WAS Liberty on IBM Cloud Private 利用ガイド

version 1.0.0

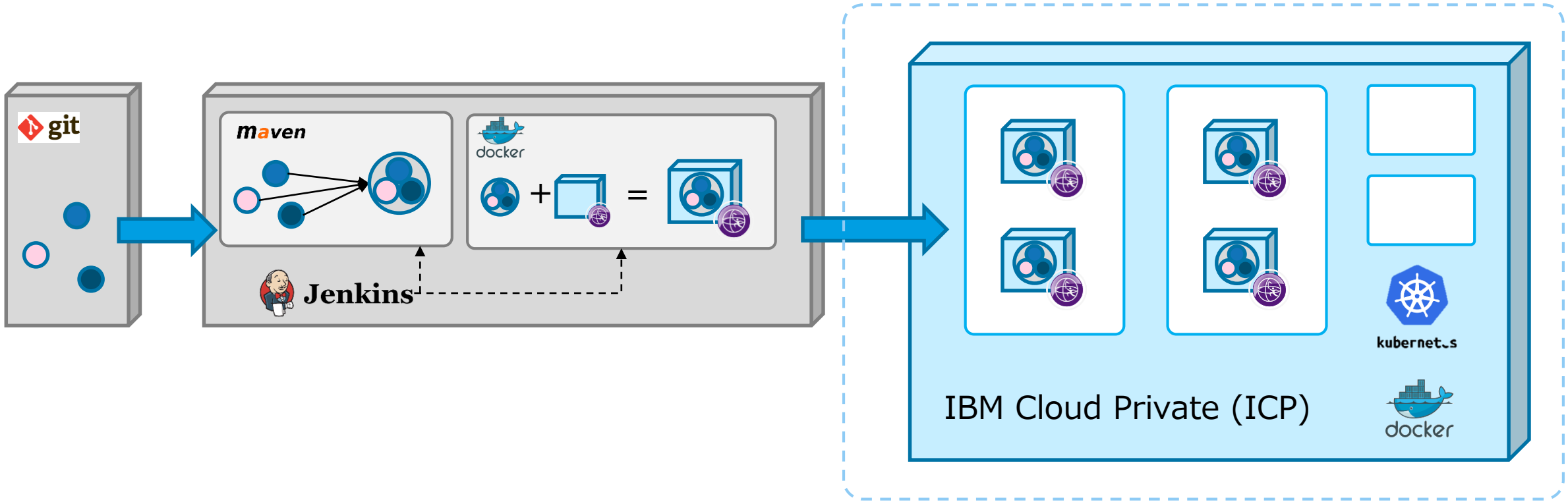
2018/07/20

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、製品の新しいリリース、FixPackなどによって動作、仕様が変わる可能性があるのでご注意下さい。この資料の内容は2018年7月現在の情報であり、下記の製品リリースに基づいています。
  - WebSphere Application Server Liberty 18.0.0.2 (18.0.0.1)
  - IBM Cloud Charts, ibm-websphere-liberty v1.4.0
  - IBM Cloud Private v2.1.0.3
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。

このガイドについて

- このガイドでは、IBM Cloud Private（ICP）環境での WAS Liberty のインストールと構成を解説しており、主に既存の WAS アプリケーションを ICP 環境にリフトする形態での利用を想定しています。
- このガイドでは、WAS Liberty で稼動するアプリケーションの開発・ビルド、および、CI/CD(継続的インテグレーション/継続的デリバリー)ツールを利用した自動化に関しては扱いません。



目次

1. IBM Cloud Private（ICP）概要

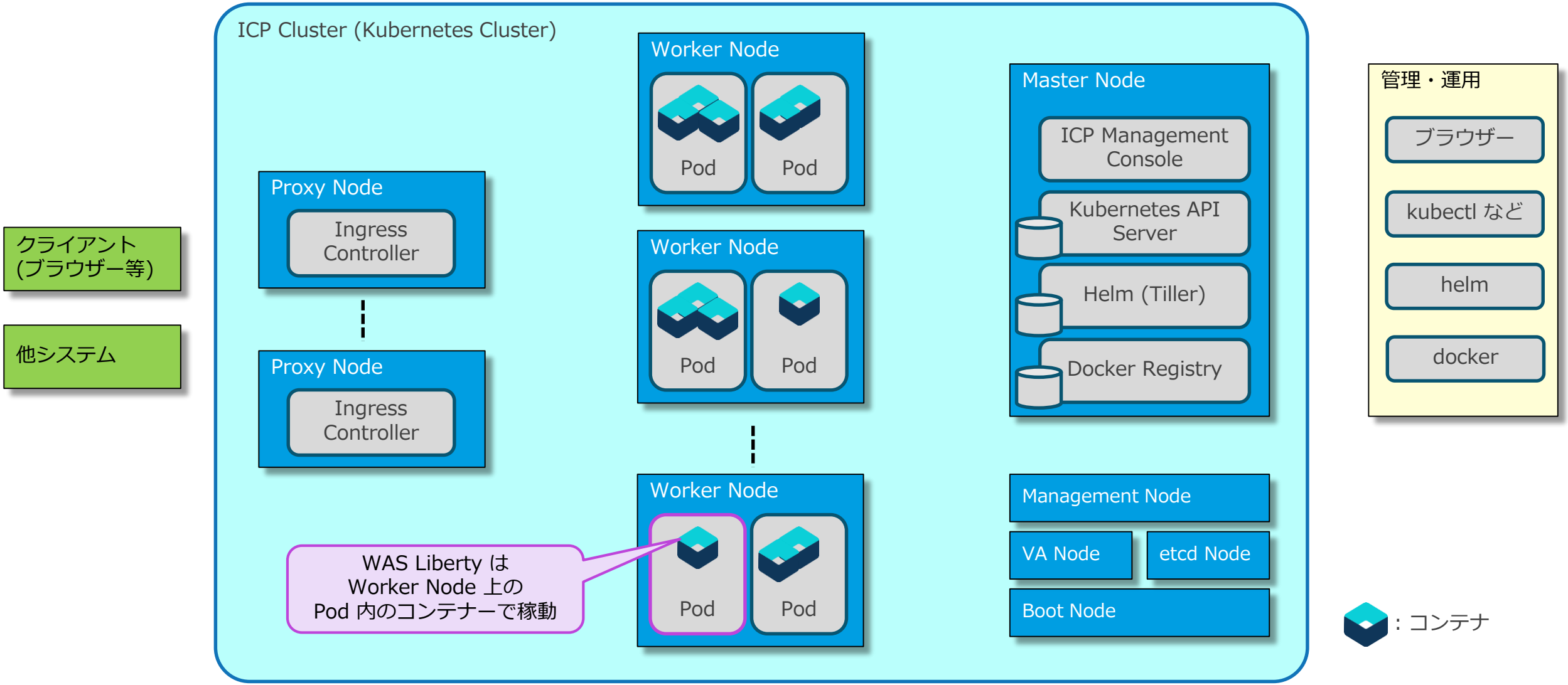
IBM Cloud Private（ICP）

- IBM Cloud Private （ICP）は、コンテナー化されたアプリケーションをオンプレミス環境で稼動されるためのプラットフォームです。

- 以下のような機能が統合されています。
  - コンテナー仮想化環境、プライベート・イメージ・レジストリー -- Docker
  - コンテナーのオーケストレーション機能 -- Kubernetes (k8s)
  - モニタリング・フレームワーク -- Prometheus/Grafana
  - ロギング・フレームワーク -- Elastic Stack
  - 管理コンソール（ICP Management Console）
  - など

IBM Cloud Private（ICP）の構成 (1)

■ 下図は ICP の構成概要を示したものです。説明は次ページを参照してください。



IBM Cloud Private（ICP）の構成 (2)

- ICP Cluster
  - 以下のノードで構成され、Master Node によって管理されます。
- Node
  - Master Node
    - クラスターを管理するノードです。複数の Master Node (3または5)を配置することで高可用性を確保できます。
  - Proxy Node
    - クライアント(ブラウザー)や他システムからの入り口となるノードです。複数の Proxy Node を配置することで高可用性を確保できます。
    - Ingress Controller が稼動し、リバース・プロキシーの機能を提供します。
  - Worker Node
    - アプリケーションやミドルウェアを稼動させるためのノードです。WAS Liberty はこのノードで稼動します。
    - 複数の Worker Node を配置することで、高可用性を確保し、処理能力を増やすことが可能です。
  - その他のノード
    - Management Node はモニタリングおよびロギング用のフレームワークを稼動させるノードです。
    - VA Node は Vulnerability Advisor の機能が稼動するノードで配置はオプションです。
    - etcd Node は Master Node から etcd のワークロードをオフロードするために配置されるオプションのノードです。
    - Boot Node はクラスターのインストール、ノード増強、クラスターの更新などで使用されるノードです。
  - 補足
    - Proxy Node と Management Node は Master Node と兼用することが可能です。
- Pod
  - 計算リソース、ネットワーク、ストレージを共有する緊密な関係を持ったコンテナーのグループで、Kubernetes の管理単位となります。
  - WAS Liberty は Pod 内のコンテナーで稼動します。
  - 各ノードで稼動する ICP (k8s) の機能も Pod として実装されているものがありますが、図ではその様には示されていません。
- コンテナー
  - Docker のコンテナ・ランタイム（インスタンス）です。

ICP の管理操作

■ ICP の管理操作は、以下のインターフェースを介して実行します

– ICP Management Console
  • ICP を管理するための GUI 画面を提供し、Web ブラウザーからアクセスします。

– Kubernetes API Server
  • 主に kubectl コマンドを使用して、ICP の管理操作(構成定義/確認/変更など)を行うために使用します。

– Helm (Tiller)
  • Chart に記述された定義に従って、アプリケーションやミドルウェアをインストール/更新/削除するために使用します。
    – *補足:Helm は k8s のパッケージ・マネージャーで、Chart(s) は Helm が使用するパッケージ・フォーマットです。k8s クラスター側で稼動するコンポーネントは Tiller と呼ばれます。*
  • helm コマンド、または、ICP Management Console から操作します。

– Docker Registry
  • Docker のイメージを管理(登録/確認/削除)するため使用します。
  • 登録(push)は docker コマンドなどで、イメージの確認/削除は ICP Management Console から行います。

ICP の管理操作：補足

- ICP Management Console へのアクセス方法や、各コマンドの利用方法に関しては、ICP の Knowledge Center を参照してください。
  - ICP Management Console へのアクセス
    - ICP Knowledge Center: Accessing your IBM Cloud Private cluster by using the management console
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/manage_cluster/cfc_gui.html
  - kubectl コマンドの利用
    - ICP Knowledge Center: Accessing your IBM Cloud Private cluster by using the kubectl CLI
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/manage_cluster/cfc_cli.html
  - helm コマンドの利用
    - ICP Knowledge Center: Setting up the Helm CLI
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/app_center/create_helm_cli.html
    - ICP Knowledge Center: Adding the internal Helm repository to Helm CLI
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/app_center/add_int_helm_repo_to_cli.html
  - イメージの操作、docker コマンドの利用
    - ICP Knowledge Center: Pushing and pulling images
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/manage_images/using_docker_cli.html
    - ICP Knowledge Center: Managing images
      - https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/manage_images/managing_images.html

基礎知識：Workload 関連

- Workload を定義することで、アプリケーションやミドルウェアを ICP クラスターで稼動させます。本ガイドに関連する Workload を以下に記載します。

  – ReplicaSet
    - テンプレートに従って同一構成の Pod を複数同時に起動し管理します。
    - Pod の名前は、ランダムなサフィックスが付与された一時的な名前となります。

  – Deployment
    - ReplicaSet を内部で使用しています。
    - ReplicaSet の履歴管理やローリング・アップデートなどの機能を提供します。

  – StatefulSet
    - Deployment/ReplicaSet と同様の機能を提供し、さらに、Pod の状態を保持する機能を持ちます。
    - Pod の名前は、インデックスで識別される永続的な名前となります。
    - Pod 毎に専用の永続領域(Persistent Volume)を割り当てることができます。

基礎知識： Service と Endpoint

- Service は、ReplicaSet や StatefulSet などが作成した論理的な Pod の集まりを抽象化する機能を提供します。
  本ガイドに関連する Service のタイプを以下に記載します。

  – type: ClusterIP
    - ICP クラスター内でのみ有効な IP アドレス（ClusterIP）を使用してサービスを公開します
    - サービスへのアクセスで必要となる Cluster IP は、Service 名を使用して DNS より取得できます

  – type: NodePort
    - ClusterIP の機能を提供します。
    - さらに、各 Node の IP アドレスで、外部からアクセス可能なポート番号（NodePort）を使用してサービスを公開します

  – Headless Service
    - clusterIP に None が指定されている Service で、Cluster IP が割り当てられません（type: ClusterIP の特殊形態です）
    - Service 名を使用して DNS を lookup すると、Service を提供する Pod の IP アドレスが返されます
    - Service を構成する Pod の IP アドレスが必要な場合などのように、特別な用途で利用されます

- Endpoint は、関連付けられた Service を提供可能な、Pod の IP アドレスのリストです。
  リストの内容は Liveness/Readiness Probe (次ページ参照)の結果に基づいて更新されます。

基礎知識： Probe

- Probe は、Pod 内のコンテナーの稼働状況をチェックし、コンテナーに対するヘルスチェック機能を提供します。System Probe を以下に記載します。

  – Liveness Probe
    - コンテナーの生死状態をチェックするための Probe です。
    - この Probe が失敗すると、コンテナーはキルされ、Pod のリスタート・ポリシーに従って処理されます。

  – Readiness Probe
    - コンテナーの Ready 状態をチェックするための Probe です。
    - この Probe が失敗すると、リクエストが該当の Pod へ割り振られなくなります。（該当の Pod の IP アドレスが Endpoint から除去されます。）

基礎知識： ConfigMap と Secret

- ConfigMap と Secret は、環境に依存する設定情報や機密性の高い情報などを保持する機能を提供します

- 保存されている情報は、環境変数、ファイル、または、コマンド・ライン引数を通じて、Pod に渡すことができるので、次のメリットがあります
  - 環境に依存する情報を Docker イメージに含める必要がなくなるので、環境毎にイメージをビルドする必要がなくなります（イメージの再利用性・可搬性が高まります）
  - 機密性の高い情報を Docker イメージに含める必要がなくなります

- ConfigMap
  - key-value 形式で情報を保持し、value として文字列やファイルが使用できます
  - ICP v2.1.0.3 (Kubernetes v1.10.0) からバイナリー・ファイルもサポートされています

- Secret
  - ConfigMap と同等の機能を提供しますが、ConfigMap よりも機密性の高い情報を保存するために使用されます

基礎知識： Storage 関連

- Pod が使用する永続領域(Persistent Volume)は２つの方法で準備できます。
  - Static Provisioning
    - 予め準備されている領域を、PersistentVolume として定義しておく方法です。
    - Pod の PersistentVolumeClaims にマッチする PersistentVolume が Pod に割り当てられます。
  - Dynamic Provisioning
    - StorageClass で定義されている provisioner から動的に領域を準備する方法です。
    - Pod の PersistentVolumeClaims で指定された StorageClass から動的に領域が準備され、Pod に割り当てられます。
- 永続領域(Persistent Volume)のマウント方法には以下のものがあります。
  - ReadWriteOnce: 単一の Pod からのみ read-write 可能
  - ReadOnlyMany: 複数の Pod から read のみ可能
  - ReadWriteMany: 複数の Pod から read-write 可能

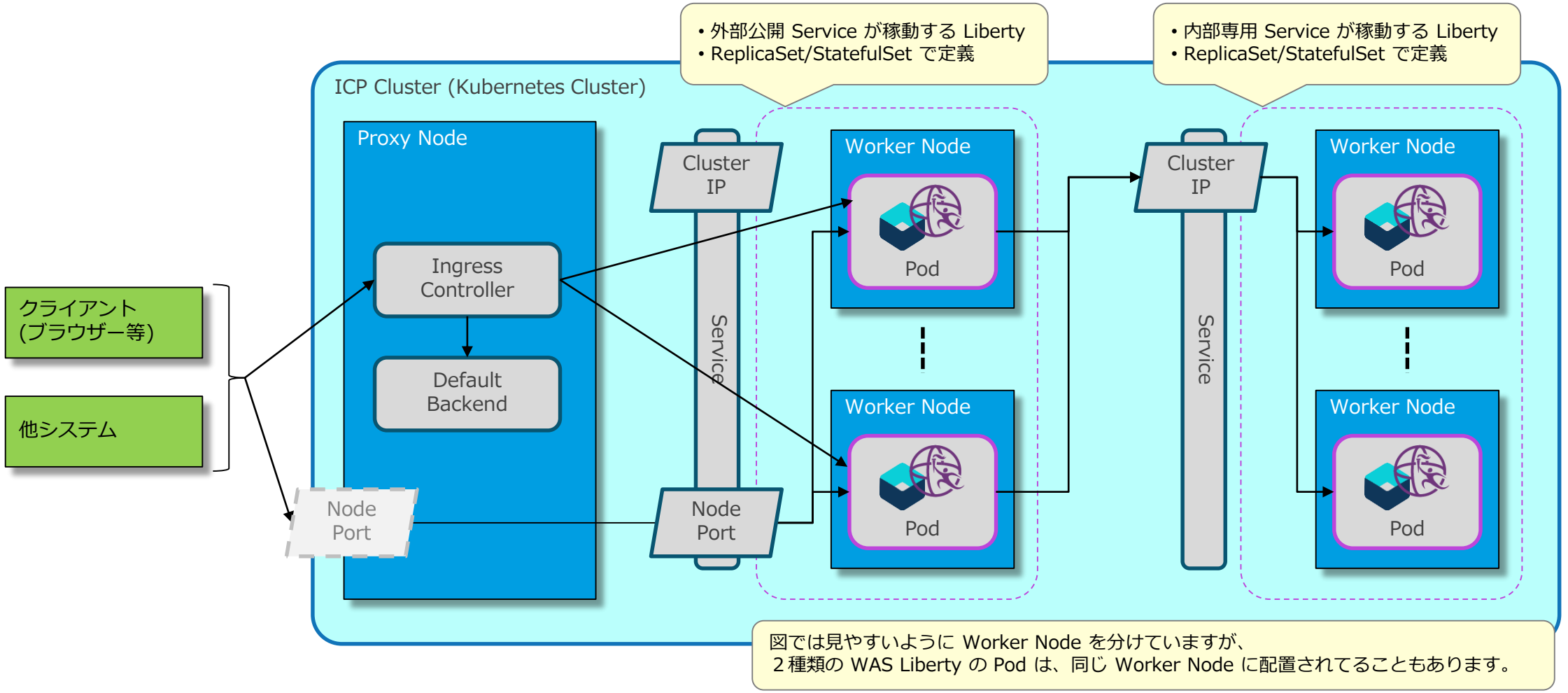
2. ICP 環境での WAS Liberty

ICP 環境での WAS Liberty の可用性と拡張性

- WAS Liberty は Worker Node 上の Pod 内のコンテナーで稼動します
- WAS Liberty の可用性・拡張性は ICP の機能で確保します
  - WAS Liberty の Network Deployment 版が提供する Collective 等の機能は使用しません
- 以下の機能は ICP の機能で実現します
  - 割振制御・再起動
    - Readiness Probe によってヘルスチェックが行われ、異常がある場合は割振り先(Endpoint)から除去されます
    - Liveness Probe を設定することで、異常がある場合は自動で再起動させることが可能です
  - WAS Liberty のクラスタリング
    - テンプレートに従って Pod を複数作成することで、目的数の Pod を稼動させ、クラスタリングします
    - 負荷分散は ClusterIP/NodePort/Ingress によって行われます (後述)
  - オート・スケーリング
    - HorizontalPodAutoscaler を定義することで、CPU 使用率を使用したオート・スケーリングが行えます
    - 但し、Liberty を Deployment/ReplicaSet としてインストールした場合（ログを永続化していない場合）のみです

ICP 環境で稼動する WAS Liberty へのアクセス：概要

■ 下図は、ICP クラスター内で稼動する２種類の WAS Liberty の Pod へのアクセスを示したものです
- 一方は ICP クラスター外部からアクセスされる Pod で、他方は ICP クラスター内部からのみアクセスされる Pod です



ICP 環境で稼動する WAS Liberty へのアクセス：外部からのアクセス

- ICP クラスター外からのアクセスには２つの方法があります。

- Ingress Controller を経由したアクセス
  - Ingress Controller (Proxy Node 上で稼動しリバース・プロキシーの役目を担う)を経由して、HTTP(S) で WAS Liberty の Pod にアクセスできます
  - ヘルスチェックと負荷分散は ICP の機能によって行われます
  - HTTP セッションのアフィニティーを維持できます（Active Cookie 方式）
  - 割り振り先が無い場合は Default Backend に割り振られます
  - Ingress Controller と Default Backend は nginx をベースとしています

