WebSphere Liberty on OpenShift Container Platform 利用ガイド ～ Helm 編 ～

version 1.0.0

2020/04/03

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2020年3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 19.0.0.12
  - IBM Charts, ibm-websphere-liberty v1.10.0
  - Red Hat OpenShift Container Platform v4.3.5 (ベータ版)
- 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。

- IBM, IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。

- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
  - Red Hat、Red Hat Enterprise Linux、Shadowmanロゴ、およびOpenShiftは、米国およびその他の国におけるRed Hat, Inc.またはその子会社の登録商標です。
  - Kubernetesは、The Linux Foundationの米国およびその他の国における登録商標です。
  - DockerおよびDockerロゴは、Docker Inc.の米国およびその他の国における商標または登録商標です。
  - JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。
  - Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
  - UNIXは、The Open Groupの米国およびその他の国における登録商標です。
  - Microsoft, Windows および Windowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。

- 当資料をコピー等で複製することは、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承諾なしではできません。

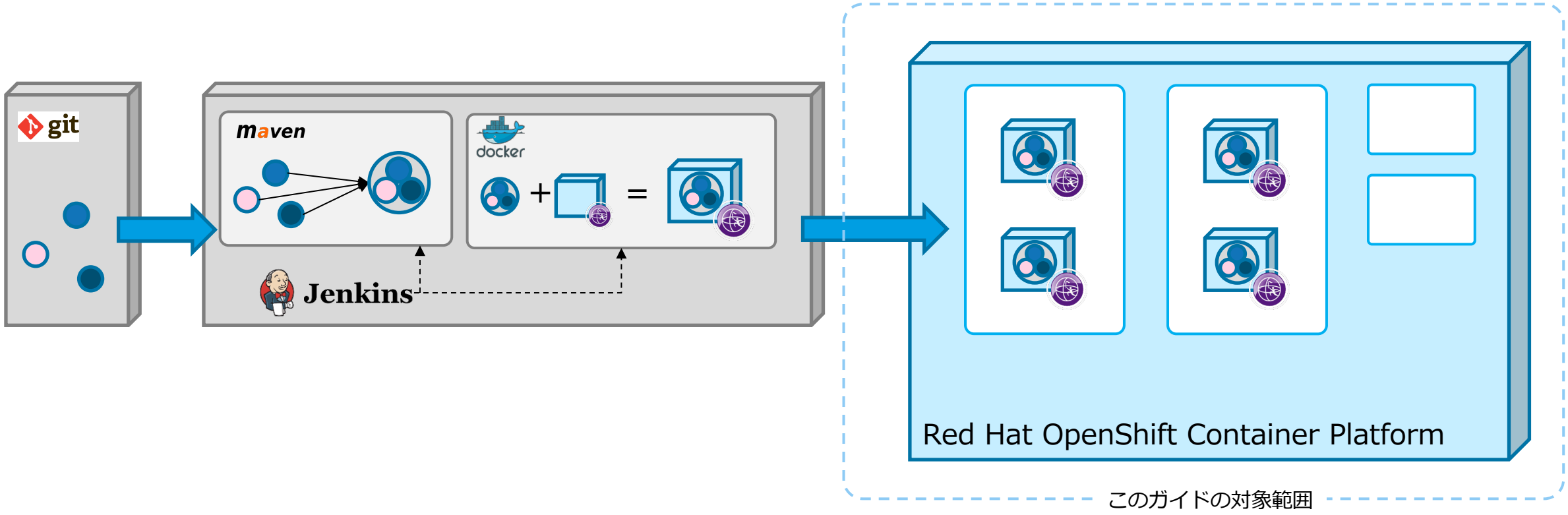
資料変更履歴

■ 本資料の変更履歴を以下に示します。

バージョン	発行日	変更内容
1.0.0	2020/04/03	初版

このガイドについて

- このガイドでは、Red Hat OpenShift Container Platform (OCP) 環境での WebSphere Liberty の構成と稼働を解説しており、主に既存のアプリケーションを OCP 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、WebSphere 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. OpenShift Container Platform 概要

OpenShift Container Platform (OCP) とは

- OpenShift Container Platform (OCP) は、コンテナー化されたアプリケーションを開発/実行するための環境で、Red Hat 社が提供するオープンソースベースの製品です。

- 以下のような特徴を持ちます。
  - エンタープライズ向け Kubernetes プラットフォーム
  - セキュリティを重視
  - サポート付き

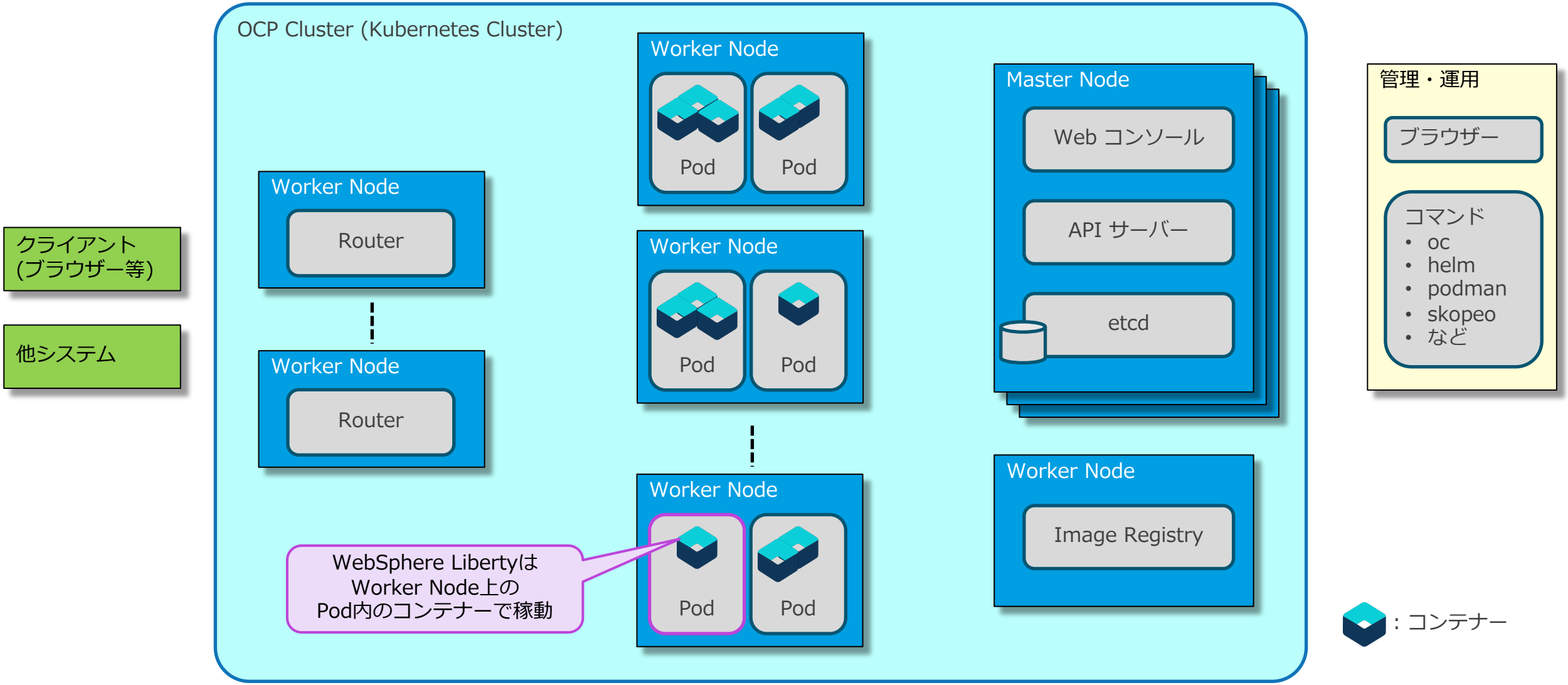
Red Hat OpenShift - 企業向け Kubernetes & Docker コンテナ基盤：特徴と利点
https://www.redhat.com/ja/technologies/cloud-computing/openshift#features

- 以下のような機能が統合されています。
  - Linux オペレーティングシステム -- Red Hat Enterprise Linux CoreOS (RHCOS)
  - ネットワーキング
  - オーケストレーション機能 -- Kubernetes (k8s) + OCP 拡張
  - コンテナー・エンジンおよびツール -- CRI-O, podman, buildah, skopeo（Docker の置き換え）
  - ロギング -- Elasticsearch/Fluentd/Kibana (EFK)
  - モニタリング -- Prometheus/Grafana
  - イメージ・レジストリー -- 統合 OpenShift Container Platform レジストリー
  - など

- IBM Cloud Pak for Applications (CP4A) は、OCP のライセンスを含みます
  - CP4A のライセンスで、WebSphere Liberty を OCP 環境で実行可能
  - WebSphere Liberty と OCP の製品サポートを受けることが可能

OpenShift Container Platform (OCP) の構成 (1)

■ 下図は OCP 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。



OpenShift Container Platform (OCP) の構成 (2)

- OCP Cluster
  - 下記の Node (OCP Cluster を構成する物理または論理マシン)で構成され、Master Node によって管理されます。
- Master Node
  - クラスターを管理するノードです。
  - 複数の Master Node を配置することで高可用性を確保できます。
- 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。
  - 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。
- Router
  - 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コンポーネントで、リバース・プロキシーの機能を提供します。
  - 複数の Router を稼働させることで高可用性を確保できます。
- Image Registry
  - コンテナーのイメージを格納するためのレジストリー（統合 OpenShift Container Platform レジストリー）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、OCP (k8s) の管理単位となります。
  - WebSphere Liberty は Pod 内のコンテナーで稼動します。
  - 各 Node で稼動する OCP (k8s) の機能も Pod として実装されているものがありますが、前の図ではその様には示されていません。
- コンテナー
  - コンテナーのインスタンスです。

OCP Product Documentation : アーキテクチャー : 3.1. OpenShift Container Platform コントロールプレーンについて
https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/architecture/control-plane

OpenShift Container Platform (OCP) の管理操作

- OCP 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。

- Web コンソール
  - OCP 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。

- API サーバー
  - 主に oc コマンドを使用して、OCP 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 コマンドを使用して、アプリケーションやミドルウェアをデプロイ(インストール)/更新/削除するためも使用します。
    helm コマンドは、Helm 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デプロイ(インストール)/更新します。

- Image Registry
  - コンテナーのイメージを管理(登録/確認/削除)するため使用する、統合 OpenShift Container Platform レジストリーです。
  - 登録(push)は podman、skopeo、docker コマンドなどで、イメージの確認/削除は oc コマンドなどで行います。

OpenShift Container Platform (OCP) の管理操作：補足

- Web コンソールへのアクセス方法や、各コマンドの利用方法に関しては、OCP の Product Documentation などを参照してください。

- Web コンソール
  - OCP Product Documentation：Web コンソール：第1章 Web コンソールへのアクセス
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/web_console/web-console

- oc コマンド
  - OCP Product Documentation：CLI ツール：第1章 OpenShift CLI (oc)
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/cli_tools/openshift-cli-oc

- helm コマンド
  - OCP Product Documentation：CLI ツール：第3章 Helm CLI
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/cli_tools/helm-cli
  - Helm - Docs - Helm Commands
    - https://helm.sh/docs/helm/helm/

- コンテナー・ツール(podman, skopeo)、および、dockerコマンド
  - OCP Product Documentation：Red Hat Enterprise Linux：コンテナーの構築、実行、および管理：8.1. podman
    - https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#podman
  - OCP Product Documentation：Red Hat Enterprise Linux：コンテナーの構築、実行、および管理：8.3. skopeo
    - https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#skopeo
  - Docker Engine：Engine リファレンス：コマンドライン・リファレンス
    - http://docs.docker.jp/engine/reference/commandline/toc.html

基礎知識：Workload

- 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 OCP 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。（詳細は各URLを参照）

- 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。
  - Kubernetes ドキュメント：コンセプト：ワークロード：コントローラー：ReplicaSet
    - https://kubernetes.io/ja/docs/concepts/workloads/controllers/replicaset/

- 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。
  - Kubernetes ドキュメント：コンセプト：ワークロード：コントローラー：Deployment
    - https://kubernetes.io/ja/docs/concepts/workloads/controllers/deployment/

- StatefulSet
  - テンプレートに従ってPodを起動し管理します。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。
  - Kubernetes ドキュメント：コンセプト：ワークロード：コントローラー：StatefulSet
    - https://kubernetes.io/ja/docs/concepts/workloads/controllers/statefulset/

基礎知識：Service と Endpoint

- Service は、Deployment 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。（詳細は下記URLを参照）
  - Kubernetes ドキュメント：コンセプト：Service、負荷分散とネットワーキング：Service
    - https://kubernetes.io/ja/docs/concepts/services-networking/service/

- 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
    - OCP 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。
  - IP アドレスのリストは Readiness Probe (次ページ参照)の結果に基づいて更新されます。
  - 上述の「Kubernetes ドキュメント：コンセプト：Service、負荷分散とネットワーキング：Service」参照

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。
  （詳細は下記URLを参照）
  - Kubernetes ドキュメント：コンセプト：ワークロード： Pod：Podのライフサイクル
    - https://kubernetes.io/ja/docs/concepts/workloads/pods/pod-lifecycle/

- 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、コンテナーは停止され、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。
  （詳細は下記URLを参照）
  - Kubernetes Document: Concepts: Storage: Volumes: ConfigMap
    - https://kubernetes.io/docs/concepts/storage/volumes/#configmap
  - Kubernetes Document: Concepts: Configuration: Secret
    - https://kubernetes.io/docs/concepts/configuration/secret/

- 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。

- 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

- Persistent Volume は、Pod に永続領域を提供します。（詳細は下記URLを参照）
  - Kubernetes Document: Concepts: Storage: Persistent Volumes
    - https://kubernetes.io/docs/concepts/storage/persistent-volumes/
  - OCP Product Documentation : ストレージ
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/storage/index

- 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. OCP 環境での WebSphere Liberty

OCP 環境での WebSphere Liberty の可用性と拡張性

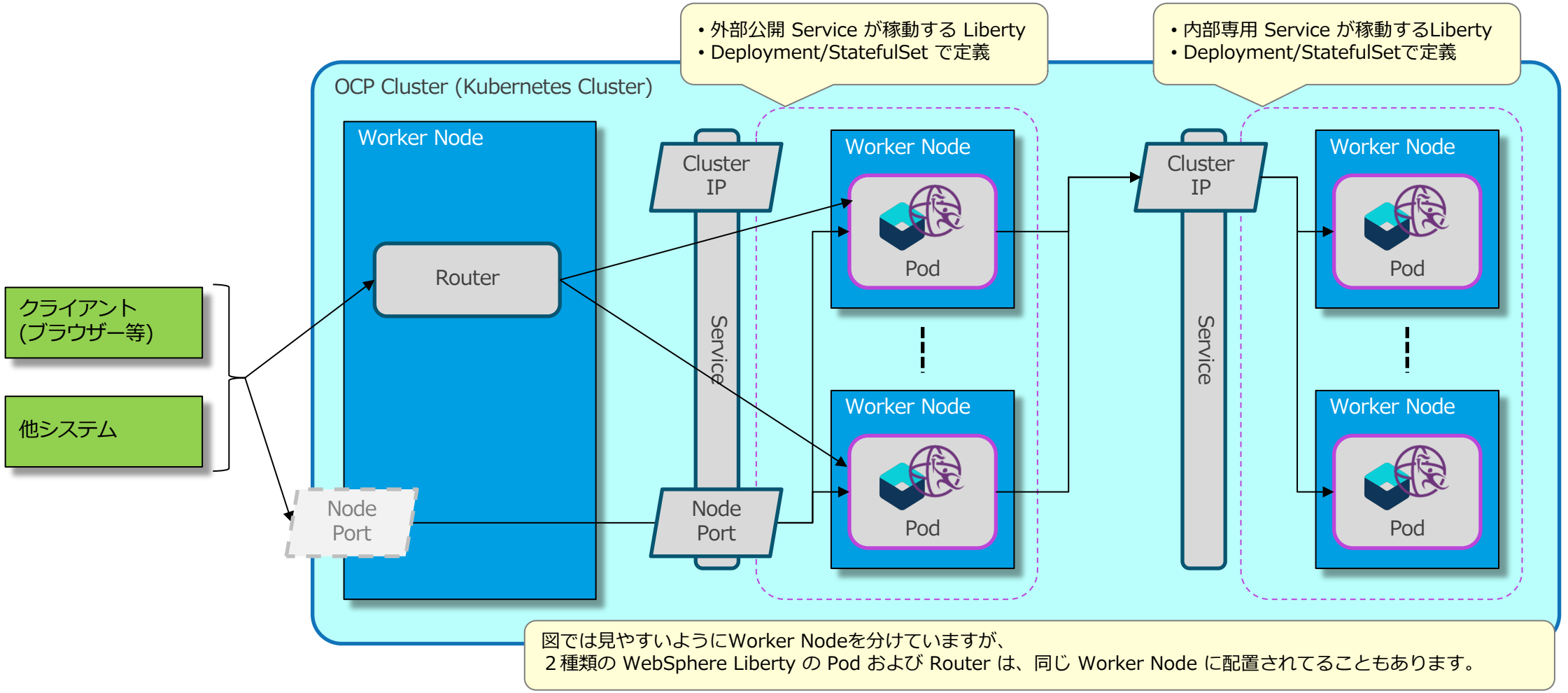
- WebSphere Liberty は Worker Node 上の Pod 内のコンテナーで稼動します。

- WebSphere Liberty の可用性・拡張性は OCP の機能で確保します。
  - WebSphere 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 OCP 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。
  - WebSphere 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/Route によって行われます。(後述)
  - オート・スケーリング
    - Horizontal Pod Autoscaler 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、WebSphere Liberty の Pod を Deployment としてデプロイした場合(ログを永続化していない場合)のみです。

OCP 環境で稼動する WebSphere Liberty へのアクセス：概要

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



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

■ OCP クラスター外からアクセスする方法は２つあります。

– Router を経由したアクセス
- Router (Worker 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 WebSphere 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 OCP 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）
- Router は HAProxy 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、任意の Node の IP と Node Port に割り当てられたポート番号で行います。
  – 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 OCP 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。

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

■ OCP 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 OCP 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. 基本的なデプロイ（１）

目次(3章)

概要

- この章では、OCP環境にWebSphere LibertyのPodをデプロイする基本的な流れと操作を解説します。

- 作業は以下の２つのパートで構成されます。
  - 準備
    - デプロイ前に準備・設定する内容です。
  - 基本的なデプロイ
    - WebSphere LibertyをOCP環境にデプロイする流れをステップバイステップで説明します。

- 最後に、IBMが提供するWebSphere Liberty用Helm Chartのパラメーターに関して説明します。

コマンドの準備

- OCP環境の操作やコンテナー・イメージのビルドに必要な以下のコマンドを準備します。

- oc コマンド
  - OCP Product Documentation : CLI ツール : 第1章 OpenShift CLI (oc)
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/cli_tools/openshift-cli-oc

- helm コマンド
  - OCP Product Documentation : CLI ツール : 第3章 Helm CLI
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html/cli_tools/helm-cli
  - Helm - Docs - Helm Commands
    - https://helm.sh/docs/helm/helm/

- dockerコマンド、または、コンテナー・ツール(podman, skopeo)
  - Docker Engine : Engine リファレンス : コマンドライン・リファレンス
    - http://docs.docker.jp/engine/reference/commandline/toc.html
  - OCP Product Documentation : Red Hat Enterprise Linux : コンテナーの構築、実行、および管理 : 8.1. podman
    - https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#podman
  - OCP Product Documentation : Red Hat Enterprise Linux : コンテナーの構築、実行、および管理 : 8.3. skopeo
    - https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/8/html-single/building_running_and_managing_containers/index#skopeo

このガイドでは、dockerコマンドを
利用した形式で記述します。

Helm Chartのダウンロード

- IBMが提供するWebSphere Liberty用のHelm Chartをダウンロードします。
  - 以下の URL から最新のHelm Chartをダウンロードします。
    - GitHub / IBM/charts/repo/stable
      - https://github.com/IBM/charts/tree/master/repo/stable
  - ガイド作成時点の最新バージョンは以下の通りでした。
    - ibm-websphere-liberty-1.10.0.tgz

OCP環境の準備（1/2）

■ 統合 OpenShift Container Platform レジストリーの準備

- このガイドは、OCP に統合されているイメージ・レジストリー（統合OCPレジストリー）を使用する前提で記述されています。
- 統合OCPレジストリーを使用するために、以下の準備・確認が必要です。

– 統合OCPレジストリーへのアクセス権

- ビルドしたコンテナー・イメージを統合OCPレジストリーにpushするには、registry-editorロールが必要になります。
- 使用するユーザーにregistry-editorロールが付与(bind)されていることを確認してください。
- OpenShift Container Platform / Registry / Accessing the registry
  - https://docs.openshift.com/container-platform/4.3/registry/accessing-the-registry.html

– 統合OCPレジストリーのエクスポーズ

- OCPクラスター外から統合OCPレジストリーにアクセスするには、統合OCPレジストリーがエクスポーズされている必要があります。
- 統合OCPレジストリーがエクスポーズされているか確認し、アクセスに必要なホスト名を確認します。
- OpenShift Container Platform / Registry / Exposing the registry
  - https://docs.openshift.com/container-platform/4.3/registry/securing-exposing-registry.html

OCP環境の準備（2/2）

■ SecurityContextConstraints (SCC) の定義

- WebSphere LibertyのHelm Chart用にカスタマイズされたSCCを定義します。
- SCCの名前は ibm-websphere-liberty-scc です。

– 以下の URL からSCCの定義を入手し、ファイルに保存します。

- 参照右記の内容
- GitHub / IBM/charts/stable/ibm-websphere-liberty/
  - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty

– ibm-websphere-liberty-scc を定義します。

- oc apply -f コマンドを実行し、SCC を定義します。

```
> oc apply -f ibm-websphere-liberty-scc.yaml
securitycontextconstraints.security.openshift.io/ibm-websphere-liberty-scc created
```

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
  name: ibm-websphere-liberty-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegedContainer: false
allowedCapabilities: []
allowedFlexVolumes: []
defaultAddCapabilities: []
fsGroup:
  type: MustRunAs
  ranges:
  - max: 65535
    min: 1
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
runAsUser:
  type: MustRunAsNonRoot
seccompProfiles:
- docker/default
seLinuxContext:
  type: RunAsAny
supplementalGroups:
  type: MustRunAs
  ranges:
  - max: 65535
    min: 1
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
priority: 0
```

OCP環境でのWebSphere Libertyのデプロイの流れ

■ WebSphere Libertyのデプロイの流れは以下のようになります。

– (1) コンテナー・イメージをビルドする
- 以下のリソースを組み込んだ、WebSphere Libertyのコンテナー・イメージをビルドします。(ビルド時の考慮点は後述を参照)
  – WebSphere Libertyの構成ファイル(server.xmlなど)
  – 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WebSphere 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)

– (2) コンテナー・イメージを統合OCPレジストリーにpushする
- (1)でビルドしたコンテナー・イメージをOCPのイメージ・レジストリーに登録(push)します。

– (3) SCCをサービス・アカウントに付与(grant)する
- SCC ibm-websphere-liberty-scc をサービス・アカウントに付与しておきます。

– (4) WebSphere LibertyのPodをデプロイする
- 以下の何れかの方法で、WebSphere LibertyのPodをデプロイします。
- 1. Helm Chartを使用する方法
  – IBM提供のHelm Chart、それを変更して作成したHelm Chart、自作のHelm Chartなどを使用してデプロイします。
  – デプロイ操作は、helmコマンドで行います。
- 2. OCPのリソースを直接定義する方法
  – OCP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、oc applyコマンドでデプロイします。
  – OCPリソースの定義情報をOCP Web コンソールの画面で入力することでデプロイすることも可能です。

– 本ガイドでは、IBM提供のWebSphere LibertyのHelm Chartを利用してデプロイする方法を説明していきます。

(1) コンテナー・イメージをビルドする（1/2）

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

  – OCP環境では、Universal Base Imageを元にしたWebSphere Libertyのイメージを使用します。
    - Docker Hub: ibmcom/websphere-liberty
      – https://hub.docker.com/r/ibmcom/websphere-liberty
    - イメージ名は ibmcom/websphere-liberty です。

  – このWebSphere Libertyのイメージには、以下の設定が施されています
    - 補足：このイメージのビルド時に使用されたDockerfileは、以下のURLで確認できます
      – https://github.com/WASdev/ci.docker

    - ユーザー名: default、ユーザーID: 1001、グループID: 0のユーザーを作成
    - WebSphere 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、WebSphere Liberty を起動

(1) コンテナー・イメージをビルドする（2/2）

– 以下のステップを記述したDockerfileを作成します。
  - WebSphere Libertyの構成ファイル(server.xml など)やアプリケーションを、コンテナー・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、コンテナー・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
  - WebSphere Libertyのフィーチャーの追加(必要な場合)

この例では、WebSphere Libertyの構成ファイル(server.xml)を/config/に、サンプル・アプリを/config/dropins/にコピーしています。
さらに、「installUtility install –acceptLicense defaultServer」コマンドを実行し、server.xmlに定義されているフィーチャーをインストールしています。

```
FROM ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi
COPY --chown=1001:0  server.xml /config/
COPY --chown=1001:0  Sample.war /config/dropins/
RUN installUtility install --acceptLicense defaultServer
```

– 「docker build -f Dockerfile –t <イメージ名>:<タグ> .」コマンドを実行し、コンテナー・イメージをビルドします。

- -f オプションで、ビルドに利用するDockerfileを指定します。
- -t mylibertyapp:1.0オプションで、作成するイメージの名前(mylibertyapp)とタグ(1.0)を指定しています。
- 最後の引数に、ビルドで使用されるファイルが格納されているパスまたはURLを指定します。今回は、ローカルのカレント・ディレクトリーにあるため「.」 を指定しています。

```
> ls
Dockerfile      Sum.war         server.xml

> docker build -f Dockerfile -t mylibertyapp:1.0 .
Sending build context to Docker daemon  7.168kB
Step 1/4 : FROM websphere-liberty:19.0.0.6-javaee8
 ---> 7a0094c39470
........

