず応答できなくなったような状態を検知）すると、Kubernetes にコンテナを再起動させて復旧します。また Readiness Probe は、コンテナのヘルスチェックに失敗（例えば、コンテナがトラフィックを受け入れられない状態を検知）すると、Kubernetes が対象となるコンテナへリクエストを振り分けないようにします。

### 参考

- Liveness Probe、Readiness Probe および Startup Probe を使用する（「Kubernetes ドキュメント」のページへ）

Liveness Probe を利用する場合、Kubernetes の Pod 定義に対して以下のように定義します。以下の例ではコンテナがリクエストを処理できるか判断する条件として、コンテナを起動してから 3 秒後に最初の Liveness Probe を実行し、8080 ポートをリッスンしているコンテナ内のサーバーに対し HTTP の GET リクエストを発行し、サーバー内の「/health/live」パスに対するハンドラが正常な応答をすることで、Kubernetes のコンテナが稼働しているかを判断します。

```
livenessProbe:
   httpGet:
      path: /health/live
      port: 8080
   initialDelaySeconds: 3
   periodSeconds: 3
```

上記のように、Liveness Probe と Readiness Probe の判断条件は、Kubernetes 利用者が定義する必要があります。

Kubernetes の仕組みを簡単に利用するために、MicroProfile には MicroProfile Health という API が存在します。この API では、Liveness 用の/health/live と Readiness 用の/health/ready のエンドポイントが用意されています。また、@Liveness や@Readiness のアノテーションを利用することにより、判断方法をカスタマイズすることができます。

これを利用することで、Kubernetes の Liveness Probe と Readiness Probe で利用する稼働状態を確認する HTTP API を簡単に用意できます。

 

### Kubernetes の ConfigMap と MicroProfile Config

クラウドネイティブの考え方では、環境に依存する情報（接続先の URL など）をアプリケーションのソースコードから切り離すことが推奨されています。これにより、アプリケーションのソースコードの編集やビルドを行わず、同一のソースコードや実行ライブラリーを利用できます。
開発したアプリケーションを素早く本番環境に展開するためには、設定情報の外出しは重要です。

この環境に依存する情報を切り離すために、Kubernetes では環境定義を一元管理できる「Config Map」という仕組みが用意されています。Pod の定義でコンテナの環境変数に Config Map の値をロードすることで、コンテナから環境に依存する情報を外出しすることが可能となります。

### 参考

- ConfigMap（「Kubernetes ドキュメント」のページへ）

同様にアプリケーションでは MicroProfile Config を利用することで、環境変数や Java のシステムプロパティなど外付けの情報を API もしくはアノテーションで参照することができます。これを利用して Config Map からロードされた環境変数を参照することで、完全に環境に依存しないアプリケーションとすることができます。

環境変数、Java のシステムプロパティなどに設定された情報は、MicroProfile Config の API から、設定方法の種類によらず統一的な方法で取得することができます。

```
@ApplicationScoped
public class ConfigSample {
   @Inject
   private Config config;
   public String getSampleProp() {
      return config.getValue("sample.prop", String.class);
   }
}
```

また、@CoonfigProperty アノテーションを使用して、設定情報をフィールドに注入して利用することもできます。

```
@ApplicationScoped
public class ConfigSample {
   @Inject
   @ConfigProperty (name="sample.prop")
   private String sampleProp;
}
```

## 今後に向けて

MicroProfile が Kubernetes の親和性が高いことをご理解いただけたでしょうか？コンテナ技術の利用が今後も拡大し、それに合わせて MicroProfile の機能は今後も拡張が予定されています。今後の MicroProfile の拡張にご期待ください。