- Node Port を使用したアクセス
  - Service (type: NodePort) を定義すると、Node Port を指定してアクセスできます
    - 外部からのアクセスは、Proxy の VIP と Node Port に割り当てられたポート番号で行います
    - 任意の Worker Node などの IP アドレスでもアクセスできますが、可用性等の観点から使用しません
    - HTTP(S) 以外の通信も可能です
    - Cluster IP も割り当てられるので、クラスター内部では Cluster IP を使用してアクセスできます
  - ヘルスチェックと負荷分散は ICP の機能によって行われます
  - HTTP セッションのアフィニティーは維持できません

ICP 環境で稼動する WAS Liberty へのアクセス：内部からのアクセス

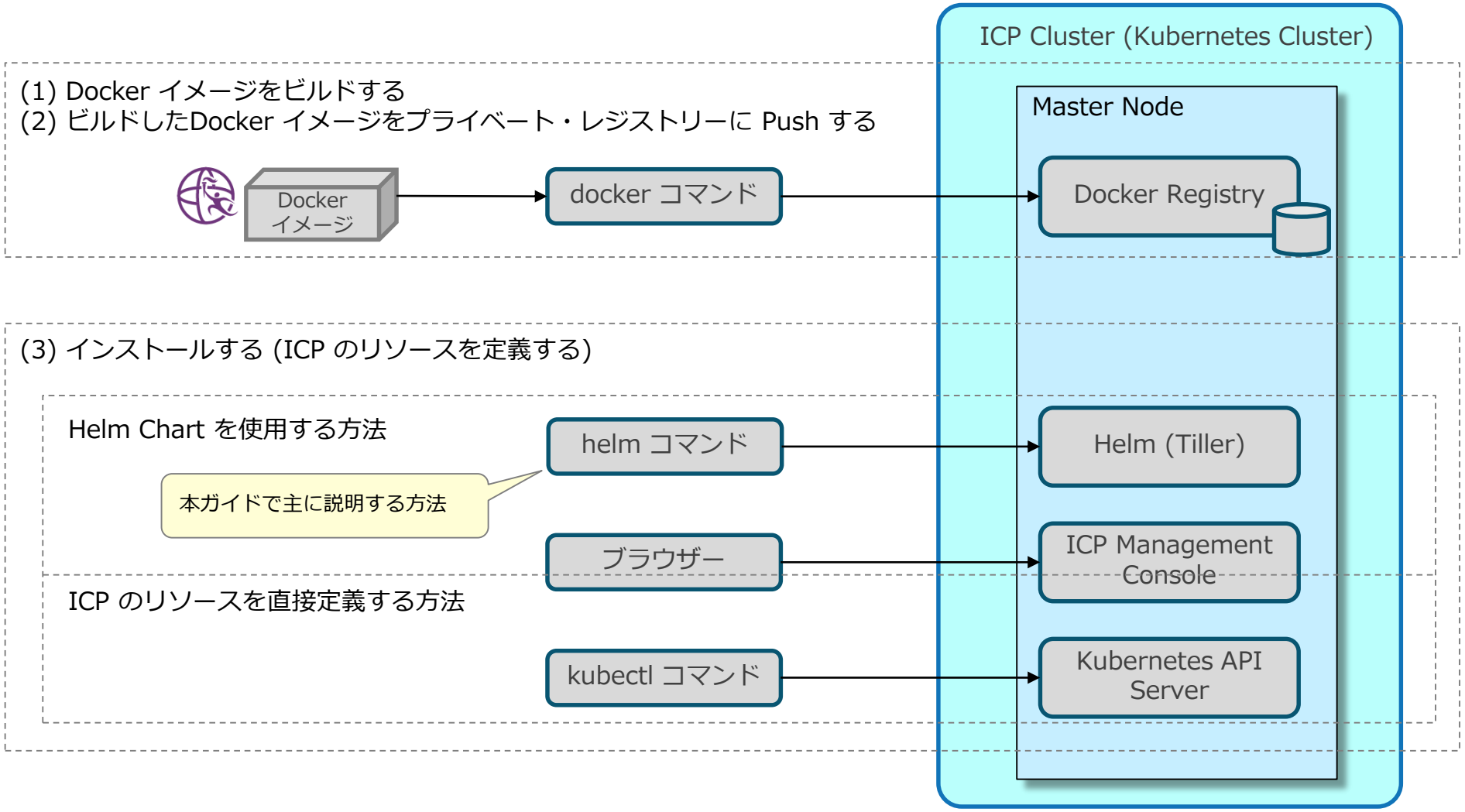
- ICP クラスター内のアクセスには Cluster IP を使用します

- Cluster IP を使用したアクセス
  - Service (type: ClusterIP) を定義すると、Cluster IP を指定してアクセスできます
    - HTTP(S) 以外の通信も可能です
  - アクセスに必要な Cluster IP は、Service 名を使用して DNS より取得できます
  - ヘルスチェックと負荷分散は ICP の機能によって行われます
  - HTTP セッションのアフィニティーは維持できません

3. 基本的なインストール

インストールの流れ (1)

■ WAS Liberty のインストールの流れを下図に示します。



インストールの流れ (2)

(1) Docker イメージをビルドする

- 以下のリソースを組み込んだ、WAS Liberty の Docker イメージをビルドします
  - WAS Liberty の構成ファイル
  - アプリケーションの war ファイルや ear ファイルなど
  - アプリケーションや WAS Liberty の稼動に必要となるその他のファイル：JDBC ドライバー、リソース・アダプター、共有ライブラリー用の jar ファイルなど
- ビルド時の考慮点は後述を参照

(2) ビルドした Docker イメージをプライベート・レジストリーに Push する

- ICP のプライベート Docker レジストリーへ push します
  - 操作に関しては、「1. IBM Cloud Private（ICP）概要」の「ICP の管理操作：補足」に示した情報を参照

(3)インストールする (ICP リソースを定義する)

- Helm Chart を使用する方法
  - IBM 提供の Chart、それを変更して作成した Chart、自作の Chart などを使用してインストールします
  - インストール操作は、helm コマンドで行うか、ICP Management Console から行います
- ICP のリソースを直接定義する方法
  - ICP リソース(ReplicaSet や Service など)の定義情報を yaml または json 形式で記述したファイルを使用して、kubectl apply コマンドでインストールします
  - ICP リソースの定義情報を ICP Management Console の画面で入力することでインストールすることも可能です
- 本ガイドでは、IBM 提供の Chart を利用して、helm コマンドでインストールする方法を主に説明していきます

WAS Liberty の Docker イメージの作成方法

■ Docker イメージの作成方法は２つあります

– IBM 提供の Docker イメージから作成する方法

- 提供されている Docker イメージには、以下の設定が施されています

  - WAS Liberty を /opt/ibm/wlp にインストール(展開)
  - サーバー defaultServer を作成
  - ログ・ディレクトリーを /logs に変更 (環境変数 LOG_DIR)
  - 出力ディレクトリーを /opt/ibm/wlp/output に変更 (環境変数 WLP_OUTPUT_DIR)
  - ディレクトリー /logs を作成
  - 出力ディレクトリーへのリンク /output を作成
  - 構成ディレクトリーへのリンク /config を作成
  - server run defaultServer コマンドで、WAS Liberty を起動

- このイメージに、WAS Liberty の構成ファイルやアプリケーションなどを組み込むことで、イメージをビルドします

– 独自の Docker イメージを作成する方法

- IBM 提供の Helm Chart は、IBM 提供の Docker イメージと組み合わせて利用することを前提として作成されています
- IBM 提供の Helm Chart と組み合わせて使用する場合は、以下の指定を Dockerfile に組み込む必要があります

```
ENV LOG_DIR /logs
ENV WLP_OUTPUT_DIR /opt/ibm/wlp/output
RUN    mkdir /logs ¥
    && ln -s $WLP_OUTPUT_DIR/defaultServer /output ¥
    && ln -s /opt/ibm/wlp/usr/servers/defaultServer /config
```

IBM 提供の Docker イメージをベースとしたビルド

- 以下のステップを Dockerfile に記述します
  - WAS Liberty の構成ファイル(server.xml など)やアプリケーションをイメージの /config にコピー
  - アプリケーションの稼動に必要となるその他のファイルをイメージにコピー
    - JDBC ドライバー、リソース・アダプター、共有ライブラリー用の jar ファイルなどを
  - WAS Liberty のフィーチャーの追加 （必要な場合）
  - WAS Liberty のライセンスの組み込み
  - 参考：Docker Hub: websphere-liberty
    - https://hub.docker.com/r/_/websphere-liberty/

Dockerfile の一例

```
FROM websphere-liberty:webProfile7

COPY target/liberty/wlp/usr/servers/defaultServer /config/

COPY /IBM/DB2Driver /IBM/DB2Driver

# Install required features if not present
RUN installUtility install --acceptLicense defaultServer

# Upgrade to production license
COPY wlp-???-license.jar /tmp/
RUN java -jar /tmp/wlp-???-license.jar --acceptLicense /opt/ibm ¥
    && rm /tmp/wlp-???-license.jar
```

IBM 提供イメージ
- websphere-liberty

指定できる tag
- kernel
- webProfile6
- webProfile7
- javaee7 (latest)
- MicroProfile
- beta

IBM 提供の Helm Chart を使用したインストール (1)

- helm コマンドで使用する Chart はいくつかの方法で指定できます。
  - 詳細は後述の「補足：helm コマンドの実行方法」を参照
- ここでは、IBM 提供の Helm Chart を展開し、そのディレクトリーを指定してインストールする方法を記載します

(1) IBM 提供の Helm Chart をダウンロードし展開します

– GitHub の IBM Cloud Charts Helm Repository の repo/stable からパッケージングされた Chart ファイル(ibm-websphere-liberty-?.?.?.tgz)をダウンロードします
  - GitHub: IBM/charts (IBM Cloud Charts Helm Repository) の URL： https://github.com/IBM/charts

– ダウンロードしたファイルを展開します

```
# tar xvf ibm-websphere-liberty-1.4.0.tgz
```

(2) Chart に定義されたパラメーターの値を指定するファイル(以下、valueFile)を作成します

  - ここでは、Chart 内の values.yaml ファイルをコピーし、必要な部分を変更します

```
# cp ./ibm-websphere-liberty/values.yaml ./hogehoge.yaml

# vi ./hogehoge.yaml
```

IBM 提供の Helm Chart を使用したインストール (2)

(3) valueFile を編集し、必要なパラメーターの値を変更します。

- 詳細は「IBM 提供の Helm Chart のパラメーター：基本的なもの」を参照

(4) helm install コマンドを実行しインストールします

- 実行時に、リリース名、valueFile、Chart を展開したディレクトリーを指定します
- ICP の場合は、--tls を指定して TLS (SSL) で通信する必要があります
- helm コマンドのセットアップに関しては、「1. IBM Cloud Private（ICP）概要」の「ICP の管理操作：補足」に示した情報を参照してください

```
# helm install --name hogehoge -f ./hogehoge.yaml ./ibm-websphere-liberty --tls
```

■ WAS Liberty は Deployment(ReplicaSet) または StatefulSet としてインストールされます

- ログの永続化を指定しなかった場合は ReplicaSet (Deployment) として、ログの永続化を指定した場合は StatefulSet としてインストールされます
- 詳細は「4. ログの永続化と転送」の章を参照

補足：helm コマンドの実行方法

- helm コマンドを使用したインストール/アップグレード操作では、使用する Chart を以下の方法で指定できます
  - *参照:Helm Docs: helm install (https://docs.helm.sh/helm/#helm-install)*

• Chart 参照 (*) :	helm install stable/mariadb
• パッケージングされた Chart ファイル :	helm install ./nginx-1.2.3.tgz
• 展開された Chart のディレクトリー :	helm install ./nginx
• Chart の URL :	helm install https://example.com/charts/nginx-1.2.3.tgz
• Chart 参照とリポジトリーの URL :	helm install –repo https://example.com/charts/ nginx

  - *(*) Chart 参照は helm コマンドを実行したユーザー環境に登録されている Chart を示すものです。ICP の Helm Chart Catalog に登録されている Chart を参照するものではありません。*

- Chart 参照を利用してインストールする場合は、以下の準備が必要です
  - ICP の Internal Helm Repository 内の Chart を利用する場合は、「ICP の管理操作：補足」に示した情報を参照にして、ICP の Internal Helm Repository をユーザー環境に追加します
  - その他のリポジトリー（IBM/charts など）内の Chart を利用する場合は、以下のようなコマンドでリポジトリーをユーザー環境に追加します

```
# helm repo add ibm-charts https://raw.githubusercontent.com/IBM/charts/master/repo/stable/
```

IBM/charts の リポジトリーを追加する例

補足：ICP Management Console で Helm Chart を使用する場合

- ICP Management Console で Helm Chart を扱うためには、ICP の Helm Chart Catalog に Chart を追加する必要があります。

- 新しい Chart や最新バージョンの Chart を ICP Management Console で使用する場合は、以下手順で ICP の Helm Chart Catalog に取り込む必要があります。

  – リモートのリポジトリーに登録されている Chart を追加・最新化する場合
    - ICP Knowledge Center: Managing Helm repositories
      – https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/app_center/manage_helm_repo.html

  – 作成した Chart を使用する場合
    - ICP Knowledge Center: Adding custom applications
      – https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/app_center/add_package.html

  – 製品提供の Chart をオフライン環境の ICP で利用する場合
    - ICP Knowledge Center: Adding featured applications to clusters without internet connectivity
      – https://www.ibm.com/support/knowledgecenter/SSBS6K_2.1.0.3/app_center/add_package_offline.html

IBM 提供の Helm Chart のパラメーター：基本的なもの (1)

■ IBM 提供 の Liberty 用 Helm Chart のパラメーターのうち、基本的なものを以下に示します。

Qualifier	Parameter	説明
image	repository	Docker イメージの名前を指定します。 ここで指定した Docker イメージが Pod 内のコンテナーで稼動することになります。アプリなどを組み込み、ビルドした Docker イメージを指定します。 通常、「<クラスター名>:<Docker レジストーのポート番号>/<ネームスペース>/<名前>」の形式になります。 例：mycluster.icp:8500/default/my-app
	tag	Docker イメージを識別するタグを指定します。
	pullPolicy	Docker イメージをリポジトリーから pull するポリシーを指定します。 使用する Docker イメージはプライベート・レジストーに登録済みなので、通常は、「IfNotPresent」を指定します。
	license	WAS Liberty のライセンスを所有している場合は、「accept」を指定します。 指定しないと、開発ライセンスでの使用となります。
service	type	service の type を指定します。 ICP で現在指定できるのは、ClusterIP または NodePort です。 詳細は、「ICP 環境で稼動する WAS Liberty へのアクセス」を参照してください。
	name	service の port 名を指定します。
	port	WAS Liberty のポート番号を指定します。デフォルトの場合、9080 (HTTP) または 9443 (HTTPS) となります。
	targetPort	Pod が WAS Liberty のポートを公開するために使用するポート番号を指定します。 Cluster IP とこのポート番号を使用して、ICP クラスター内からアクセスできるようになります。
ssl	enabled	service として公開したポートで、SSL (HTTPS) が有効になっているか否かを指定します。
	useClusterSSLConfiguration	クラスター SSL 構成の使用有無を指定します。使用しない場合は false を指定します。(詳細は後述)
	createClusterSSLConfiguration	クラスター SSL 構成を作成するように指示します。通常時は false を指定します。（詳細は後述）

IBM 提供の Helm Chart のパラメーター：基本的なもの (2)

Qualifier	Parameter	説明
ingress	enabled	Ingress Controller (Proxy Node 上で稼動するリバース・プロキシー)経由でのアクセス有無を指定します。 詳細は、「ICP 環境で稼動する WAS Liberty へのアクセス」を参照してください。
	secureBackends	Ingress と WAS Liberty の間の通信をセキュアにするか否かを指定します。 （service として公開したポートで、SSL (HTTPS) が有効になっているか否かを指定します。）
	rewriteTarget	以下の path で指定するパスの、書き換え先のパスを指定します。 WAS Liberty へのリクエストは、ここで指定したパスに書き換えられて転送されることになります。
	path	Ingress Controller は、このパス配下へのアクセスを WAS Liberty に転送します。
microprofile	health.enabled	true を指定すると、WAS Liberty のコンテナーに対する Readiness Probe のパスが / から /health に変わります。 主に、MicroProfile Health に対応したマイクロサービスを WAS Liberty で稼動させる場合に true を指定します。
replicaCount		Pod のレプリカ数を指定します。ここで指定した数の Pod が同時に稼動するようになります。 Pod の可用性を確保する場合は、最低でも 2 以上を指定することになります。さらに、同時リクエスト数（ワークロード）に応じて、レプリカ数を増やします。 尚、autoscaling を有効にした場合は、ここで指定したレプリカ数は無視されます。
persistence	----	後述
logs	----	後述
autoscaling	----	後述
resources	----	後述

Helm Chart が作成したリソースの確認

■ 作成されたリソースの一覧と状況は、helm status コマンドで確認できます

```
# helm status hogehoge --tls
```

helm status コマンドの出力例

```
LAST DEPLOYED: Thu Jun 28 02:51:39 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
NAME                       TYPE       CLUSTER-IP  EXTERNAL-IP  PORT(S)   AGE
hogehoge-ibm-websphere-l   ClusterIP  10.0.0.85   <none>       9080/TCP  1m

==> v1beta1/Deployment
NAME                       DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
hogehoge-ibm-websphere-l   2        2        2           1          1m

==> v1beta1/Ingress
NAME                       HOSTS  ADDRESS       PORTS  AGE
hogehoge-ibm-websphere-l   *      10.132.2.237  80     1m

==> v1/Pod(related)
NAME                                    READY  STATUS   RESTARTS  AGE
hogehoge-ibm-websphere-l-fd76767f8-cdztg  0/1    Running  0         1m
hogehoge-ibm-websphere-l-fd76767f8-hmdc4  1/1    Running  0         1m


NOTES:
1. Get the application URL by running these commands:
  export INGRESS_IP=$(kubectl get ing --namespace default hogehoge-ibm-websphere-l -o
jsonpath="{.status.loadBalancer.ingress[0].ip}")
  export APP_PATH=/infra-test-ingress
  echo https://$INGRESS_IP$APP_PATH
```

サービスの名前、Cluster IP、ポート

Deployment の名前と状況

Ingress の名前と概要

Pod の名前と状況

Helm Chart を使用した運用操作

■ helm コマンドを使用して Pod の運用ができます

- 以下の例では変更内容を示すために --set で値を指定していますが、変更内容を残す必要がある場合は、valueFile の値を更新して実行する方法をお勧めします。

– レプリカ数の増減

```
# helm upgrade hogehoge -f ./hogehoge.yaml --set replicaCount=3 ./ibm-websphere-liberty --tls
```

helm upgrade で変更できる内容には制限があります

– 全 Pod の停止（レプリカ数を 0 に設定）

```
# helm upgrade hogehoge -f ./hogehoge.yaml --set replicaCount=0 ./ibm-websphere-liberty --tls
```

– Docker イメージの更新

```
# helm upgrade hogehoge -f ./hogehoge.yaml --set image.tag=v200 ./ibm-websphere-liberty --tls
```

- イメージの名前やタグを変更すると、strategy/updateStrategy に従って、Pod が更新されます (strategy/updateStrategy の変更に関しては後述)
  - RollingUpdate: ローリング・アップデートが行われ直ぐにイメージが更新されます
  - OnDelete: Pod が削除されるまで古いイメージのまま動き続けます

ReplicaSet

```
spec:
  ……
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

StatefulSet

```
spec:
  ……
  updateStrategy:
    type: OnDelete
```

Helm Chart を使用した運用操作 （続き）

– リリースされている内容/valueの表示

```
# helm get hogehoge --tls
```

```
# helm get values hogehoge --tls
```

– 更新履歴の確認

```
# helm history hogehoge --tls
```

– 履歴内のリビジョンを復元

```
# helm rollback hogehoge 2 --tls
```

復元するリビジョンの番号(左記の例では2)を指定して実行する

– アンインストール

- 履歴を残している場合は、helm rollback コマンドで復元できます
- 履歴が残っていると、同じリリース名を指定してインストール(helm install)できません