> docker images | grep myliberty
mylibertyapp              1.0                 709ce7a8c295        6 minutes ago       520MB
```

(2) コンテナー・イメージを統合OCPレジストリーにpushする（1/2）

■ ビルドしたコンテナー・イメージを統合OCPレジストリーにpushします。

– 「docker login <統合OCPレジストリーのホスト名>」コマンドで、統合OCPレジストリーへログインします。

```
> docker login default-route-openshift-image-registry.····· -u ユーザー名 -pパスワード(または、トークン)
```

– 「docker tag <ソースの名前:タグ> <統合OCPレジストリーのホスト名>/<ネームスペース>/<名前:タグ>」コマンドで、ビルドしたイメージに統合OCPレジストリーのホスト名とネームスペース(プロジェクト)を示す名前を付与します。

```
> docker tag mylibertyapp:1.0 default-route-openshift-image-registry.·····/wlp-sample/mylibertyapp:1.0

> docker images | grep mylibertyapp
mylibertyapp                                                      1.0   709ce7a8c295   About an hour ago   520MB
default-route-openshift-image-registry.·····/wlp-sample/mylibertyapp   1.0   709ce7a8c295   About an hour ago   520MB
```

– 「docker push <名前:タグ>」コマンドで、統合OCPレジストリーへコンテナー・イメージを登録(push)します。

```
> docker push default-route-openshift-image-registry.·····/wlp-sample/mylibertyapp:1.0
The push refers to repository [default-route-openshift-image-registry.·····/wlp-sample/mylibertyapp]
81bfea1502fb: Pushed
·····
```

(2) コンテナー・イメージを統合OCPレジストリーにpushする（2/2）

- 「oc get imagestream」コマンドで、コンテナー・イメージが統合OCPレジストリーにpushされたことを確認します。
  - さらに詳し情報は 「oc describe imagestream」コマンドで確認できます。

```
> oc get imagestream
NAME            IMAGE REPOSITORY                                                    TAGS   UPDATED
mylibertyapp    image-registry.openshift-image-registry.svc:5000/wlp-sample/mylibertyapp    1.0    1 hour ago

> oc describe imagestream mylibertyapp
Name:                   mylibertyapp
Namespace:              wlp-sample
Created:                1 hour ago
Labels:                 <none>
Annotations:            <none>
Image Repository:       image-registry.openshift-image-registry.svc:5000/wlp-sample/mylibertyapp
Image Lookup:           local=false
Unique Images:          1
Tags:                   1

1.0
  no spec tag

  * image-registry.openshift-image-registry.svc:5000/wlp-sample/mylibertyapp @sha256:085cab·····69d     1 hour ago
```

イメージ名の先頭部分は、OCPクラスター内部において統合OCPレジストリーにアクセスするためのサービス名とポート番号に変わっています。

(3) SCCをサービス・アカウントに付与する

■ 準備作業で定義したSCCをサービス・アカウントに付与(grant)します。

- ibm-websphere-liberty-sccをWebSphere Libertyが使用するサービス・アカウントに付与しておきます。これにより、WebSphere Libertyのコンテナーが指定されたセキュリティー制約の下で実行されるようになります。

– WebSphere Libertyが使用するサービス・アカウントは、 WebSphere LibertyのPodのデプロイ時にRBAC(ロールベースアクセス制御)を有効にするか否かで異なります。

- RBACを有効化した場合のサービス・アカウント：「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字
- RBACを無効化した場合のサービス・アカウント： default

– 「oc adm policy add-scc-to-user ibm-websphere-liberty-scc -z <サービス・アカウント名>」を実行し、サービス・アカウントに SCC（ibm-websphere-liberty-scc）を付与します。

- RBACを有効化している場合の例

```
> oc project wlp-sample

> oc adm policy add-scc-to-user ibm-websphere-liberty-scc -z mylibertyapp-ibm-websphe
```

Helmリソース名をmylibertyappにする場合

- RBACを無効化している場合の例

```
> oc project wlp-sample

> oc adm policy add-scc-to-user ibm-websphere-liberty-scc -z default
```

(4) WebSphere LibertyのPodをデプロイする（1/4）

■ Helm Chartを使ってOCPリソースを定義することで、ビルドしたコンテナー・イメージを使用するWebSphere LibertyのPodをデプロイします。

- 以下の例では、準備としてダウンロードしたHelm Chart が、カレント・ディレクトリー内に配置されている前提になっています。

– Helm Chartから変数ファイルを抽出します。

```
> helm inspect values ./ibm-websphere-liberty-1.10.0.tgz > mylibertyapp.yaml
```

– 抽出した変数ファイルのパラメーターを要件に合わせて修正します。
または、必要なパラメーターだけを含む、以下のような変数ファイルを作成します。

- 変更するパラメーターの説明は、本章の「WebSphere LibertyのHelm Chartのパラメーター：基本的なもの」、および、後続の章を参照

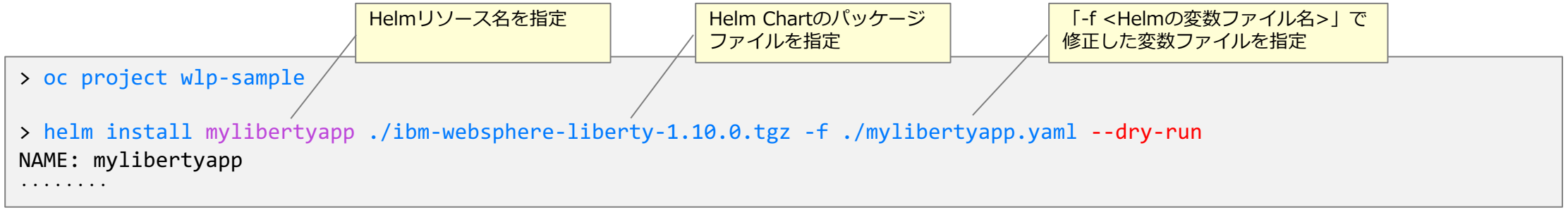
```
image:
  repository: image-registry.openshift-image-registry.svc:5000/wlp-sample/mylibertyapp
  tag: "1.0"
  pullPolicy: Always
  license: "accept"
service:
  enabled: true
  port: 9080
  targetPort: 9080
  type: NodePort
ssl:
  enabled: false
rbac:
  install: false
```

イメージ名の先頭部分は、OCPクラスター内部において統合OCPレジストリーにアクセスするためのサービス名とポート番号で指定

この例では、以下の点をデフォルトから変更しています。
・ビルドしたイメージの名前とタグを指定
・テスト用にpullPolicyはAlwaysに変更
・ライセンス済みに設定
・サービスでHTTP(9080)ポートを公開
・SSLは無効化
・RBACは無効化

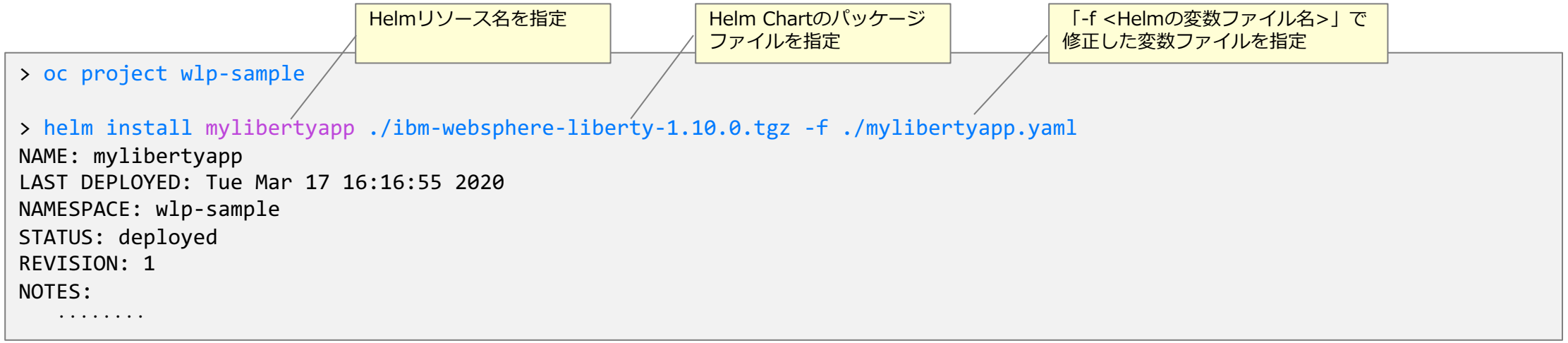
(4) WebSphere LibertyのPodをデプロイする（2/4）

– (オプション) 修正した変数ファイルと「--dry-run」オプションを指定して「helm install」コマンドを実行し、作成されるOCPリソースを事前に確認します。



– 修正した変数ファイルを指定して「helm install」コマンドを実行し、デプロイします。

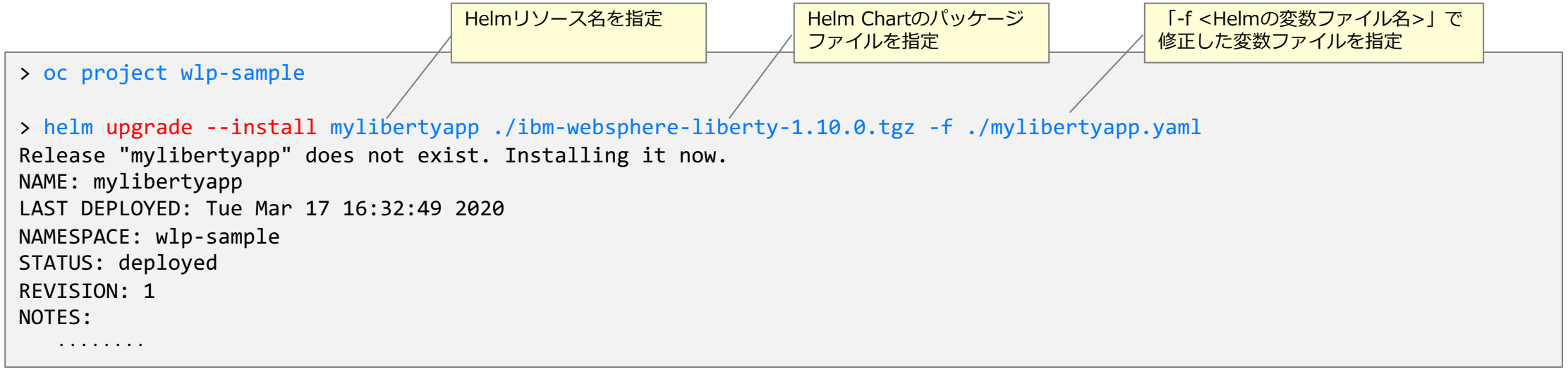
• Helm Chartの指定方法に関しては、本章の「helmコマンドでのHelm Chartの指定方法」を参照



(4) WebSphere LibertyのPodをデプロイする（3/4）

– 補足：「helm install」コマンドの代わりに「helm upgrade --install」コマンドを使用することもできます。

- 「helm upgrade --install」コマンドは以下のような動作をするため、初期デプロイ時、アップグレード時に同じコマンドで操作できます。
  – 同じ名前のHelmリソースがデプロイされていない場合は、新規デプロイする。
  – 同じ名前のHelmリソースがデプロイされている場合は、Helmの変数ファイルの更新部分のみpatchする。



(4) WebSphere LibertyのPodをデプロイする（4/4）

– 正常にデプロイできたか確認します。

- 「helm list」コマンドで、ネームスペース(プロジェクト)内のHelmリソースの名前を確認できます。

```
> helm list
NAME            NAMESPACE     REVISION    UPDATED                 STATUS      CHART                                APP VERSION
mylibertyapp    wlp-sample    1           2020-03-17 ····J ST     deployed    ibm-websphere-liberty-1.10.0    19.0.0.6
```

- 「oc get all -l app=<ラベル値>」コマンドで、作成された主要なOCPリソースの一覧と状況が確認できます。
  – ラベル値は「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字です。

```
> oc get all -l app=mylibertyapp-ibm-websphe
NAME                                          READY   STATUS    RESTARTS   AGE
pod/mylibertyapp-ibm-websphe-5755b7467b-sgrlw   1/1     Running   0          16m

NAME                                TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/mylibertyapp-ibm-websphe    NodePort   172.21.193.150   <none>        9080:30692/TCP   16m

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mylibertyapp-ibm-websphe    1/1     1            1           16m

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/mylibertyapp-ibm-websphe-5755b7467b    1         1         1       16m
```

作成された「Pod」の名前と状況

作成された「Service」の名前と状況

作成された「Deployment」の名前と状況

(5) 動作を確認する

■ デプロイしたWebSphere LibertyのPodの動作を確認します。

- Helm Chartの変数ファイルでingress関連のパラメーターを調整していないので、Ingressが作成されておらず、Routeも生成されていません。（ ingress関連のパラメーターに関しては、次の章で説明します。）
- ここでは、手作業でRouteを定義し、WebSphere Libertyにアクセスしてみます。

– 作成されたServiceを確認し、Routeを作成し、RouteのHOST名を確認します。

```
> oc get service
NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
mylibertyapp-ibm-websphe    NodePort    172.21.193.150   <none>        9080:30692/TCP   16h

> oc expose service mylibertyapp-ibm-websphe
route.route.openshift.io/mylibertyapp-ibm-websphe exposed

> oc get route
NAME                        HOST/PORT                                        PATH     SERVICES                    PORT    ····
mylibertyapp-ibm-websphe    mylibertyapp-ibm-websphe-wlp-sample.·····                 mylibertyapp-ibm-websphe    http    ····
```

– 表示されたHOST名を使用して、 WebSphere Libertyにアクセスします。

```
> curl -s http://mylibertyapp-ibm-websphe-wlp-sample.·····
<!DOCTYPE html>
<html lang="en" xml:lang="en">
<head>
·····
```

(6) WebSphere Libertyをアンデプロイする

■ 動作が確認でき不要になった場合は、WebSphere LibertyのPodをアンデプロイします。

– 「helm uninstall <Helmリリース名>」コマンドを実行し、アンデプロイします。
  • Helm Chartによって作成されたOCPリソースがすべて削除されます。
  • helm v2 までの「helm delete」と同様の動作ですが、 Helmリリースの履歴情報も含めて完全に削除されます。

```
> helm list
NAME            NAMESPACE     REVISION    UPDATED                 STATUS     CHART                              APP VERSION
mylibertyapp    wlp-sample    1           2020-03-17 ····J ST     deployed   ibm-websphere-liberty-1.10.0       19.0.0.6

> helm uninstall mylibertyapp
release "mylibertyapp" uninstalled
```

– 今回は、Routeを手作業で作成したので、Routeを削除します。

```
> oc get route
NAME                        HOST/PORT                                          PATH     SERVICES                    PORT    ····
mylibertyapp-ibm-websphe    mylibertyapp-ibm-websphe-wlp-sample.·····                   mylibertyapp-ibm-websphe    http    ····

> oc delete route mylibertyapp-ibm-websphe
route.route.openshift.io "mylibertyapp-ibm-websphe" deleted
```

(補足)その他のhelmコマンド

■ その他のhelmコマンドをいくつかご紹介します。

– リリースされている内容の表示
  • Helm Chartが作成したOCPリソースのマニフェストの表示

```
# helm get manifest mylibertyapp
```

  • デプロイ時に指定したHelmのパラメーターの設定値の表示

```
# helm get values mylibertyapp
```

– 更新履歴の確認

```
# helm history mylibertyapp
```

– 履歴内のリビジョンを復元

```
# helm rollback mylibertyapp 2
```

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

■ Helmコマンドの利用方法は以下のURLを参照してください。
  • Helm - Docs - Helm Commands
    – https://helm.sh/docs/helm/helm/

WebSphere LibertyのHelm Chartのパラメーター：基本的なもの（1/4）

■ IBM提供のWebSphere LibertyのHelmチャートの基本的なパラメーター

Qualifier	Parameter	デフォルト値	説明
image	repository	ibmcom/websphere-liberty	コンテナー・イメージの名前を指定します。 ここで指定したコンテナー・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 コンテナー・イメージを指定します。 通常、「image-registry.openshift-image-registry.svc:5000/<ネームスペース>/<名前>」の形式になります。 例：image-registry.openshift-image-registry.svc:5000/wlp-sample/my-app
	tag	javaee8-ubi-min	コンテナー・イメージを識別するタグを指定します。
	pullPolicy	IfNotPresent	コンテナー・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。
	license	""	WebSphere 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。
	readinessProbe	{}	(4章を参照)
	livenessProbe	{}	(4章を参照)
	extraEnvs	{}	コンテナーを起動する際に引き渡したい環境変数を設定します。yaml構文で記載します。
	lifecycle	{}	Lifecycle EventのPostStartとPreStopで実行する処理を指定できます。
	serverOverridesConfigMapName	""	(8章を参照)
	extraVolumeMounts	[]	追加でPodにVolumeをマウントしたい場合に指定します。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

WebSphere LibertyのHelm Chartのパラメーター：基本的なもの（2/4）

■ IBM提供のWebSphere LibertyのHelmチャートの基本的なパラメーター

Qualifier	Parameter	デフォルト値	説明
deployment	annotations	{}	Deploymentに指定するannotationを指定します。yaml構文での指定となります。
	labels	{}	Deploymentに指定するlabelを指定します。yaml構文での指定となります。
service	enabled	true	Serviceを利用するかどうかを指定します。デフォルトは利用します。
	type	NodePort	Serviceのtypeを指定します。「ClusterIP」または「NodePort」です。(2章を参照)
	name	""	Service名を指定します。(DNSのAレコードになります。)
	port	9443	ServiceがWebSphere 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、OCP クラスター内からアクセスできるようになります。
	targetPort	9443	WebSphere 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。
	labels	{}	Serviceに追加するラベルを指定します。
	annotations	{}	Serviceに追加するannotationを追加します。
	extraPorts	[]	Serviceに追加するPortを記載します。
	extraSelectors	{}	追加のLabel Selectorを指定します。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

WebSphere LibertyのHelm Chartのパラメーター：基本的なもの（3/4）

■ IBM提供のWebSphere LibertyのHelmチャートの基本的なパラメーター

Qualifier	Parameter	デフォルト値	説明
ssl	enabled	true	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	false	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。 (14章を参照)
	createClusterSSLConfiguration	false	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。 (14章を参照)
ingress	enabled	false	Router経由でのアクセス有無を指定します。 (4章および2章を参照)
	rewriteTarget	"/"	(4章を参照)
	path	"/"	(4章を参照)
	host	""	(4章を参照)
	secretName	""	(4章を参照)
	labels	{}	Ingressリソースに追加するlabelを指定します。
	annotations	{}	Ingressリソースに追加するlabelを指定します。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

WebSphere LibertyのHelm Chartのパラメーター：基本的なもの（4/4）

■ IBM提供のWebSphere LibertyのHelmチャートの基本的なパラメーター

Qualifier	Parameter	デフォルト値	説明
microprofile	health.enabled	false	古いバージョンのチャートにおいて、Readiness Probeのパスを調整するために使用されていたもので、trueを指定すると、Readiness Probeのパスが/から/healthに変わります 現行バージョンでは、images.readinessProbeで詳細に指定できるので、このパラメーターを使用しない(デフォルトのfalseから変更しない)ことをお勧めします。
monitoring	---	---	(7章を参照)
replicaCount	-	1	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	---	---	(6章を参照)
logs	---	---	(6章を参照)
autoscaling	---	---	(11章を参照)
resources	---	---	(11章を参照)
env	jvmArgs	""	JVM_ARGS環境変数を設定します。
rbac	install	true	RBAC(ロールベースアクセス制御)の有効化/無効化を指定します。 有効化すると、Helmリリース単位でサービス・アカウントとロールが作成されます。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

helmコマンドでのHelm Chartの指定方法

■ helm install/upgradeコマンドでは、使用するHelm Chartを以下の方法で指定できます。

- Helm - Docs - Helm Install
  - https://helm.sh/docs/helm/helm_install/

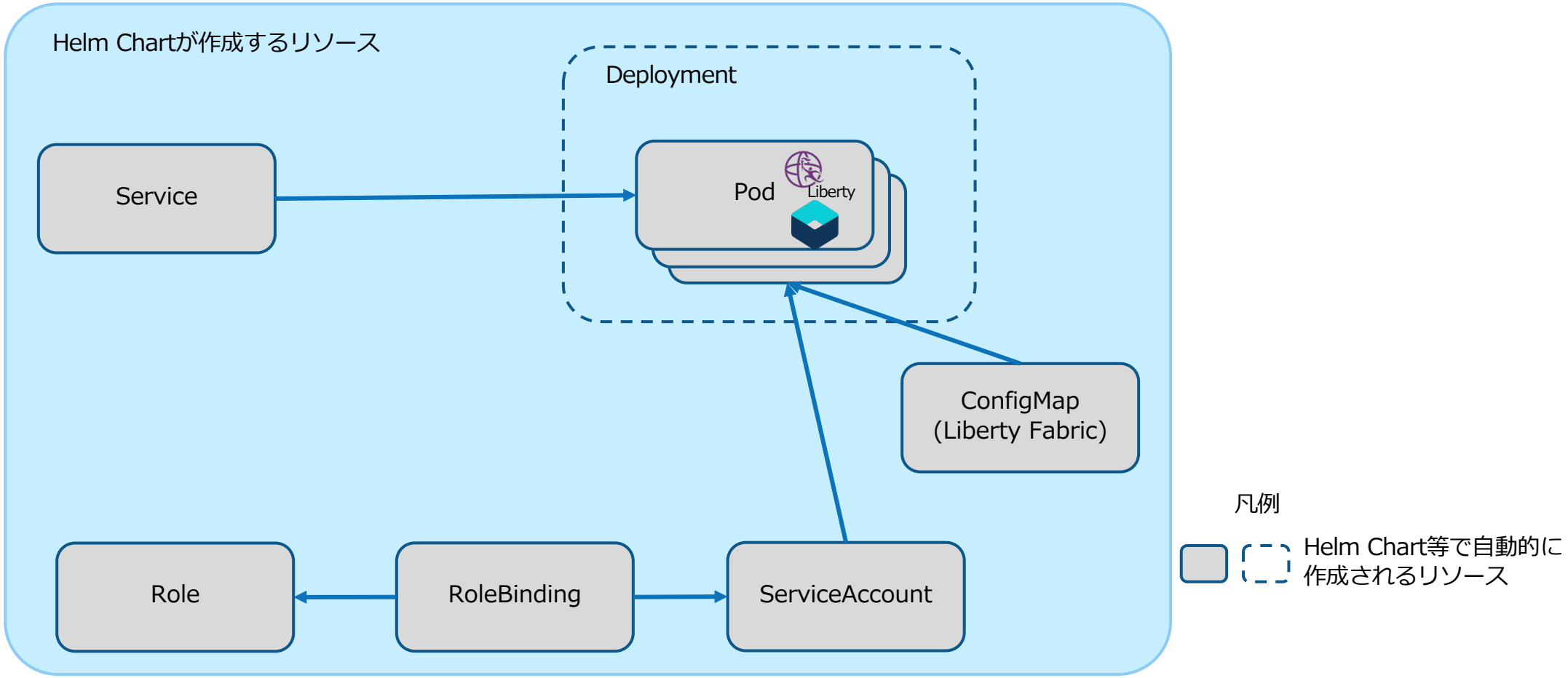
Chart指定方法	helm installコマンドの例	備考
Chart参照	helm install example/mariadb	• 登録されているHelmリポジトリー内のHelm Chartを指定してデプロイする方法です。 • 登録されているHelmリポジトリーはhelm repo listで確認できます。
パッケージングされたChartファイル	helm install ./nginx-1.2.3.tgz	• helmコマンド実行端末上に配置しているtgzでパッケージングされたHelm Chartを指定してデプロイする方法です。
展開されたChartのディレクトリー	helm install ./nginx	• helmコマンド実行端末上に配置している展開済みのHelm Chartを指定して導入する方法です。(helm createを実行、もしくは、tgzのHelm Chartを展開した場合)
ChartのURL	helm install https://example.com/charts/nginx-1.2.3.tgz	• 外部ネットワークに接続しているhelmコマンド実行端末から、外部のHelm ChartをURLで直接指定して導入する方法です。
Chart参照とリポジトリーのURL	helm install –repo https://example.com/charts/ mynginx nginx	• 外部ネットワークに接続しているhelmコマンド実行端末から、外部のHelmリポジトリー内にあるHelm ChartをリポジトリーのURLとChart名を指定して導入する方法です。

– 本ガイドでは、ダウンロードしたHelm Chartを展開しないでそのまま指定する方法(「パッケージングされたChartファイル」)を主に利用しています。
– ダウンロードしたHelm Chartをカスタマイズする場合は、「展開されたChartのディレクトリー」を指定する方法も利用しています。

(補足)Helm Chartによって作成されるOCPリソース：例１（1/2）

■ リソースの概要図

– デフォルトの設定値でデプロイした場合に、Helm Chartによって作成されるOCPリソース



(補足)Helm Chartによって作成されるOCPリソース：例１（2/2）

■ リソースの一覧

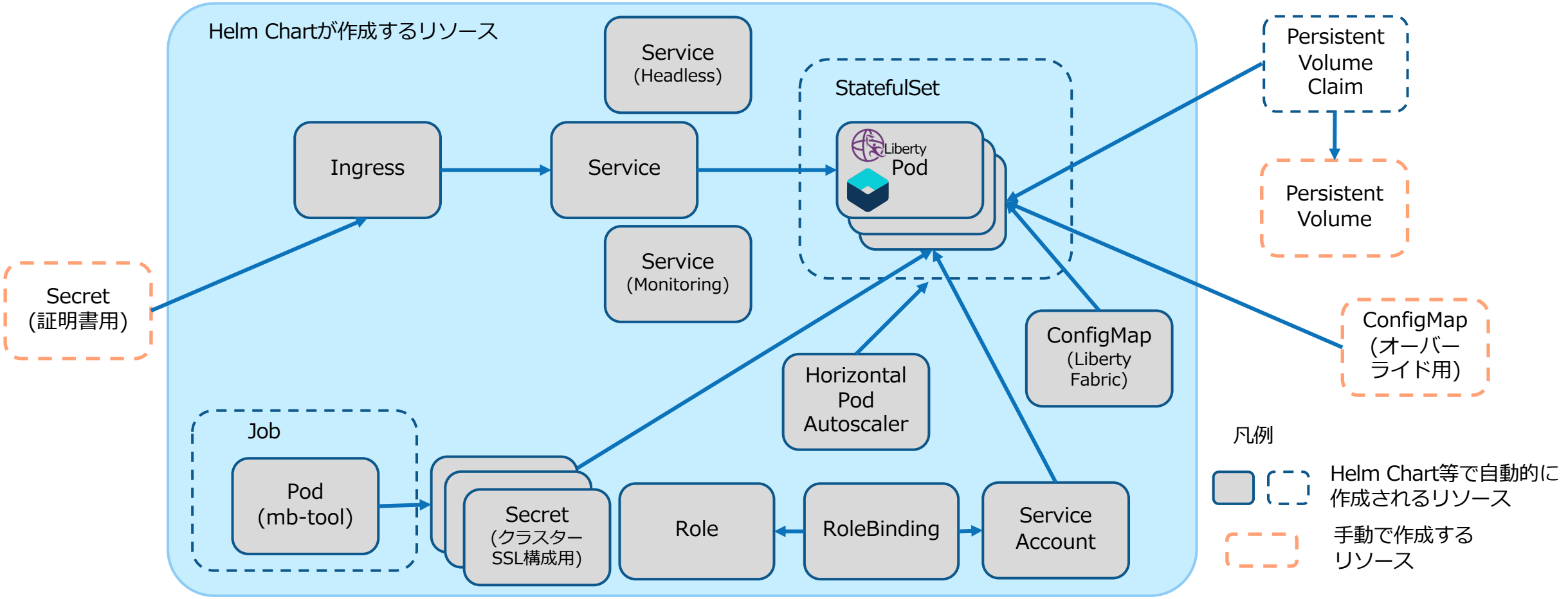
– デフォルトの設定値でデプロイした場合に、Helm Chartによって作成されるOCPリソース

リソース名	Helm Chartの変数ファイルで関連するパラメーター	備考
Deployment(Pod)		デフォルトではWebSphere LibertyはDeploymentとして実行される
Service	service.enable=true	WebSphere Libertyへのアクセスを負荷分散するServiceリソース
ConfigMap		Liberty Fabricとも呼ばれる、設定のカスタマイズのためのserver.xmlフラグメントを格納したConfigMap (8章を参照)
ServiceAccount	rbac.install=true	WebSphere LibertyのPodを実行するServiceAccount
Role	rbac.install=true	WebSphere LibertyのPodの実行に必要な権限を定義したRole
RoleBinding	rbac.install=true	ServiceAccountにRoleをバインドするためのRoleBinding

(補足)Helm Chartによって作成されるOCPリソース：例２（1/3）

■ リソースの概要図

– 設定値を変更してデプロイした場合に、Helm Chartによって作成されるOCPリソースの例



(補足)Helm Chartによって作成されるOCPリソース：例２（2/3）

■ リソースの一覧：Helm Chart等による作成分

– 設定値を変更してデプロイした場合に、Helm Chartによって作成されるOCPリソースの例

リソース名	Helm Chartの変数ファイルで関連するパラメーター	備考
StatefulSet(Pod)	logs.persistLogs=true logs.persistTransactionLogs=true	ログを永続化をする場合、WebSphere LibertyはStatefulSetとして実行される
PrsistentVolumeClaim	logs.persistLogs=true logs.persistTransactionLogs=true	ログを永続化した場合に、StatefulSetによって作成され、対象のPVとPodの紐付けを行うリソース
Service(Headless)		StatefulSetの場合、Headless Serviceがあわせて作成される
Service(Monitoring)	monitoring.enabled=true	モニタリングを有効にした場合、HTTP経由でWebSphere LibertyにアクセスするためのServiceが作成される
Ingress	ingress.enable=true	クラスター外からWebSphere LibertyへのRouter経由でのアクセスを定義するためのIngressリソース
Job(Pod)	ssl.createClusterSSLConfiguration=true	クラスターSSL構成を作成するためにPodを実行するJob
Secret (クラスターSSL構成用)		複数のWebSphere Libertyで共通の証明書を利用する(クラスターSSL構成)場合に、そのキーストアやパスワードなどを格納したSecret群
HorizontalPodAutoscaler	autoscaling.enabled=true	負荷状況に応じてPodの数を増やす場合に設定するHPAリソース (HPAを利用する場合はStatefulSetではなくDeploymentを利用する必要がある。)

(補足)Helm Chartによって作成されるOCPリソース：例２（3/3）

■ リソースの一覧：手動作成分

– Helm Chartによって作成されず、必要に応じてユーザーが作成する必要のあるOCPリソース

リソース名	Helm Chartの変数ファイルで指定するパラメーターと設定値	備考
PersistentVolume	n/a	永続ログや静的コンテンツなどを格納するためのリソース
PrsistentVolumeClaim	n/a	静的コンテンツ用PVなどを利用する場合に作成し、対象のPVとPodの紐付けを行うリソース
ConfigMap (オーバーライド用)	n/a	server.xmlフラグメントをデプロイ後に編集するConfigMap (8章を参照)
Secret(証明書用)	n/a	Secretに格納しIngressに設定する (4章を参照)

4. 基本的なデプロイ（2）

目次(4章)

- 概要
- Readiness Probeの設定
- Liveness Probeの設定
- レプリカ数の設定
- Serviceの設定
- Ingress（Route）の設定
  - 概要
  - 全てHTTP通信の形態
  - Edge Terminationの形態

概要

- ここでは、WebSphere LibertyのHelm Chartが提供する機能のうち、利用頻度が高いものを取り上げて説明します。
- イメージのビルドやHelmを使ったデプロイに関しては前章で説明済みなので、主として変数ファイルでの指定方法を説明します。
- 変数ファイルのパラメーターに関しては、前章の「WebSphere LibertyのHelm Chartのパラメーター：基本的なもの」を参照してください。

Readiness Probeの設定

- Readiness Probeは、Helm Chartの変数ファイルで以下の指定を行うことで設定できます。
  - 指定する内容はKubernetesのドキュメントを参照してください。
    - Kubernetes Documentation / Configure Liveness and Readiness Probes
      - https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/
  - initialDelaySecondsには、WebSphere Libertyおよびアプリケーションの起動・初期化に必要な時間を考慮した値を設定します。
    - 不要なReadiness Probeの発生を抑止できます。