```
# helm delete hogehoge --tls
```

履歴を残す場合

```
# helm delete hogehoge --purge --tls
```

履歴も含め完全に削除する場合

4. ログとダンプの永続化と転送

WAS Liberty が出力するログとダンプ

- WAS Liberty が出力するログとダンプを以下に示します
  - 以下の URL の情報も参照してください。
    - developerWorks: WAS V9 Liberty基盤設計セミナー資料 ー「8. パフォーマンス/問題判別」
      - https://www.ibm.com/developerworks/jp/websphere/library/was/liberty_v9_infradesign/

(*) IBM 提供の Docker イメージを使用し、出力先をカスタマイズしていない場合の出力先です。

ログ	説明	デフォルトの出力先 (*)
console.log	コンソール出力が書き込まれるログです。IBM 提供の Docker イメージでは server run コマンドでサーバーを起動しているので、ファイルには出力されません。	コンソール
messages.log	WAS Liberty からのログ出力が記録されます。 アプリケーションが System.out および System.err に出力した内容も記録されます。	/logs/messages*.log
trace.log	詳細ログを指定した場合にトレースが記録されます。	/logs/trace*.log
FFDC ログ	FFDC ログです。WAS Liberty 内部のシステム・エラー情報を詳細に記録します。	/logs/ffdc/*.log
GC ログ	冗長ガーベッジ・コレクション(Verbose GC)を有効にすると出力されます。デフォルトでは出力されません。 -Xverbosegclog を指定しないとコンソール(console.log)へ出力されます。 -Xverbosegclog で出力先ファイルを指定しないと、/output/verbosegc.*.txt へ出力されます。	(コンソール) または (/output/verbosegc.*.txt)
http_access.log	WAS Liberty の HTTP トランスポートのアクセス・ログです。デフォルトでは出力されません。 出力先を指定しないと、/output/logs/http_access*.log に記録されます。	(/output/logs/http_access*.log)
トランザクション ログ	XA トランザクション（2 フェーズ・コミット, 2PC）の管理用のログです。	/output/tranlog/*/log*

ダンプ	説明	デフォルトの出力先 (*)
Java Dump	Java Core とも呼ばれます。問題判別用に JVM の内部情報がテキスト形式で出力されます。	/output/javacore.*.txt
Heap Dump	ヒープ中のオブジェクトの種類、サイズ、アドレス参照関係などの情報がバイナリー形式で出力されます。	/output/heapdump.*.phd
System Dump	システム・ダンプ(core)です。出力先は OS 設定に依存します。	n/a

ログとダンプの永続化が必要になるケース

- XA トランザクション（2 フェーズ・コミット, 2PC）を使用している場合
  - インダウト状態のトランザクションを回復するには、トランザクション・ログが必須になります
- 障害発生時にログとダンプを確実に残しておきたい場合
  - Pod のコンテナー内に出力されたログやダンプは、Pod が停止すると消えてしまい、障害発生後は確認できない可能性があります
  - 確実に残しておくには永続化が必須となります
- パフォーマンス管理に必要なログを従来と同じ形式で残したい場合
  - 従来から使用しているパフォーマンス分析ツールなどを継続利用したい場合、従来と同じ形式で残しておく必要があります。
- 従来からの出力形式でのログ保存が必須の場合
  - アクセス・ログなど、特定の形式でのログ保存が要件となっていることも想定されます

ログとダンプの永続化方法

- ログとダンプを永続化させる方法を以下に示します。
- ログの永続化
  - Helm Chart のパラメーター設定で永続化できるログ (ログの出力先をデフォルトから変更しない前提)
    - /logs 内に出力されるログ： messages.log，trace.log，FFDC ログ
    - /output/tranlog 内に出力されるログ： トランザクション・ログ
  - 出力先を /logs 内に指定することで永続化できるログ
    - http_access.log
    - GC ログ
- ダンプの永続化
  - ログの永続化用の領域(/logs)を流用することで永続化する方法
    - Helm Chart のパラメーター設定でログを永続化させ、/logs にダンプを出力するように WAS Liberty を構成します
    - ダンプによって、ログ出力領域が圧迫される可能性があります
  - 専用の永続化用の領域を設ける方法
    - Helm Chart を変更し、専用の領域を設ける方法が考えられます
    - ダンプによるログ出力領域の圧迫を抑止できます

ログの永続化：永続領域(Persistent Volume)の準備

■ ログの永続化に必要な永続領域(Persistent Volume)は、以下の何れかの方法で準備します。

– Dynamic Provisioning を使用する場合
  • ICP クラスターに定義されている StorageClass を確認します。(詳細は ICP クラスターの管理者に確認)

– Static Provisioning を使用する場合は、必要な数の PersistentVolume (PV) を定義します
  • ICP Management Console で定義する場合は、[Menu > Platform > Storage]と選択して[Storage]画面に進み、[Create PersistentVolume]ボタンを押下して作成します。(定義内容は yaml ファイルの例を参照)
  • kubectl apply コマンドで作成する場合は、以下のような yaml ファイルを使用します
    – *kubectl コマンドのセットアップに関しては、「1. IBM Cloud Private (ICP) 概要」の「ICP の管理操作：補足」に示した情報を参照してください*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hogehoge-pv-1
  labels:
    type: hogehoge-pv
spec:
  storageClassName: was-logs
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /nfs/hogehoge-pv-1
    server: 10.20.30.40
```

PV の名前です。定義する PV 毎に変えます。

インストール時に label が指定された場合は、PVC とのマッチ条件の 1 つとなります。

PV の StorageClassName を指定します。
インストール時に StorageClassName が指定された場合は、PVC とのマッチ条件の 1 つとなります。

storage のサイズは用途に応じて調整します。PVC とのマッチ条件の 1 つです。

ログ永続化用の PV は 1 つの Pod が専有するので、ReadWriteOnce を指定します。

PersistentVolumeClaim が削除されたときの動作を指定します。
NFS を PV として使用する場合、有効な指定は Retain のみです。

NFS のディレクトリーを PV として定義する場合の例です。
path に指定したディレクトリーは予め作成しておく必要があります。path は PV 毎に変えます。

ログの永続化：トランザクション・ログの永続化

- WAS Liberty のトランザクション・ログを永続化する場合は以下の考慮が必要です
  - server.xml に以下の指定を追加します。
    - WAS Liberty の起動時にインダウトの解消が開始され、完了するまでアプリが始動されなくなります。

```
<transaction recoverOnStartup="true" waitForRecovery="true" />
```

  - IBM 提供の Helm Chart は、デフォルトのログ配置を想定しています。
    transactionLogDirectory は変更できません。
    - デフォルトのトランザクション・ログの配置： ${server.output.dir}/tranlog/
  - WAS Liberty のトランザクション関連の設定に関しては、以下の URL の情報も参照してください。
    - developerWorks: WAS V9 Liberty基盤設計セミナー資料 ー「6. 外部連携」の「トランザクション」の部分
      - https://www.ibm.com/developerworks/jp/websphere/library/was/liberty_v9_infradesign/

ログの永続化：インストールと運用

■ IBM 提供の Helm Chart のパラメーターに設定する値

– 以下の指定を行います
  • persistence.* でログ用の永続領域の情報を指定します
  • logs.persistLogs および logs.persistTransactionLogs で永続化するログを指定します
– 詳細は次ページの「IBM 提供の Helm Chart のパラメーター：ログ関連」を参照してください

■ インストール・運用

– ログを永続化しない場合と同様ですが、以下の差異があります
  • Pod は StatefulSet としてインストールされ、Pod の名前はインデックスで識別される永続的な名前となります

```
# kubectl get pod
NAME                                  READY     STATUS    RESTARTS   AGE
hogehoge-ibm-websphere-l-0            1/1       Running   0          1m
hogehoge-ibm-websphere-l-1            1/1       Running   0          1m
```

  • Pod はインデックス順に順次起動されます （デフォルトの podManagementPolicy が OrderedReady のため）
  • Headless Service が追加で作成されます（Headless Service が StatefulSet を使用する場合の前提のため）

```
# kubectl get service
NAME                          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
hogehoge-ibm-websphere-l      ClusterIP   None          <none>        9080/TCP         1m
hogehoge-ibm-websphere-l-np   ClusterIP   10.0.0.204    <none>        9080/TCP         1m
```

サフィックス –np が付く方が
Pod アクセス用です

IBM 提供の Helm Chart のパラメーター：ログ関連

■ ログの永続化に関連するパラメーター

Qualifier	Parameter	説明
persistence	name	Persistence Volume Claim (PVC) の名称の接頭語として使用されます。
	useDynamicProvisioning	Dynamic Provisioning を使用する場合は true を、Static Provisioning を使用する場合は false を指定します。
	storageClassName	使用する StorageClass を指定します。 （Dynamic Provisioning の場合は、使用する StorageClass の名前を指定します。） （Static Provisioning の場合は、使用する PV に定義されている StorageClassName の値を指定します。）
	selector.label	Static Provisioning の場合に、使用する PV を選択するための、ラベルの名前を指定します。 （使用する PV に定義されているラベルの名前を指定します。）
	selector.value	Static Provisioning の場合に、使用する PV を選択するための、ラベルの値を指定します。 （使用する PV に定義されているラベル(selector.labelで指定したラベル)の値を指定します。）
	size	永続領域のサイズを指定します。（単位指定：10のn乗 E, P, T, G, M, K または 2のn乗 Ei, Pi, Ti, Gi, Mi, Ki）
logs	persistLogs	WAS Liberty のログ出力領域を永続領域上に割り当てる場合は true を選択します。 true を指定すると、Docker イメージの /logs に永続領域内の logs ディレクトリーがマウントされます。
	persistTransactionLogs	WAS Liberty のトランザクション・ログ出力領域を永続領域上に割り当てる場合は true を選択します。 true を指定すると、Docker イメージの /output/tranlog に永続領域内の tranlog ディレクトリーがマウントされます。

■ コンソール・ログ出力に関連するパラメーター

Qualifier	Parameter	説明
logs	consoleFormat	WAS Liberty のコンソール・ログのフォーマットを指定します。 json 形式(Helm Chart のデフォルト)、または、従来からの出力形式である basic 形式が指定できます。
	consoleLogLevel	WAS Liberty のコンソール・ログの出力レベルを指定します。 info (Helm Chart のデフォルト), audit, warning, error または off の何れかを指定します。
	consoleSource	WAS Liberty のコンソール・ログに出力する内容を message, trace, accessLog, ffdc の組み合わせで指定します。 Helm Chart のデフォルトは、message,trace,accessLog,ffdc です。consoleFormat が json の場合にのみ有効です。 アクセス・ログをコンソール・ログとして出力するには、WAS Liberty 側の構成も必要になります。

ログの永続化：永続領域の管理 (1)

- Pod が起動する時に PersistentVolumeClaim (PVC) が作成され、Claim にマッチする PersistentVolume (PV) と関連付けられます。
  - kubectl get pv コマンドで PV の状態を、kubectl get pvc コマンドで PVC の状態を確認できます

右記の PVC に関連付けられている状態

PVC に関連付けられておらず、割り当て可能な状態

```
# kubectl get pv
NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
hogehoge-pv-1                 1Gi        RWO            Retain           Available
hogehoge-pv-2                 1Gi        RWO            Retain           Bound       default/hogehoge-pvc-hogehoge-ibm-websphere-l-1
hogehoge-pv-3                 1Gi        RWO            Retain           Bound       default/hogehoge-pvc-hogehoge-ibm-websphere-l-0
```

```
# kubectl get pvc
NAME                                          STATUS    VOLUME           CAPACITY   ACCESS MODES
hogehoge-pvc-hogehoge-ibm-websphere-l-0       Bound     hogehoge-pv-3    1Gi        RWO
hogehoge-pvc-hogehoge-ibm-websphere-l-1       Bound     hogehoge-pv-2    1Gi        RWO
```

右記の PV に関連付けられている状態

  - ICP Management Console の場合は、[Menu > Platform > Storage]と選択して[Storage]画面に進み、該当のタブを選択することで、PV と PVC の状態を確認できます。

ログの永続化：永続領域の管理 (2)

- 以下の操作を行っても、作成された PVC は残り、PV との関連付けも残ります（Pod の状態を保持し続けるため）
  - Pod のレプリカ数の削減
  - アンインストール (helm delete --purge)

- 不要になった PVC や、PV との関連付けは手動で削除する必要があります
  - PVC の削除は kubectl delete pvc コマンドで、PV との関連付けは kubectl edit pv コマンドで行います

```
# kubectl delete pvc hogehoge-pvc-hogehoge-ibm-websphere-l-0
persistentvolumeclaim "hogehoge-pvc-hogehoge-ibm-websphere-l-0" deleted

# kubectl edit pv hogehoge-pv-2
persistentvolume "hogehoge-pv-2" edited
```

表示されたエディター画面で spec 内の claimRef の定義を削除して、保存します

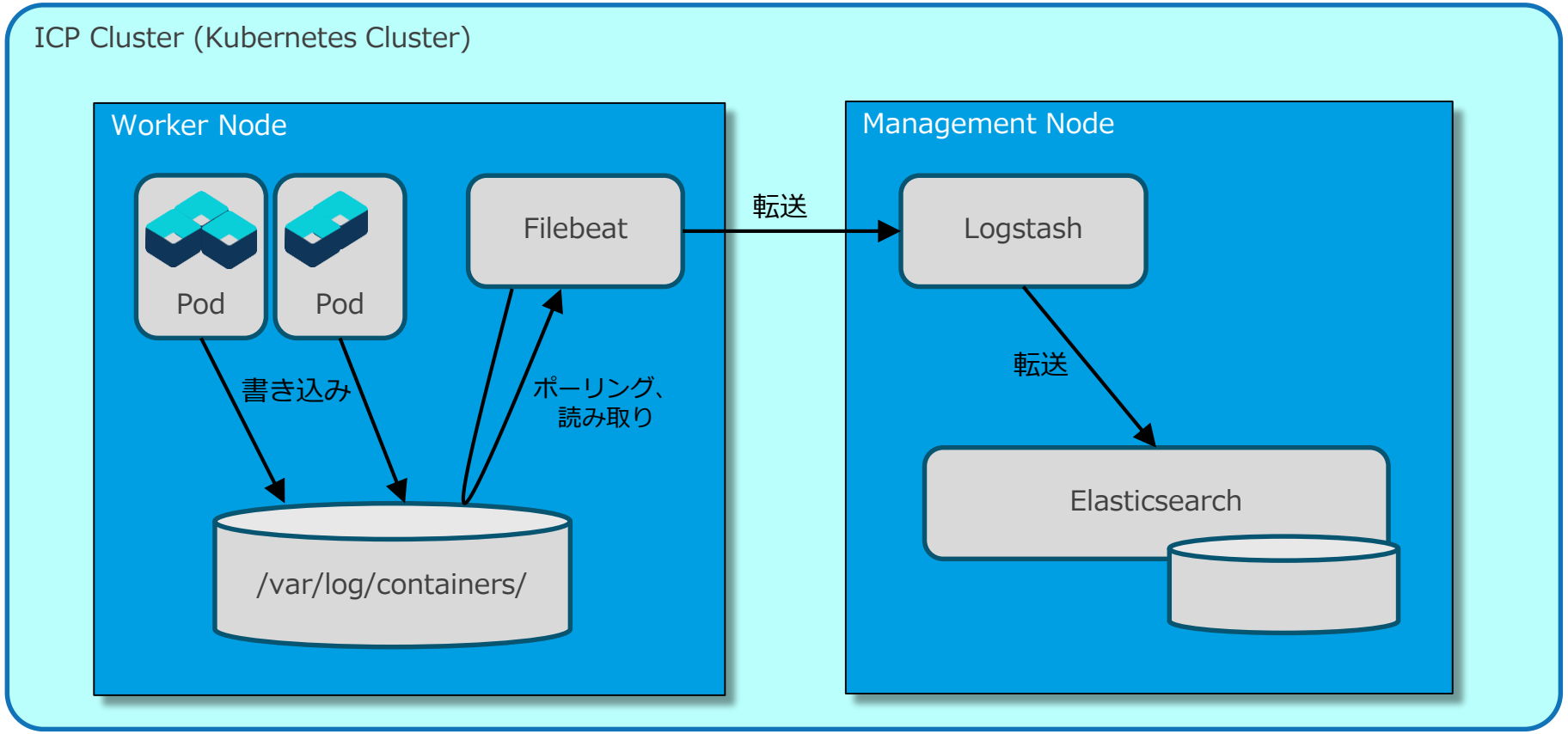
delete/edit 対象を間違えないように、十分注意してください

```
… … … …
spec:
  … … … …
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: hogehoge-pvc-hogehoge-ibm-websphere-l-1
    namespace: default
    resourceVersion: "11180941"
    uid: 59bba296-62f9-11e8-ae22-06fbcac104c4
  … … … …
```

  - ICP Management Console の場合は、前述の画面で行います

ログの転送：ICP のログ転送機能

- Pod 内で稼動するコンテナーのコンソール出力は、Management Node で稼動する Logstash に自動的に転送され、Elasticsearch に保管されます。



- WAS Liberty のコンソール出力も、自動的に Elasticsearch に保管されます

ログの転送：転送するログの設定

- IBM 提供の Helm Chart のパラメーターに設定する値
  - 以下のパラメーターで調整できます。デフォルトで、転送可能な全てのログが転送される設定になっています。
    - logs.consoleFormat
    - logs.consoleLogLevel
    - logs.consoleSource
  - 詳細は前述の「IBM 提供の Helm Chart のパラメーター：ログ関連」を参照してください

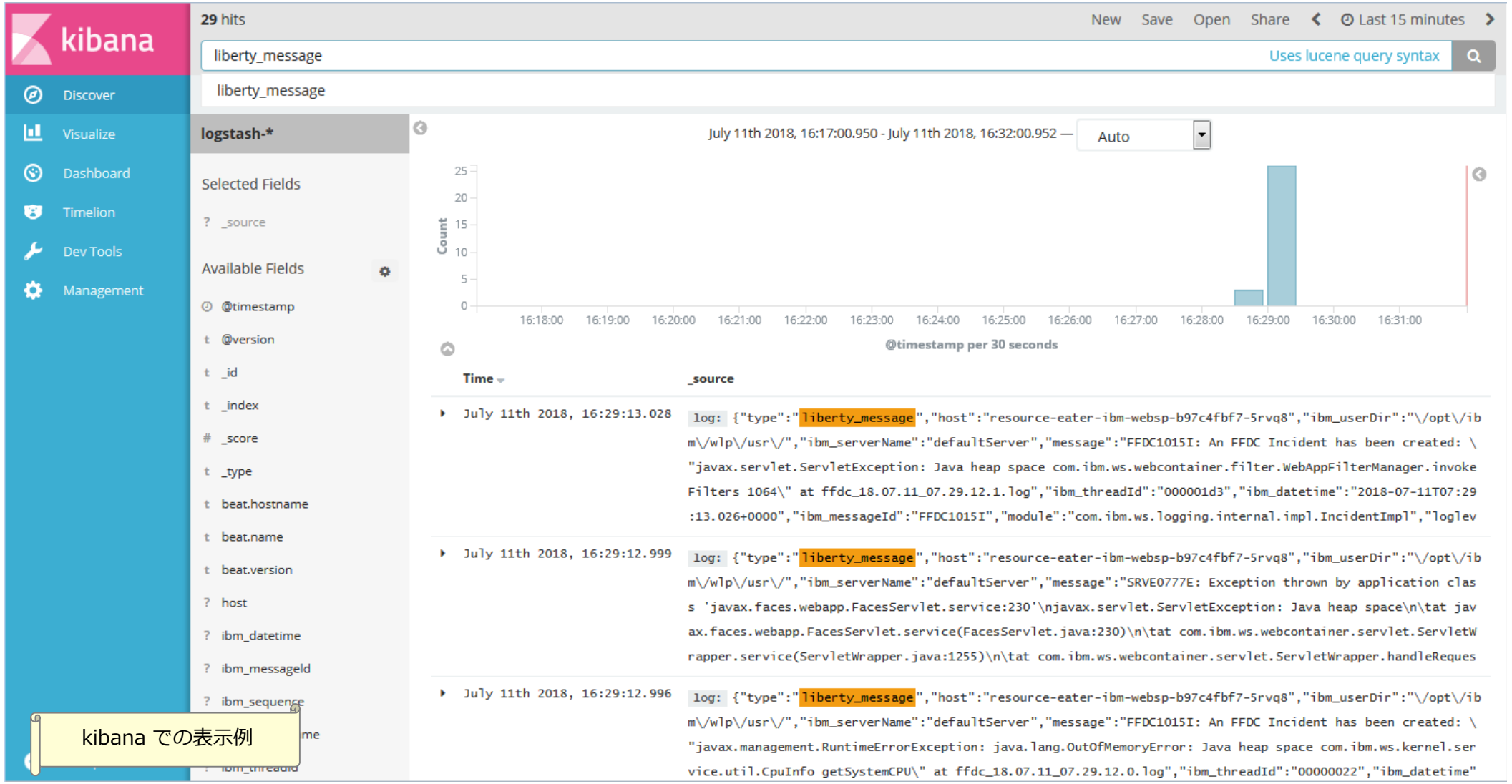
- アクセス・ログを転送するには、WAS Liberty の設定でアクセス・ログ出力を有効にする必要があります。

```
<httpAccessLogging id="accessLogging" filePath="${env.X_LOG_DIR}/http_access.log"/>
<httpEndpoint id="defaultHttpEndpoint" accessLoggingRef="accessLogging"/>
```

- 参考情報
  - WAS Liberty Knowledge Center: HTTP アクセスのロギング
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_http_accesslogs.html
  - WAS Liberty Knowledge Center: ロギングおよびトレース
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html

ログの転送：転送されたログの表示

- Elasticsearch へ転送されたログは kibana で確認できます

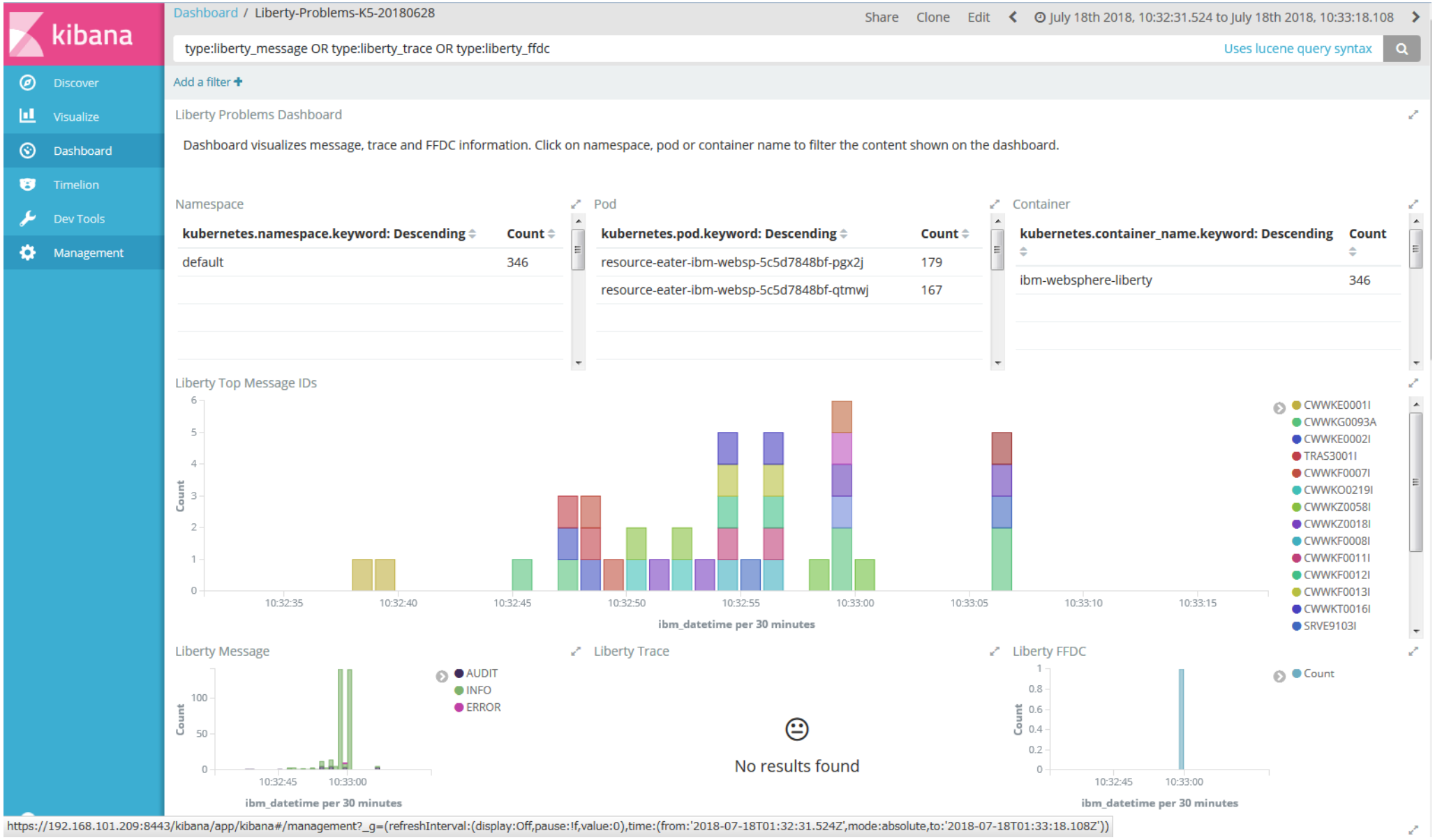


ログの転送：サンプル・ダッシュボードの利用

- サンプル・ダッシュボードを Kibana にインポートすると、Liberty のログをビジュアルに表示できるようになります
  - サンプル・ダッシュボード
    - GitHub: WASdev/sample.dashboards/IBMCloudPrivate_2.1.0.2/Kibana_5.x/
      - https://github.com/WASdev/sample.dashboards/tree/master/IBMCloudPrivate_2.1.0.2/Kibana_5.x
  - 以下の２つのダッシュボードが提供されています（次ページの表示例を参照）
    - Liberty Problems Dashboard　メッセージ、トレース、FFDC の情報をビジュアル化するダッシュボード
    - Liberty Traffic Dashboard　アクセス・ログをビジュアル化するダッシュボード

- 補足
  - Liberty のログが Elasticsearch に格納されていない状態でサンプル・ダッシュボードをインポートすると、インポート時にエラー・メッセージが表示されます
    - アクセス・ログが Elasticsearch に送られていない場合、Liberty Traffic Dashboard の定義に対してエラーが表示されますが、Liberty Problems Dashboard は正しく表示できます
  - サンプル・ダッシュボードがうまく表示できない場合は、kibana で以下の操作を行ってください
    1. Liberty のログが表示できることを確認する
    2. インデックス・パターンを削除し、再作成する（Time Filter フィールド名として ibm_timestamp を指定する必要があります）
    3. ダッシュボード定義を再インポートする

ログの転送：Liberty Problems Dashboard の表示例



ログの転送：Liberty Traffic Dashboard の表示例

5. クラスター SSL 構成の利用

クラスター SSL 構成とは

- クラスター SSL 構成を利用すると、ICP クラスター内で稼動する WAS Liberty が、共通の SSL 構成情報を使用できるようになります
- クラスター SSL 構成は、以下の要素で構成されます

要素	格納方法	Secret または ConfigMap の 名前	/ キー	説明
鍵ストア	Secret	mb-keystore	key.jks	WAS Liberty が SSL 通信で使用する鍵が格納されているファイルです。
鍵ストアの パスワード	Secret	mb-keystore-password	password	鍵ストアのパスワードです。 WAS Liberty が提供する securityUtility encode コマンドを使用して xor 方式等でエンコードしたものを格納するようにします。
トラスト・ストア	Secret	mb-truststore	trust.jks	信頼できる署名者証明書(証明書の署名検査に使用される証明書)が格納されているファイルです。
トラスト・ストアの パスワード	Secret	mb-truststore-password	password	トラスト・ストアのパスワードです。 WAS Liberty が提供する securityUtility encode コマンドを使用して xor 方式等でエンコードしたものを格納するようにします。
WAS Liberty の 構成ファイル	ConfigMap	liberty-config	keystore.xml	鍵ストアとトラスト・ストアの情報が記述されている、WAS Liberty の構成ファイルです。

- 参考：WAS Liberty Knowledge Center: SSL 通信の使用可能化
  - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_sec_ssl.html
- 参考：WAS Liberty Knowledge Center: SSL 構成の属性
  - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_ssl.html

クラスター SSL 構成を有効化した WAS Liberty のインストールと構成

- Chart のパラメーター ssl.useClusterSSLConfiguration に true を指定してインストールすると、WAS Liberty に下記の構成が追加され、クラスター SSL 構成が有効化されます
  (事前に、クラスター SSL 構成を作成しておく必要があります。作成方法は後述。)

  – クラスター SSL 構成のファイルは、以下のパスに配置されます
    - 鍵スト：　/etc/wlp/config/keystore/key.jks
    - トラスト・ストア：　/etc/wlp/config/truststore/trust.jks
    - 構成ファイル：　/config/configDropins/defaults/keystore.xml

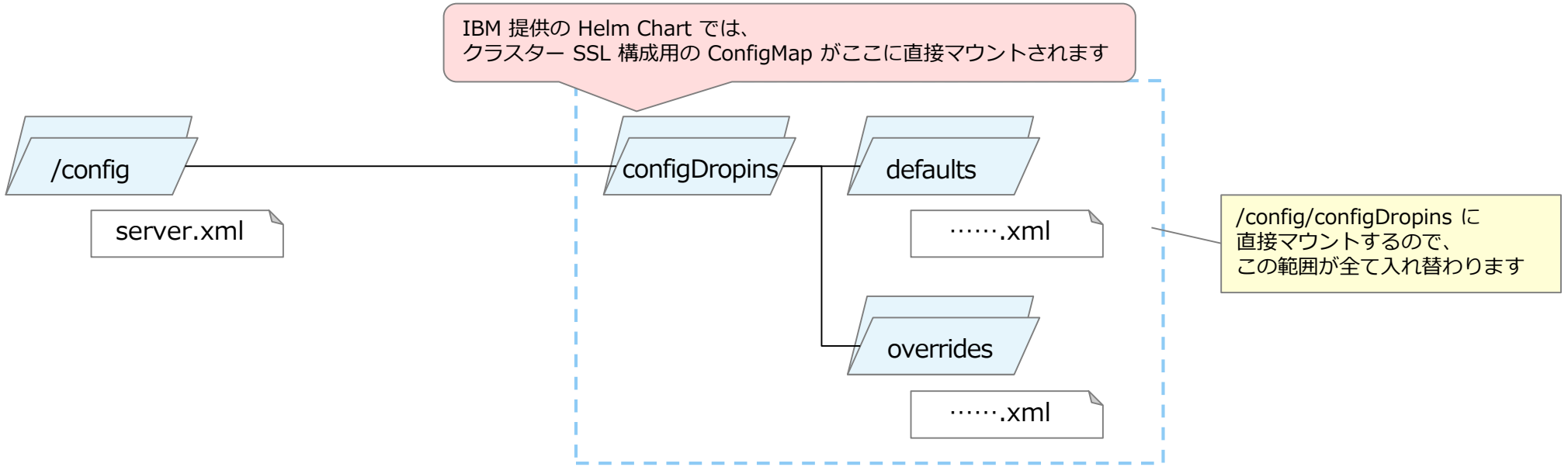
構成ファイル keystore.xml の内容

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" trustStoreRef="defaultTrustStore"/>
  <keyStore id="defaultKeyStore"   location="/etc/wlp/config/keystore/key.jks"     password="${env.MB_KEYSTORE_PASSWORD}"   />
  <keyStore id="defaultTrustStore" location="/etc/wlp/config/truststore/trust.jks" password="${env.MB_TRUSTSTORE_PASSWORD}" />
</server>
```

  – クラスター SSL 構成のパスワードは、以下の環境変数に設定されます
    - 鍵ストアのパスワード：　MB_KEYSTORE_PASSWORD
    - トラスト・ストアのパスワード：　MB_TRUSTSTORE_PASSWORD

クラスター SSL 構成を有効化する場合の注意点

- クラスター SSL 構成を有効化する際は、以下の点に注意が必要です
  - クラスター SSL 構成用の ConfigMap が /config/configDropins にマウントされるため、Docker イメージ内の /config/configDropins 配下に格納された構成ファイルは使われなくなります



  - クラスター SSL 構成用の構成ファイル keystore.xml は、/config/configDropins/defaults に配置されるので、server.xml 内に重複した定義が存在すると無視されてしまい、クラスター SSL 構成が有効化されません

  - 後述の「Helm Chart の作成と改良」と「ConfigMap と Secret の利用」も参照してください

クラスター SSL 構成の作成（１）

- クラスター SSL 構成は、以下のいずれかの方法で作成できます
  尚、この方法で作成される証明書は、有効期限が１年間の自己証明書なので、注意が必要です
  - Chart のパラメーター ssl.createClusterSSLConfiguration に true を指定してインストールを行う
  - Microservice Builder ファブリックをインストールする
    - 補足：いずれの場合も、イメージ ibmcom/mb-tools がジョブとして実行されて、クラスター SSL 構成が生成されます
- クラスター SSL 構成を再作成する場合は、既存の Secret、ConfigMap、および、生成ジョブを以下のコマンドで削除してから行います
  - 参考：WAS Liberty Knowledge Center: IBM Cloud Private 用の Liberty での SSL の使用可能化
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_ssl_helm.html

```
# kubectl delete secrets mb-keystore mb-keystore-password mb-truststore mb-truststore-password

# kubectl delete configmap liberty-config

# kubectl delete job liberty-secret-generator-deploy
```

クラスター SSL 構成の作成（２）

- ジョブが生成した鍵の使用が適さない場合は、鍵ストアとトラスト・ストアを準備し、下記のコマンドでクラスター SSL 構成用の Secret を置き換えます
  - 参考：WAS Liberty Knowledge Center: サード・パーティー証明書を使用した SSL の使用可能化
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_auto_ssl3.html
  - 参考：WAS Liberty Knowledge Center: securityUtility コマンド
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_securityutil.html

```
# kubectl delete secrets mb-keystore mb-keystore-password mb-truststore mb-truststore-password

# kubectl create secret generic mb-keystore   --from-file=key.jks=./key.jks
# kubectl create secret generic mb-truststore --from-file=trust.jks=./trust.jks

# kubectl create secret generic mb-keystore-password   --from-literal=password="……"
# kubectl create secret generic mb-truststore-password --from-literal=password="……"
```

パスワードは、securityUtility encode コマンドでエンコードしたものを指定します

6. オートスケーリングとリソース調整

オートスケーリング

- IBM 提供の Helm Chart では、Horizontal Pod Auto-Scaler (HPA) を使用したオートスケーリングの定義が可能です

Qualifier	Parameter	説明
autoscaling	enabled	オートスケーリングを有効にするか否かを指定します。 オートスケーリングを有効にした場合、replicaCount に指定したレプリカ数は無視されるようになります。
	minReplicas	レプリカ数の最小値と最大値を指定します。
	maxReplicas	
	targetCPUUtilizationPercentage	CPU 使用率の目標値(*)をパーセントで指定します。 (*) CPU の要求量に対する割合で、全Podの平均値です。

- また、コンテナーが使用するリソースの要求量と上限を指定できます

Qualifier	Parameter	説明
resources	constraints.enabled	リソース制約を有効にするか否かを指定します。
	requests.cpu requests.memory	CPU とメモリーの要求量を指定します。 指定しない場合は、limits で指定した値、または、環境のデフォルト値が使用されます。
	limits.cpu limits.memory	CPU とメモリーの上限を指定します。

- 尚、下記の場合の条件の場合にのみ、オートスケーリングの動作が確認できました
  - リソース制約を有効化（constraints.enabled=true）し、
  - Deployment/ReplicaSet としてインストールした場合（ログを永続化していない場合）

オートスケーリングの動作例（１）

■ HPA の動作状況の概要は kubectl get (HAP名) –w でモニターできます

```
# kubectl get hpa
NAME                           REFERENCE                                 TARGETS           MINPODS   MAXPODS   REPLICAS   AGE
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp   <unknown> / 50%   1         3         0          25s

# kubectl get hpa resource-eater-ibm-websp-hpa -w
NAME                           REFERENCE                                 TARGETS           MINPODS   MAXPODS   REPLICAS   AGE
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp   <unknown> / 50%   1         3         1          35s
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         16% / 50%   1         3         1          1m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          1% / 50%   1         3         1          2m
....
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          0% / 50%   1         3         1          15m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         63% / 50%   1         3         1          17m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         63% / 50%   1         3         2          18m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp        100% / 50%   1         3         2          18m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp        100% / 50%   1         3         2          19m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         50% / 50%   1         3         2          21m
....
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         50% / 50%   1         3         2          21m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         95% / 50%   1         3         3          23m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp        100% / 50%   1         3         3          23m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp        100% / 50%   1         3         3          24m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         60% / 50%   1         3         3          24m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         60% / 50%   1         3         3          25m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         33% / 50%   1         3         3          25m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         33% / 50%   1         3         3          26m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          0% / 50%   1         3         3          26m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          0% / 50%   1         3         3          27m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          0% / 50%   1         3         3          27m
....
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp          0% / 50%   1         3         1          28m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         97% / 50%   1         3         1          28m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         99% / 50%   1         3         2          32m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         99% / 50%   1         3         2          32m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         51% / 50%   1         3         2          33m
....
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         50% / 50%   1         3         2          36m
resource-eater-ibm-websp-hpa   Deployment/resource-eater-ibm-websp         50% / 50%   1         3         2          36m
```

負荷を増減させた場合の挙動

出力例はカラムを合わせて整形しています

オートスケーリングの動作例（２）

- HPA の状態は kubectl describe (HAP名) で確認できます
  - Conditions:　HPA の現在の状態です
  - Events:　　HPA のイベントの履歴です

負荷を増減させた後に状態を確認したところ

```
# kubectl describe hpa resource-eater-ibm-websp-hpa
Name:                                                  resource-eater-ibm-websp-hpa
Namespace:                                             default
Labels:                                                app=resource-eater-ibm-websp
                                                       chart=ibm-websphere-liberty-1.4.0
                                                       heritage=Tiller
                                                       release=resource-eater
Annotations:                                           <none>
CreationTimestamp:                                     Fri, 13 Jul 2018 15:21:39 +0900
Reference:                                             Deployment/resource-eater-ibm-websp
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  50% (251m) / 50%
Min replicas:                                          1
Max replicas:                                          3
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:
  Type     Reason                        Age                From                       Message
  ----     ------                        ----               ----                       -------
  Warning  FailedGetResourceMetric       38m (x2 over 38m)  horizontal-pod-autoscaler  unable to get metrics for resource cpu: no metrics returned from resource metrics API
  Warning  FailedComputeMetricsReplicas  38m (x2 over 38m)  horizontal-pod-autoscaler  failed to get cpu utilization: unable to get metrics for resource cpu: no metrics ···
  Normal   SuccessfulRescale             16m                horizontal-pod-autoscaler  New size: 3; reason: cpu resource utilization (percentage of request) above target
  Normal   SuccessfulRescale             11m                horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
  Normal   SuccessfulRescale             7m (x2 over 21m)   horizontal-pod-autoscaler  New size: 2; reason: cpu resource utilization (percentage of request) above target
```

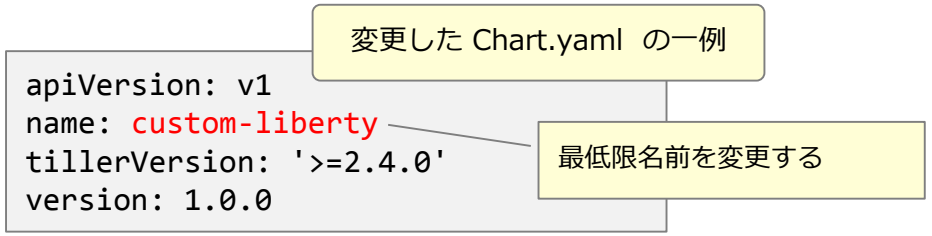
7. Helm Chart の作成と改良

Helm Chart の作成

- IBM 提供の Helm Chart が要件に合わない場合は、提供さている Chart を元にして別の Chart を作成するか、新たに独自の Chart を作成することになります
  - その他に、Chart を使用せずに、直接リソースを定義する方法などもありますが、本ガイドでは扱いません
- ここでは、IBM 提供の Helm Chart を元にして別の Chart を作成する方法を記載します

(1) まず、IBM 提供の Helm Chart をダウンロードし、展開します （前述参照）
(2) 次に、展開されたディレクトリーの名前を変更し、Chart.yaml を編集して Chart 名などを変更します
  • value のデフォルト値やメタデータを変更する場合は、values.yaml および values-metadata.yaml を更新します
(3) Chart に必要な変更を施します（templates ディレクトリー内のファイルを編集します）

```
# tar xvf ibm-websphere-liberty-1.4.0.tgz

# mv ibm-websphere-liberty custom-liberty
# vi ./custom-liberty/Chart.yaml
# vi ./custom-liberty/values.yaml
# vi ./custom-liberty/values-metadata.yaml