```
readinessProbe:
    httpGet:
      path: /app/ready
      port: 9080
      scheme: HTTP
    initialDelaySeconds: 20                 # コンテナーが起動してからProbeを始めるまでの時間
    timeoutSeconds: 2                       # Probeのレスポンスを待つ時間
    periodSeconds: 10                       # Probeを実行する間隔
    successThreshold: 1                     # Failした後、何回連続で成功すればReadiness Probeが成功したとみなすか
    failureThreshold: 5                     # Readiness Probeが何回連続で失敗した場合にPodを割り振り対象から除外するか
```

Liveness Probeの設定

- Liveness Probeは、Helm Chartの変数ファイルで以下の指定を行うことで設定できます。
  - 指定する内容はKubernetesのドキュメントを参照してください。
    - Kubernetes Documentation / Configure Liveness and Readiness Probes
      - https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/
  - initialDelaySecondsには、WebSphere Libertyおよびアプリケーションの起動・初期化に必要な時間を考慮した値を設定する必要があります。
    - 短すぎると、OCPによってWebSphere Libertyが再起動されてしまいます。
    - 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。ログを永続化していない場合に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。

```
livenessProbe:
     httpGet:
       path: /app/live
       port: 9080
     scheme: HTTP
     initialDelaySeconds: 20              # コンテナーが起動してからProbeを始めるまでの時間
     timeoutSeconds: 2                    # Probeのレスポンスを待つ時間
     periodSeconds: 10                    # Probeを実行する間隔
     successThreshold: 1                  # Failした後、何回連続で成功すればLiveness Probeが成功したとみなすか
     failureThreshold: 5                  # Liveness Probeが何回連続で失敗した場合にPodを再起動させるか
```

レプリカ数の設定

- Podのレプリカ数は、Helm Chartの変数ファイルで以下の指定を行うことで設定できます。
  - WebSphere Libertyは、DeploymentまたはStatefulSetとしてデプロイされます。
  - 何れの場合も、Helm Chartの変数ファイルの「replicaCount」でレプリカ数を指定できます。

```
replicaCount: 2                                  # Podのreplica数を2に変更(デフォルトは1)
```

  - 但し、変数ファイルで「autoscaling.enabled=true」を指定することで、オートスケーリングを有効化した場合は、「replicaCount」の指定は無視されます。
    - autoscaling.enabledのデフォルト値はfalseです。

Serviceの設定

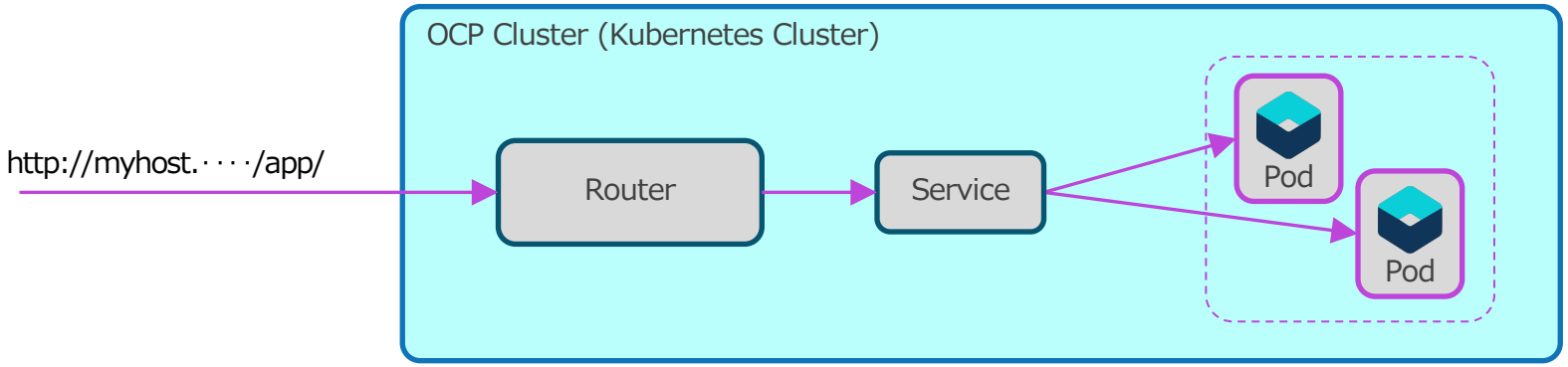
- Serviceの設定は、Helm Chartの変数ファイルで以下の指定を行うことで設定できます。
  - 「ssl.enabled」の値と「service.targetPort」の値に矛盾がないように指定します。
  - 「type」には、OCPクラスター外からNodePortを利用したアクセスが必要か否かによって値を指定します。
    - NodePort:　NodePortを利用してOCPクラスター外からアクセスする場合
    - ClusterIP:　NodePortを利用してOCPクラスター外からアクセスしない場合

```
service:
  enabled: true
  name: ""                                  # サービスの名前(デフォルトは「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字)
  port: 9080                                # デフォルトは9443(HTTPの場合は9080に変更)
  targetPort: 9080                          # デフォルトは9443(HTTPの場合は9080に変更)
  type: ClusterIP                           # ClusterIPまたはNodePortを指定（デフォルトはNodePort）

ssl:
  enabled: false                            # デフォルトはtrue(HTTPの場合はfalseに変更)
  useClusterSSLConfiguration: false         # クラスターSSL構成に関しては、後の章で説明
  createClusterSSLConfiguration: false      # クラスターSSL構成に関しては、後の章で説明
```

Ingress（Route）の設定：概要（１）

■ Routeを定義すると、Router経由でWebSphere Libertyにアクセスできるようになります。



■ Routeは以下の２つの方法で作成できます。

– 手作業で作成する方法
  • Helm ChartでWebSphere LibertyのPodをデプロイした後に、oc exposeやoc create routeコマンド等を利用してRouteを作成します。
  • 定義したRouteは、helm uninstallでは削除されないので、oc delete routeコマンド等を利用して別途削除する必要があります。

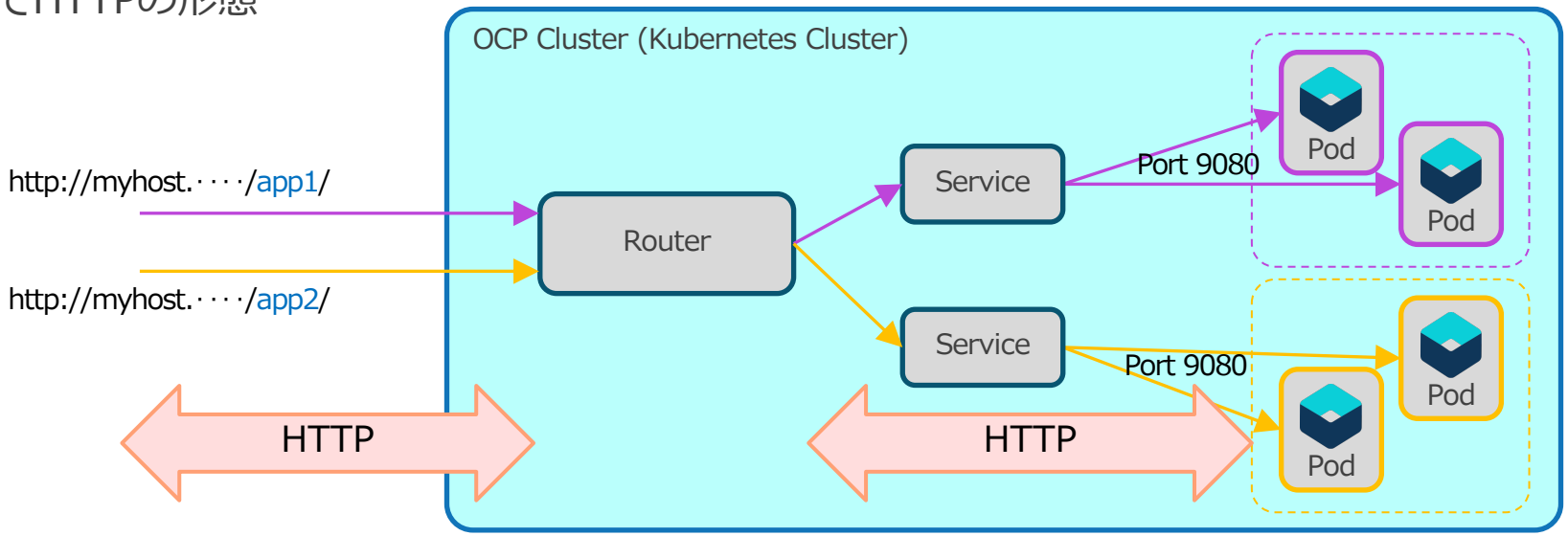
– IngressからRouteを作成する方法
  • WebSphere LibertyのHelm ChartでIngressを作成します。作成されたIngressがRouteに変換可能な場合、OCPがIngress定義からRoute定義を自動的に生成します。
  • 生成されたRouteは、Ingressの変更・削除に追従して変更・削除されるので、 helm uninstallでRouteも削除されます。

■ このガイドでは、後者の方法を説明します。

Ingress（Route）の設定：概要（2）

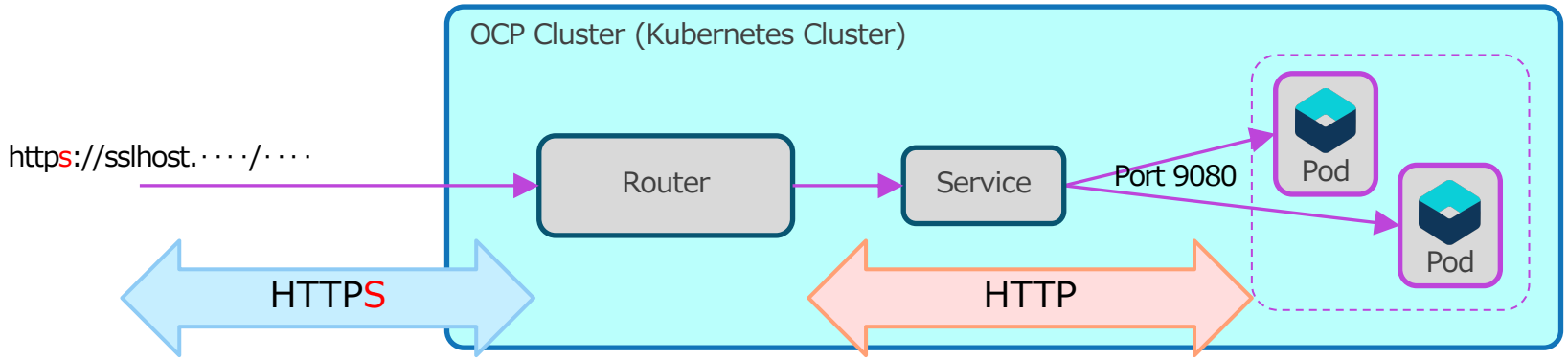
■ IngressからRouteを作成する方法では、以下の２つの形態が構成できます。

– 全てHTTPの形態



パスによる振り分けが利用できます。

– Edge Terminationの形態（RouterまでがHTTPSで、RouterとPod間はHTTPの形態）



insecureEdgeTerminationPolicy は Redirect になります。
(httpアクセスは、httpsにリダイレクトされます。)

Ingress（Route）の設定：全てHTTPの形態

■ 全てHTTPで接続する形態は、Helm Chartの変数ファイルで以下の指定を行うことで構成できます。

- service.targetPortにはWebSphere LibertyのHTTP通信用のポート番号を指定します。
- ssl.enabledにはfalseを指定します。
- ingress.enabledにはtrueを指定します。
- ingress.hostとingress.pathの指定が必須です。
- ingress. rewriteTargetの内容はRoute定義には反映されません。

```
service:
  enabled: true
  name: ""
  port: 9080                                    # HTTP接続なので9080に変更
  targetPort: 9080                              # HTTP接続なので9080に変更
  type: ClusterIP

ssl:
  enabled: false                                # HTTPSを利用しないのでfalseに変更
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false

ingress:
  enabled: true                                 # Ingressを使用するのでtrueに変更
  rewriteTarget: "/"                            # Routerにはrewrite機能がないので指定しても無視されます。
  path: "/app1"                                 # コンテキスト・ルートを記載
  host: "myhost.····"                           # ホスト名を指定（Routeを生成するために必須）
  secretName: ""                                # すべてHTTP通信なので指定しません。Secretは自動的には作成されません。
```

Ingress（Route）の設定：Edge Terminationの形態（1/2）

- 矛盾なくEdge Terminationの形態を構成するには、Helm Chartの修正が必要になります。
  - Helm Chartの修正方法に関しては、次章を参照してください。
- 以下のようなHelm Chartの変数ファイルを使用すると、軽微な矛盾はあるものの、 Edge Terminationの形態を構成できます。（但し、一時的な利用に留めてください。）
  - 軽微な矛盾：作成されるServiceのport名、ポート番号が設定される環境変数名などが、SSL用の名前となります。
  - ingress.secretNameで指定するSecretの作成方法は次ページへを参照

```
service:
  enabled: true
  name: ""
  port: 9080                                    # HTTP接続なので9080に変更
  targetPort: 9080                              # HTTP接続なので9080に変更
  type: ClusterIP

ssl:
  enabled: true                                 # 回避策としてtrueを指定
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false

ingress:
  enabled: true                                 # Ingressを使用するのでtrueに変更
  rewriteTarget: "/"                            # Routerにはrewrite機能がないので指定しても無視されます。
  path: "/app1"                                 # コンテキスト・ルートを記載
  host: "sslhost.····"                          # ホスト名を指定（Routeを生成するために必須）
  secretName: "sslhost-tls-secret"              # サーバー証明書と鍵が格納されているSecretの名前
```

Ingress（Route）の設定：Edge Terminationの形態（2/2）

■ Edge Terminationで使用するSecret

– WebSphere Libertyをデプロイする前に、Routerが使用するサーバー証明書と鍵が保管されているSecretを作成しておきます。
- Secretは自動的には作成されません。
- デプロイによってIngressが作成されると、Secretの内容がRoute定義に反映されます。
- 証明書と鍵は「PEM-encoded X.509, RSA (2048) secret」として登録します。

– 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。
- oc create secret tls (Secretの名前) --key (鍵用のPEMファイル) --cert (証明書用のPEMファイル)

```
# oc create secret tls sslhost-tls-secret --key server.key.clear.pem --cert server-chain.cert.pem
```

5. Helm Chartの変更

目次(5章)

- Helm Chartの変更
- WebSphere LibertyのHelm Chartの構造
- Helm Chartの変更手順
- Helm Chartの変更例
  - DeploymentのStrategy.rollingUpdateの変更
  - StatefulSetのpodManagementPolicyなどの変更
  - RouterのEdge Termination構成用の変更
  - Helm Chart v1.10.0 の問題への対応

Helm Chartの変更

- IBMが提供するWebSphere LibertyのHelm Chartを利用することで、主要な要件を満たすことができますが、全ての要件を満たせるわけではありません。

- 提供されているHelm Chartで要件が満たせない場合は、以下のような対応が必要となります。
  - 提供されているWebSphere LibertyのHelm Chartをカスタマイズする（修正する）
  - 要件を満たすHelm Chartを新規に作成する
  - 要件にあった別のHelm ChartやOperatorを利用する
  - Helm ChartやOperatorを使用せず、マニフェスト・ファイル等で直接リソースを定義する

- 以降では、提供されているWebSphere LibertyのHelm Chartを修正し、要件にあった別のHelm Chartを作成する方法を幾つかの例を元に説明します。

WebSphere LibertyのHelm Chartの構造（1/2）

- Helm Chartでは、依存関係のあるChartをSubchartとして指定し、利用することができます。
  - Helm Documentation / CHART DEPENDENCIES
    - https://helm.sh/docs/charts/#chart-dependencies

- WebSphere LibertyのHelm Chartは、「ibm-shared-liberty」をSubchartとして利用する構成となっています。
  - Subchart「ibm-shared-liberty」は、WebSphere LibertyとOpen Libertyの共通部分を抜き出して作成されており、WebSphere LibertyのHelm Chartのディレクトリー内にパッケージ化されて取り込まれています。
  - WebSphere LibertyのHelm Chartは、「ibm-shared-liberty」を Subchartとして利用し、WebSphere Liberty固有のテンプレートの追加や、設定値を引き渡す構成になっています
    - Helm Documentation / Subcharts and Global Values
      - https://helm.sh/docs/chart_template_guide/#subcharts-and-global-values
  - 依存関係は、Helm Chart内にある「requirement.yaml」に記載されています。

（requirement.yamlの抜粋）

```
dependencies:
- name: ibm-shared-liberty
repository: "@slt" ## where 'slt' is [NAME] from the cmd: helm repo add [flags] [NAME] [URL]
version: 1.10.0
alias: slt
```

WebSphere LibertyのHelm Chartの構造（2/2）

■ WebSphere LibertyのHelm Chartの構造

WebSphere LibertyのHelm Chart

```
ibm-websphere-liberty
├── Chart.yaml              # chartのバージョン、名前の設定
├── LICENSE
├── LICENSES
│   ├── LICENSE-hazelcast
│   ├── LICENSE-non-ibm
│   ├── LICENSE-notices
│   └── LICENSE-websphere-Liberty
├── README.md
├── RELEASENOTES.md         # リリースノート
├── charts
│   └── ibm-shared-liberty-1.10.0.tgz
├── ibm_cloud_pak
│   ├── manifest.yaml
│   └── qualification.yaml
├── requirements.yaml       # このchartを実行するための前提chartの指定
├── templates
│   ├── NOTES.txt           # デプロイ時に表示される情報
│   ├── _slt-config.tpl     # Shared Liberty用の設定ファイル
│   ├── shared.yaml         # shard libertyテンプレートの取り込み設定
│   └── tests
│       ├── basic-endpoint-test.yaml
│       ├── microprofile-health-test.yaml
│       └── persistent-volume-test.yaml
├── values-metadata.yaml
└── values.yaml             # 設定のデフォルト値
```

tgzを展開

WebSphere LibertyのHelm Chart内に含まれるSubchart「ibm-shared-liberty」

```
ibm-shared-liberty/             # ibm-shared-libertyのSubchart
├── Chart.yaml                  # Chartのバージョン、名前の設定
├── README.md
└── templates
    ├── _affinity.tpl           # Mode Affinityのテンプレート
    ├── _config.tpl             # デフォルト設定ファイルのテンプレート
    ├── _configmap.tpl          # configmapのテンプレート
    ├── _deployment.tpl         # Deploymentのテンプレート
    ├── _hpa.tpl                # 水平autoscaleのテンプレート
    ├── _ingress.tpl            # Ingressのテンプレート
    ├── _job.tpl                # Jobのテンプレート
    ├── _notes.tpl              # NOTES.TXTのテンプレート
    ├── _role-binding.tpl       # RBACの紐づけ設定のテンプレート
    ├── _role.tpl               # RBACのrole設定のテンプレート
    ├── _security-context.tpl   # securityのテンプレート
    ├── _service-account.tpl    # Service Accountのテンプレート
    ├── _service.tpl            # Serviceのテンプレート
    ├── _shared.tpl             # shard libertyテンプレートの取り込み
    ├── _utils.tpl              # 設定ファイルの確認や設定値の設定
    └── tests
        └── _tests.tpl
```

Helm Chartの変更手順

- WebSphere LibertyのHelm Chartを変更する手順の流れは以下のようになります。
  - (1) WebSphere LibertyのHelm Chartを展開します。
    - WebSphere LibertyのHelm Chart(ibm-websphere-liberty-1.10.0.tgz)と、Subchart「ibm-shared-liberty」(ibm-shared-liberty-1.10.0.tgz)を展開します。
  - (2) Helm ChartのChart名を変更します。
    - オリジナルのHelm Chartのバージョンアップの際に混乱を招く可能性があるので、Helm Chart名を変更します。Helm Chart内にある「Chart.yaml」のnameとディレクトリー名を新しい名前に変更します。
  - (3) Helm Chart内のSubchart「ibm-shared-liberty」のテンプレートを修正します。
    - Subchart「ibm-shared-liberty」内の変更が必要なテンプレート・ファイルを修正します。
  - (4) 変更したHelm Chartの構文が正しいか確認をします。
    - 「helm lint」コマンドでHelm Chartの構文チェックを行います。
  - (5) (オプション)Helm Chartを再度パッケージ化(tgz)します。
    - 修正したHelm Chartをパッケージ化(tgz)します。「helm package」コマンドを使用します。

- デプロイの手順は、提供されているHelm Chartと同じです
  - (6) Helm Chartから変数ファイルを抽出します。
  - (7) 抽出した変数ファイルのパラメーターを要件に合わせて修正します。
  - (8) 修正した変数ファイルを使用して、デプロイします。

Helm Chartの変更例：Deploymentのstrategy.rollingUpdateの変更（1/4）

- WebSphere LibertyのHelm Chartでは、Deploymentの「spec.strategy.rollingUpdate」の設定値を変更できません。
  - spec.strategy.rollingUpdateで指定できる内容
    - 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)（デフォルト：25%）
    - maxUnavailable:　更新時に、使用不可状態にできる Pod の最大数(割合)（デフォルト：25%）
  - 参考
    - Kubernetes Documentation / Deployment
      - https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

- ここでは、Helm Chartを修正し、「spec.strategy.rollingUpdate」の設定を変更する例を紹介します。
  - 補足：ログを永続化しない場合、WebSphere LibertyのpodはDeploymentとしてデプロイされ、ここで変更した内容が有効になります。
  - 補足：ここで変更したHelm Chartを利用する場合は、ログを永続化せずにDeploymentとしてデプロイする必要があります。

Helm Chartの変更例：Deploymentのstrategy.rollingUpdateの変更（2/4）

■ 手順

– (1) WebSphere LibertyのHelm Chartを展開します。
  • WebSphere LibertyのHelm Chart(ibm-websphere-liberty-1.10.0.tgz)と、Subchart「ibm-shared-liberty」(ibm-shared-liberty-1.10.0.tgz)を展開します。

– (2) Helm ChartのChart名を変更します。
  • 展開したHelm Chartのルートのディレクトリー名と「Chart.yaml」ファイルの「name」の設定値を変更します。
  • ディレクトリー名と「Chart.yaml」の「name」の設定値は同じである必要があります。バージョン変更も可能です。

ディレクトリー名の変更

```
$ mv ibm-websphere-liberty ibm-websphere-liberty-ru
```

ディレクトリーを任意の名前に変更します。
(左記の例では、rollingupdateを表すruをつけて
ibm-websphere-liberty-ruとしてみました。)

ibm-websphere-liberty-ru/Chart.ymlの変更

```
....
maintainers:
- name: WebSphere Liberty
name: ibm-websphere-liberty-ru
....
version: 1.10.0
```

nameにディレクトリー名と同じ名前を設定します。

Helm Chartの変更例：Deploymentのstrategy.rollingUpdateの変更（3/4）

– (3) Helm Chart内のSubchart「ibm-shared-liberty」のテンプレートを修正します。
- Deploymentの定義を修正するので、 Subchart「ibm-shared-liberty」 内のtemplates/_deployment.tplを修正します。

ibm-websphere-liberty-ru/charts/ibm-shared-liberty/templates/_deployment.tplの変更

```
......
spec:
  strategy:
    rollingUpdate:
      maxSurge: 50%
      maxUnavailable: 0%
    type: RollingUpdate
  {{ if $stateful }}
  serviceName: {{ include "slt.utils.servicename" (list $root) }}
  {{ end }}
  ......
```

spec.strategy.rollingUpdateのmaxSurgeとmaxUnavailableの設定内容を明示的に追記します。
(StatefulSetの場合の外側に記載するように気をつけてください。)

– (4) 変更したHelm Chartの構文が正しいか確認をします。
- Helm Chartの構文チェックを「helm lint」コマンドで行います。エラー出力がないことを確認します。

```
$ helm lint ibm-websphere-liberty-ru
==> Linting ibm-websphere-liberty-ru

1 chart(s) linted, 0 chart(s) failed
```

Helm Chartの変更例：Deploymentのstrategy.rollingUpdateの変更（4/4）

- (5) (オプション)Helm Chartを再度パッケージ化(tgz)します。
  - 修正したHelm Chartをパッケージ化(tgz)します。「helm package」コマンドを使用します。
  - Helm Chartをパッケージ化する前に、Subchart 「ibm-shared-liberty」の「ibm-shared-liberty-1.10.0.tgz」ファイルが残っている場合は、このファイルを削除してください。(変更前のSubchartのtgzファイルを削除してから全体をパッケージ化します。)
  - 「helm package」コマンドが完了すると修正したHelm Chartのtgzファイル(ibm-websphere-liberty-ru-1.10.0.tgz)が作成されます。

```
$ helm package ibm-websphere-liberty-ru
Successfully packaged chart and saved it to: /‥/‥/ibm-websphere-liberty-ru-1.10.0.tgz
```

■ 以上で、Helm Chartの変更は完了です。今までと同様の手順でデプロイします。

- (6) Helm Chartから変数ファイルを抽出します。
- (7) 抽出した変数ファイルのパラメーターを要件に合わせて修正します。
  - ここで変更したHelm Chartを利用する場合は、ログを永続化せずにDeploymentとしてデプロイする必要があります。
- (8) 修正した変数ファイルを使用して、デプロイします。

Helm Chartの変更例：StatefulSetのpodManagementPolicy等の変更（1/4）

- WebSphere LibertyのHelm Chartでは、StatefulSetの「spec.podManagement」と「spec.updateStrategy.type」の設定値を変更できません。
  - 「spec.podManagement」で指定できる値（デフォルト：OrderedReady）
    - 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、停止はインデックスが大きい順になります。)
    - Parallel:　Podを並列起動/並列停止します。
  - 「spec.updateStrategy.type」で指定できる値（デフォルト： RollingUpdate）
    - 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。
  - 参考
    - Kubernetes Documentation / StatefulSets
      - https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

- ここでは、Helm Chartを修正し、「spec.podManagement」と「spec.updateStrategy.type」の設定を変更する例を紹介します。
  - ここでは、「spec.podManagement」を「Parallel」に、 「spec.updateStrategy.type」を「 OnDelete 」に変更します。
  - 補足：ログを永続化する場合、WebSphere LibertyはStatefulSetとしてデプロイされ、ここで変更した内容が有効になります。

Helm Chartの変更例：StatefulSetのpodManagementPolicy等の変更（2/4）

■ 手順

– (1) WebSphere LibertyのHelm Chartを展開します。
  • WebSphere LibertyのHelm Chart(ibm-websphere-liberty-1.10.0.tgz)と、Subchart「ibm-shared-liberty」 (ibm-shared-liberty-1.10.0.tgz)を展開します。

– (2) Helm ChartのChart名を変更します。
  • 展開したHelm Chartのルートのディレクトリー名と「Chart.yaml」ファイルの「name」の設定値を変更します。
  • ディレクトリー名と「Chart.yaml」の「name」の設定値は同じである必要があります。バージョン変更も可能です。

ディレクトリー名の変更