# vi ./custom-liberty/templates/deployment.yaml
# vi ./custom-liberty/templates/xxxxx.yaml
```



- 作成した Chart を ICP の Internal Helm Repository に登録する方法に関しては、前述の「補足：ICP Management Console で Helm Chart を使用する場合」を参照してください

Readiness Probe の改良

■ Readiness Probe の設定内容

– IBM 提供の Helm Chart では、右図の Readiness Probe が設定されますが、以下の問題点があります

- microprofile.health.enabled を true に変更しなかった場合、Readiness Probe は http(s)://<host>/ への GET 要求になります。この要求に対して、WAS Liberty は自身の初期化が完了すると 200 OK を返してしまいます。このため、アプリの起動が完了する前に Readiness Probe が正常完了してしまい、アプリ起動完了前に要求の割り振りが開始されてしまいます。

```
readinessProbe:
  failureThreshold: 3
  httpGet:
    path: / または /health
    port: <service.targetPortの指定値>
    scheme: HTTP または HTTPS
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 1
```

青文字の部分は Chart で指定しているものではなく、
デフォルト値が反映されている部分

■ Readiness Probe の改良

– Chart の deployment.yaml 内の readinessProbe の path を、アプリのヘルスチェックが可能なパスに変更します

– その他の readinessProbe のデフォルト値に関しても、変更が必要な項目を設定するように、Chart を変更します

- 参照：kubernetes – Tasks – Configure Liveness and Readiness Probes
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

readinessProbe の path を変更した例

readinessProbe の設定をデフォルトから変更した例

```
readinessProbe:
  httpGet:
  {{ if .Values.microprofile.health.enabled }}
    path: /health
  {{ else }}
    path: /app/healthcheck
  {{ end }}
    port: {{ .Values.service.targetPort }}
    {{ if .Values.ssl.enabled}}
    scheme: HTTPS
    {{ end }}
  initialDelaySeconds: 15
  timeoutSeconds: 3
```

Chart の deployment.yaml の変更例

Liveness Probe の追加

■ Liveness Probe の追加

– IBM 提供の Helm Chart では Liveness Probe が設定されません。
このため、Readiness Probe が失敗し、長時間に渡り割り振り対象から除外されたままでも、自動で再起動されることはありません。
– livenessProbe を追加することで、Pod 内のコンテナーが自動で再起動されるように指定できます。
– 但し、再起動により、コンテナー内に出力されたログなどは消失します。問題判別等の目的で再起動したくない場合は、Liveness Probe は追加しないでください。

- 参照：Kubernetes – Tasks – Configure Liveness and Readiness Probes
  – https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

```
  .........
livenessProbe:
  httpGet:
  {{ if .Values.microprofile.health.enabled }}
    path: /health
  {{ else }}
    path: /app/healthcheck
  {{ end }}
    port: {{ .Values.service.targetPort }}
    {{ if .Values.ssl.enabled}}
    scheme: HTTPS
    {{ end }}
  initialDelaySeconds: 15
  failureThreshold: 30
readinessProbe:
  .........
```

Chart の depolyment.yaml に Liveness Probe を追加した例：

Readiness Probe と同様のチェックを行うが、
failureThreshold の値を Readiness Probe より大きくすることで、
長時間回復しない場合に自動で再起動するように指定した例

StatefulSet のデフォルト設定の変更

- Liberty が StatefulSet としてインストールされる場合(ログを永続化する場合)に、以下の変更が必要になることがあります

- podManagementPolicy の指定
  - 以下の２種類の指定が可能です
    - OrderedReady: Pod を順次起動/順次停止します (起動はインデックスが小さい順、停止は大きい順)
    - Parallel: Pod を並列起動/並列停止します
  - 変更しないと OrderedReady となり、順次起動/順次停止です
    - デフォルトが OrderedReady で、Chart で指定していないので OrderedReady になります

- updateStrategy の指定
  - 以下の２種類の指定が可能です
    - RollingUpdate: 自動的にローリング・アップデートを行います
    - OnDelete: Pod が手動で削除されるまで更新されません
  - 変更しないと OnDelete になり、自動的には更新されません
    - deployment.yaml 内で apiVersion として apps/v1beta1 が指定されているため、古いデフォルト値の OnDelete が使用されます
    - apps/v1 のデフォルトは RollingUpdate です

    - 参照：Kubernetes – Concepts – StatefulSets
      - https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

apiVersion の指定が古い
(ibm-websphere-liberty v1.4.0 の場合)

```
{{ if $stateful }}
apiVersion: apps/v1beta1
kind: StatefulSet
{{ else }}
apiVersion: extensions/v1beta1
kind: Deployment
{{ end }}
metadata:
  ……
spec:
  {{ if $stateful }}
  serviceName: {{ template "fullname" . }}
  podManagementPolicy: Parallel
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  {{ end }}
  ……
```

Chart の deployment.yaml の変更例

Deployment のデフォルト設定の変更

- Liberty が Deployment(ReplicaSet) としてインストールされる場合(ログを永続化しない場合)に、以下の変更が必要になることがあります

- strategy の rollingUpdate の調整
  - Chart で指定が無いため、右図のデフォルト値が使用されます
    - deployment.yaml 内で apiVersion として extensions/v1beta1 が指定されているため、古いデフォルト値が使用されます
    - apps/v1 のデフォルトは maxSurge=25% および maxUnavailable=25% です
      - *maxSurge　更新時に、指定されたレプリカ数を超えて生成することができる Pod の数(割合)*
      - *maxUnavailable　更新時に、使用不可状態にできる Pod の最大数(割合)*
  - 必要に応じて、調整します

- strategy の type の変更
  - 同様に、更新方法をRollingUpdate から Recreate に変更することも可能です

    - 参照：Kubernetes – Concepts – Deployments
      - https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

apiVersion の指定が古い
(ibm-websphere-liberty v1.4.0 の場合)

```
apiVersion: extensions/v1beta1
kind: Deployment
 ……
spec:
  ……
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

青文字の部分は Chart で指定しているものではなく、
デフォルト値が反映されている部分

maxSurge と maxUnavailable を変更した例

```
  ……
spec:
  {{ if $stateful }}
  serviceName: {{ template "fullname" . }}
  ……
  {{ else }}
  strategy:
    rollingUpdate:
      maxSurge: 50%
      maxUnavailable: 0%
    type: RollingUpdate
  {{ end }}
  ……
```

Chart の deployment.yaml の変更例

クラスター SSL 構成用の構成ファイル keystore.xml の配置方法の変更

- keystore.xml の配置方法を変更すると、クラスター SSL 構成を有効化した際のデメリットを解消できます（参照：「クラスター SSL 構成を有効化する場合の注意点」）
  - イメージの /config/configDropins 配下の構成ファイルが有効になります(残ります)
  - keystore.xml の配置先を /config/configDropins/overrides に変更することで、クラスター SSL 構成の適用を強制できます
  - 他の ConfigMap や Secret の構成ファイルを /config/configDropins 配下に配置できるようになります
- 但し、この配置方法を使用した場合は、クラスター SSL 構成用の ConfigMap に対する変更が動的に反映されなくなります

Chart の deployment.yaml の変更例 (1/2)

```
.....
 template:
   .....
   spec:
     .....
     volumes:
       .....
       - name: ssl-config
         configMap:
           name: liberty-config
         items:
         - key: keystore.xml
           path: defaults/keystore.xml
     .....
```

削除

Chart の deployment.yaml の変更例 (2/2)

```
.....
volumeMounts:
.....
- name: ssl-config
  mountPath: /config/configDropins/defaults/keystore.xml
  subPath: keystore.xml
{{ end }}
.....
```

変更

適用を強制する場合は defaults をoverrides に変更

Ingress の変更：名前ベースの仮想ホストの利用（１）

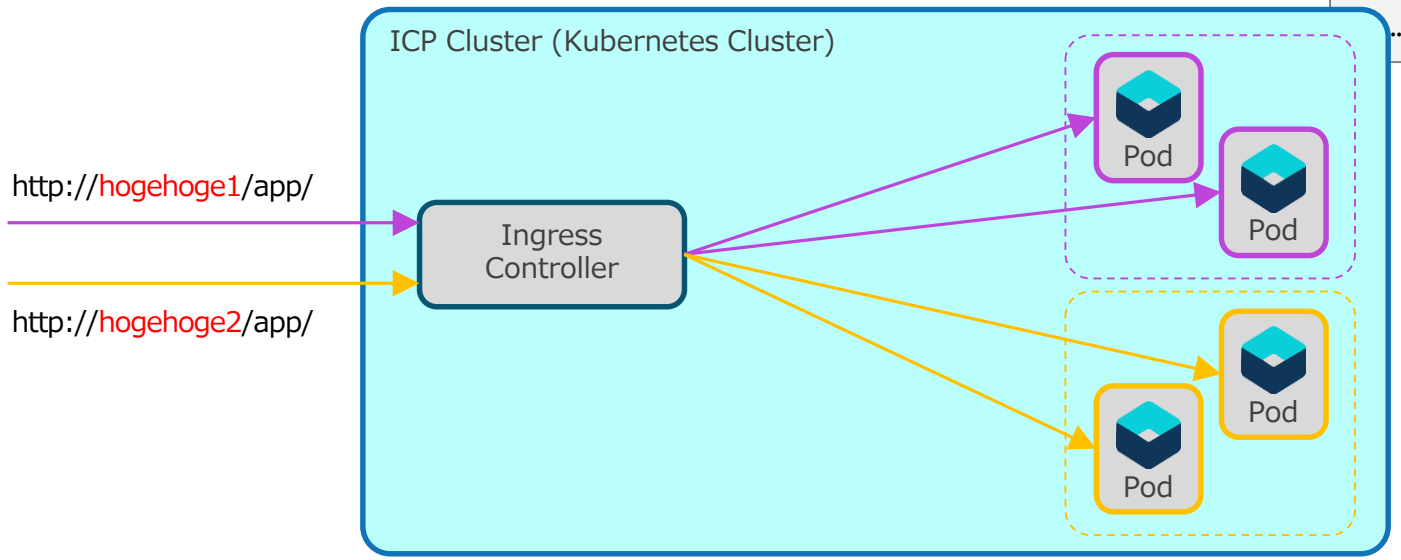
- ICP の Ingress Controller は名前ベースの仮想ホスト(Name-based Virtual Host)に対応しています
  - ホスト名に基づいて、同じパスを持つ複数のサービスに処理を割り振ることが可能です
  - ingress.yaml の spec.rules に host を追加することで、名前ベースの仮想ホストが利用可能になります
  - 但し、HTTPS (SSL) 通信で名前ベースの仮想ホストを利用するには、次ページの変更も必要です
    - 参照：kubernetes – Concepts - Ingress
      - https://kubernetes.io/docs/concepts/services-networking/ingress/

Chart の values.yaml の例

```
ingress:
  enabled: true
  secureBackends: true
  rewriteTarget: "/"
  host: ""
  path: "/liberty"
```

Chart の ingress.yaml の変更例

```
.....
spec:
  rules:
  - host: {{ .Values.ingress.host }}
    http:
      paths:
      - path: {{ .Values.ingress.path }}
        backend:
          serviceName: {{ template "fullname" . }}
          servicePort: {{ .Values.service.port }}
...
```



valueFile での指定例

```
ingress:
  enabled: true
  secureBackends: true
  rewriteTarget: "/"
  host: "hogehoge1"
  path: "/liberty"
```

Ingress の変更：名前ベースの仮想ホストの利用（2）

■ HTTPS (TLS) 通信で名前ベースの仮想ホストを利用する場合は、更に、次の定義が必要です

– サーバー証明書と鍵を TLS Secret として登録します
  • NGINX Ingress Controller の仕様にあわせて、証明書と鍵は「PEM-encoded X.509, RSA (2048) secret」として登録します
    – *参考:NGINX Ingress Controller - TLS/HTTPS*
    – *https://kubernetes.github.io/ingress-nginx/user-guide/tls/*
  • TLS Secret は以下のようなコマンドで PEM ファイルから定義できます

```
# kubectl create secret tls hogehoge1-tls-secret --key hogehoge1.key.clear.pem --cert hogehoge1-chain.cert.pem
```

– TLS Secret とホスト名をマップするための Ingress を作成します
  • 以下のような yaml フィルを作成し、kubectl apply –f で定義します

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hogehoge1-tls-ingress
spec:
  tls:
  - hosts:
      - hogehoge1
    secretName: hogehoge1-tls-secret
  rules:
  - host: hogehoge1
```

ICP v2.1.0.3 では、左記の Ingress を定義すると
Ingress Controller がダウンする現象が確認されました。
定義した Ingress を削除すると回復します。

8. 静的コンテンツの扱い

静的コンテンツを扱う方法

- 下記の方法があります

- WAS Liberty の File Serving 機能(デフォルトで有効)を使用する方法

  (a) war モジュールに含める方法
  - アプリケーションの war モジュール内に静的コンテンツを含めることで、簡単に対応できます。
  - 静的コンテンツを修正するには、war モジュールと Docker イメージを再ビルドし、Pod が使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。

  (b) コンテナー・イメージに格納し、格納先を拡張文書ルートとして指定する方法
  - Docker イメージに静的コンテンツのディレクトリーを作り、その中に静的コンテンツを含め、File Serving 機能の拡張文書ルートとして指定することで対応します。
  - 静的コンテンツを修正するには、Docker イメージを再ビルドし、Pod が使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。

  (c) 永続領域に置き、そのマウント先を拡張文書ルートとして指定する方法
  - 永続領域に静的コンテンツを配置して Pod にマウントし、File Serving 機能の拡張文書ルートとして指定することで対応します。
  - 静的コンテンツの修正は、永続領域内のコンテンツの更新で行えます。また、静的コンテンツが多い/大きい場合にも適した方法となります。

- WAS Liberty の代わりに、IBM HTTP Server などの Web サーバーを利用する方法

  – 同様に、次の２つの方法が想定されますが、メリット・デメリットは WAS Liberty の場合と同様です。

  (b) コンテナー・イメージに含める方法

  (c) 永続領域に置く方法

File Serving 機能を使用する方法：拡張文書ルートの指定方法

■ 拡張文書ルートは、以下の何れかの方法で指定できます

– *注記：この機能(File Serving 機能の拡張文書ルート)は WAS が従来から提供している機能です。WAS Liberty の Knowledge Center 等での記載が確認できておりませんが、指定が有効に機能していることは確認済みです。*

– war モジュールの WEB-INF/ibm-web-ext.xml で指定

```
<?xml version="1.0" encoding="UTF-8"?>
<web-ext
    xmlns="http://websphere.ibm.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee http://websphere.ibm.com/xml/ns/javaee/ibm-web-ext_1_0.xsd"
    version="1.0">
  <fileServingEnabled="true">
  <file-serving-attribute name="extendedDocumentRoot" value="/htdocs" />
</web-ext>
```

– Liberty の servre.xml 内の <web-ext> で指定

```
  <webApplication name="infra-test" location="${app.location}">
    <web-ext enable-file-serving="true">
      <file-serving-attribute name="extendedDocumentRoot" value="/htdocs"/>
    </web-ext>
  </webApplication>
```

• 参考： WAS traditional Knowledge Center: ファイル・サービス

– *https://www.ibm.com/support/knowledgecenter/ja/SSAW57_9.0.0/com.ibm.websphere.nd.multiplatform.doc/ae/cweb_flserv.html*

静的コンテンツ用の永続領域の利用例 （１）

- 静的コンテンツ用の永続領域(Persistent Volume)を定義し、WAS Liberty の File Serving 機能の拡張文書ルートとして前ページで定義した /htdocs にマウントする例を示します

- まず、静的コンテンツ用の PersistentVolume (PV) と PersistentVolumeClaim (PVC) を定義します。
  - 例のような yaml ファイルを作成し、kubectl apply コマンドで作成します
    - *ICP Management Console で定義する場合は、[Menu > Platform > Storage]と選択して[Storage]画面に進み、PV および PVC のタブから作成します。*
  - 例では、label (key名: "type") をマッチング条件の１つとして指定しています
  - Pod は静的コンテンツを更新する必要がないので、accessMode は ReadOnlyMany を指定しています

PV の定義例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hogehoge-pv-htdocs
  labels:
    type: hogehoge-pv-htdocs
spec:
  accessModes:
  - ReadOnlyMany
  capacity:
    storage: 2Gi
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /nfs/hogehoge-pv-htdocs
    server: 10.20.30.40
```

PVC の定義例

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: hogehoge-pv-htdocs-pvc
spec:
  storageClassName: ""
  selector:
    matchExpressions:
    - key: type
      operator: In
      values:
      - hogehoge-pv-htdocs
  accessModes: ["ReadOnlyMany"]
  resources:
    requests:
      storage: "2Gi"
```

静的コンテンツ用の永続領域の利用例 （2）

■ 次に、永続領域をマウントする指定を Chart の deployment.yaml に追加します。

- IBM 提供の Helm Chart を元にして新しい Chart を作成し、deployment.xml に下記の変更を加えます
  - *IBM 提供の Chart では、生成される内容を if 文で制御していますが、今回の変更を適用するには、if 文の位置を移動させる必要があります。*
  - *例では、必ず静的コンテンツ用の永続領域がマウントされ、PVC の名前も固定になっていますので、特定用途でのみ使用可能な Chart になります。多様な用途で使用する Chart として変更する場合は、マウント有無、PVC 名、マウント先などをパラメーター指定できるように改良する必要があります。*

```
  .........
template:
  metadata:
    labels:
      app: {{ template "fullname" . }}
  spec:
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
    volumes:
      - name: htdocs
        persistentVolumeClaim:
          claimName: hogehoge-pv-htdocs-pvc
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
      - name: keystores
        secret:
          secretName: mb-keystore
          .........
```

移動

Chart の deployment.yaml の変更例 (1/2)

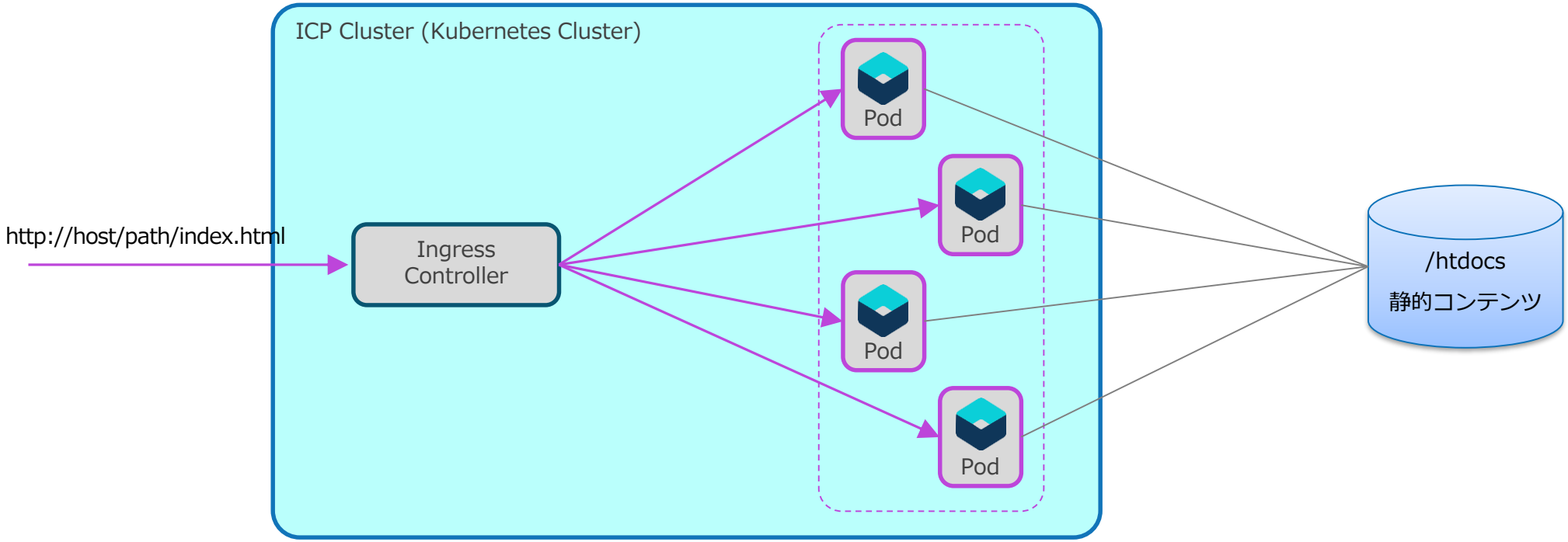
```
        .........
        {{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
        volumeMounts:
        - mountPath: /htdocs
          name: htdocs
          readOnly: true
        {{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
        {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration -}}
        - name: keystores
          mountPath: /etc/wlp/config/keystore
          readOnly: true
      .........
```

移動

Chart の deployment.yaml の変更例 (2/2)

静的コンテンツ用の永続領域の利用例 （3）

- 変更した Helm Chart を使用してインストールします
  - 永続領域に配置された静的コンテンツが WAS Liberty の File Serving 機能によって配信されるようになります



9. IBM HTTP Server (IHS) の利用

IBM HTTP Server (IHS) に関する情報

- IHS を Docker 環境で利用するための情報を以下に示します
  - IHS の Docker イメージに関する情報
    - 以下の URL で情報が公開されており、WAS Plug-in を含む Docker イメージが公開されています
    - 参考：Docker Hub: ibmcom/ibm-http-server
      - https://hub.docker.com/r/ibmcom/ibm-http-server/
    - 参照：GitHub: Dockerfiles for IBM HTTP Server
      - https://github.com/WASdev/ci.docker.ibm-http-server
    - 参照：GitHub: Building an IBM HTTP Server v8.5.5 production image from binaries
      - https://github.com/WASdev/ci.docker.ibm-http-server/tree/master/production
  - IHS のアーカイブからのインストールに関する情報
    - アーカイブ・ファイルを使用すると、ほぼ解凍するだけで IHS と WAS Plug—in のインストールが完了します
    - 参考：IHS Knowledge Center: Installing IBM HTTP Server from an archive
      - https://www.ibm.com/support/knowledgecenter/en/SSEQTJ_9.0.0/com.ibm.websphere.ihs.doc/ihs/tihs_archive_intall.html
  - IHS に関連するその他の情報
    - 参考：IHS Knowledge Center: Configuring IBM HTTP Server for Liberty
      - https://www.ibm.com/support/knowledgecenter/en/SSEQTJ_9.0.0/com.ibm.websphere.ihs.doc/ihs/tihs_install_config_liberty.html
    - 参考：PI77874: PLUGIN OFFLOAD/ONLOAD FOR SSL
      - http://www-01.ibm.com/support/docview.wss?uid=swg1PI77874

- これらの情報を参考にし、アーカイブからインストールする方法で、独自の IHS の Dockerイメージを作成する方法を以降に記載します
  - ICP 環境での IHS の利用をお勧めするものではなく、既存環境からの移行等による一時的な利用を想定して、ご紹介しています

IHS の Docker イメージの作成

- IHS のイメージを作成する Dockerfile の例を示します。この Dockerfile では、以下の操作を行っています
  - OS の更新と unzip のインストール
  - アーカイブからの IHS のインストール（WAS の Plug-in を含む）
  - ディレクトリーの準備（ WAS Liberty と同様の構成）
    - /htdocs　　コンテンツ用
    - /logs　　ログ用
    - /config　　構成用
  - 起動スクリプトの追加
  - 起動コマンドの指定

- 以下の操作を、必要に応じて追加する必要があります
  - カスタマイズした httpd.conf を /config に COPY する
  - 静的コンテンツを /htdocs に配置する (コンテンツをイメージに内に含める場合)

Dockerfile の例

```
FROM ubuntu:16.04

# Update and Install unzip
RUN apt-get update && apt-get install -y unzip wget

# Install IHS (unzip IHS zip-file)
ARG ihs_zip_file=9.0.0-WS-IHS-ARCHIVE-linux-x86_64-FP007.zip
COPY ${ihs_zip_file} /tmp/
RUN    mkdir /opt/ibm ¥
    && cd /opt/ibm ¥
    && unzip /tmp/${ihs_zip_file} ¥
    && rm    /tmp/${ihs_zip_file} ¥
    && ./IHS/postinstall.sh

# Setup Directories
RUN    rm -r /opt/ibm/IHS/htdocs ¥
    && mkdir /htdocs ¥
    && ln -s /htdocs /opt/ibm/IHS/htdocs ¥
    && rm -r /opt/ibm/IHS/logs ¥
    && mkdir /logs ¥
    && ln -s /logs /opt/ibm/IHS/logs ¥
    && mkdir /output/logs/state -p ¥
    && ln -s /opt/ibm/IHS/conf /config

# Append SetEnv to httpd.conf
RUN echo SetEnv ssl-map-mode offload >> /config/httpd.conf

# Add shell
COPY ihsstart.sh /work/

ENV PATH /opt/ibm/IHS/bin:$PATH
CMD ["/work/ihsstart.sh"]
```

予めダウンロードしておいた、IHS のアーカイブ・ファイルの名前

IHS & Plug-in のインストールと postinstall.sh の実行

ディレクトリーの準備
（後述の Liberty との連携で使用するディレクトリーの定義(青文字部分)を含む）

後述の Liberty との連携用
（Liberty と常に http で通信するように、SetEnv を httpd.conf に追加）

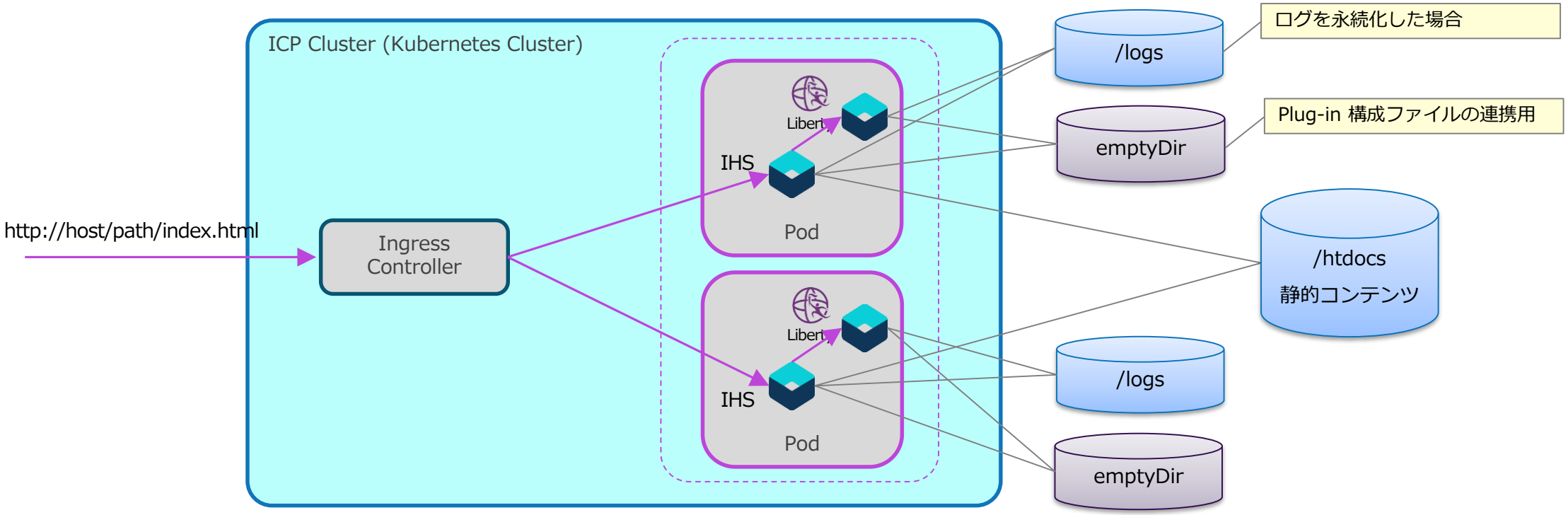
前述のサイトで公開されている ihsstart.sh をカスタマイズ(*)して使用
(*) インストール・ディレクトリーを /opt/IBM/HTTPServer/ から /opt/ibm/IHS/ に変更

IHS 用の Helm Chart の作成

- IHS 用の Helm Chart は提供されていませんが、
  WAS Liberty 用の Helm Chart を参考に(変更することで)容易に作成できます
- 作成する際のポイントは以下の通りです
  - 前ページに示した IHS のイメージは WAS Liberty と同様のディレクトリー構造としているので、以下の指定は WAS Liberty の Chart と同じです
    - /logs に出力されるログを永続化する
    - 静的コンテンツ用の永続領域を /htdocs にマウントする
  - 以下の指定は IHS では不要です
    - コンソールへのログ出力を設定する環境変数
    - トランザクション・ログの永続化
    - クラスター SSL 構成
  - 前ページに示した IHS のイメージでは http 通信のみが可能です
- WAS Liberty 用の Helm Chart を使用して IHS をインストールすることも可能です
  - ポート番号 80 で、http 通信のみが可能な Liberty と見立ててインストールする事となります。

IHS を WAS Liberty の前段に配置する構成

- Pod 内で IHS のコンテナーと WAS Liberty のコンテナーを稼動させ、Plug-in で連携させる方法を説明します
  - IHS と WAS Liberty 間は、Pod 内に閉じた、 1 対 1 構成となります
  - IHS が静的コンテンツを処理し、WAS Liberty が動的コンテンツを処理する、従来構造の Web システムとなります
    - Plug-in 構成ファイルは、Pod 内で emptyDir (非永続領域) を共有することで、WAS Liberty のコンテナーから IHS のコンテナーへ連携します
    - Plug-in を経由するので、ServerIOTimeout などの Plug-in の構成パラメーターが利用できるようになります
  - ログを永続化させる場合は、/logs 用の永続領域を Pod 内で共有できます



IHS を WAS Liberty の前段に配置する構成：IHS のイメージの変更

- WAS Liberty が生成した Plug-in 構成ファイルが IHS 側の所定の位置に随時コピーされるように、IHS の Docker イメージに含める ihsstart.sh を以下の内容に差し替えます

```
#! /bin/bash

#################################################################
startServer()
{
    echo "Starting IBM HTTP Server "
    # Starting IBM HTTPServer
    /opt/ibm/IHS/bin/apachectl start
    if [ $? = 0 ]
    then
       echo "IBM HTTP Server started successfully"
    else
       echo "Failed to start IBM HTTP Server"
    fi
}

#################################################################
stopServer()
{
    echo "Stopping IBM HTTP Server "
    # Stopping IBM HTTPServer
    /opt/ibm/IHS/bin/apachectl graceful-stop
    if [ $? = 0 ]
    then
       echo "IBM HTTP Server stopped successfully"
    fi
}

#################################################################
```

続き

```
while [ ! -e "/output/logs/state/plugin-cfg.xml" ]
do
   echo "Waiting for plugin-cfg.xml generation"
   sleep 5
done

cp -a /output/logs/state/plugin-cfg.xml /opt/ibm/IHS/plugin/config/webserver1/plugin-cfg.xml
chmod o+r /opt/ibm/IHS/plugin/config/webserver1/plugin-cfg.xml
echo "plugin-cfg.xml was generated and copied"
oldFulltime=`ls --full-time /output/logs/state/plugin-cfg.xml`

startServer

trap "stopServer" SIGTERM

sleep 10

tail -f /logs/error_log &
tail -f /logs/http_plugin.log &

while [ -f "/logs/httpd.pid" ]
do
    newFulltime=`ls --full-time /output/logs/state/plugin-cfg.xml`
    if [[ $newFulltime != $oldFulltime ]]
    then
        cp -a /output/logs/state/plugin-cfg.xml /opt/ibm/IHS/plugin/config/webserver1/plugin-cfg.xml
        chmod o+r /opt/ibm/IHS/plugin/config/webserver1/plugin-cfg.xml
        echo "plugin-cfg.xml was updated and copied"
        oldFulltime=$newFulltime
    fi
    sleep 10
done
```

Plug-in 構成ファイルの生成を待つ

Plug-in 構成ファイルをコピー

IHS 起動、trap 設定、sleep（変更前と同様）

ログをコンソールへ tail 出力

Plug-in 構成ファイルが更新されたらコピー
（アプリケーションの特性上、起動完了後に Plug-in 構成ファイルが更新されることがない場合は、 除去しても支障ありません。）

IHS を WAS Liberty の前段に配置する構成：WAS Liberty 側の設定

- WAS Liberty の server.xml で pluginConfiguration を IHS 側の構成に合わせて指定します
  - pluginInstallRoot
    - Plug-in のインストール先
    - IHS をアーカイブ・インストールした場合の Plug-in のインストール先 /opt/ibm/IHS/plugin を設定
  - webserverName
    - IHS をアーカイブ・インストールした場合にデフォルトで作成される Web サーバー名 webserver1 を設定
  - logFileName
    - IHS のログ出力用ディレクトリー /logs 内に作成するように設定（例：/logs/http_plugin.log）

```
<pluginConfiguration pluginInstallRoot="/opt/ibm/IHS/plugin"
                     webserverName="webserver1"
                     logFileName="/logs/http_plugin.log" />
```

pluginConfiguration の指定を含む構成ファイルを
ConfigMap を使用して /config/configDropins 配下に配置することで、
この構成を追加・上書きすることも可能です

  - その他の設定項目に関しては、以下の URL を参照
    - WAS Liberty Knowledge Center: pluginConfiguration - Web Server Plugin (pluginConfiguration)
      - https://www.ibm.com/support/knowledgecenter/SSAW57_liberty/com.ibm.websphere.liberty.autogen.nd.doc/ae/rwlp_config_pluginConfiguration.html

IHS を WAS Liberty の前段に配置する構成：Helm Chart の変更

- IBM 提供の Helm Chart を元にして新しい Chart を作成します。以下の追加が最低限必要になります。
  - IHS 用コンテナーの定義
  - Plug-in 構成ファイルを連携するための volume 定義（下記例で name が state の emptyDir）
  - 上記 volume のコンテナーへのマウント定義（Liberty 用と IHS 用の両方でマウント）

Chart の deployment.yaml の変更例

```
  ……
template:
  metadata:
    labels:
      app: {{ template "fullname" . }}
  spec:
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfigu
    volumes:
      - name: htdocs
        persistentVolumeClaim:
          claimName: hogehoge-pv-htdocs-pvc
      - name: state
        emptyDir:
          medium: "Memory"
          sizeLimit: "1Mi"
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfigu
      - name: keystores
        secret:
          ……
```

IHS 用コンテナーの定義

Plug-in 構成ファイルの連携用

移動

```
  ……
containers:
- name: ihs
  image: "{{ .Values.ihs.image.repository }}:{{ .Values.ihs.image.tag }}"
  imagePullPolicy: {{ .Values.ihs.image.pullPolicy }}
  volumeMounts:
  - mountPath: /htdocs
    name: htdocs
    readOnly: true
  - mountPath: /output/logs/state
    name: state
  {{ if .Values.logs.persistLogs }}
  - mountPath: /logs
    name: {{ .Values.persistence.name | trunc 63 | lower | trimSuffix "-" | quote }}
    subPath: logs
  {{ end }}
- name: liberty
  livenessProbe:
    ……
    ……
    ……
  {{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
  volumeMounts:
  - mountPath: /output/logs/state
    name: state
  {{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
  {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration -}}
  - name: keystores
    ……
```

Liberty 用コンテナーの名前を簡略化 （オプション）

移動

IBM 提供の Chart では、生成される内容を if 文で制御していますが、
今回の変更を適用するには、if 文の位置を移動させる必要があります。

IHS を WAS Liberty の前段に配置する構成：インストール

- Pod として公開するサービスは IHS のポート番号 80 宛の http 通信となるので、インストール時に下記のパラメーターの指定に注意が必要です。
  - service.port
  - ssl.enabled
  - ingress.secureBackends
  - ingress.rewriteTarget
- 前ページの Chart 変更例の場合、Chart の values.yaml / values-metadata.yaml および、valueFile に、IHS のイメージ関連のパラメーター定義の追加が必要になります。
  - ihs.image.repository
  - ihs.image.tag
  - ihs.image.pullPolicy

valueFile での追加指定の例
(Chart の value.yaml にも定義)

```
ihs:
  image:
    repository: mycluster.icp:8500/default/ihs
    tag: v100
    pullPolicy: IfNotPresent

……
```

- 前ページに示した Chart の変更例には以下の制約があります
  - 例では、IHS 用のコンテナーに対する Liveness Probe と Readiness Probe の定義を省略しています。
  - Liberty 用コンテナーの Liveness Probe と Readiness Probe は IHS 経由の path で指定することになります。（Liberty に到達しない path を指定すると、IHS 用コンテナーのみが Probe 対象となります。）
  - Chart でのインストール時に指定できるリソース制限は、Liberty 用コンテナーにのみ適用されます。

補足：Plug-in 構成ファイルのアクセスに関して

- Plug-in は、以下のタイミングで Plug-in 構成ファイルにアクセスします
  - IHS 起動時： IHS 起動ユーザー(root)で read します
  - その後： httpd.conf で指定されたユーザー(nobody)で変更を確認し、read します
- WAS Liberty が生成するファイルには、other への権限が付与されないため、ユーザー nobody では Plug-in 構成ファイルを read できません
  - WAS Liberty は、デフォルトで、実行ユーザーの umask で指定された other の権限を全て無効化します
  - 実行ユーザーの umask で指定された other の権限を適用するには、環境変数 WLP_SKIP_UMASK に true を指定する必要があります
    - WAS Liberty Knowledge Center: サーバー・コマンド・オプション： WLP_SKIP_UMASK 環境変数
      - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_server.html
- IBM が提供する WAS Liberty の Docker イメージでは /opt/ibm/wlp/output にスティッキー・ビット(T)が設定されているため、ユーザー nobody では Plug-in 構成ファイルの更新有無を確認できません（ファイル更新チェックでエラーが発生します）
- 上記の点を踏まえ、本ガイドで例示した ihsstart.sh では、生成された Plug-in 構成ファイルを所定のディレクトリーにコピーし、権限を変更するようにしています。

10. ConfigMap と Secret の利用

ConfigMap と Secret の用途

- ConfigMap と Secret は、例えば、以下のような情報を保存するために利用できます
  - ConfigMap： 環境毎に異なる情報
    - データベース接続情報（データソースの定義情報）のうち、機密性が低い情報を格納する
    - 本番環境でのみ構成が必要になる、セッション・パーシスタンスの構成情報を格納する
  - Secret：機密性の高い情報・環境毎に異なる情報
    - データベース接続用の認証情報（パスワードなど）を格納する
    - SSL 通信用の鍵ストアとそのパスワードを格納する
- ConfigMap と Secret に保存されている情報は、環境変数やファイルを通じて、Pod 内で稼動する WAS Liberty やアプリケーションに渡すことができます
- 参照：前述の「基礎知識： ConfigMap と Secret」

ConfigMap と Secret のマウント・ポイントに関する注意点

- 以下のディレクトリーに ConfigMap や Secret を直接マウントすると、WAS Liberty の起動に失敗します（2018年7月時点、Liberty 18.0.0.1, 18.0.0.2 の状況）
  - /config/configDropins/defaults
  - /config/configDropins/overrides
- 以下の方法で対応する必要があります
  - /config/configDropins にマウントする
    - この方法を使用すると、/config/configDropins 配下の既存の構成ファイルが全て使用されなくなります
  - volumeMount 定義で subPath を指定し、ファイル単位で上記ディレクトリーに配置する
    - この方法を使用すると、 ConfigMap や Secret に対する変更が動的に反映されなくなります
    - 後述の例では、この方法を使用しています
- 参照：起動失敗時のログ出力抜粋

messeges.log

```
[6/20/18 3:48:16:450 UTC] 0000001e com.ibm.ws.logging.internal.impl.IncidentImpl               I FFDC1015I: An FFDC Incident has been created:
 "com.ibm.wsspi.kernel.service.location.MalformedLocationException: Can not reference ".." when creating a descendant (path=..data/) com.ibm.ws.config.xml.internal.WSConfigXMLActivator 120"
 at ffdc_18.06.20_03.48.16.0.log
[6/20/18 3:48:16:461 UTC] 0000001e com.ibm.ws.config.xml.internal.WSConfigXMLActivator         A CWWKG0010I: The server defaultServer is shutting down because of a previous initialization error.
```

FFDC ログ