```
$ mv ibm-websphere-liberty ibm-websphere-liberty-od
```

ディレクトリーを任意の名前に変更します。
(左記の例では、OnDeleteを表すodをつけて
ibm-websphere-liberty-odとしてみました。)

ibm-websphere-liberty-od/Chart.ymlの変更

```
....
maintainers:
- name: WebSphere Liberty
name: ibm-websphere-liberty-od
....
version: 1.10.0
```

nameにディレクトリー名と同じ名前を設定します。

Helm Chartの変更例：StatefulSetのpodManagementPolicy等の変更（3/4）

– (3) Helm Chart内のSubchart「ibm-shared-liberty」のテンプレートを修正します。
- StatefulSetの定義は、Depoymentと同じテンプレート・ファイルに記述されているので、 Subchart「ibm-shared-liberty」 内のtemplates/_deployment.tplを修正します。

ibm-websphere-liberty-od/charts/ibm-shared-liberty/templates/_deployment.tplの変更

```
{{ if $stateful }}
apiVersion: apps/v1
kind: StatefulSet
{{ else }}
......
spec:
  {{ if $stateful }}
  serviceName: {{ include "slt.utils.servicename" (list $root) }}
  podManagementPolicy: Parallel
  updateStrategy:
    type: OnDelete
  {{ end }}
```

テンプレート内のif文でStatefulSetの場合の設定箇所に追加する必要があります。
podManagementPolicyとupdateStrategy.typeの設定をspec:のすぐ下の{{ if $stateful }}と{{ end }}の間に追記します。

– (4) 変更したHelm Chartの構文が正しいか確認をします。
- Helm Chartの構文チェックを「helm lint」コマンドで行います。エラー出力がないことを確認します。

```
$ helm lint ibm-websphere-liberty-od
==> Linting ibm-websphere-liberty-od

1 chart(s) linted, 0 chart(s) failed
```

Helm Chartの変更例：StatefulSetのpodManagementPolicy等の変更（4/4）

- (5) (オプション)Helm Chartを再度パッケージ化(tgz)します。
  - 修正したHelm Chartをパッケージ化(tgz)します。「helm package」コマンドを使用します。
  - Helm Chartをパッケージ化する前に、Subchart 「ibm-shared-liberty」の「ibm-shared-liberty-1.10.0.tgz」ファイルが残っている場合は、このファイルを削除してください。(変更前のSubchartのtgzファイルを削除してから全体をパッケージ化します。)
  - 「helm package」コマンドが完了すると修正したHelm Chartのtgzファイル(ibm-websphere-liberty-od-1.10.0.tgz)が作成されます。

```
$ helm package ibm-websphere-liberty-od
Successfully packaged chart and saved it to: /‥/‥/ibm-websphere-liberty-od-1.10.0.tgz
```

■ 以上で、Helm Chartの変更は完了です。今までと同様の手順でデプロイします。

- (6) Helm Chartから変数ファイルを抽出します。
- (7) 抽出した変数ファイルのパラメーターを要件に合わせて修正します。
  - ここで変更したHelm Chartを利用する場合は、ログを永続化しStatefulSetとしてデプロイする必要があります。
- (8) 修正した変数ファイルを使用して、デプロイします。

Helm Chartの変更例：RouterのEdge Termination構成用の変更（1/2）

- WebSphere LibertyのHelm Chartでは、RouterのEdge Termination構成が適切にできません。
  - 前章に記載したような方法で回避することができますが、矛盾がある構成となり、適切ではありません。

- Subchart内のテンプレート・ファイル「templates/_ingress.tpl」に以下の修正を施すことで、RouterのEdge Termination構成が適切に行えるようになります。
  - Helm Chartの変更手順は前述の例を参照してください。

変更前

```
......
spec:
  {{- if and $root.Values.ssl.enabled $root.Values.ingress.host }}
  tls:
  - secretName: {{ $root.Values.ingress.secretNa
    {{- if $root.Values.ingress.host }}
    hosts:
    - {{ $root.Values.ingress.host }}
    {{- end -}}
  {{- end }}
  rules:
    ......
```

ingress.secretNameとingress.hostが指定されている場合は、spec.tlsの内容がIngress定義に反映されるように、if文の条件を変更します。

変更後

```
......
spec:
  {{- if and $root.Values.ingress.secretName $root.Values.ingress.host }}
  tls:
  - secretName: {{ $root.Values.ingress.secretName }}
    {{- if $root.Values.ingress.host }}
    hosts:
    - {{ $root.Values.ingress.host }}
    {{- end -}}
  {{- end }}
  rules:
    ......
```

Helm Chartの変更例：RouterのEdge Termination構成用の変更（2/2）

- 変更後のHelm Chartを利用すると、下記のような変数ファイルでEdge Terminationの形態を構成できます。
  - ingress.secretNameで指定するSecretの作成方法は前述を参照

```
service:
  enabled: true
  name: ""
  port: 9080                                    # HTTP接続なので9080に変更
  targetPort: 9080                              # HTTP接続なので9080に変更
  type: ClusterIP

ssl:
  enabled: flase                                # HTTP接続なのでfalseを指定
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false

ingress:
  enabled: true                                 # Ingressを使用するのでtrueに変更
  rewriteTarget: "/"                            # Routerにはrewrite機能がないので指定しても無視されます。
  path: "/app1"                                 # コンテキストルートを記載
  host: "sslhost.····"                          # ホスト名を指定（Routeを生成するために必須）
  secretName: "sslhost-tls-secret"              # サーバー証明書と鍵が格納されているSecretの名前
```

Helm Chartの変更例：Helm Chart v1.10.0 の問題への対応

- WebSphere LibertyのHelm Chart v1.10.0 では、image.extraVolumeMountsの展開に問題があり、 変数ファイルでimage.extraVolumeMountsを指定した場合にエラーが発生することがあります。
  - 変数ファイルでimage.extraVolumeMountsを正しい構文で指定しても、"mapping values are not allowed in this context"というエラーが発生し、デプロイできません。

- Subchart内のテンプレート・ファイル「templates/_deployment.tpl」に以下の修正を施すことで、エラーが解消します。
  - Helm Chartの変更手順は前述の例を参照してください。

変更前

```
    ......
{{- with $root.Values.image.extraVolumeMounts }}
{{ toYaml . | indent 8 }}
{{- end -}}
    ......
```

image.extraVolumeMountsを展開している部分で、
「{{- end -}}」を「{{- end }}」に修正します。

変更後

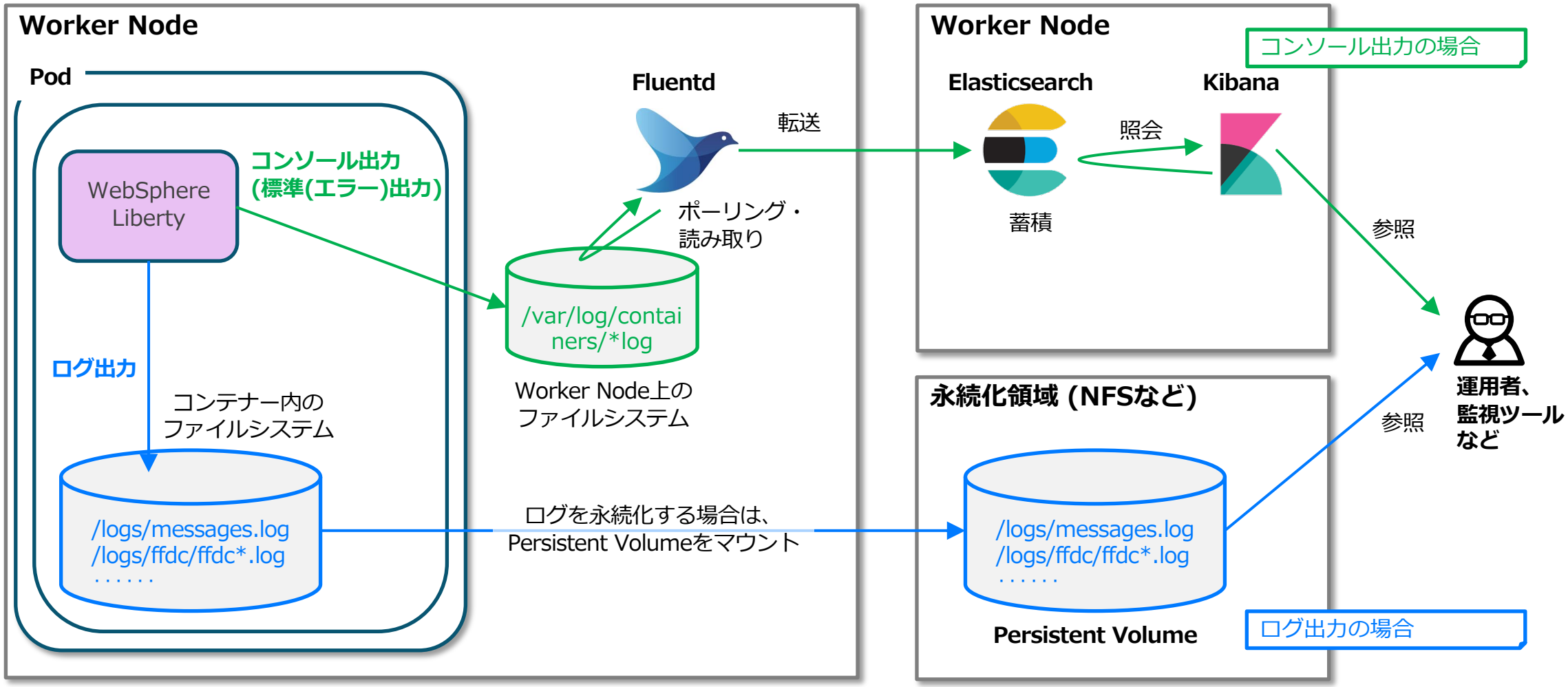
```
    ......
{{- with $root.Values.image.extraVolumeMounts }}
{{ toYaml . | indent 8 }}
{{- end }}
    ......
```

6. ログとダンプの永続化と転送

目次(6章)

OCP環境でのWebSphere Libertyのコンソール出力とログ出力

■ 「コンソール出力」と「ログ出力」は以下の様に処理されます。



WebSphere Libertyが出力するログとダンプ

■ WebSphere Libertyが出力するログとダンプは以下の通りです。

– 以下のURLの情報も参照してください。

• developerWorks: WAS(WebSphere Application Server) 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コンテナー内でのデフォルトの出力先です。

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 提供のイメージでは server run コマンドで WebSphere Liberty 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	WebSphere 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。WebSphere 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	WebSphere 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
  - コンテナー内に出力されたログやダンプは、コンテナーが停止すると消えてしまいます。このため、障害状況によってはログやダンプを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

■ ログの永続化方法

– 以下に示すログは、Helm Chartのパラメーター設定で永続化できます。

– ログの永続化用の領域は「/logs」と「/output/tranlog」となります。

- 「/logs」内に出力されるログ: message.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

– 以下に示すログは、出力先をログの永続化用の領域の「/logs」内に指定することで永続化できます。

- 「/logs」内に出力先を変更することで永続化できるログ: http_access.log、GCログ

■ ダンプの永続化方法

– ログの永続化用の領域(/logs)を流用してダンプを永続化できます。

- Helm Chart のパラメーター設定でログを永続化させます(/logs) 。また、ダンプを「/logs」に出力するように WebSphere Liberty を構成します。
  – WebSphere Liberty Knowledge Center / コマンド行からの Liberty サーバー・ダンプの生成
  – https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_dump_server.html
- ダンプ出力によって、ログの永続化領域（「logs」）が圧迫される可能性があるので、十分な容量を確保しておく必要があります。

– ダンプ専用の永続化用の領域を設ける方法でダンプを永続化できます。

- Helm Chart のパラメーター設定でダンプ専用の永続化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。

トランザクション・ログの永続化

- WebSphere 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。
    - WebSphere 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。
    - 参照：GitHub / IBM/charts/stable/ibm-websphere-liberty / Transaction logs
      - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#transaction-logs

```
    <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るため、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/
  - WebSphere 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。
    - 参照：developerWorks: WAS(WebSphere Application Server) 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/

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、以下のいずれかの方法で準備します。
  - Dynamic Provisioningを使用する場合
    - OCP クラスターに定義されているStorage Classを確認します。(詳細はOCPクラスターの管理者に確認)
  - Static Provisioningを使用する場合
    - HelmでLibertyをデプロイする前に、必要な数のPersistent Volume (PV)を定義します。

Persistent Volume用のマニフェストファイル(yaml)の例

```
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: /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。

Helmのデプロイ時に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。
Helmのデプロイ時に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。
(例ではNFSを使用しています。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。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Persisting logs 参照
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#persisting-logs

ログ永続化の設定例：概要、イメージの準備

■ 概要

– 以下のログを永続化する手順を記載します。
  • /logsに保管されるログ（message.log、trace.log、FFDC）
  • /output/tranlogに保管されるトランザクション・ログ
– Static Provisioningを使用し、LabelでPVとPVCをマッチングします。
– NFSを使用したPVを利用します。

■ イメージの準備

– この例では、トランザクション・ログも永続化するので、server.xmlまたはserver.xmlフラグメントに以下の記述を追加します。

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

– イメージをビルドし、レジストリーにpushします。

ログ永続化の設定例：PVの準備

■ PVを作成します。

– PV用のマニフェストファイル(例: hogehoge-pv-1.yaml)を作成します。

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

– 準備したマニフェストファイル(例: hogehoge-pv-1.yaml)からPVを定義します。

```
> oc apply -f hogehoge-pv-1.yaml
persistentvolume/hogehoge-pv-1 created
```

ログ永続化の設定例：変数ファイルの準備

■ Helm Chartの変数ファイルを準備します。

- 変更するパラメーターの説明は、本章の「WebSphere LibertyのHelm Chartのパラメーター：ログ関連」を参照

– ログの永続化に関連する以下の設定を変更します。

- 「logs.persistLogs」と「logs.persistTransactionLogs」をtrueに変更します。

– PVC定義に関連する以下の設定を変更します。

- ここでは、Dynamic Provisioningを使用しないので、「persistence.useDyanamicProvisioning」をfalseとします。
- PVとPVCのマッチングをstorageClassではなく、labelで行うため、「persistence.selector.label」と「persistence.selector.value」にPVと同じ値を設定します。
- 「persistence.size」にはPVの「spec.capacity.storage」を超えない値を指定します。

```
logs:
  persistLogs: true
  persistTransactionLogs: true
  consoleFormat: json
  consoleLogLevel: info
  consoleSource: message,trace,accessLog,ffdc

persistence:
  name: "liberty-pvc"
  size: "1Gi"
  useDynamicProvisioning: false
  fsGroupGid:
  storageClassName: ""
  selector:
    label: "type"
    value: "hogehoge-pv"
```

ログ永続化の設定例：デプロイ

■ デプロイし、ログが永続化されていることを確認します。

– 準備した変数ファイルを使用して、Helm installまたはupgradeを実行します。

```
> helm upgrade --install liberty-log-test01 ibm-websphere-liberty-1.10.0.tgz -f values-liberty-log-test01.yaml
Release "liberty-log-test01" does not exist. Installing it now.
NAME: liberty-log-test01
......
```

– ログが永続化されていることを確認します。

• NFSサーバー側で、WebSphere Libertyのログとトランザクション・ログが永続化されていることを確認します。

(例) ログ

```
> ls -l /nfs/liberty/hogehoge-pv-1/logs/
合計 24
-rw-r-----. 1 1001 bin 16800  3月 13 13:46 messages.log
-rw-r-----. 1 1001 bin   977  3月 13 13:46 trace.log
```

(例) トランザクション・ログ

```
> ls -l /nfs/liberty/hogehoge-pv-1/tranlog/
合計 0
drwxr-x---. 2 1001 bin 61  3月 13 13:46 partnerlog
drwxr-x---. 2 1001 bin 61  3月 13 13:46 tranlog

> ls -l /nfs/liberty/hogehoge-pv-1/tranlog/tranlog/
合計 8
-rw-r-----. 1 1001 bin       0  3月 13 13:46 DO NOT DELETE LOG FILES
-rw-r-----. 1 1001 bin 1048576  3月 13 13:46 log1
-rw-r-----. 1 1001 bin 1048576  3月 13 13:46 log2
```

作成されるワークロードの違い

■ ログを永続化しない場合と永続化した場合では以下の差異があります。

– ログを永続化しない場合: Podは「Deployment」としてデプロイされます。

(例) Podのレプリカ数を2とした場合(replicaCount: 2)

```
> oc get deployment
NAME                        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
liberty-log-test01-ibm-w    2         2         2            2           3m5s

> oc get pods
NAME                                      READY   STATUS    RESTARTS   AGE
liberty-log-test01-ibm-w-86958c8df9-7pwwk   1/1     Running   0          3m8s
liberty-log-test01-ibm-w-86958c8df9-f8skh   1/1     Running   0          3m8s
```

ログを永続化しない場合は、「Deployment」としてデプロイされます。

Podの名前は、<Deployment name>-<ランダムな文字列> となります。

– ログを永続化した場合:　Podは「Statefulset」としてデプロイされます。

- 「Statefulset」としてデプロイされた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はインデックス順に順次起動されます。（Statefulsetのデフォルトの「podManagementPolicy」が「OrderedReady」のため）

(例) Podのレプリカ数を2とした場合(replicaCount: 2)

```
> oc get statefulset
NAME                        DESIRED   CURRENT   AGE
liberty-log-test01-ibm-w    2         2         14m

> oc get pod
NAME                          READY   STATUS    RESTARTS   AGE
liberty-log-test01-ibm-w-0    1/1     Running   0          14m
liberty-log-test01-ibm-w-1    1/1     Running   0          14m
```

ログを永続化する場合は、「Statefulset」としてデプロイされます。

Podの名前は、<Statefulset name>-index となります。

作成されるサービスの違い

■ ログを永続化しない場合と永続化した場合では以下の差異があります。

– ログを永続化しない場合: Helmの変数ファイルで定義した「Service」が作成されます。

(例) Helmの変数ファイルで「Service.type: Nodeport」(dafault)と設定した場合

```
> oc get service
NAME                          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
liberty-log-test01-ibm-w      NodePort   10.0.228.44   <none>        9443:31685/TCP   7s
```

– ログを永続化した場合:　Helmの変数ファイルで定義した「Service」に加えて「Headless Service」も作成されます。

- OCPクラスター外部からPodへのアクセスは通常の「Service」経由でアクセスします。
- ログを永続化した場合、StatefulSetとしてデプロイされます。Headless Serviceは、StatefulSetの前提として作成されます。

(例) Helmの変数ファイルで「Service.type: Nodeport」(dafault)と設定した場合

```
> oc get service
NAME                          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
liberty-log-test01-ibm-w      NodePort    10.0.13.11    <none>        9443:30411/TCP   5s
liberty-log-test01-ibm-w-sts  ClusterIP   None          <none>        9443/TCP         5s
```

Headless Serviceも追加で作成されます。(spec.ClusterIP: None)

PVとPVCの管理（1/3）

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

(例) 3つのPVを作成し、レプリカ数を2とした場合(replicaCount: 2)

```
> oc get pv
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                                                  ....
......
hogehoge-pv-1       1Gi        RWO            Retain           Available
hogehoge-pv-2       1Gi        RWO            Retain           Bound       handson/liberty-pvc-liberty-log-test01-ibm-w-1
hogehoge-pv-3       1Gi        RWO            Retain           Bound       handson/liberty-pvc-liberty-log-test01-ibm-w-0
......
```

PVがPVCに関連付けられておらず、割り当て可能な状態

PVが右記のPVCに関連付けられている状態

注意：
oc get pv を実行すると、クラスター内の全てのPVがリストされます。

  - 「oc get pvc」コマンドでPVCの状態を確認できます。

```
> oc get pvc
NAME                                    STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
liberty-pvc-liberty-log-test01-ibm-w-0   Bound    hogehoge-pv-3   1Gi        RWO                           96s
liberty-pvc-liberty-log-test01-ibm-w-1   Bound    hogehoge-pv-2   1Gi        RWO                           73s
```

PVCが、右記のPVに関連付けられている状態

PVとPVCの管理（2/3）

- Podの状態を保持し続けるために、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のレプリカ数の削減
  - アンデプロイ（helm uninstall <Helmリリース名> ）

- 不要になったPVCは手動で削除する必要があります。
  - PVCの削除は「oc delete pvc <PVC名>」で行います。

(例) アンイデプロイ後のPVCの削除

```
> oc delete pvc liberty-pvc-liberty-log-test01-ibm-w-0
persistentvolumeclaim "liberty-pvc-liberty-log-test01-ibm-w-0" deleted

> oc delete pvc liberty-pvc-liberty-log-test01-ibm-w-1
persistentvolumeclaim "liberty-pvc-liberty-log-test01-ibm-w-1" deleted
```

PVとPVCの管理（3/3）

- PVのpersistentVolumeReclaimPolicyがRetainの場合、
  不要になったPVは、削除するか、PVCとの関連付け情報を削除し再利用できます。

  – PVの削除は「oc delete pv <PV名>」コマンドで行います。

```
> oc delete pv hogehoge-pv-3
persistentvolume "hogehoge-pv-3" deleted
```

  – PVCとの関連付け情報の削除は、「oc edit pv <PV名>」コマンドなどで行います。
    • claimRefの部分を削除します。

```
> oc edit pv hogehoge-pv-2
persistentvolume/hogehoge-pv-2 edited

> oc get pv hogehoge-pv-2
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
hogehoge-pv-2   1Gi        RWO            Retain           Available
```

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

PVのSTATUSが再び「Available」になり割り当て可能な状態に戻る

```
......
spec:
  ......
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: liberty-pvc-liberty-log-test01-ibm-w-1
    namespace: handson
    resourceVersion: "14635363"
    uid: cdac6c95-9177-11e9-91ea-0050568cf015
......
```

WebSphere Liberty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	デフォルト値	説明
persistence	name	"liberty-pvc"	Persistence Volume Claim (PVC) の名称の接頭語として使用されます。
	useDynamicProvisioning	true	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。
	fsGroupGid	(blank)	File System Group IDを指定します。 (永続領域のFile System Group IDを変更する必要がある場合に指定します。WebSphere Libertyは、root(gid=0)グループに所属するdefault(uid=1001)ユーザーで稼働します。)
	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。但し、StorageClassを使用せず、ラベルで使用するPVを指定する場合は""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	"1Gi"	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	false	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、コンテナーの /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	false	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、コンテナーの /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。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

WebSphere LibertyのHelm Chartのパラメーター：ログ関連

■ 「コンソール出力」に関連するパラメーター

Qualifier	Parameter	デフォルト値	説明
logs	consoleFormat	json	WebSphere 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	info	WebSphere 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	message, trace, accessLog, ffdc	WebSphere Liberty のコンソール・ログに出力する内容を message, trace, accessLog, ffdc, audit 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、WebSphere Liberty 側の構成も必要になります。 auditログをコンソール・ログとして出力するには「audit-1.0」featureも必要になります。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

コンソール出力の制御

- コンソールへの出力形式や出力内容は、以下のHelm Chartのパラメーターで調整できます。
  - 詳細は前述の「WebSphere LibertyのHelm Chartのパラメーター：ログ関連」 の「コンソール出力」のパートを参照してください。
  - デフォルトでmessage、trace、accessLog、ffdcが転送される設定になっています。(accessLogを転送するには下記の構成が必要)
  - auditも表示させたい場合はauditも追加で設定します。(auditは「audit-1.0」のfeatureも必要)
  - logs.consoleFormat
  - logs.consoleLogLevel
  - logs.consoleSource

- アクセス・ログをコンソール出力させるには、WebSphere 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。
  - server.xmlに以下の指定を追加します。

```
    <httpAccessLogging id="accessLogging" filePath="/logs/http_access.log"/>
    <httpEndpoint id="defaultHttpEndpoint" accessLoggingRef="accessLogging"/>
```

  - WebSphere 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
  - WebSphere 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

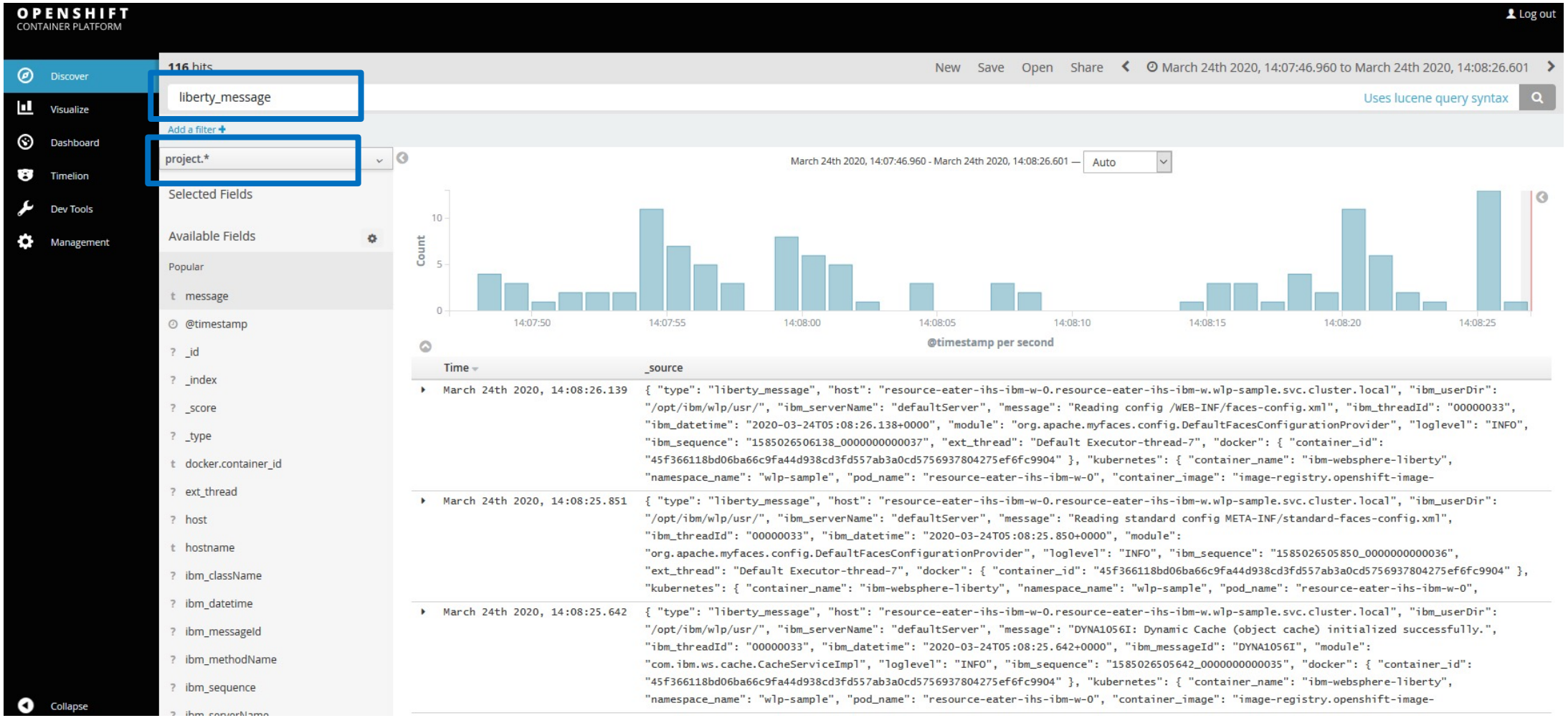
JSON形式のコンソール出力を構文解析するために必要なOCPの設定

- WebSphere Libertyが出力したJSON形式のコンソール出力を構文解析し、Elasticsearchに転送するには、Fluentdログコレクターの構成を変更する必要があります。
  - この構成変更を行うには、クラスターロギングを管理外の状態に設定する必要があります。

- 詳細は、下記のOCPのドキュメンテーションを参照してください。
  - OpenShift Container Platform > 4.3 > ロギング > 8.7.5. ログコレクション JSON 構文解析の設定
    - https://access.redhat.com/documentation/ja-jp/openshift_container_platform/4.3/html-single/logging/index#cluster-logging-collector-json_cluster-logging-collector

コンソール出力のKibanaでの表示

- Elasticsearchへ転送された「コンソール出力」はKibanaで表示できます。
  - Kibanaの左側のメニューから「Discover」を選択し、Index Patternとして「project.*」を選び、「liberty_message」でフィルターした場合の表示例



サンプル・ダッシュボード

- JSON形式のコンソール出力をKibanaで可視化するために、サンプル・ダッシュボードが提供されています。
  - GitHub / IBM/charts/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards/
    - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards

提供されるサンプル・ダッシュボード ファイル名(json)	サンプル・ダッシュボード名	備考
ibm-websphere-liberty-kibana5-problems-dashboard.json	• 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
ibm-websphere-liberty-kibana5-traffic-dashboard.json	• 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
ibm-websphere-liberty-kibana5-audit-dashboard.json	• Liberty Audit AUTHN-AUTHZ Dashboard • Liberty Audit JMS Dashboard • Liberty Audit JMX Dashboard • Liberty Audit Dashboard • Liberty Audit SECURITY_MEMBER_MGMT Dashboard	• auditの情報をビジュアル化するダッシュボード

本ガイド作成時点では、サンプル・ダッシュボードが正しく表示できませんでした。

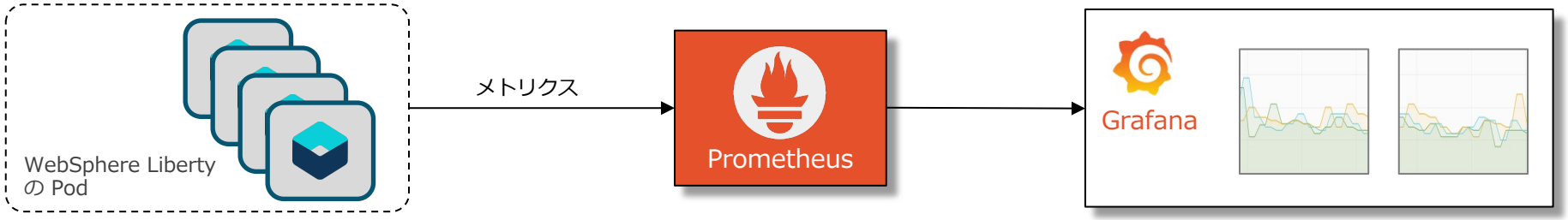
7. モニタリング

目次(7章)

OCP環境でのWebSphere Libertyのモニタリング

- 以下の３つのステップでPrometheusによるモニタリングが可能になります。
  - (1) WebSphere Liberty のフィーチャー「mpMetrics-1.1」および「monitor-1.0」を構成する
    - WebSphere Liberty Knowledge Center / MicroProfile メトリックを使用したモニタリング
      - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mp_metrics_monitor.html
  - (2) Helm Chartの変数ファイルで「monitoring.enabled」を有効にしてWebSphere LibertyのPodをデプロイする
    - GitHub / IBM/charts/stable/ibm-websphere-liberty / monitoring
      - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#monitoring
  - (3) ServiceMonitorを定義する。

- 取集されたメトリクスをGrafanaで可視化するために、サンプル・ダッシュボードが提供されています。
  - GitHub / IBM/charts/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards/
    - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards



WebSphere LibertyのHelm Chartのパラメーター：モニタリング関連

■ 「モニタリング」に関連するパラメーター

Qualifier	Parameter	デフォルト値	説明
monitoring	enabled	false	WebSphere Liberty のモニタリングを有効化するかを指定します。 有効化する場合は、trueを指定します。 有効化する場合は、WebSphere Liberty側の構成で「mpMetrics-1.1」と「monitor-1.0」フィーチャーも必要になります。

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

モニタリングの設定例：概要

■ モニタリングで使用するPrometheusとGrafanaの準備

– OCP 4.3のクラスター・モニタリング用のPrometheusとGrafanaは、以下のような制約があります。

- Prometheus： モニタリングできるのはOCPのコンポーネントのみです。
  但し、テクノロジー・プレビュー機能を有効にするとカスタム・サービスのモニタリングも可能となります。
- Grafana： 提供されているダッシュボードのみが使用でき、ダッシュボードの追加や変更はできません。

– このガイドでは、プロジェクト専用のPrometheusとGrafanaをデプロイする方法を利用します。

- 【注意】この方法でデプロイしたPrometheusとGrafanaは、OCPのサポート対象とはなりません。
- 【注意】メトリクス・データやダッシュボード定義を永続化する場合は、追加の設定作業が必要です。

■ WebSphere Libertyの構成、イメージのビルド、デプロイ

– フィーチャー「mpMetrics-1.1」および「monitor-1.0」を構成したイメージをビルドします。

– Helm Chartの変数ファイルで「monitoring.enabled=true」を指定してデプロイします。

■ ServiceMonitorの定義

– WebSphere Libertyがメトリクス情報の収集対象となるように、ServiceMonitorを定義します。

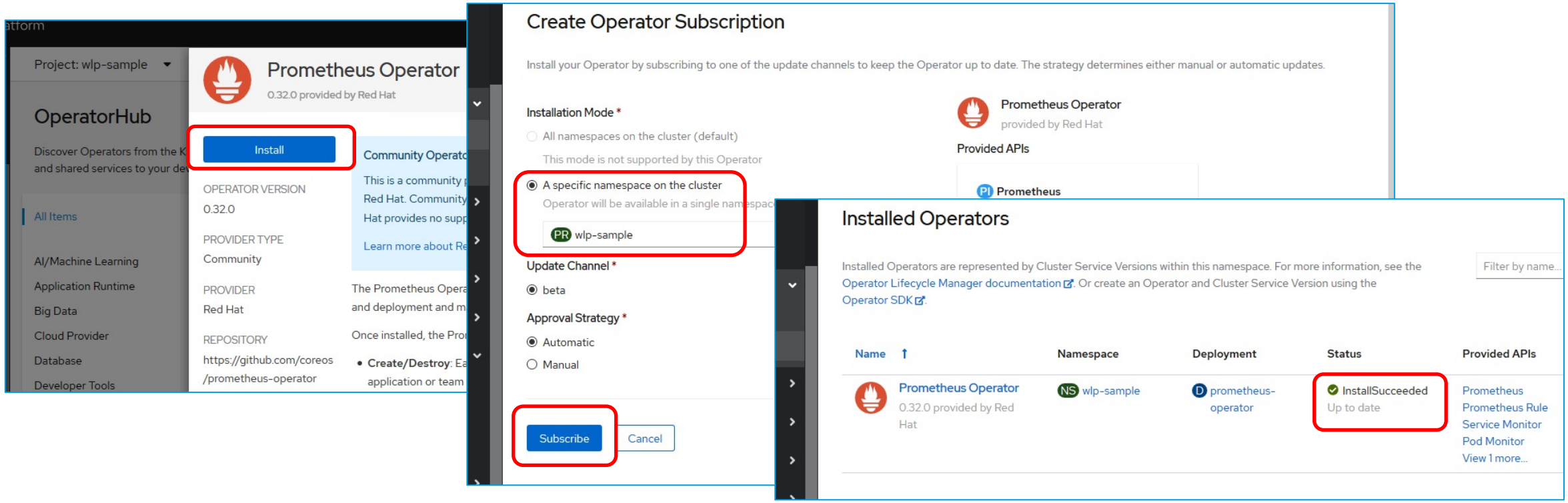
■ サンプル・ダッシュボードの利用

– Grafanaにサンプル・ダッシュボードをインポートし、利用します。

モニタリングの設定例：Prometheusの準備（1/3）

- モニタリングで使用するPrometheusを準備します。手順は、以下の通りです。

- Prometheus Operatorのインストール
  - OCP の Webコンソールから「Operators > OperatorHub」を選択し、「Prometheus Operator」をインストールします。
  - インストール先プロジェクトとして、WebSphere Libertyと同じプロジェクトを指定します。



モニタリングの設定例：Prometheusの準備（2/3）

■ Prometheusインスタンスの作成

– 以下のような yamlファイル(prometheus.yaml)を作成します。

• ネームスペースは、先ほどPrometheus Operatorをインストールしたネームスペースを指定します。

```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example
  labels:
    prometheus: k8s
  namespace: wlp-sample
spec:
  replicas: 2
  serviceAccountName: prometheus-k8s
  securityContext: {}
  serviceMonitorNamespaceSelector:
    matchNames:
    - wlp-sample
  serviceMonitorSelector: {}
  ruleSelector: {}
  alerting:
    alertmanagers:
      - namespace: wlp-sample
        name: alertmanager-main
        port: web
```

特定のネームスペース内に定義されているServiceMonitorのみを対象とするように指定

```
> oc apply -f prometheus.yaml
prometheus.monitoring.coreos.com/example created

> oc get pod -w
NAME                                   READY   STATUS      RESTARTS   AGE
prometheus-example-0                   3/3     Running     1          79s
prometheus-example-1                   3/3     Running     1          79s
prometheus-operator-797dfcd456-plwcr   1/1     Running     0          11m
```

– oc applyコマンドを実行し、yamlファイルからカスタムリソースを作成します。

– Prometheus OperatorがPrometheusインスタンスを作成し、PrometheusがReady状態になります。

モニタリングの設定例：Prometheusの準備（3/3）

■ PrometheusのRouteの作成

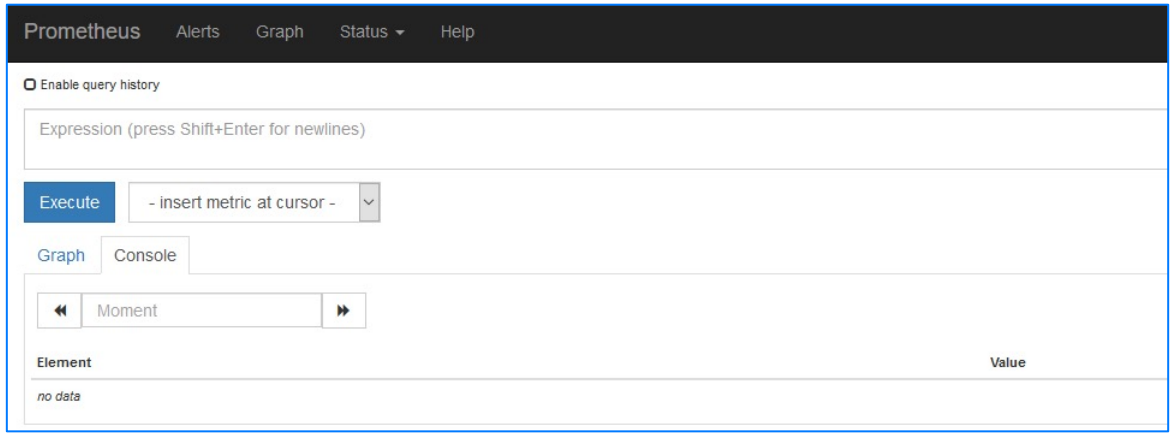
– Prometheusインスタンス用のServiceが作成されているので、exposeしてRouteを作成します。

```
> oc get service
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
prometheus-operated   ClusterIP   None            <none>        9090/TCP   1m

> oc expose service prometheus-operated
route.route.openshift.io/prometheus-operated exposed

> oc get route
NAME                  HOST/PORT                                PATH   SERVICES              PORT   TERMINATION   WILDCARD
prometheus-operated   prometheus-operated-wlp-sample.····             prometheus-operated   web                  None
```

– 作成されたRouteにhttpでアクセスすると、Prometheusのコンソールが表示できます。



– 以上で、Prometheusの準備は完了です。

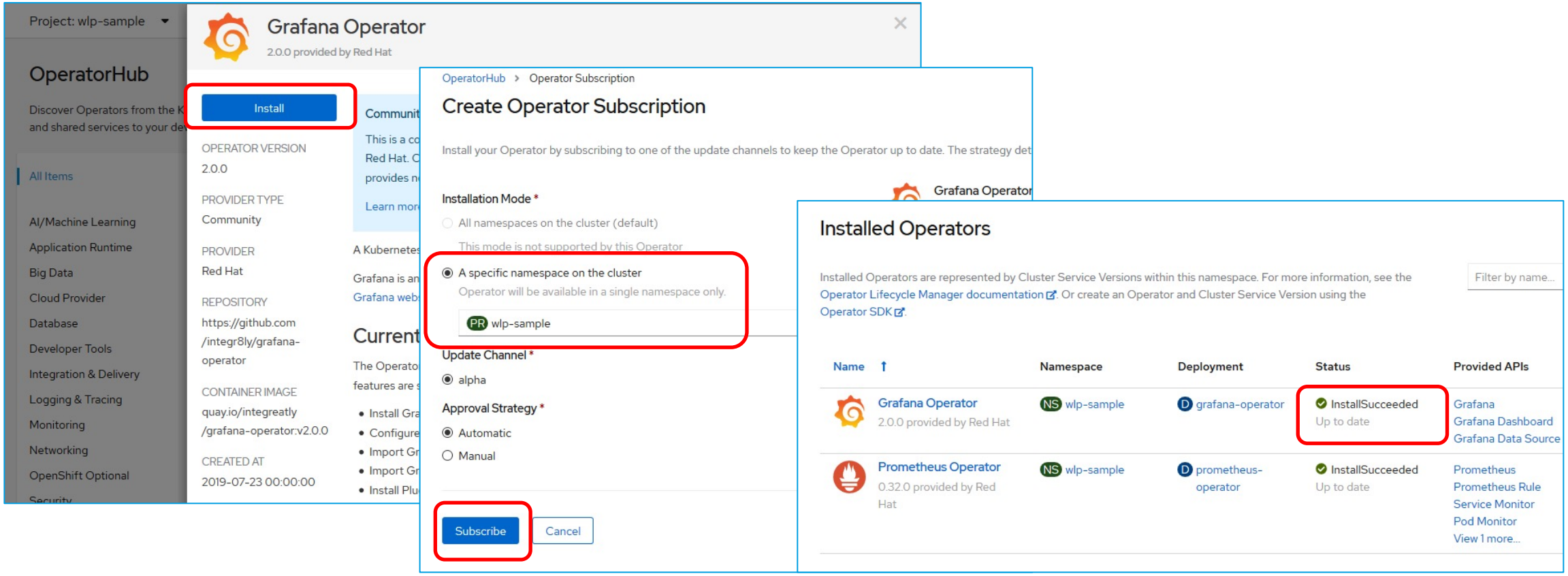
モニタリングの設定例：Grafanaの準備（1/4）

■ モニタリングで使用するGrafanaを準備します。手順は、以下の通りです。

■ Grafana Operatorのインストール

– OCP の Webコンソールから「Operators > OperatorHub」を選択し、「Grafana Operator」をインストールします。

– インストール先プロジェクトとして、先ほどインストールしたPrometheusと同じプロジェクトを指定します。



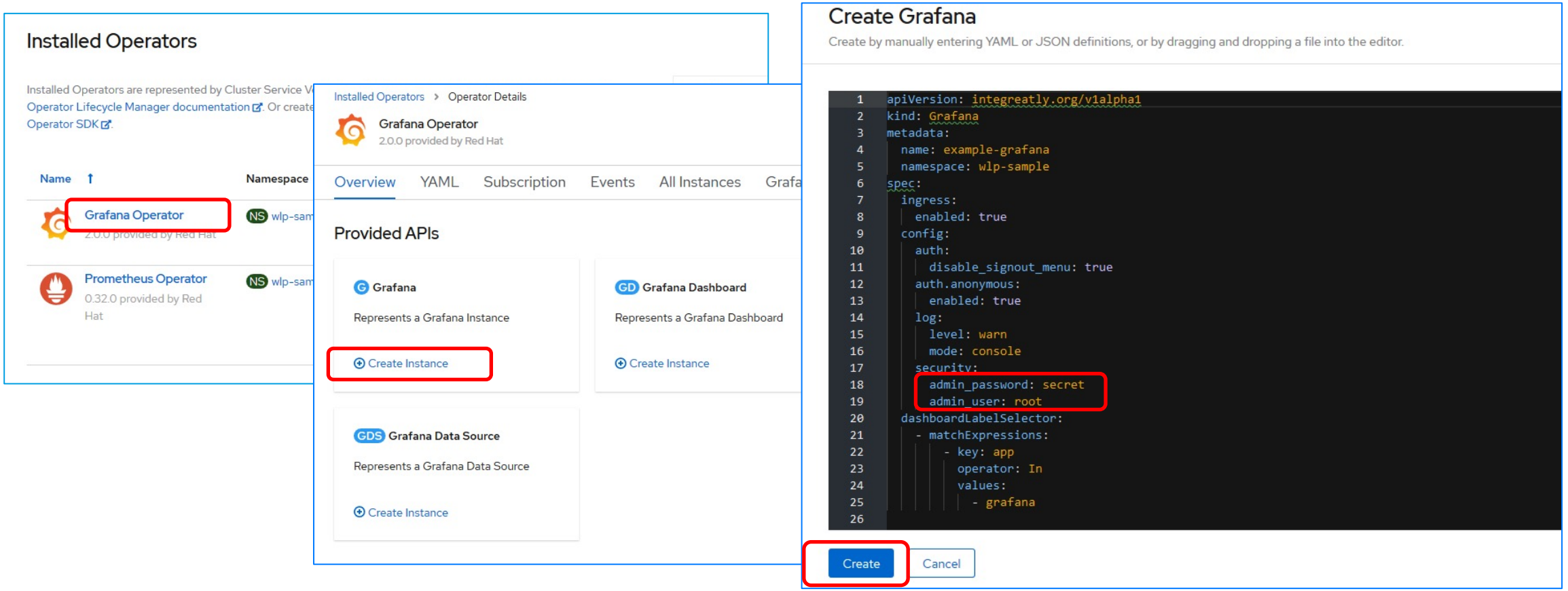
モニタリングの設定例：Grafanaの準備（2/4）

■ Grafanaインスタンスの作成

– OCP の Webコンソールから「Operators > Installed Operators」を選択し、表示された「Grafana Operator」を選択します。

– 「Grafana」の下にある「Create Instance」を選択し、Grafanaインスタンスを作成します

• 作成時に管理ユーザ名とpasswordを確認・変更します。



モニタリングの設定例：Grafanaの準備（3/4）

■ Grafanaコンソールへのアクセス

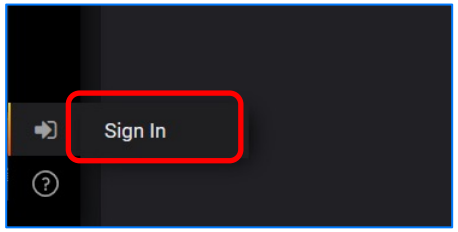
– Grafana OperatorがGrafanaインスタンスを作成し、GrafanaがReady状態になります。

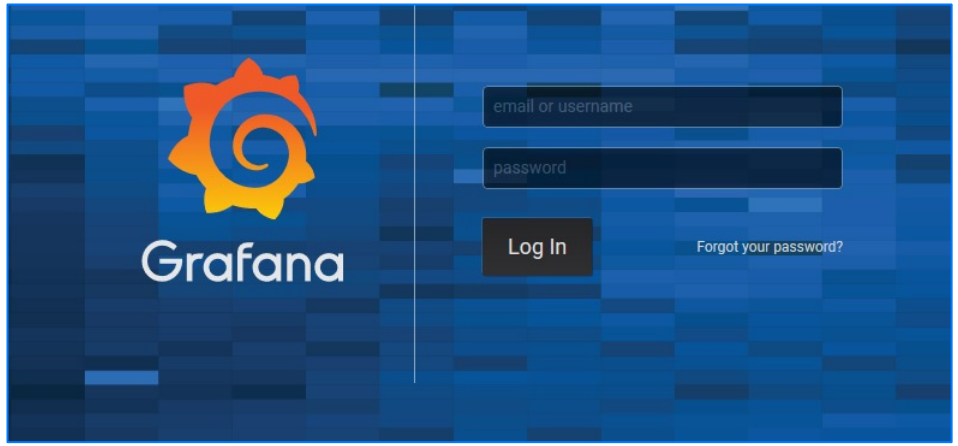
```
> oc get pod
NAME                                  READY   STATUS      RESTARTS   AGE
grafana-deployment-795d56b7d4-bqbcv   1/1     Running     0          26s
grafana-operator-544487858d-v9ssg     1/1     Running     0          9m55s
prometheus-example-0                  3/3     Running     1          32m
prometheus-example-1                  3/3     Running     1          32m
prometheus-operator-797dfcd456-vszsj  1/1     Running     0          32m

> oc get route
NAME                 HOST/PORT                                  PATH   SERVICES              PORT    TERMINATION   WILDCARD
grafana-route        grafana-route-wlp-sample. .····            grafana-service       grafana   edge          None
prometheus-operated  prometheus-operated-wlp-sample.····        prometheus-operated   web                     None
```

– Grafana用のRouteも作成されるのでホスト名を確認し、httpsでアクセスします。

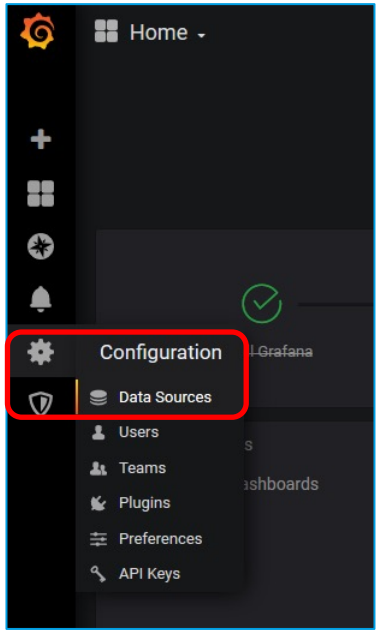
– Grafana の画面の左下から「sign in」を選び、
定義した管理ユーザでログインします。

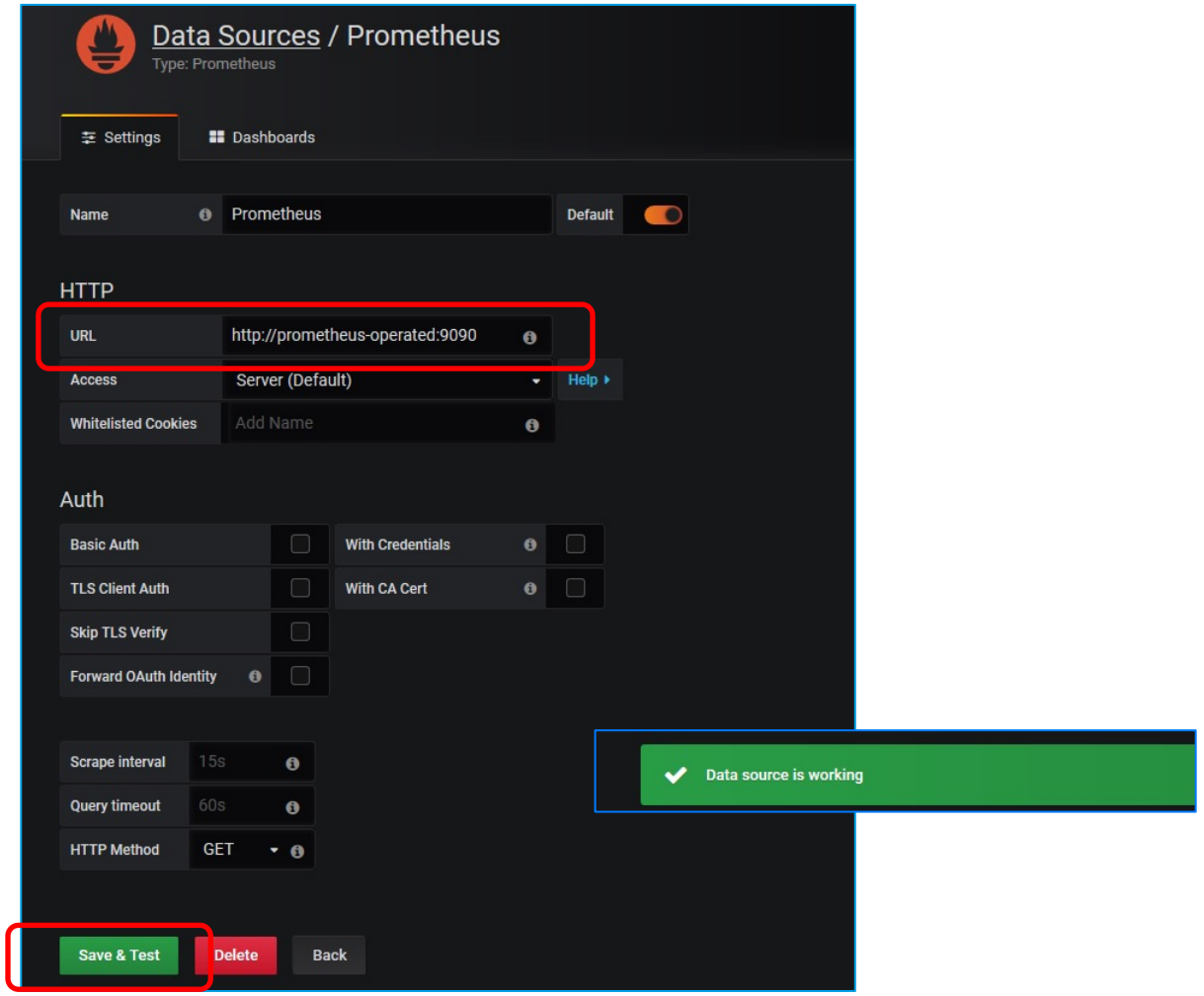




モニタリングの設定例：Grafanaの準備（4/4）

- GrafanaのData source定義
  - Grafana の画面の左から「Configuration > Data Sources」を選び、「Add data source」ボタンを押下し、Data source type から「Prometheus」を選択します。
  - URL に「http://prometheus-operated:9090」を指定して、Data source を定義します。
  - Prometheusへの接続に成功すると、"Data source is working" と表示されます。





モニタリングの設定例：WebSphere Libertyの設定とデプロイ（1/2）

- WebSphere Libertyを構成しデプロイします。手順は、以下の通りです。

- イメージの準備
  - server.xmlまたはserver.xmlフラグメントで、フィーチャー「mpMetrics-1.1」および「monitor-1.0」を有効化しメトリクスが取得できるように構成します。

```
<server>
    <featureManager>
        <feature>mpMetrics-1.1</feature>
        <feature>monitor-1.0</feature>
    </featureManager>

    <mpMetrics authentication="false" />
</server>
```

「featureManager」エレメント内に「mpMetrics-1.1」と「monitor-1.0」フィーチャーを追加します。

認証なしで/metricsにアクセスできるように構成する必要があります。
（「GitHub / IBM/charts/stable/ibm-websphere-liberty / monitoring」参照）

  - イメージをビルドし、レジストリーにpushします。
    - ベース・イメージにフィーチャーが含まれていない場合は、「RUN installUtility install --acceptLicense defaultServer」をビルド時に実行する必要があります。

モニタリングの設定例：WebSphere Libertyの設定とデプロイ（2/2）

■ デプロイ

– Helm Chartの変数ファイルに以下の指定を追加(変更)します。

• その他の指定は、モニタリングを有効化しない場合と同じです。

```
monitoring:
  enabled: true
```

– 変更した変数ファイルを使用して、Helm installまたはupgradeを実行します。

• helm install/upgradeの実行方法は、モニタリングを有効化しない場合と同じです。

```
> helm upgrade --install infra-test-http ibm-websphere-liberty-1.10.0.tgz -f infra-test-http.yaml
Release "infra-test-http" does not exist. Installing it now.
NAME: infra-test-http
LAST DEPLOYED: Wed Mar 11 09:04:49 2020
NAMESPACE: wlp-sample
STATUS: deployed
......
```

■ 以上で、WebSphere Libertyの設定とデプロイは完了です。

– 起動されたPodがReady状態になることを確認してください。

モニタリングの設定例：ServiceMonitorの定義（1/4）

■ ServiceMonitorを定義します。手順は、以下の通りです。

■ 作成されたServiceの確認

– WebSphere Libertyをデプロイすると、下記のServiceが作成されます。

• ベース名は、「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字、または、変数ファイルで「service.name」に指定した名前

サービス名	説明
ベース名	• アプリケーションへのアクセスで使用されるサービスで、必ず作成されます。 • ClusterIP、または、NodePortとして定義されます。 • Helm Chartの変数ファイルで「ssl.enabled=true」を指定した場合、ポート名がSSL通信用の名前(https)になります。
ベース名-sts	• StatefulSetとしてデプロイされた時に、追加で作成されるサービスです。 • Headless Serviceとして定義されます。
ベース名-http-clusterip	• Helm Chartの変数ファイルで「ssl.enabled=true」と「monitoring.enabled=true」が指定された場合に作成されます。 • httpでアクセス可能なサービスとして、Prometheusによるメトリクス収集用に作成されます。

– 作成されたServiceの名前を確認します。