```
------Start of DE processing------ = [6/20/18 3:48:16:206 UTC]
Exception = com.ibm.wsspi.kernel.service.location.MalformedLocationException
Source = com.ibm.ws.config.xml.internal.WSConfigXMLActivator
probeid = 120
Stack Dump = com.ibm.wsspi.kernel.service.location.MalformedLocationException: Can not reference ".." when creating a descendant (path=..data/)
        at com.ibm.wsspi.kernel.service.utils.PathUtils.normalizeDescendentPath(PathUtils.java:193)
        at com.ibm.ws.kernel.service.location.internal.ResourceUtils.getChildResource(ResourceUtils.java:68)
```

WAS Liberty の構成情報の優先順位

- WAS Liberty の構成情報の優先順位は以下のようになります
  - 低い優先順位で指定した構成情報は、高い順位の構成情報で上書きされます
  - 参考：WAS Liberty Knowledge Center: 構成ドロップイン (dropins) フォルダーを使用したサーバー構成の指定
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_dropins.html
  - 参考： WAS Liberty Knowledge Center: server.xml ファイルでの外部 XML ファイルからの構成情報の組み込み
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_config_include.html

高．/config/configDropins/overrides 内の xml ファイルでの指定

中．server.xml での指定
（明示的に include した構成の競合処理方法は onConflict 属性で個別に指定可能）

低．/config/configDropins/defaults 内の xml ファイルでの指定

環境変数の優先順位

- ConfigMap や Secret の value を環境変数を通じて WAS Liberty に渡す場合、以下の優先順位に配慮する必要があります
  - 低い優先順位で指定した値は、高い順位の値で上書きされます

高．Docker イメージ内の server.env での設定

中．インストール時の設定

低．Docker イメージでの設定

Chart で ConfigMap や Secret の value を設定した場合など

– イメージ内の server.env で指定されている値は、変更できません

ConfigMap と Secret の使用例（1）

- 概要
  - ConfigMap 内の xml ファイルを /config/configDropins/overrides に追加することで、 データソース定義と認証データをオーバーライドする例を示します
  - xml ファイルは、subPath を指定してファイル単位で追加します
  - また、クラスター SSL 構成用の ConfigMap も subPath を指定してファイル単位で配置するように変更します
- この例では、各要素に id が付与された、下図の server.xml を前提としています

```
<server>

     … … … … … … … … … …
     … … … … … … … … … …

    <authData id="SampleAD" user="root" password="?????"/>

    <dataSource id="SampleDS" jndiName="jdbc/sample" containerAuthDataRef="SampleAD" jdbcDriverRef="SampleJD">
        <properties URL="jdbc:mysql://localhost/sample"/>
    </dataSource>

    <jdbcDriver id="SampleJD">
        <library>
            <fileset dir="/mysql-connector" includes="mysql-connector-java-5.1.45-bin.jar"/>
        </library>
    </jdbcDriver>

</server>
```

ConfigMap と Secret の使用例（2）

- データソースと認証データをオーバーライドするために、下図のような xml ファイル (sampledb.xml)を作成します
  - この例では、DB 接続ユーザーとパスワードを環境変数から取得するようにしています

```
<?xml version="1.0" encoding="UTF-8"?>
<server>

    <authData id="SampleAD" user="${env.DB_USER}" password="${env.DB_PSWD}"/>

    <dataSource id="SampleDS" jndiName="jdbc/sample" containerAuthDataRef="SampleAD" jdbcDriverRef="SampleJD">
        <properties URL="jdbc:mysql://sampledb-mariadb:3306/my_database"/>
    </dataSource>

</server>
```

ConfigMap と Secret の使用例（3）

- 作成した構成ファイル(sampledb.xml)とDB接続ユーザーの情報を ConfigMap として作成します

```
# kubectl create configmap sampledb-liberty-config ¥
      --from-file=sampledb.xml=./sampledb.xml ¥
      --from-literal=DB_USER=root
```

- DB接続ユーザーのパスワードを Secret として作成します
  - パスワードは WAS Liberty の securityUtility encode コマンドでエンコードしたものを指定します

```
# kubectl create secret generic sampledb-password --from-literal=DB_PSWD="……"
```

ConfigMap と Secret の使用例（4）

- Helm Chart を作成します
  - IBM 提供の Helm Chart を元にして新しい Chart を作成し、ConfigMap 用の volume 定義を追加し、ConfigMap と Secret からの環境変数の設定を追加します
  - 更に、クラスター SSL 構成用の ConfigMap の volume 定義を修正(一部削除)します
    - *IBM 提供の Chart では、生成される内容を if 文で制御していますが、今回の変更を適用するには、if 文の位置を移動させる必要があります。*
    - *例では、ConfigMap と Secret の名前が固定になっていますので、特定用途でのみ使用可能な Chart になります。*

Chart の deployment.yaml の変更例 (1/3)

```
……
template:
  metadata:
    labels:
      app: {{ template "fullname" . }}
  spec:
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
    volumes:
      - name: sampledb-config
        configMap:
          name: sampledb-liberty-config
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
      - name: keystores
        secret:
          secretName: mb-keystore
          ……
      - name: ssl-config
        configMap:
          name: liberty-config
          items:
          - key: keystore.xml
            path: defaults/keystore.xml
    {{ end }}
    ……
```

移動

削除

Chart の deployment.yaml の変更例 (2/3)

```
……
imagePullPolicy: {{ .Values.image.pullPolicy }}
env:
- name: DB_USER
  valueFrom:
    configMapKeyRef:
      name: sampledb-liberty-config
      key: DB_USER
- name: DB_PSWD
  valueFrom:
    secretKeyRef:
      name: sampledb-password
      key: DB_PSWD
- name: LICENSE
……
```

ConfigMap と Secret の使用例（5）

- Helm Chart を作成します（続き）
  - 今回追加した ConfigMap 用の volumeMount 定義を追加します
  - 更に、既存のクラスター SSL 構成用の ConfigMap の volumeMount を変更します

Chart の deployment.yaml の変更例 (3/3)

```
.....
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
volumeMounts:
- name: sampledb-config
  mountPath: /config/configDropins/overrides/sampledb.xml
  subPath: sampledb.xml
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
{{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration -}}
- name: keystores
  mountPath: /etc/wlp/config/keystore
  readOnly: true
- name: truststores
  mountPath: /etc/wlp/config/truststore
  readOnly: true
- name: ssl-config
  mountPath: /config/configDropins/defaults/keystore.xml
  subPath: keystore.xml
{{ end }}
.....
```

移動

変更

ConfigMap と Secret の使用例（6）

- 変更した Helm Chart でインストールします
  - 差し替えたデータソース定義と認証データが使用されるようになります
  - 構成情報の処理状況は /logs/messages.log で確認できます

```
… … … … … … … … … … … … …
A CWWKE0001I: The server defaultServer has been launched.
A CWWKG0093A: Processing configuration drop-ins resource: /opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults/keystore.xml
A CWWKG0093A: Processing configuration drop-ins resource: /opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/sampledb.xml
… … … … … … … … … … … … …
W CWWKG0011W: The configuration validation did not succeed. Found conflicting settings for SampleAD instance of authData configuration.
  Property password has conflicting values:
    Secure value is set in file:/opt/ibm/wlp/usr/servers/defaultServer/server.xml.
    Secure value is set in file:/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/sampledb.xml.
  Property password will be set to the value defined in file:/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/sampledb.xml.
  Property user has conflicting values:
    Value root is set in file:/opt/ibm/wlp/usr/servers/defaultServer/server.xml.
    Value ${env.DB_USER} is set in file:/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/sampledb.xml.
  Property user will be set to ${env.DB_USER}.
… … … … … … … … … … … … …
A CWWKF0011I: The server defaultServer is ready to run a smarter planet.
… … … … … … … … … … … … …
```

messeges.log 抜粋

11. DB を使用したセッション・パーシスタンス

WAS Liberty のセッション・パーシスタンス機能

■ セッション・パーシスタンス機能

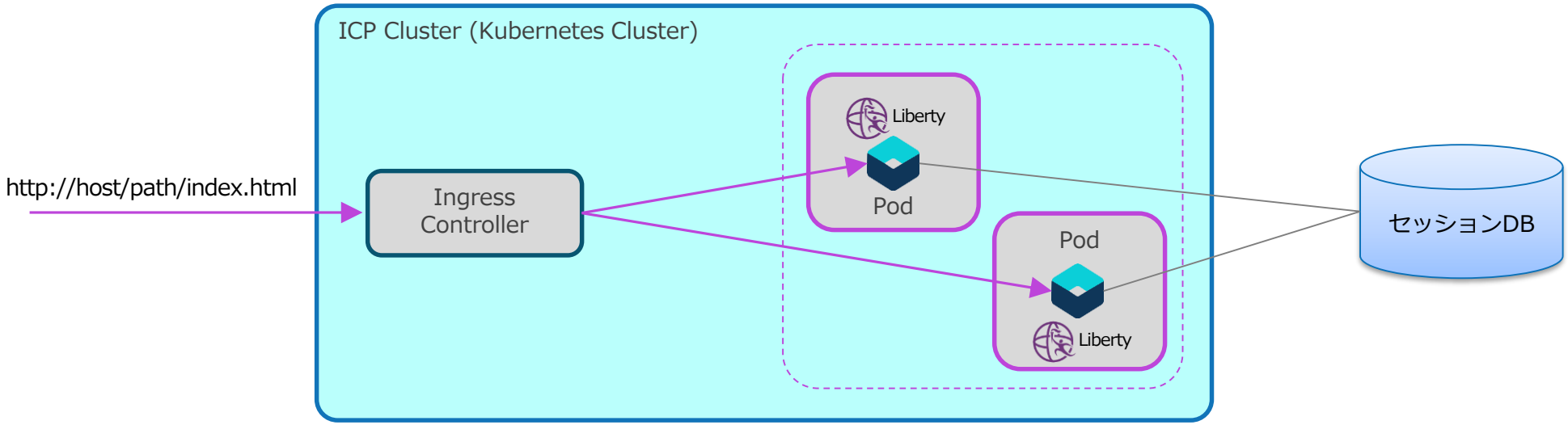
- WAS Liberty が保持する HTTP セッション・オブジェクトを外部保管する機能です
- WAS Liberty の障害時に、別の WAS Liberty が HTTP セッション・オブジェクトを引き継ぎ、HTTP セッションを継続することが可能になります
- 例えば、以下のような要件を満たすために、セッション・パーシスタンスを利用します
  - HTTP セッションの高可用性を確保したい場合
  - HTTP セッションのアフィニティーが十分には確保できない環境で稼動させる必要がある場合
  - 無停止でサービスを提供するが、定期保守などで部分的な停止が必要な場合
- WAS Liberty がサポートするセッション・パーシスタンス方法は、以下の通りです
  - データベース
  - WebSphere eXtreme Scale (WXS)
  - JCache (Liberty 18.0.0.2 以降で対応)
- セッション・パーシスタンスを利用するには、アプリケーション側での対応も必要になります
  - HTTP セッションに保管するデータは Serializable なオブジェクトである必要があります
  - パフォーマンスへの影響を減らすために、HTTP セッション・データは軽量である必要があります

■ データベース（セッションDB）の利用

- ICP 環境でも、従来環境と同様の構成定義を行うことで、データベースを使用したセッション・パーシスタンスが実現できます
  - WAS Liberty Knowledge Center: Liberty のセッション・パーシスタンスの構成
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_admin_session_persistence.html

ConfigMap と Secret を使用したセッションDBの構成例（１）

- セッション・パーシスタンスが構成されていない WAS Liberty のイメージに対して、ConfigMap と Secret を使用してセッションDBの構成定義を追加し、セッション・パーシスタンスを実装する例を記載します
  - 但し、フィーチャー「sessionDatabase-1.0」は WAS Liberty のイメージに含まれている必要があります



ConfigMap と Secret を使用したセッションDBの構成例（２）

- まず、セッションDBを構成するために、下図のような xml ファイルを作成します
  - sessiondb-config.xml　　セッションDBを構成するためのファイル（ConfigMap として格納）
  - sessiondb-secret.xml　　パスワードなどのセキュアな情報を含むファイル（Secret として格納）

sessiondb-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server>

    <featureManager>
        <feature>sessionDatabase-1.0</feature>
    </featureManager>

    <dataSource id="SessionDS" jdbcDriverRef="SessionJD">
        <properties URL="${env.SESSION_DB_URL}"
                    user="${SESSION_DB_USER}"
                    password="${SESSION_DB_PSWD}" />
    </dataSource>
    <jdbcDriver id="SessionJD">
        <library>
            <fileset dir="/SessionJD/" includes="*.jar"/>
        </library>
    </jdbcDriver>

    <httpSessionDatabase id="SessionDB"
                         dataSourceRef="SessionDS"
                         writeFrequency="TIME_BASED_WRITE"
                         writeInterval="10s"/>
    <httpSession storageRef="SessionDB"/>

</server>
```

sessiondb-secret.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server>

    <variable name="SESSION_DB_USER" value="root"/>
    <variable name="SESSION_DB_PSWD" value="……."/>

</server>
```

パスワードは WAS Liberty の securityUtility encode コマンドで
エンコードしたものを指定します

ConfigMap と Secret を使用したセッションDBの構成例（３）

- sessiondb-config.xml と DB接続用URLの情報を ConfigMap として作成します

```
# kubectl create configmap sessiondb-liberty-config ¥
      --from-file=sessiondb-config.xml=./sessiondb-config.xml ¥
      --from-literal=SESSION_DB_URL=jdbc:mysql://sessiondb-mariadb:3306/my_database
```

- sessiondb-secret.xml を Secret として作成します

```
# kubectl create secret generic sessiondb-liberty-secret --from-file=sessiondb-secret.xml=./sessiondb-secret.xml
```

- 追加で、JDBC ドライバーを Secret として登録します
  - Kubernetes v1.10.0 以降で ConfigMap にバイナリー・ファイルを保存できるように改良されましたが、それ以前は、バイナリー・ファイルの格納に Secret が使用されていました。ここで示す方法は、古い方法となります。
  - また、例として JDBC ドライバーをイメージにマウントしていますが、予めイメージ内に入れておく方が一般的と思われます。

```
# kubectl create secret generic sessiondb-jdbc-driver --from-file=./SessionJD/
```

ディレクトリー ./SessionJD/ 内に JDBC ドライバーの jar が格納されています

ConfigMap と Secret を使用したセッションDBの構成例（4）

- Helm Chart を作成します
  - IBM 提供の Helm Chart を元にして新しい Chart を作成し、ConfigMap と Secret 用の volume 定義を追加し、ConfigMap からの環境変数の設定を追加します
  - 更に、クラスター SSL 構成用の ConfigMap の volume 定義を修正(一部削除)します
    - *IBM 提供の Chart では、生成される内容を if 文で制御していますが、今回の変更を適用するには、if 文の位置を移動させる必要があります。*
    - *例では、ConfigMap と Secret の名前が固定になっていますので、特定用途でのみ使用可能な Chart になります。*

Chart の deployment.yaml の変更例 (1/3)

```
……
template:
  ……
  spec:
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
    volumes:
      - name: sessiondb-config
        configMap:
          name: sessiondb-liberty-config
      - name: sessiondb-secret
        secret:
          secretName: sessiondb-liberty-secret
      - name: sessiondb-driver
        secret:
          secretName: sessiondb-jdbc-driver
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
      - name: keystores
          ……
      - name: ssl-config
        configMap:
          name: liberty-config
          items:
            - key: keystore.xml
              path: defaults/keystore.xml
    {{ end }}
    ……
```

移動

削除

Chart の deployment.yaml の変更例 (2/3)

```
……
imagePullPolicy: {{ .Values.image.pullPolicy }}
env:
- name: SESSION_DB_URL
  valueFrom:
    configMapKeyRef:
      name: sessiondb-liberty-config
      key: SESSION_DB_URL
- name: LICENSE
……
```

ConfigMap と Secret を使用したセッションDBの構成例（5）

- Helm Chart を作成します（続き）
  - 今回追加した ConfigMap と Secret 用の volumeMounts を追加します
  - 更に、既存のクラスター SSL 構成用の ConfigMap の volumeMount を変更します

Chart の deployment.yaml の変更例 (3/3)

```
.........
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
volumeMounts:
- name: sessiondb-config
  mountPath: /config/configDropins/overrides/sessiondb-config.xml
  subPath: sessiondb-config.xml
- name: sessiondb-secret
  mountPath: /config/configDropins/overrides/sessiondb-secret.xml
  subPath: sessiondb-secret.xml
- name: sessiondb-driver
  mountPath: /SessionJD
  readOnly: true
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
{{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration -}}
- name: keystores
  mountPath: /etc/wlp/config/keystore
  readOnly: true
- name: truststores
  mountPath: /etc/wlp/config/truststore
  readOnly: true
- name: ssl-config
  mountPath: /config/configDropins/defaults/keystore.xml
  subPath: keystore.xml
{{ end }}
.........
```

移動

変更

ConfigMap と Secret を使用したセッションDBの構成例（6）

- 変更した Helm Chart でインストールします
- セッション・パーシスタンスの稼働状況を確認します
  - 起動すると、セッションDB内にセッション・パーシスタンス用のテーブル sessions が作成されます
  - アプリケーションが HTTP セッションを作成すると、HTTP セッション・データがテーブルに保管されます

セッションDBの確認例

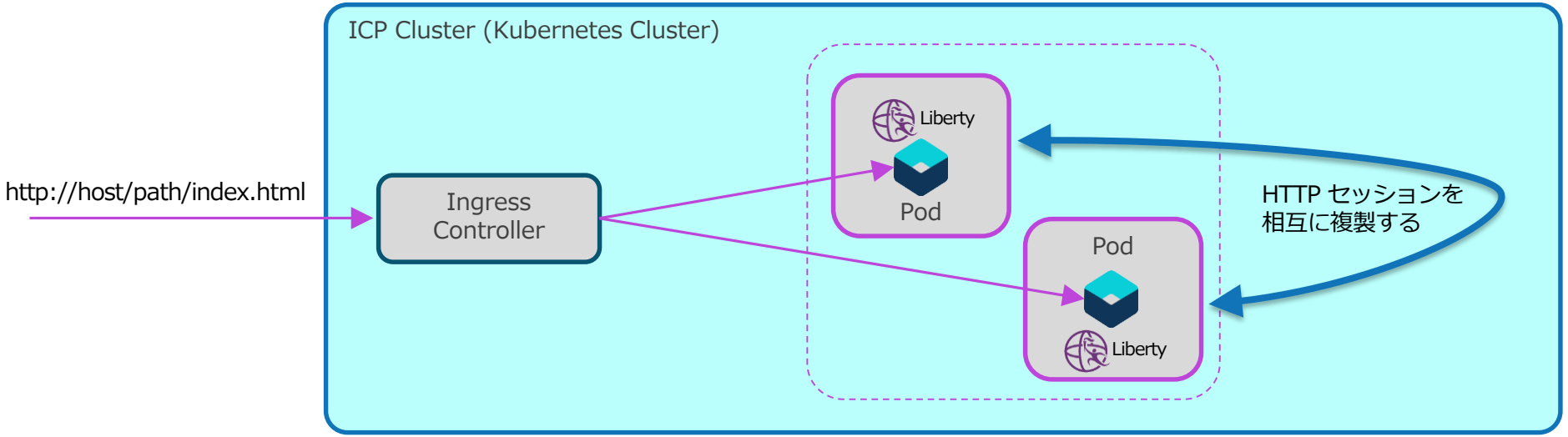
```
mysql> use my_database
Database changed

mysql> select id, propid, appname, listenercnt, lastaccess, creationtime, maxinactivetime, username, length(small), length(medium), length(large) from sessions;
+------------------------+------------------------+--------------+-------------+--------------+--------------+-----------------+-----------+--------------+----------------+---------------+
| id                     | propid                 | appname      | listenercnt | lastaccess   | creationtime | maxinactivetime | username  | length(small) | length(medium) | length(large) |
+------------------------+------------------------+--------------+-------------+--------------+--------------+-----------------+-----------+--------------+----------------+---------------+
| default_host/          | default_host/          | default_host/ |        NULL | 1529630466096 |         NULL |              -1 | NULL      |          NULL |           NULL |          NULL |
| r4-dxq4qLV9pKjvAf_mdsDL | r4-dxq4qLV9pKjvAf_mdsDL | default_host/ |           1 | 1529630159538 | 1529629822617 |            1800 | anonymous |           169 |           NULL |          NULL |
+------------------------+------------------------+--------------+-------------+--------------+--------------+-----------------+-----------+--------------+----------------+---------------+
2 rows in set (0.27 sec)set (0.01 sec)

mysql>
```

  - HTTP セッションが存在する WAS Liberty の Pod を、kubectl delete pod コマンド等で停止させても、別 Pod 内で稼動する WAS Liberty に HTTP セッションが引き継がれます

12. JCache を使用したセッション・パーシスタンス

JCache を使用したセッション・パーシスタンス

- WAS Liberty 18.0.0.2（2018年6月末リリース）から、JCache を使用したセッション・パーシスタンス機能が提供されています
  - Liberty Knowledge Center: Configuring Liberty session persistence with JCache
    - https://www.ibm.com/support/knowledgecenter/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_admin_session_persistence_jcache.html
  - Liberty Knowledge Center: JCache Session Persistence
    - https://www.ibm.com/support/knowledgecenter/SSAW57_liberty/com.ibm.websphere.liberty.autogen.nd.doc/ae/rwlp_feature_sessionCache-1.0.html
  - Open Liberty Blog: Adding distributed in-memory session caching to your Java apps
    - https://openliberty.io/blog/2018/03/22/distributed-in-memory-session-caching.html
- ここでは、JCache 実装としてオープンソース版の Hazelcast を使用して、相互レプリケーション方式(Peer-to-Peer構成)のセッション・パーシスタンスの構成例を紹介します
  - Hazelcast は上記 URL で紹介されている JCache 実装の１つです



Kubernetes 環境での Hazelcast の利用

- Hazelcast を Kubernetes 環境で利用するための情報が、下記の URL で紹介されています
  - GitHub: hazelcast/hazelcast-kubernetes
    - https://github.com/hazelcast/hazelcast-kubernetes
- Hazelcast は Hazelcast クラスター内のメンバー間でデータをレプリケーションします。
- Kubernetes 環境の場合、以下の方法でメンバーを発見します。
  - REST API Request
    - Kubernetes Service Discovery REST API を使用してメンバーを発見します
    - この方法を使用するにはサービスが必要になります
    - WAS Liberty を IBM 提供の Helm Chart でインストールすると、サービスが定義されるので、それを利用できます
  - DNS Lookup
    - ヘッドレス・サービスを Kubernetes DNS で lookup することでメンバーを発見します
    - この方法を使用する場合は、ヘッドレス・サービスが必要になります
    - WAS Liberty が StatefulSet としてインストール(*)すると、StatefulSet の前提としてヘッドレス・サービスが定義されるので、それを利用できます
      - *logs.persistLogs または logs.persistTransactionLogs に true を指定すると、StatefulSet としてインストールされます*
- Hazelcast を Kubernetes 環境でするには、hazelcast-kubernetes とその依存モジュールが必要になります
  - Maven Repository: com.hazelcast » hazelcast-kubernetes
    - https://mvnrepository.com/artifact/com.hazelcast/hazelcast-kubernetes

WAS Liberty の構成

- JCache を使用したセッション・パーシスタンスを使用するには、以下の構成定義を WAS Liberty に追加する必要があります