```
> oc get service
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana-service               ClusterIP   172.21.148.225   <none>        3000/TCP   16h
prometheus-operated           ClusterIP   None             <none>        9090/TCP   16
infra-test-http-ibm-webs      ClusterIP   172.21.180.218   <none>        9080/TCP   6s
```

モニタリングの設定例：ServiceMonitorの定義（2/4）

■ 作成されたServiceの確認（続き）

– ServiceMonitor定義で必要になるServiceの情報を確認します。

• 名前が「ベース名-http-clusterip」のServiceが作成されていた場合は、その内容を確認します。

```
> oc get service infra-test-http-ibm-webs -o yaml
apiVersion: v1
kind: Service
metadata:
labels:
    app: infra-test-http-ibm-webs
    chart: ibm-websphere-liberty-1.10.0
    heritage: Helm
    release: infra-test-http
  name: infra-test-http-ibm-webs
  namespace: wlp-sample
  ......
spec:
  clusterIP: 172.21.180.218
  ports:
  - name: http
    port: 9080
    protocol: TCP
    targetPort: 9080
  selector:
    app: infra-test-http-ibm-webs
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

appラベルが定義されており、
値は「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字になります。

名前がhttpのポートが定義されています。

モニタリングの設定例：ServiceMonitorの定義（3/4）

■ ServiceMonitorの定義

– ServiceMonitorを定義するために、下記のようなyamlファイル(service-monitor.yaml)を作成します。

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: infra-test-http-ibm-webs
  labels:
    k8s-app: infra-test-http-ibm-webs
    app:     infra-test-http-ibm-webs
  namespace: wlp-sample
spec:
  namespaceSelector:
    matchNames:
      - wlp-sample
  selector:
    matchLabels:
      app: infra-test-http-ibm-webs
  endpoints:
    - port: http
      path: /metrics
      interval: 30s
```

- ServiceMonitorの名前です。
  ここでは、Serviceと同じ名前を指定しています。
- ServiceMonitorに付与するラベルです。
  ここでは、Serviceと同じラベルを定義し、追加でk8s-appのラベルも定義しています。
- Serviceを特定のネームスペースから選択するように指定します。
  ここでは、WebSphere Libertyをデプロイしたネームスペースを指定します。
- Serviceを特定するためのラベルの条件を指定します。
  ここでは、先ほどデプロイしたWebSphere Liberty用Serviceが検出されるように条件を指定しています。
- Serviceに定義されているポートの名前とメトリクスが公開されいるパスを指定します。

– oc applyコマンドを実行し、yamlファイルからSerivceMonitorを作成します。

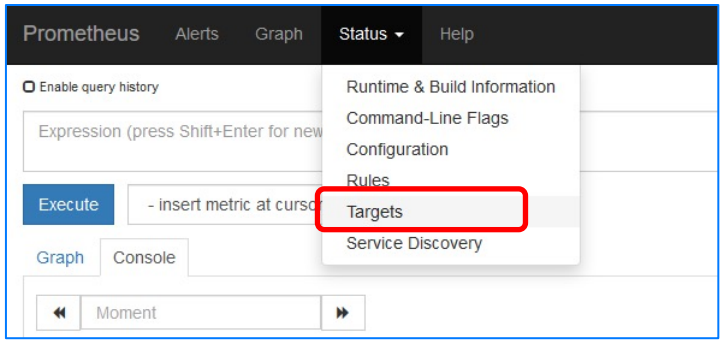
```
> oc apply -f service-monitor.yaml
servicemonitor.monitoring.coreos.com/infra-test-http-ibm-webs created
```

モニタリングの設定例：ServiceMonitorの定義（4/4）

■ モニタリングの確認

– 作成したServiceMonitorが機能し、 Prometheusがメトリクスを収集しているか確認します。

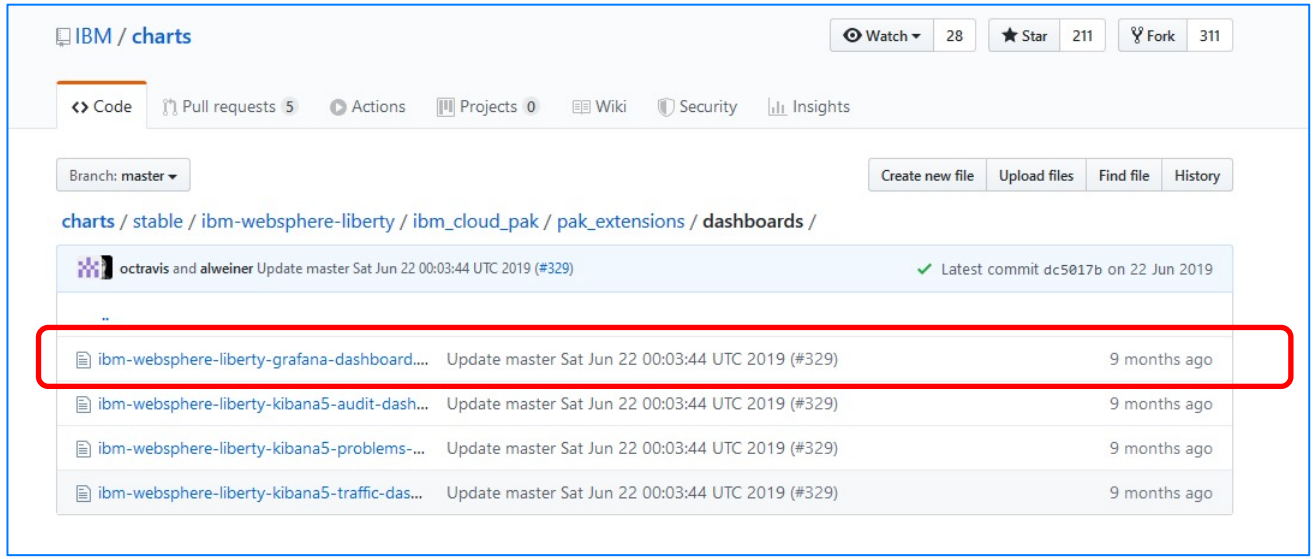
- Prometheusのコンソールにhttpでアクセスします。(アクセス方法は前述)
- コンソールの上部から「Status > Targets」を選択し、Targets画面を表示します。
- モニタリング対象のPodがリストされ、Statusが upになっていることを確認します。



モニタリングの設定例：サンプル・ダッシュボードの利用（1/3）

- サンプル・ダッシュボードをGrafanaにインポートし、メトリクスを確認します。
  手順は、以下の通りです。

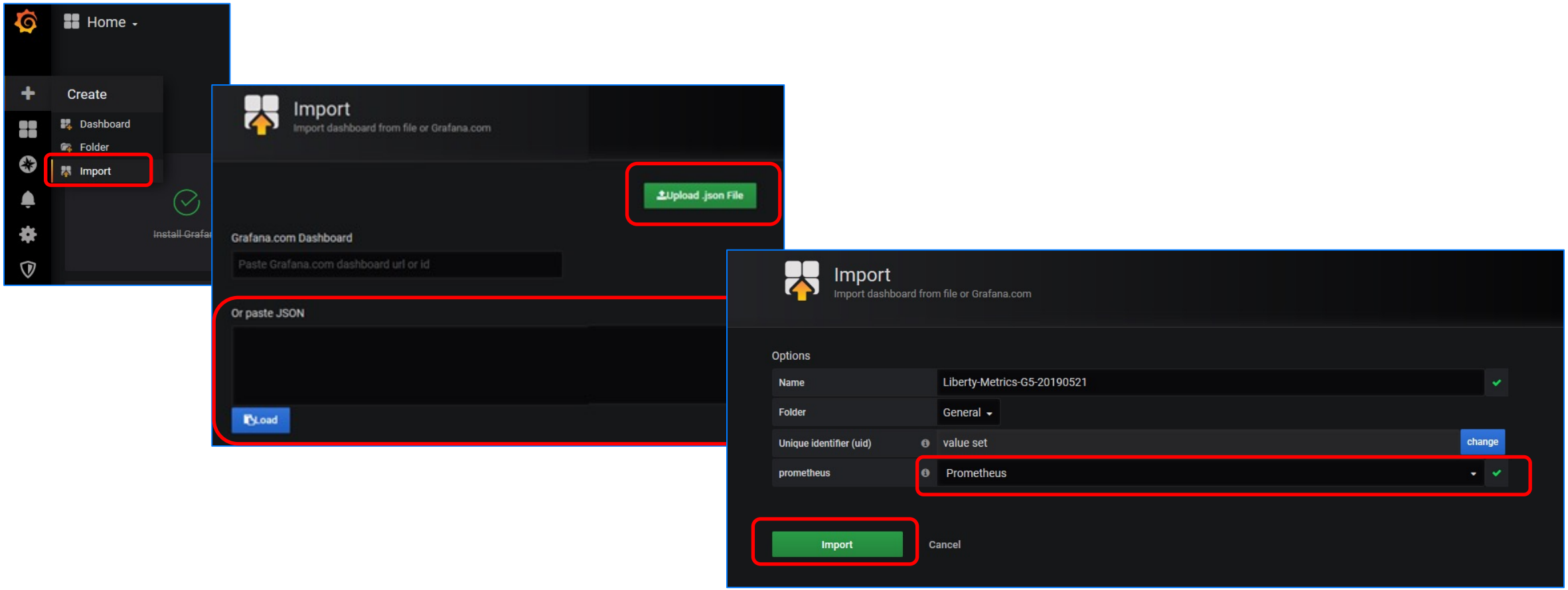
- サンプル・ダッシュボード定義の入手
  - 以下のURLにアクセスし、サンプル・ダッシュボードのjsonファイルを取得します。
    - GitHub / IBM/charts/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards/
      - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty/ibm_cloud_pak/pak_extensions/dashboards



モニタリングの設定例：サンプル・ダッシュボードの利用（2/3）

■ サンプル・ダッシュボードのインポート

– Grafanaにログインし(ログイン方法は前述)、画面の左から「Create > Import」を選びます。
– Import画面にサンプル・ダッシュボードの内容を張り付けるか、アップロードします
– 続きの画面で、Data sourceとして使用するPrometheusを選択し、インポートします。

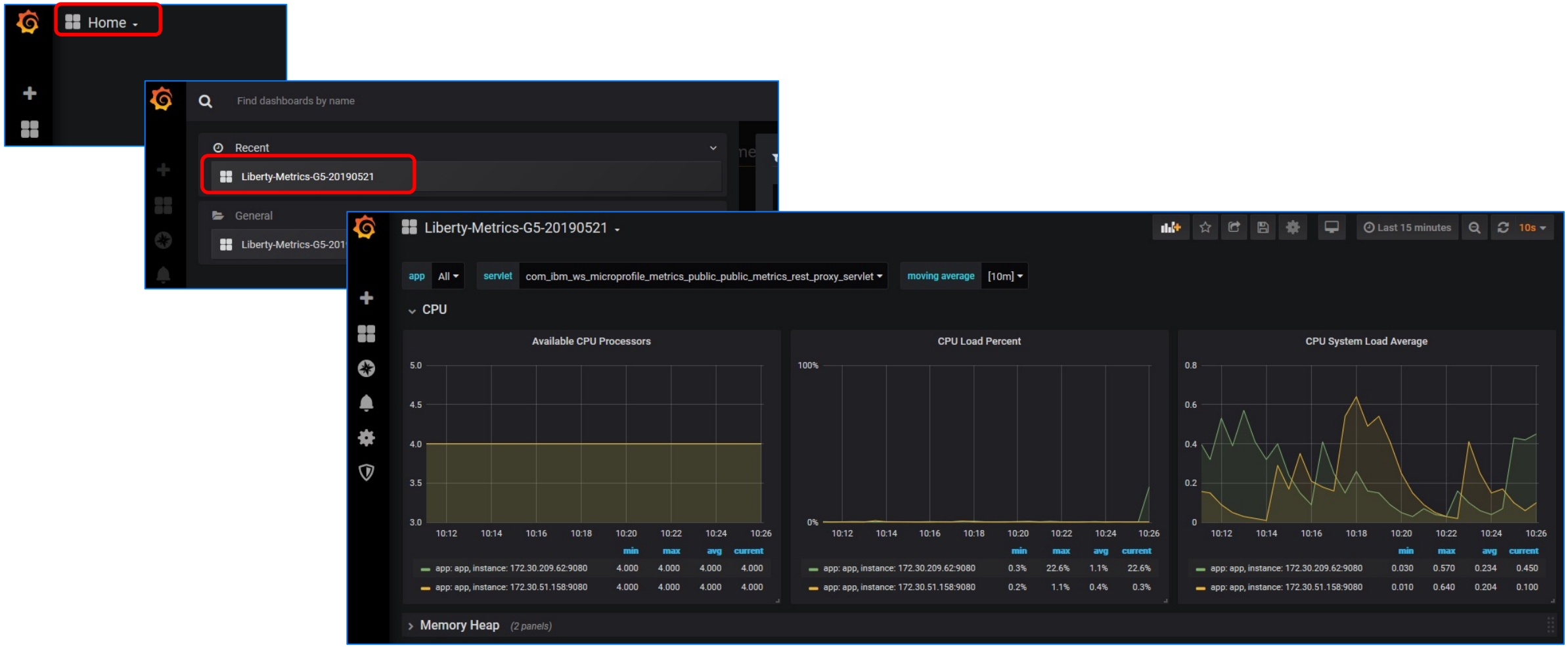


モニタリングの設定例：サンプル・ダッシュボードの利用（3/3）

■ サンプル・ダッシュボードの表示

– GrafanaのHomeドロップダウンをクリックし、サンプル・ダッシュボードを表示します。

• サンプル・ダッシュボードの表示例・操作例は、補足を参照



(補足)WebSphere Libertyで利用可能なメトリクス

- WebSphere Libertyの「mpMetrics-1.1」フィーチャーは、MicroProfile Metrics 1.1仕様に準拠しています。

スコープ	REST APIのURL	Prometheusのメトリクス名	説明
(ルート)	/metrics	n/a	全スコープのモニタリング情報
base	/metrics/base	base:<メトリクス名>	JVMなどのモニタリング情報 mpMetrics-1.1で取得可能 （**MicroProfile Metrics 1.1で定義されているメトリクス**）
vendor	/metrics/vendor	vendor:<メトリクス名>	アプリケーション・サーバーなどのモニタリング情報 mpMetrics-1.1 + monitor-1.0で取得可能 （**Libertyによって提供されるLiberty固有のメトリクス**）
application	/metrics/application	application:<メトリクス名>	アプリケーションのモニタリング情報 （**アプリケーション開発者が実装するメトリクス**）

WebSphere Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mon.html
WebSphere Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター / MicroProfile メトリックを使用したモニタリング
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mp_metrics_monitor.html
WebSphere Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター / MicroProfile メトリックを使用したモニタリング / MicroProfile Metrics REST API
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_mp_metrics_rest_api.html
WebSphere Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター / MicroProfile メトリックを使用したモニタリング / MicroProfile Metrics 1.1 ベンダー・メトリック
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_monitor_metrics_rest_api.html
WebSphere Liberty Knowledge Center / Liberty 環境でのアプリケーションの開発 / アプリケーションへのメトリックの追加 / MicroProfile メトリック計測 API
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_mp_metrics_api.html

(補足)WebSphere Libertyで利用可能なメトリクス（baseスコープ）

- General JVM Stats
  - memory.usedHeap
  - memory.committedHeap
  - memory.maxHeap
  - gc.global.count
  - gc.global.time
  - gc.scavenge.count
  - gc.scavenge.time
  - jvm.uptime
- Thread JVM Stats
  - thread.count
  - thread.daemon.count
  - thread.max.count
- Thread Pool Stats
  - vendorスコープで定義
- ClassLoading JVM Stats
  - classloader.currentLoadedClass.count
  - classloader.totalLoadedClass.count
  - classloader.totalUnloadedClass.count
- Operating System
  - cpu.availableProcessors
  - cpu.systemLoadAverage
  - cpu.processCpuLoad

base スコープ：MicroProfile Metrics で定義されているもの

WebSphere Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター / MicroProfile メトリックを使用したモニタリング
（「Metrics for Eclipse MicroProfile 1.1 仕様」のリンク先資料の「Chapter 4. Required Metrics」参照）
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mp_metrics_monitor.html

(補足)WebSphere Libertyで利用可能なメトリクス（vendorスコープ）

■ Webアプリケーション・メトリクス
- servlet.%s.request.total
- servlet.%s.responseTime.total
  - (%: ServletStats MXBeanインスタンス名)

■ スレッド・プール・メトリクス
- threadpool.%s.activeThreads
- threadpool.%s.poolSize
  - (%: ThreadPoolStats MXBeanインスタンス名)

■ セッション管理メトリクス
- session.%s.create.total
- session.%s.liveSessions
- session.%s.activeSessions
- session.%s.invalidated.total
- session.%s.invalidatedbyTimeout.total
  - (%: SessionStats MXBeanインスタンス名)

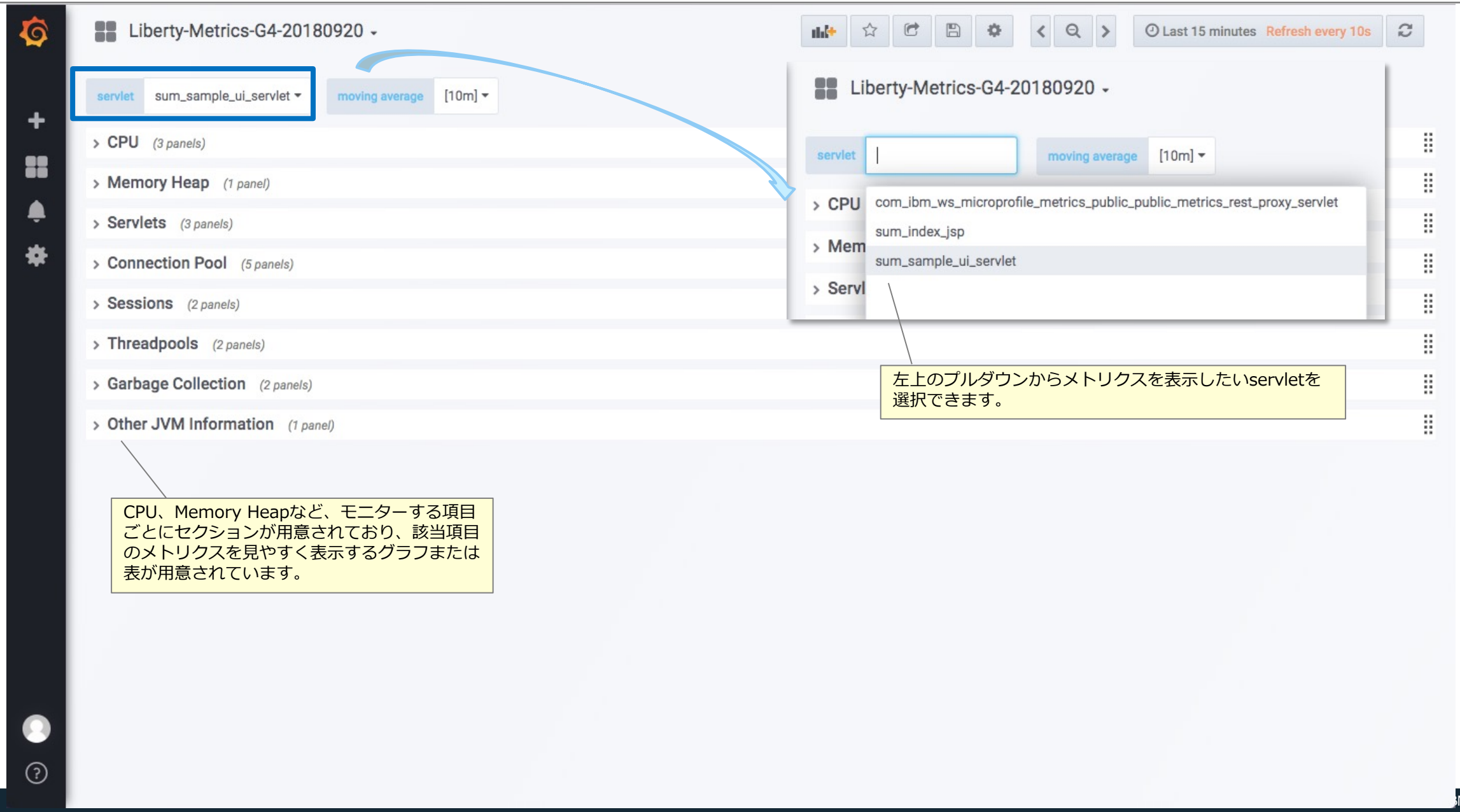
■ Connection Poolメトリクス
- connectionpool.%s.create.total
- connectionpool.%s.destroy.total
- connectionpool.%s.managedConnections
- connectionpool.%s.connectionHandles
- connectionpool.%s.freeConnections
- connectionpool.%s.waitTime.total
- connectionpool.%s.queuedRequests.total
- connectionpool.%s.inUseTime.total
- connectionpool.%s.usedConnections.total
  - (%: ConnectionPoolStats MXBeanインスタンス名)

■ JAX-WSメトリクス
- (リンク先参照)

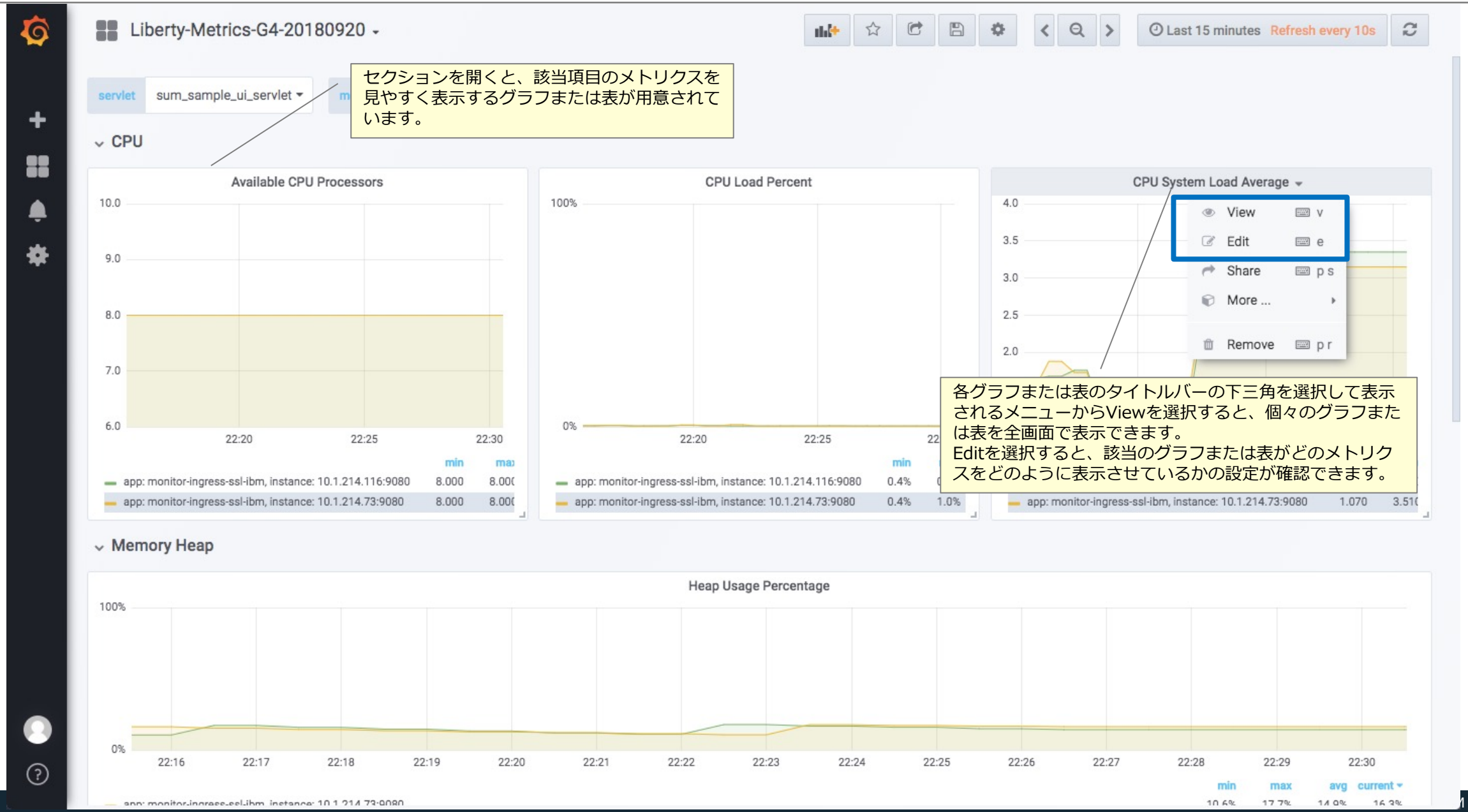
vendor スコープ： WebSphere Liberty 固有のメトリクス
(詳細は下記リンクを参照)

WebSphere Liberty Knowledge Center / MicroProfile Metrics 1.1 ベンダー・メトリック
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_monitor_metrics_rest_api.html

(補足)サンプル・ダッシュボードの表示例（全般）

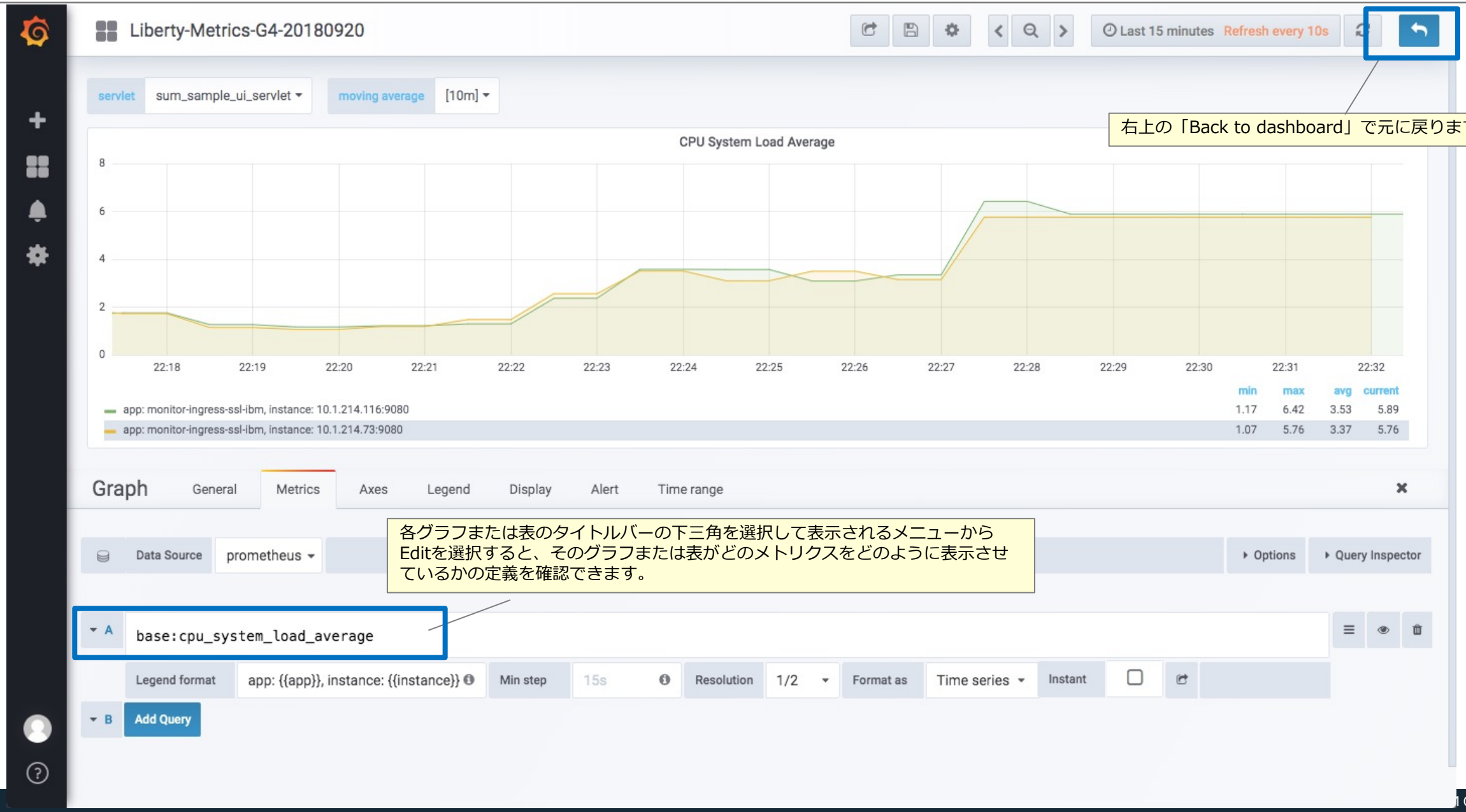


(補足)サンプル・ダッシュボードの表示例（全般）



(補足)サンプル・ダッシュボードの表示例（全般）

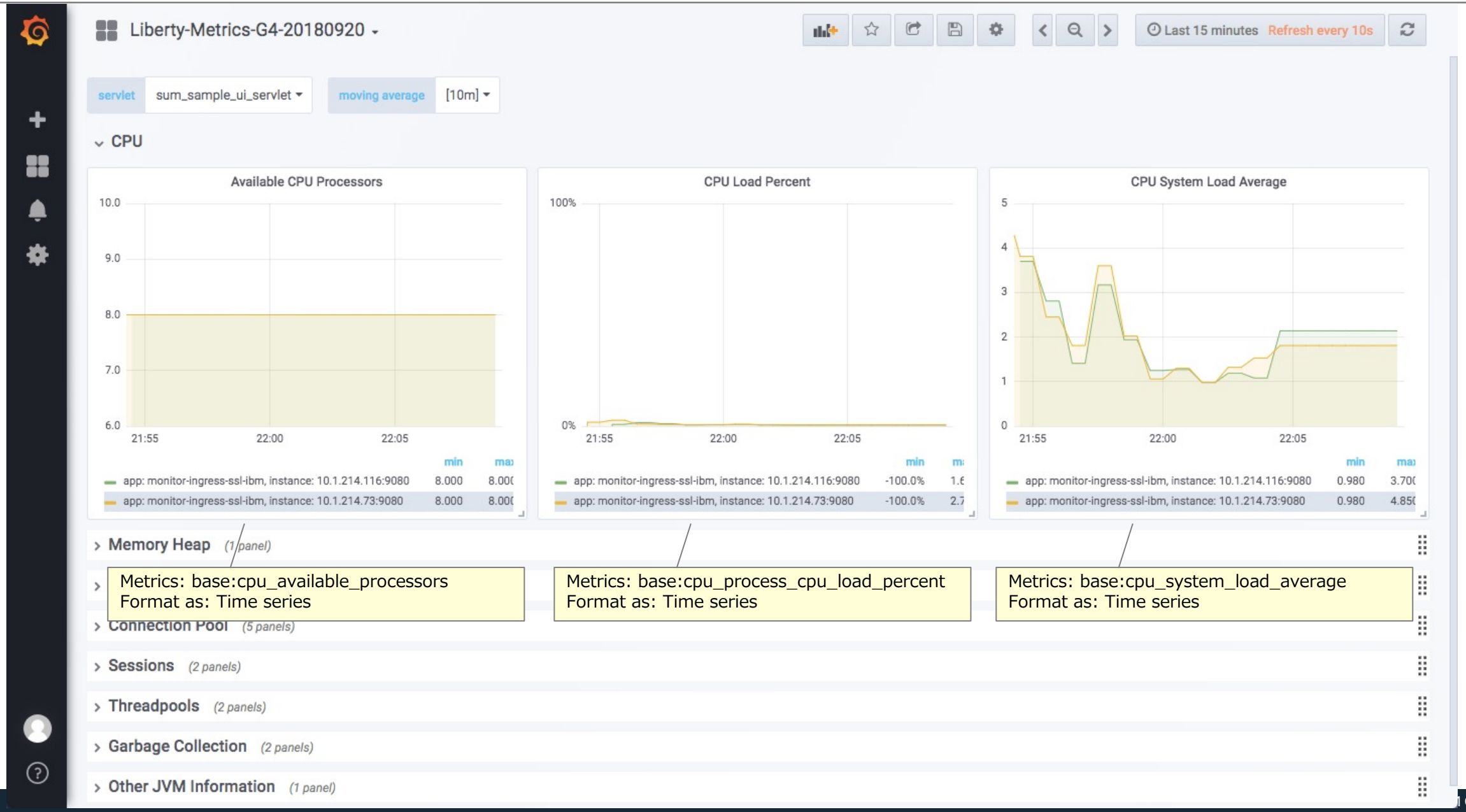
(補足)サンプル・ダッシュボードの表示例（全般）



右上の「Back to dashboard」で元に戻ります。

各グラフまたは表のタイトルバーの下三角を選択して表示されるメニューから
Editを選択すると、そのグラフまたは表がどのメトリクスをどのように表示させ
ているかの定義を確認できます。

(補足)サンプル・ダッシュボードの表示例（CPU）

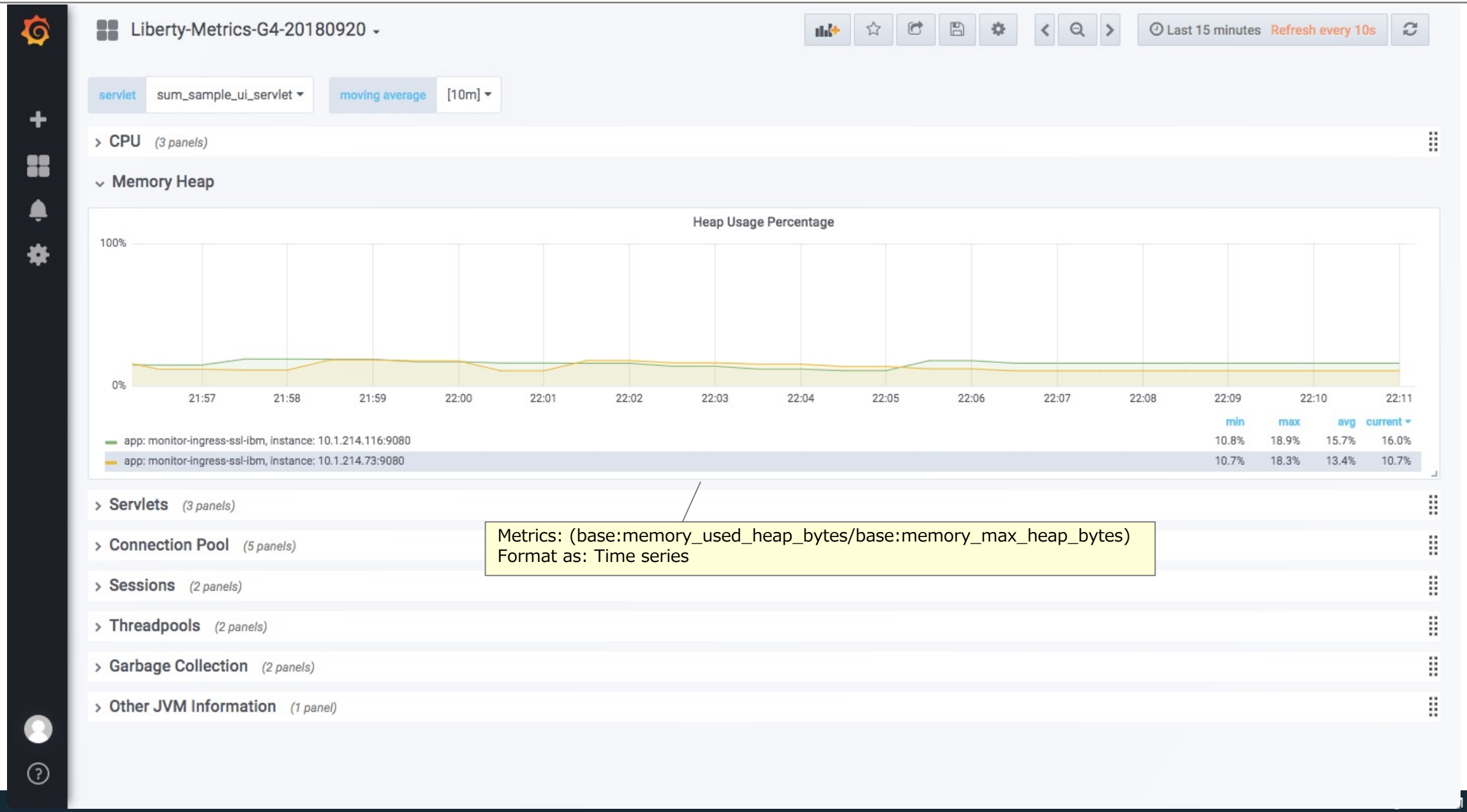


Metrics: base:cpu_available_processors
Format as: Time series

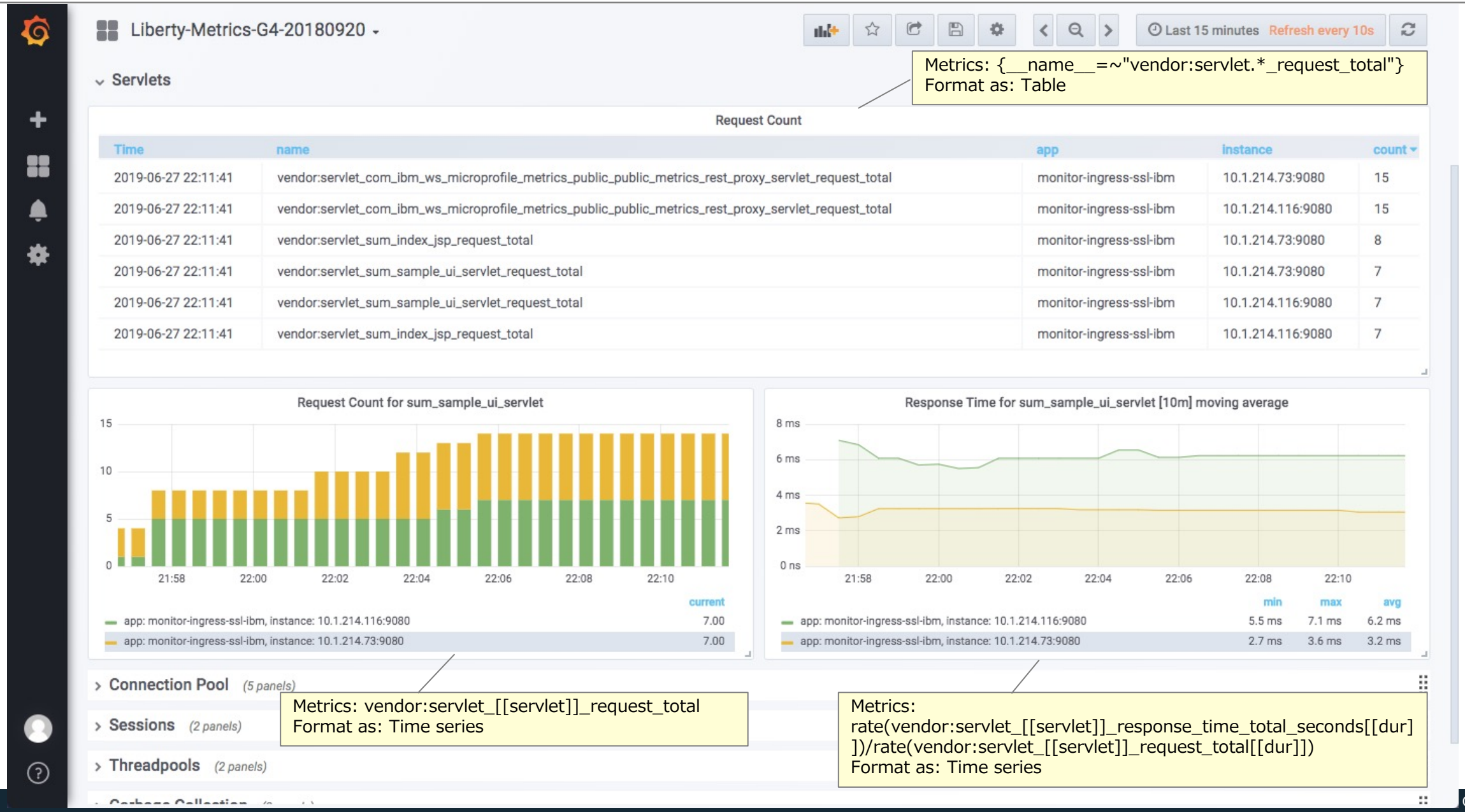
Metrics: base:cpu_process_cpu_load_percent
Format as: Time series

Metrics: base:cpu_system_load_average
Format as: Time series

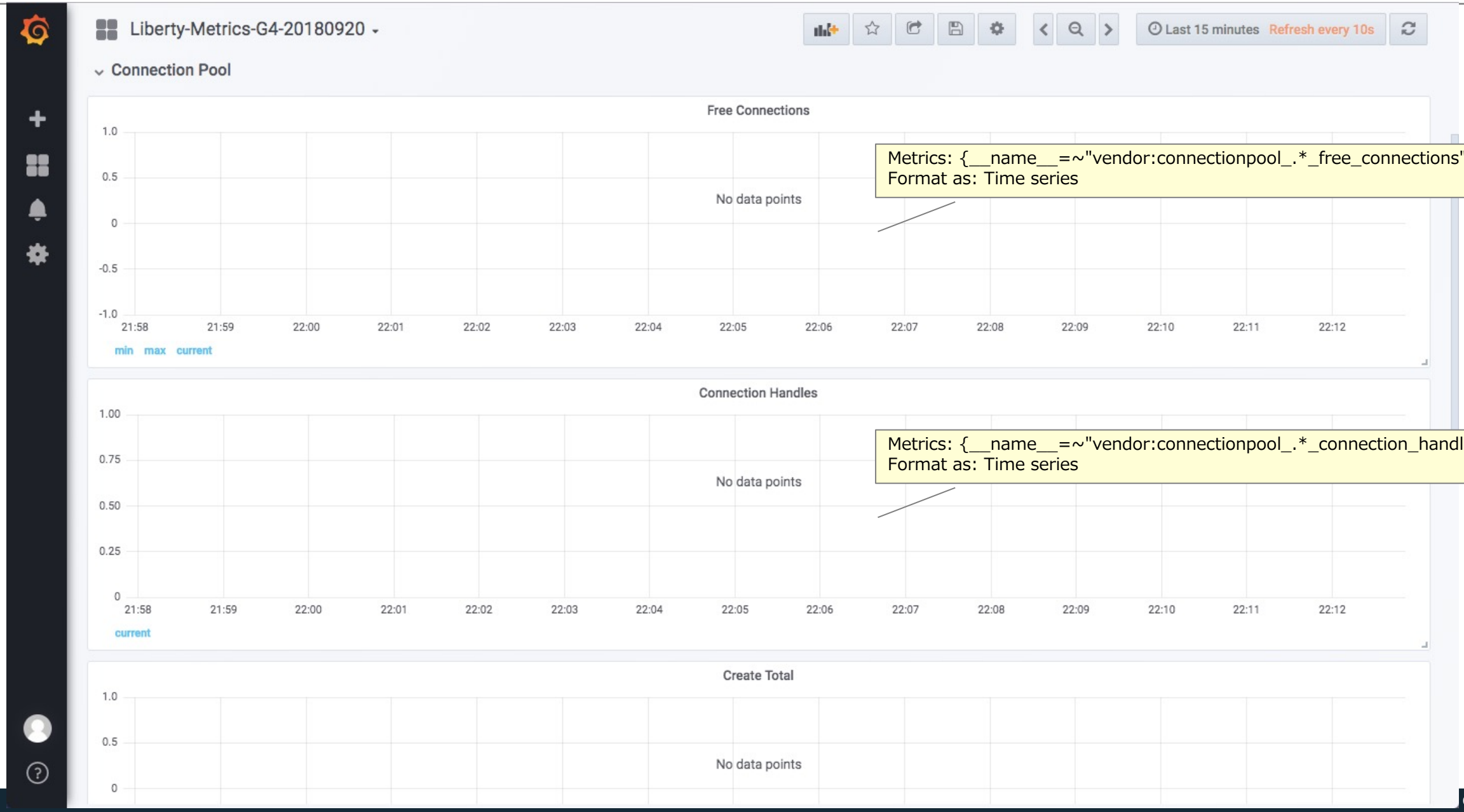
(補足)サンプル・ダッシュボードの表示例（Memory Heap）



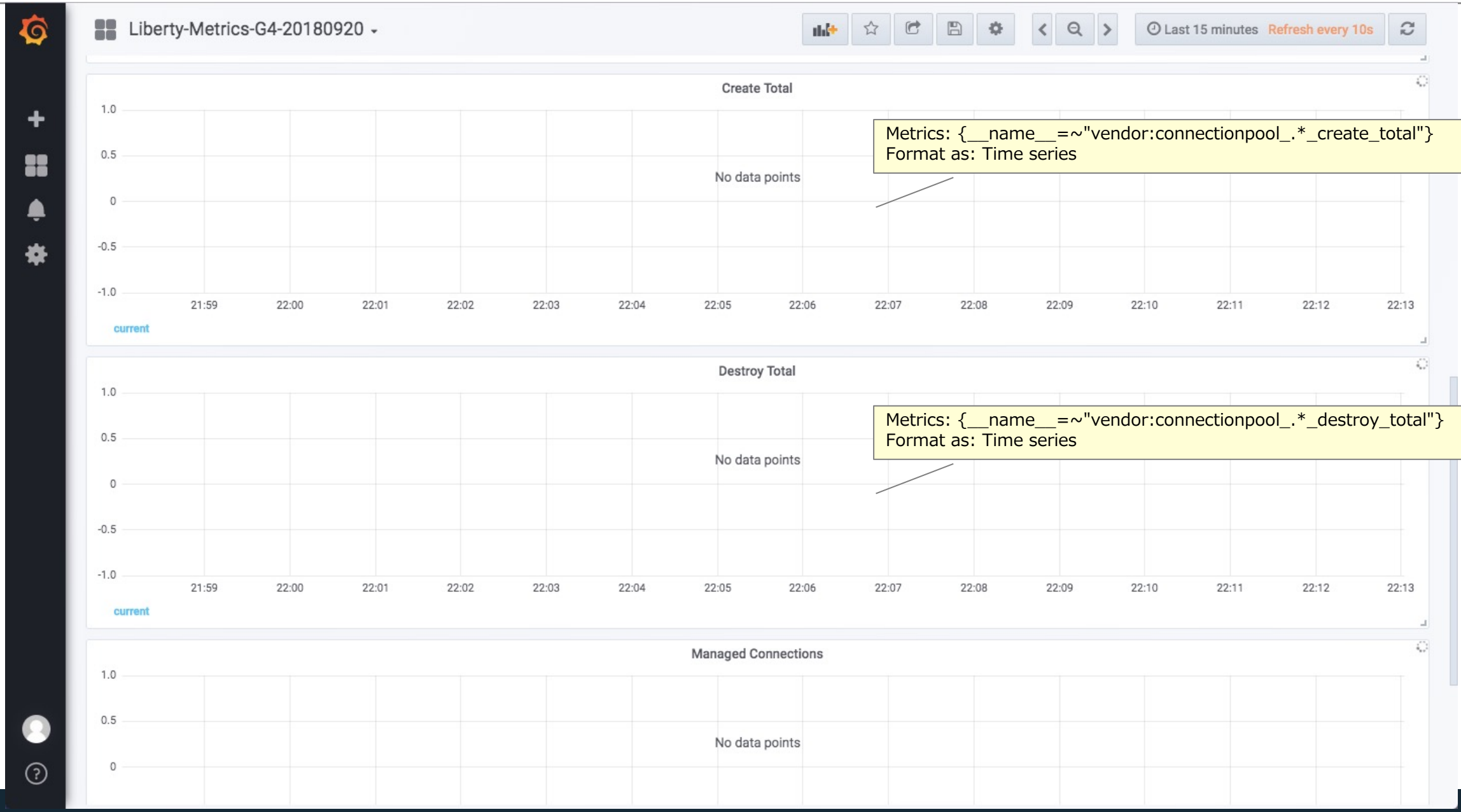
(補足)サンプル・ダッシュボードの表示例（Servlets）



(補足)サンプル・ダッシュボードの表示例（Connection Pool）



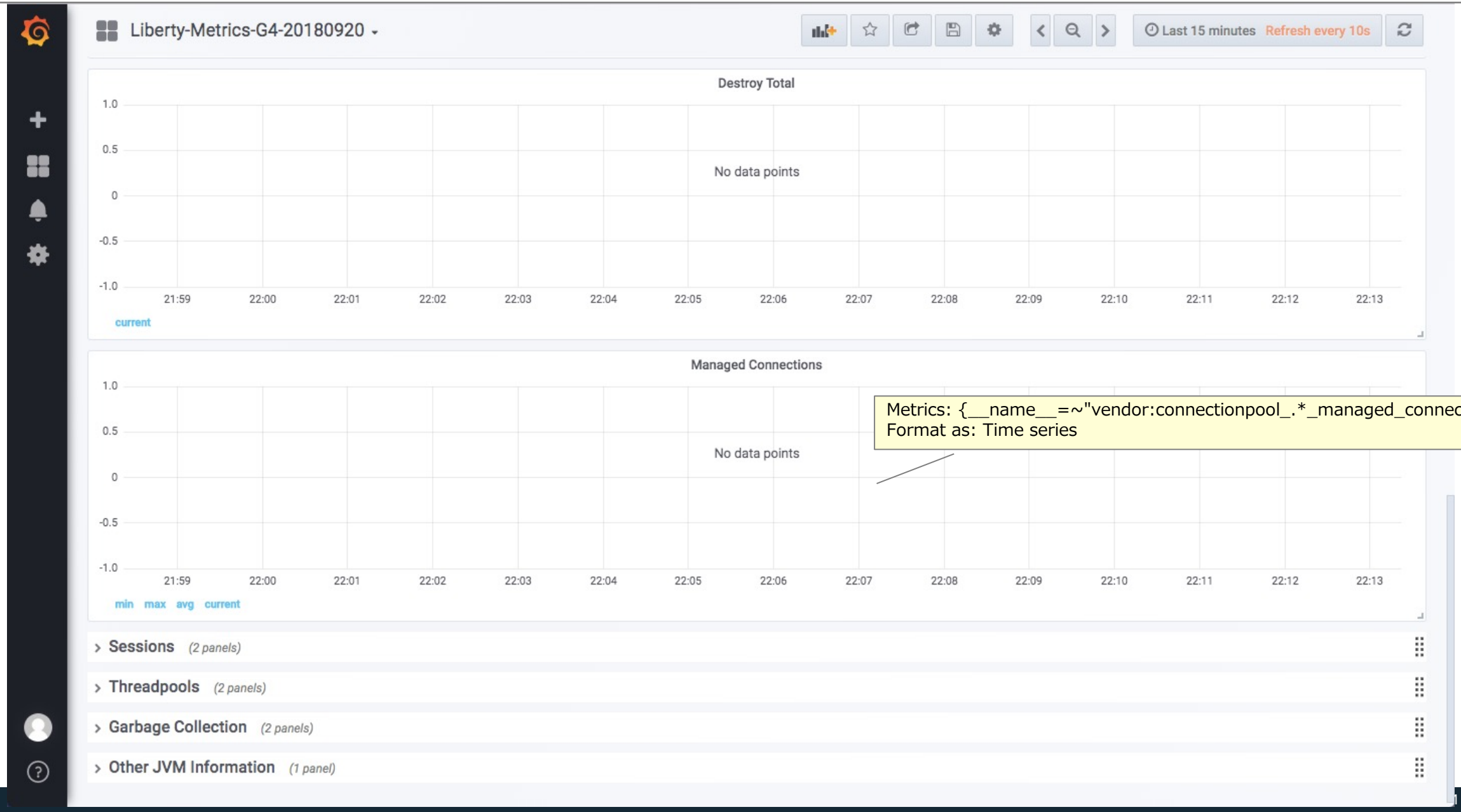
(補足)サンプル・ダッシュボードの表示例（Connection Pool）



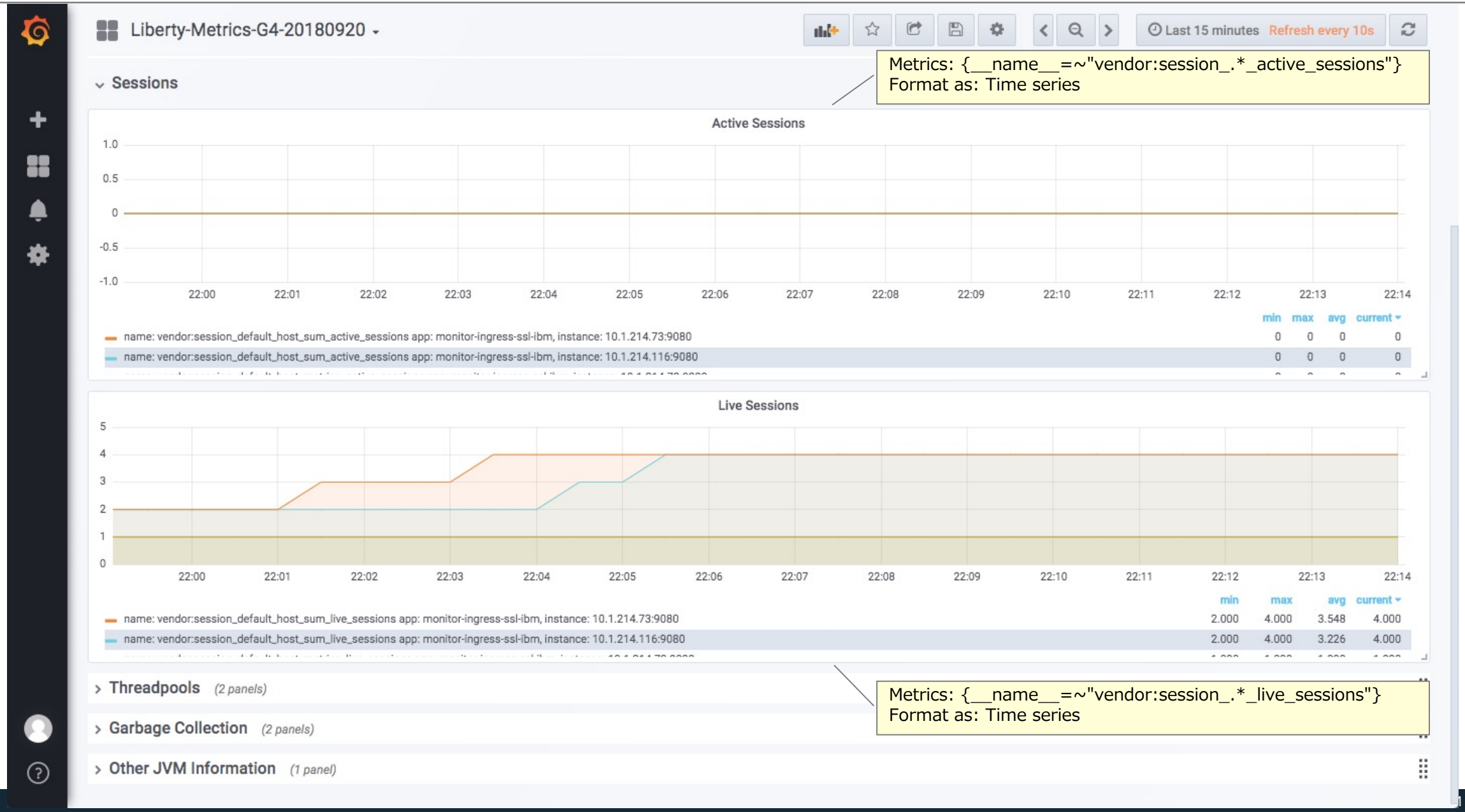
Metrics: {__name__=~"vendor:connectionpool_.*_create_total"}
Format as: Time series

Metrics: {__name__=~"vendor:connectionpool_.*_destroy_total"}
Format as: Time series

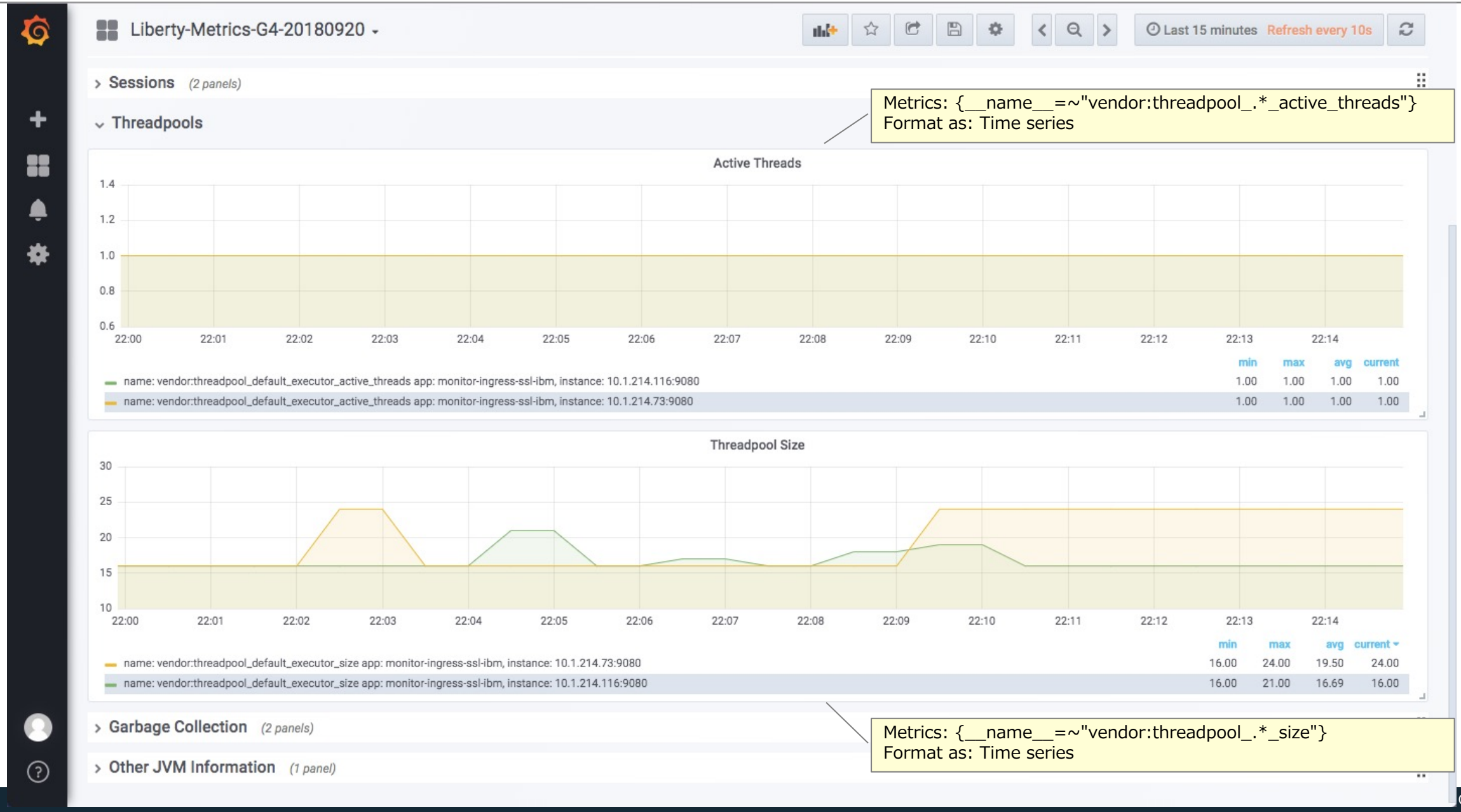
(補足)サンプル・ダッシュボードの表示例（Connection Pool）



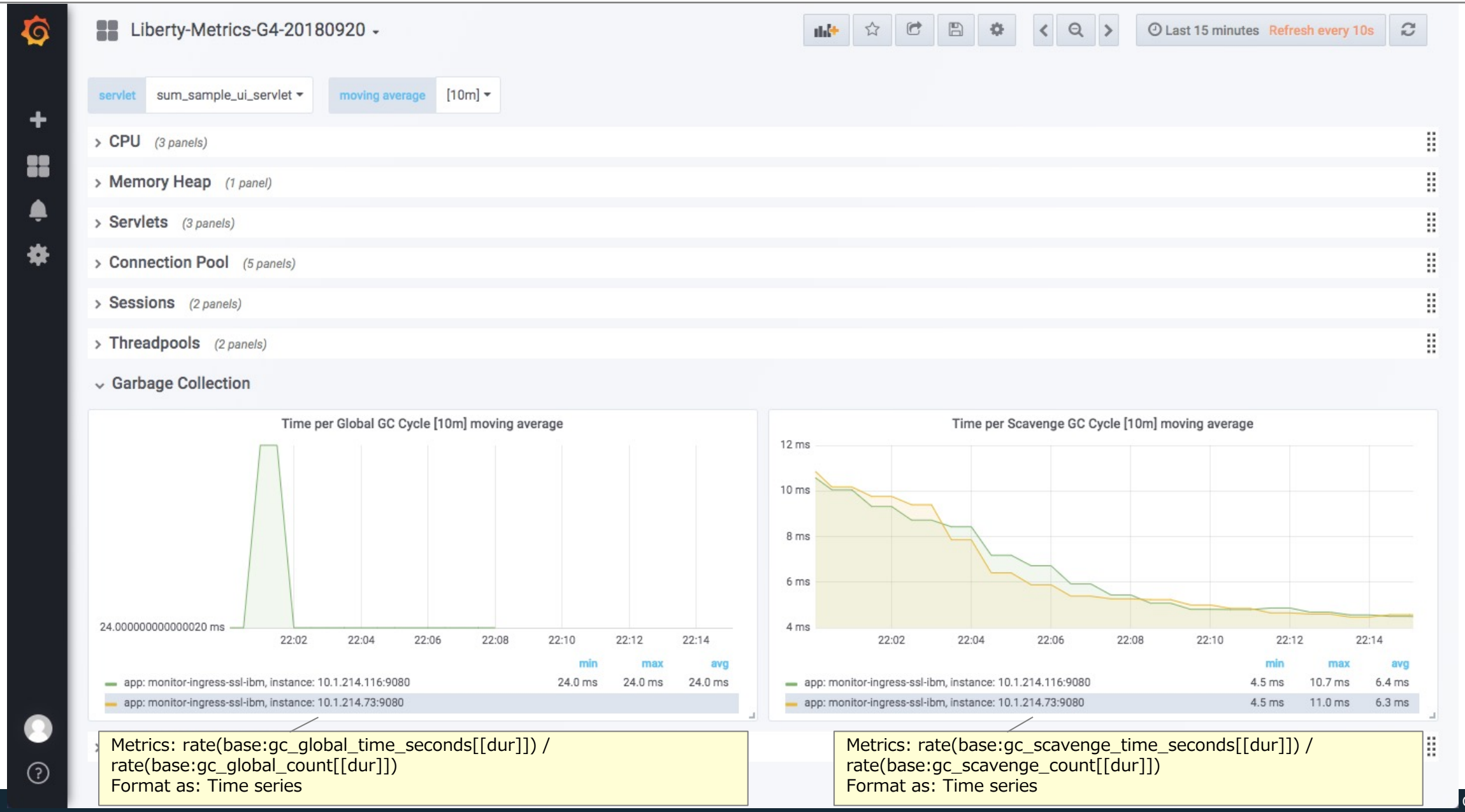
(補足)サンプル・ダッシュボードの表示例（Sessions）



(補足)サンプル・ダッシュボードの表示例（Threadpools）



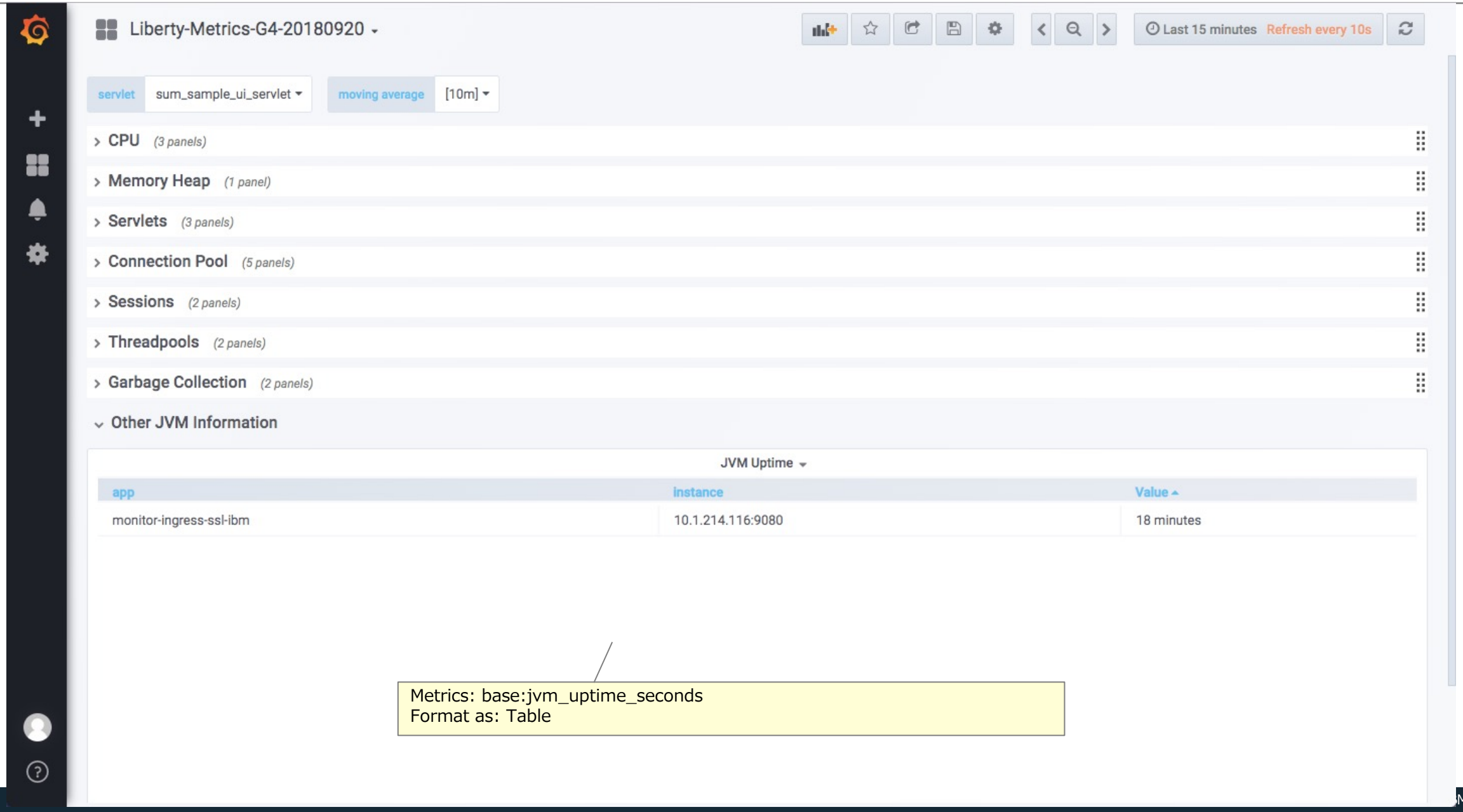
(補足)サンプル・ダッシュボードの表示例（Garbage Collection）



Metrics: rate(base:gc_global_time_seconds[[dur]]) /
rate(base:gc_global_count[[dur]])
Format as: Time series

Metrics: rate(base:gc_scavenge_time_seconds[[dur]]) /
rate(base:gc_scavenge_count[[dur]])
Format as: Time series

(補足)サンプル・ダッシュボードの表示例（Other JVM Info.）



(補足)環境変数MP_METRICS_TAGSを使用したメトリクスのタグ付け

■ 環境変数MP_METRICS_TAGSを使用するとメトリクスにタグを付与することができます。

– ユースケース

- メトリクスにDeploymentや環境などを示す任意の情報をタグとして付与することができます。
- Grafanaのサンプルダッシュ・ボードに関して一部の表をタグでソートして表示することができます。

– 参考リンク

- WebSphere Liberty Knowledge Center / MicroProfile メトリック計測 API / メタデータによるメトリックの記述
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_mp_metrics_api.html

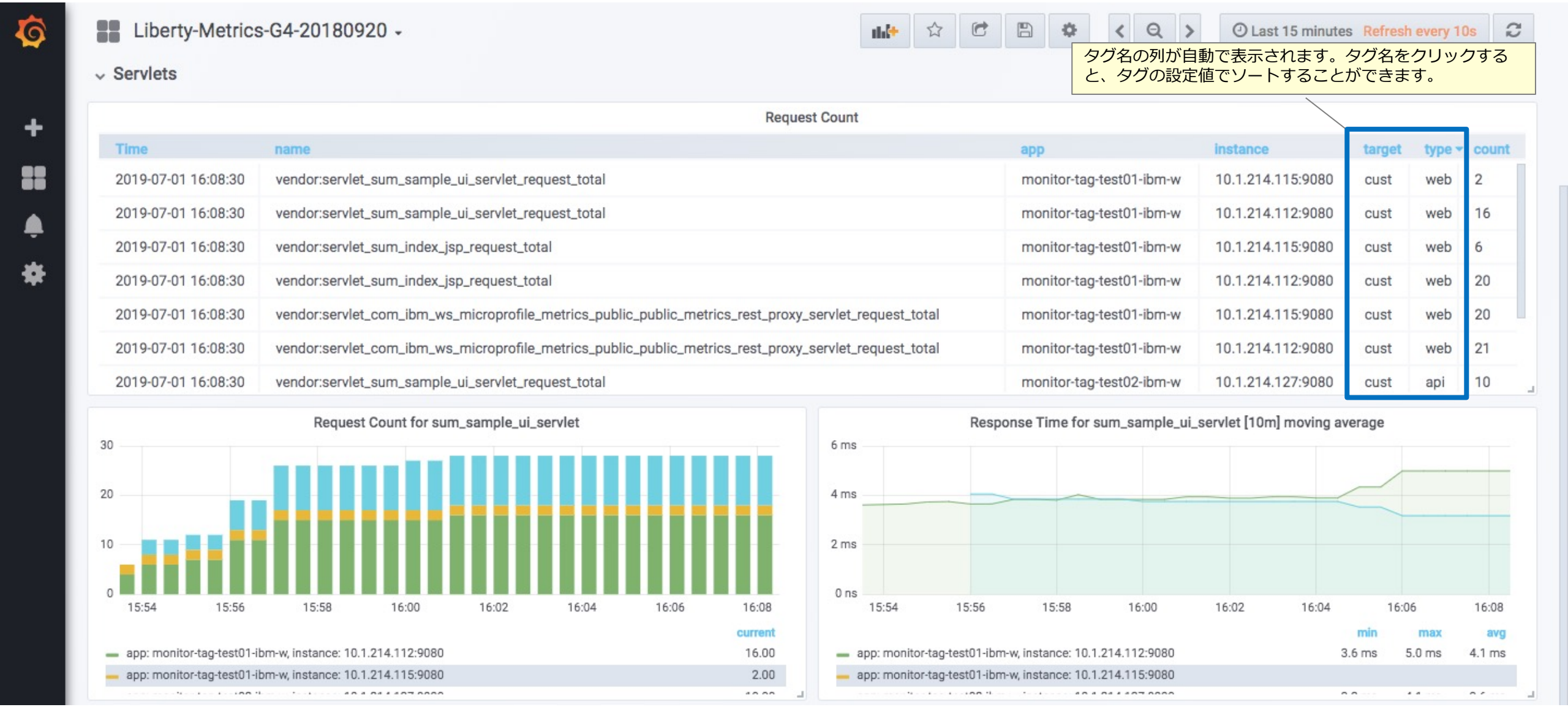
– 設定箇所

- 環境変数MP_METRICS_TAGSは、Helm Chartの変数ファイルの「image.extraEnvs」で設定することができます。

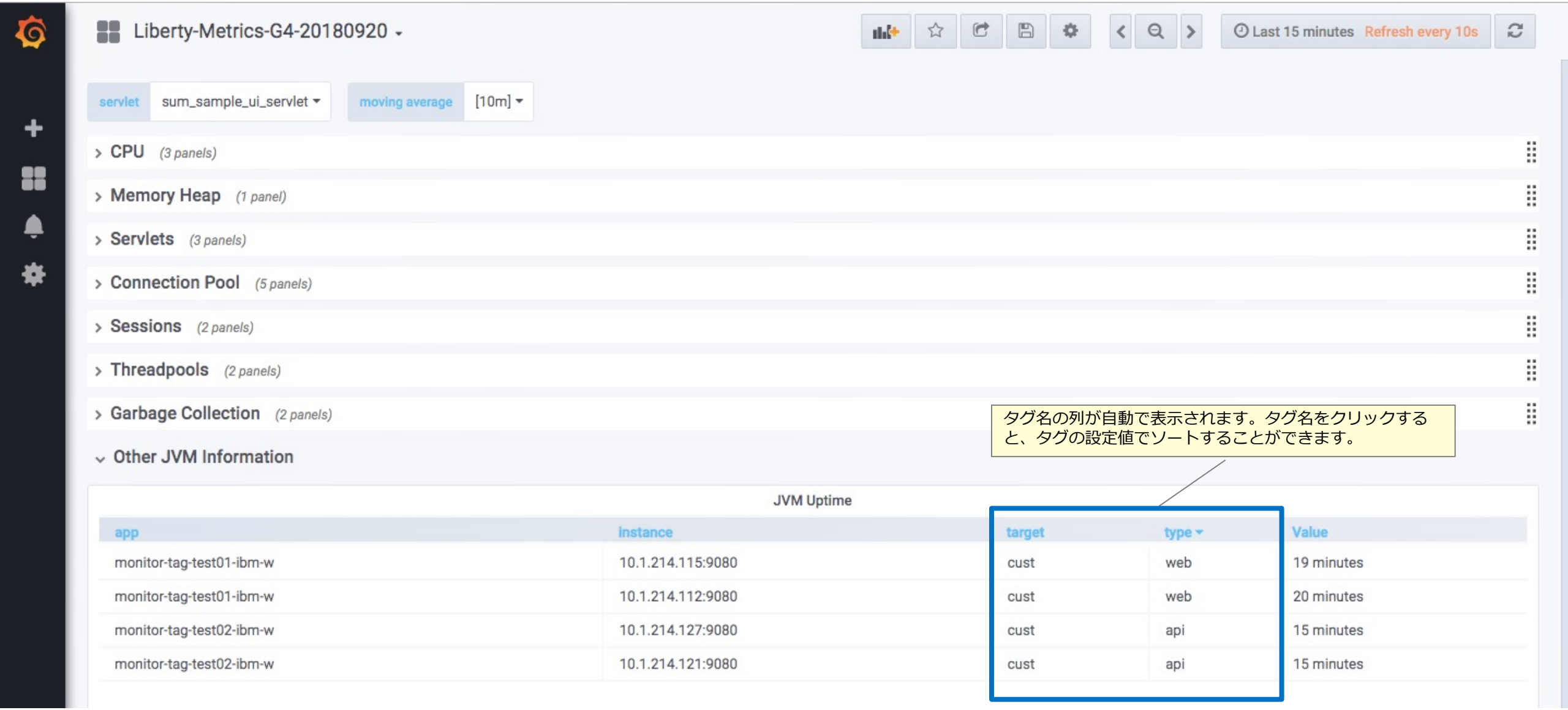
(例)Helmの変数ファイルの例 (MP_METRICS_TAGSに、アプリケーションの「type」として「web」、「target」として「cust」というタグを付与する場合)