```
<?xml version="1.0" encoding="UTF-8"?>
<server>

  <featureManager>
    <feature>sessionCache-1.0</feature>
  </featureManager>

  <!-- point to the Hazelcast configuration file with the 'uri' attribute -->
  <httpSessionCache libraryRef="JCacheLib" uri="file:${server.config.dir}/hazelcast-config.xml"/>

  <library id="JCacheLib">
    <file name="/usr/lib/hazelcast-kubernetes-0.0.1-SNAPSHOT-jar-with-dependencies.jar"/>
  </library>

</server>
```

sessionCache-1.0 のフィーチャーを追加

Hazelcast の構成ファイルの URI を指定

hazelcast-kubernetes の jar とその依存モジュールの jar が必要

この例は、hazelcast-kubernetes と全ての依存モジュールを含む jar を maven でビルド(パッケージング)して使用した場合のもの

Hazelcast の構成

- Hazelcast の構成例は以下のようになります。メンバーの発見方式によって、discovery-strategy の property に指定する内容が変わります

```
<hazelcast xmlns="http://www.hazelcast.com/schema/config"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.hazelcast.com/schema/config
                               http://www.hazelcast.com/schema/config/hazelcast-config-3.11.xsd">
  <properties>
    <!-- only necessary prior Hazelcast 3.8 -->
    <property name="hazelcast.discovery.enabled">true</property>
  </properties>
  <network>
    <join>
      <!-- deactivate normal discovery -->
      <multicast enabled="false"/>
      <tcp-ip    enabled="false" />
      <!-- activate the Kubernetes plugin -->
      <discovery-strategies>
        <discovery-strategy
              enabled="true"
              class="com.hazelcast.kubernetes.HazelcastKubernetesDiscoveryStrategy">
          <properties>
            <!-- configure discovery service API lookup -->



          </properties>
        </discovery-strategy>
      </discovery-strategies>
    </join>
  </network>
</hazelcast>
```

XML ネームスペース等の指定を省略すると
正しく認識されないことがあります

REST API Request で発見する場合

```
<property name="service-name">hogehoge-custom-liberty</property>
<property name="namespace">default</property>
```

DNS Lookupで発見する場合

```
<property name="service-dns">hogehoge-custom-liberty</property>
<property name="service-dns-timeout">5</property>
```

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（１）

- この例では、以下の前提で、JCache と Hazelcast を使用したセッション・パーシスタンスを構成します
  - ConfigMap の作成は Helm Chart で行います
    - WAS Liberty をインストールすると必要な ConfigMap が作成され、アンインストールすると ConfigMap も削除されます
  - Hazelcast の発見方式は、２つの方式の稼動をテストするために、次のように異なる方式としています
    - REST API Request 方式： WAS Liberty が Deployment/ReplicaSet としてインストールされる場合
    - DNS Lookup方式： WAS Liberty が StatefulSet としてインストールされる場合
  - hazelcast-kubernetes とその依存モジュールを含む jar ファイルは、予め Docker イメージ内に配置されているものとします
  - WAS Liberty のフィーチャー「sessionCache-1.0」は、予め Docker イメージに含まれているものとします

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（2）

- Helm Chart を作成します
  - IBM 提供の Helm Chart を元にして新しい Chart を作成します
  - templates ディレクトリー内に、ConfigMap 用のテンプレート session-cache-by-hazelcast.yaml を作成します

```
{{- $stateful := or .Values.logs.persistTransactionLogs .Values.logs.persistLogs -}}
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "fullname" . }}
  labels:
    chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
    app: {{ template "fullname" . }}
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
data:
  session-cache.xml: |
    <?xml version="1.0" encoding="UTF-8"?>
    <server>
      <featureManager>
        <feature>sessionCache-1.0</feature>
      </featureManager>
      <!-- point to the Hazelcast configuration file with the 'uri' attribute -->
      <httpSessionCache libraryRef="JCacheLib" uri="file:${server.config.dir}/hazelcast-config.xml"/>
      <library id="JCacheLib">
        <file name="/usr/lib/hazelcast-kubernetes-0.0.1-SNAPSHOT-jar-with-dependencies.jar"/>
      </library>
    </server>
```

変数 stateful の設定
templates ディレクトリー内の他のファイル(deployment.yaml など)からコピー

templates ディレクトリー内の他のファイル(deployment.yaml など)からコピー

WAS Liberty の構成情報
(session-cache.xml) の定義

次ページに続く…

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（3）

前ページからの続き

```
hazelcast-config.xml: |
  <hazelcast xmlns="http://www.hazelcast.com/schema/config"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.hazelcast.com/schema/config
                                 http://www.hazelcast.com/schema/config/hazelcast-config-3.11.xsd">
    <properties>
      <!-- only necessary prior Hazelcast 3.8 -->
      <property name="hazelcast.discovery.enabled">true</property>
    </properties>
    <network>
      <join>
        <!-- deactivate normal discovery -->
        <multicast enabled="false"/>
        <tcp-ip    enabled="false" />
        <!-- activate the Kubernetes plugin -->
        <discovery-strategies>
          <discovery-strategy
                enabled="true"
                class="com.hazelcast.kubernetes.HazelcastKubernetesDiscoveryStrategy">
            <properties>
              <!-- configure discovery service API lookup -->
              {{ if $stateful }}
                <property name="service-dns">{{ template "fullname" . }}</property>
                <property name="service-dns-timeout">5</property>
              {{ else }}
                <property name="service-name">{{ template "fullname" . }}</property>
                <property name="namespace">{{ .Release.Namespace }}</property>
              {{ end }}
            </properties>
          </discovery-strategy>
        </discovery-strategies>
      </join>
    </network>
  </hazelcast>
```

Hazelcast の構成情報
(hazelcast-config.xml) の定義

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（4）

- Helm Chart の deployment.yaml に変更を加えます
  - ConfigMap 用の volume 定義を追加します
  - 更に、クラスター SSL 構成用の ConfigMap の volume 定義を修正(一部削除)します
    - *IBM 提供の Chart では、生成される内容を if 文で制御していますが、今回の変更を適用するには、if 文の位置を移動させる必要があります。*

Chart の deployment.yaml の変更例 (1/2)

```
  ……
template:
  ……
  spec:
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
    volumes:
      - name: session-cache-by-hazelcast
        configMap:
          name: {{ template "fullname" . }}
    {{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration }}
      - name: keystores
          ……
      - name: ssl-config
        configMap:
          name: liberty-config
          items:
            - key: keystore.xml
              path: defaults/keystore.xml
    {{ end }}
    ……
```

移動

削除

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（5）

- Helm Chart の deployment.yaml に変更を加えます（続き）
  - 今回追加した ConfigMap 用の volumeMounts を追加します
  - 更に、既存のクラスター SSL 構成用の ConfigMap の volumeMount を変更します

Chart の deployment.yaml の変更例 (2/2)

```
.....
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
volumeMounts:
- name: session-cache-by-hazelcast
  mountPath: /config/configDropins/overrides/session-cache.xml
  subPath: session-cache.xml
- name: session-cache-by-hazelcast
  mountPath: /config/hazelcast-config.xml
  subPath: hazelcast-config.xml
{{ if or $stateful (and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration) }}
{{ if and .Values.ssl.enabled .Values.ssl.useClusterSSLConfiguration -}}
- name: keystores
  mountPath: /etc/wlp/config/keystore
  readOnly: true
- name: truststores
  mountPath: /etc/wlp/config/truststore
  readOnly: true
- name: ssl-config
  mountPath: /config/configDropins/defaults/keystore.xml
  subPath: keystore.xml
{{ end }}
.....
```

移動

変更

JCache と Hazelcast を使用したセッション・パーシスタンスの構成例（6）

- 変更した Helm Chart でインストールします
- セッション・パーシスタンスの稼働状況を確認します
  - 起動すると、Hazelcast がメンバーを認識している状況が /logs/messages.log に出力されます

messages.log の出力例

```
... ... ... ...
I [10.1.199.101]:5701 [dev] [3.8.2] Kubernetes Discovery activated resolver: ServiceEndpointResolver
I [10.1.199.101]:5701 [dev] [3.8.2] Activating Discovery SPI Joiner
... ... ... ...
I [10.1.199.101]:5701 [dev] [3.8.2]

Members [1] {
        Member [10.1.199.101]:5701 - 9171b855-fc15-4069-9ace-5e0160830e9b this
}

I [10.1.199.101]:5701 [dev] [3.8.2] [10.1.199.101]:5701 is STARTED
I [10.1.199.101]:5701 [dev] [3.8.2] Initializing cluster partition table arrangement...
I [10.1.199.101]:5701 [dev] [3.8.2] Added cache config: CacheConfig{name='com.ibm.ws.session.meta.default_host%2F', managerPrefix='/hz/file:/opt/ibm/wlp/usr/servers/defaultServer//hazelcast-
config.xml/', inMemoryFormat=BINARY, backupCount=1, hotRestart=HotRestartConfig{enabled=false, fsync=false}}
... ... ... ...
I [10.1.199.101]:5701 [dev] [3.8.2]

Members [2] {
        Member [10.1.199.101]:5701 - 9171b855-fc15-4069-9ace-5e0160830e9b this
        Member [10.1.105.204]:5701 - 15cc4415-6d80-478f-85a7-516773a75328
}

... ... ... ...
```

  - HTTP セッションが存在する WAS Liberty の Pod を、kubectl delete pod コマンド等で停止させても、別 Pod 内で稼動する WAS Liberty に HTTP セッションが引き継がれます

【参考】
Ingress Controller のカスタマイズ

Ingress Controller が使用するデフォルトの証明書の変更（１）

- Ingress Controller は Proxy Node で稼動する DaemonSet として定義されています
  - Ingress Controller のネームスペースは kube-system です
- Ingress Controller の起動引数に「--default-ssl-certificate=(ネームスペース)/(TLS Secret名)」を追加することで、デフォルトの証明書を変更できます
  - 参考：NGINX Ingress Controller - Command line arguments
    - https://kubernetes.github.io/ingress-nginx/user-guide/cli-arguments/
- 以下に手順を記載します
  - まず、サーバー証明書と鍵を TLS Secret として登録します
    - NGINX Ingress Controller の仕様にあわせて、証明書と鍵は「PEM-encoded X.509, RSA (2048) secret」として登録します
      - *参考:NGINX Ingress Controller - TLS/HTTPS*
      - https://kubernetes.github.io/ingress-nginx/user-guide/tls/
    - TLS Secret は以下のようなコマンドで PEM ファイルから定義できます
      （下記の例では、ネームスペース kube-system 内に定義しています）

```
# kubectl create secret tls proxy-tls-secret --key proxy.key.clear.pem --cert proxy-chain.cert.pem -n kube-system
```

Ingress Controller が使用するデフォルトの証明書の変更（2）

– 次に、起動引数を追加します
  • 起動引数の追加は kubectl edit DaemonSet コマンドで行います

```
# kubectl edit DaemonSet nginx-ingress-controller -n kube-system
daemonset "nginx-ingress-controller" edited
```

表示されたエディター画面で spec.template.spec.containers 内の args に引数を追加し、保存します

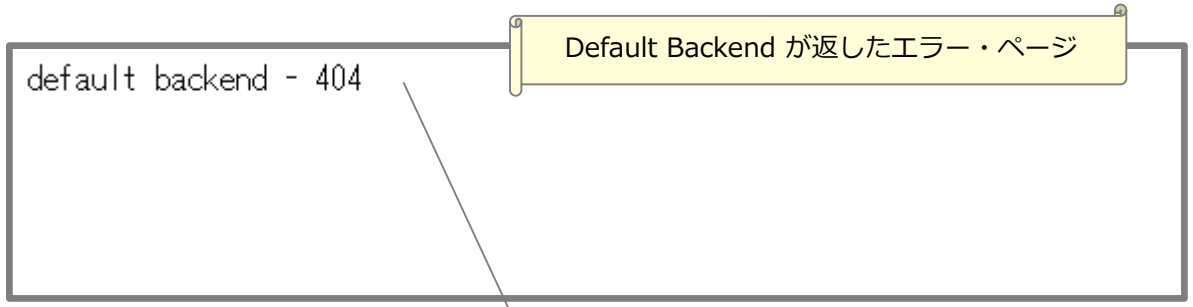
```
 … … …
spec:
  … … …
  template:
      … … …
    spec:
      containers:
      - args:
        - /nginx-ingress-controller
        - --default-backend-service=$(POD_NAMESPACE)/default-http-backend
        - --default-ssl-certificate=kube-system/proxy-tls-secret
        - --configmap=$(POD_NAMESPACE)/nginx-ingress-controller
        - --report-node-internal-ip-address=true
        - --annotations-prefix=ingress.kubernetes.io
          … … …
```

  • ICP Management Console の場合は、 [DaemonSets]画面から追加します

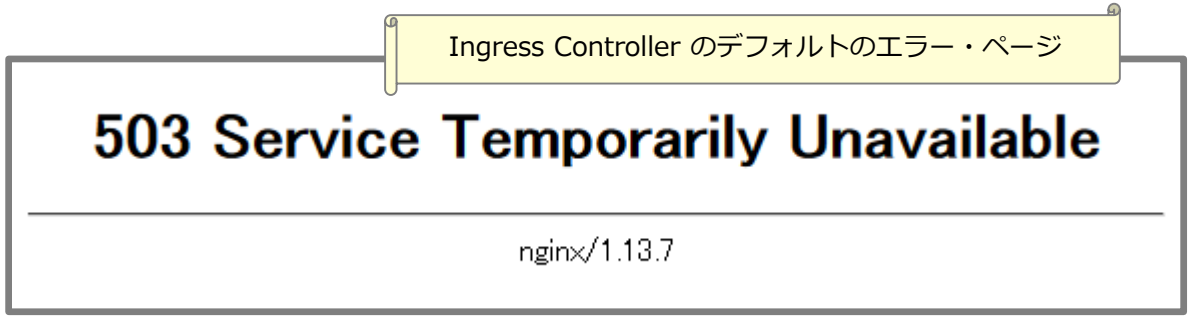
– 変更を保存すると、Ingress Controller が自動的に再起動され、変更が反映されます

エラー・ページのカスタマイズ：デフォルトのエラー・ページ

- Ingress Controller はデフォルトで次のエラー・ページを使用します
  - Default Backend が返したページ
    - 割り振り先サービスが無い URL へのアクセスで 404 エラーを返す場合
  - Ingress Controller のデフォルトのエラー・ページ
    - その他の場合（割り振り先のサービスを提供する Pod が全て停止している場合など）



デフォルトの Default Backend は、HTTP ステータス・コードに関係なく、
固定のエラー･メッセージ「default backend – 404」を返します



エラー・ページのカスタマイズ： custom-http-errors の指定

- Ingress Controller のデフォルトのエラー・ページの代わりに、Default Backend が生成したエラー・ページを使用するように構成できます
  - ネームスペース kube-system 内の ConfigMap nginx-ingress-controller に custom-http-errors の指定を追加すると、指定された HTTP ステータス・コード用のエラー・ページとして Default Backend が生成したページが使用されるようになります
    - エラー・ページをカスタマイズするには、Default Backend を差し替える必要があります
    - 404 などのアプリケーションが返す可能性がある HTTP ステータス・コードを指定すると、アプリケーションが返したエラー・ページも Default Backend が生成したページで置き換わります
    - 参考：NGINX Ingress Controller - ConfigMap
      - https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/
  - 追加方法
    - kubectl edit ConfigMap コマンドで追加します
    - ICP Management Console の場合は、 [ConfigMaps]画面から追加します
    - 変更を保存すると、Ingress Controller が変更を検知し、自動的に反映されます

```
# kubectl edit ConfigMap nginx-ingress-controller  -n kube-system
configmap "nginx-ingress-controller" edited
```

表示されたエディター画面で data 内に custom-http-errors の指定を追加し、保存します

```
apiVersion: v1
data:
  custom-http-errors: "500,503"
  disable-access-log: "true"
  keep-alive-requests: "10000"
  upstream-keepalive-connections: "64"
kind: ConfigMap
metadata:
  … … …
```

エラー・ページのカスタマイズ：Default Backend の準備

- エラー・ページをカスタマイズするために、Default Backend として機能する Pod を準備します
  - 以下の URL に記載されている情報を元に作成します
    - 参考：NGINX Ingress Controller – Default backend
      - https://kubernetes.github.io/ingress-nginx/user-guide/default-backend/
    - 参考：NGINX Ingress Controller - Custom errors
      - https://kubernetes.github.io/ingress-nginx/user-guide/custom-errors/
    - 補足：割り振り先サービスが無い URL へのアクセスで返される 404 エラー・ページの場合は、X-Original-URI 以外のヘッダーはセットされていませんでした

- WAS Liberty で Default Backend を実装することも可能です
  - 上記要件を満たす war モジュールは容易に開発できます
    - 但し、JSP フィーチャーを無効化するか、.jsp や .jspx などが JSP プロセッサーで処理されないように構成する必要があります
  - IBM 提供の Helm Chart またはそれをベースに作成した Chart でインストールできます
    - ingress.enabled=false を指定します
    - Readiness Probe のパスには注意が必要です。Chart を変更してパスを /healthz に変更するか、microprofile 用の Readiness Probe を使用する必要があります
  - Chart でインストールすると、Worker Node で稼動することになります
    （デフォルトの Default Backend は Proxy Node で稼動しています）

エラー・ページのカスタマイズ：Default Backend の差し替え

- Ingress Controller の起動引数「--default-backend-service =(ネームスペース)/(サービス名)」を変更することで、 Default Backend を差し替えます
  - 参考：NGINX Ingress Controller - Command line arguments
    - https://kubernetes.github.io/ingress-nginx/user-guide/cli-arguments/

– 変更方法
  - 起動引数の変更は kubectl edit DaemonSet コマンドで行います

```
# kubectl edit DaemonSet nginx-ingress-controller -n kube-system
daemonset "nginx-ingress-controller" edited
```

表示されたエディター画面で spec.template.spec.containers 内の args の --default-backend-service の指定を変更し、保存します

```
 … … …
spec:
  … … …
  template:
      … … …
    spec:
      containers:
      - args:
        - /nginx-ingress-controller
        - --default-backend-service=default/custom-default-backend-i
        - --default-ssl-certificate=kube-system/proxy-tls-secret
        - --configmap=$(POD_NAMESPACE)/nginx-ingress-controller
        - --report-node-internal-ip-address=true
        - --annotations-prefix=ingress.kubernetes.io
          … … …
```

デフォルト値：$(POD_NAMESPACE)/default-http-backend

  - ICP Management Console の場合は、 [DaemonSets]画面から追加します

– 変更を保存すると、Ingress Controller が自動的に再起動され、変更が反映されます

IBM Cloud Private

クラウド・ネイティブの能力をオンプレに