```
image:
 ....
  extraEnvs:
  - name: MP_METRICS_TAGS
    value: "type=web,target=cust"
......
```

(補足)MP_METRICS_TAGSでタグ付与した場合の表示例（Servlets）



(補足)MP_METRICS_TAGSでタグ付与した場合の表示例（Other JVM Info.）



JVM Uptime

app	instance	target	type	Value
monitor-tag-test01-ibm-w	10.1.214.115:9080	cust	web	19 minutes
monitor-tag-test01-ibm-w	10.1.214.112:9080	cust	web	20 minutes
monitor-tag-test02-ibm-w	10.1.214.127:9080	cust	api	15 minutes
monitor-tag-test02-ibm-w	10.1.214.121:9080	cust	api	15 minutes

8. ConfigMap と Secret の利用

目次(8章)

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内で稼働するWebSphere Libertyやアプリケーションに渡すことができます。
- 参照：第１章の「基礎知識：ConfigMap と Secret」

構成情報と環境変数の優先順位：構成情報

- WebSphere Libertyの構成情報(server.xmlまたはserver.xmlフラグメント)は、複数の箇所に記述できます。

- WebSphere 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。
    - WebSphere 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
    - WebSphere 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server.xmlフラグメントでの指定

中．/config内の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server.xmlフラグメントでの指定

構成情報と環境変数の優先順位：環境変数

- ConfigMapやSecretに定義されたvalueを環境変数を通じてWebSphere Libertyに渡すことができます。

- ConfigMapやSecretに定義されたvalueを環境変数を通じてWebSphere 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環境変数の設定値は、高い順位の設定値で上書きされます。

高．イメージ内のserver.envファイルでの設定

中．Helm Chartの変数ファイルでのデプロイ時の設定

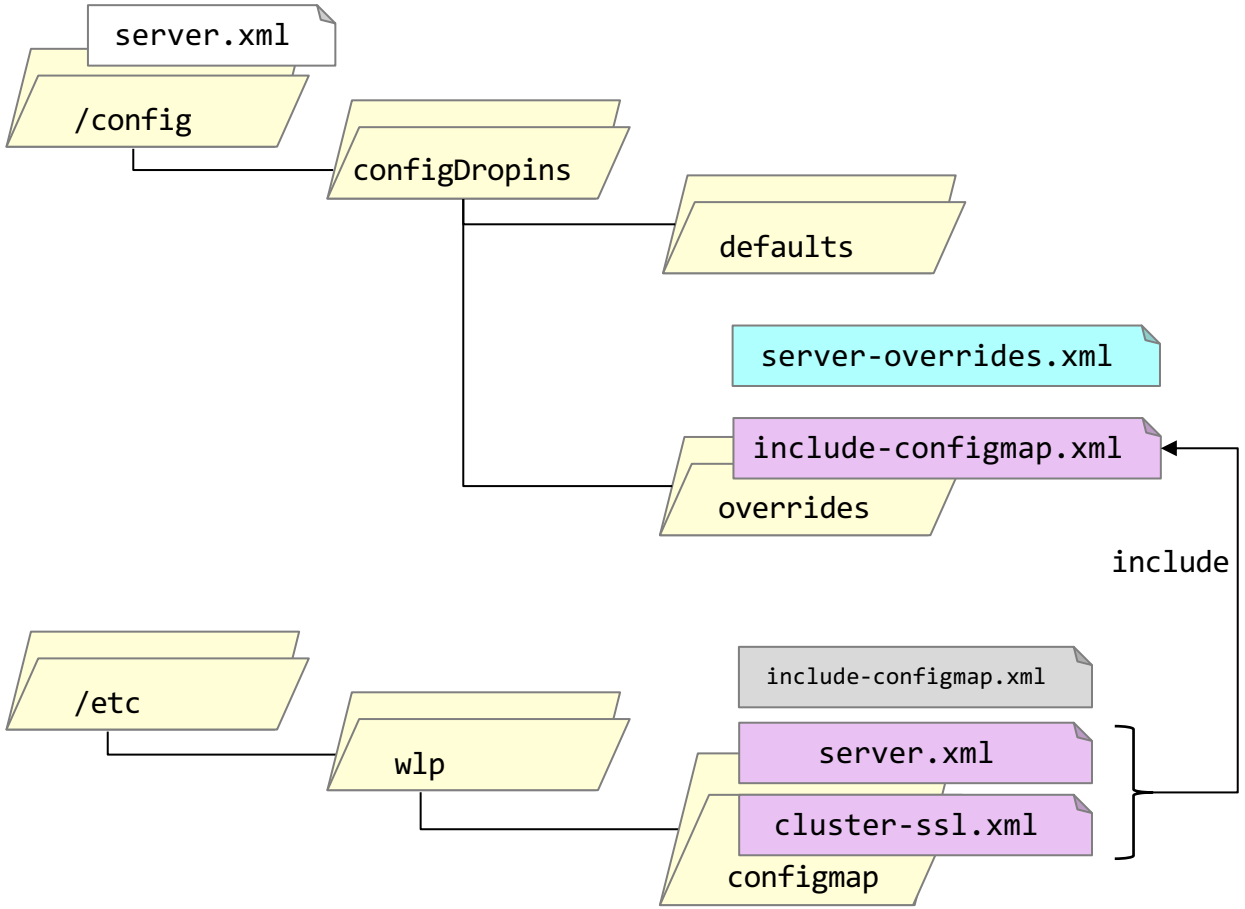
Helm ChartでConfigMapやSecretのvalueを設定した場合など

低．イメージでの環境変数の設定

  - イメージ内のserver.envで指定されている環境変数の設定値は変更できません。

WebSphere LibertyのHelm Chartが生成/使用するConfigMap

- Helm Chartを使ってデプロイすると、ConfigMap内の構成情報がWebSphere Liberty のコンテナーに以下の様にマウントされます。



ユーザーが作成したConfigMap内のserver-overrides.xmlがsubPathを指定してマウント(*1)されます。
ConfigMapの名前は、Helm Chartの変数ファイルで指定します。

Helm Chartが作成するConfigMapの内容が配置されます
ConfigMapの名前は、「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字になります。
このConfigMap内のinclude-configmap.xmlには、/etc/wlp/configmap内のserver.xmlとcluster-ssl.xmlをincludeするタグが記述されており、/config/configDropins/overridesにsubPathを指定してマウント(*1)されます。
さらに、このConfigMapの内容全てが、/etc/wlp/configmapにsubPathを指定しないでマウント(*2)されます。

- server.xmlは、空の状態でConfigMap内に作成されます。
- cluster-ssl.xmlは、クラスSSL構成を有効にした場合にのみ、ConfigMap内に作成されます。
- include-configmap.xmlもマウントされていますが、こちらは使用されません。

(*1)subPathを指定してマウント: ConfigMapを変更しても、コンテナーに動的に反映されません。
(変更を反映するにはPodを削除して再作成する必要があります。)
(*2)subPathを指定せずにマウント: ConfigMapを変更すると、コンテナーに動的に反映されます。

(参考)WebSphere LibertyのHelm Chartが生成するConfigMap関連の情報

■ Helm Chartが生成するConfigMap定義とPodテンプレート定義の抜粋は以下になります。

```
# <リリース名>-ibm-websphereのConfigMap
# 各エントリは /etc/wlp/configmap にマウントされる
# include-configmap.xml については、
# /config/configDropins/overrides/ にもマウントされる

apiVersion: v1
kind: ConfigMap

data:
################################################################################
# Liberty Fabric
################################################################################
  include-configmap.xml: |-
    <server>
      <include optional="true" location="/etc/wlp/configmap/server.xml"/>
      <include optional="true" location="/etc/wlp/configmap/cluster-ssl.xml"/>
    </server>

  server.xml: |-
    <server>
      <!-- Customize the running configuration. -->
    </server>

  cluster-ssl.xml: |-
    <server>
      <featureManager>
        <feature>ssl-1.0</feature>
      </featureManager>
     (省略)
    </server>
```

```
# LibertyのDeployment/StatefulSet

apiVersion: apps/v1
kind: Deployment

spec:
  template:
    spec:
      volumes:
      - name: liberty-overrides
        configMap:
          name: myliberty-ibm-websphere
          items:
          - key: include-configmap.xml
            path: include-configmap.xml
      - name: liberty-config
        configMap:
          name: myliberty-ibm-websphere

      containers:
      - name: ibm-websphere-liberty
        volumeMounts:
        - name: liberty-overrides
          mountPath: /config/configDropins/overrides/include-configmap.xml
          subPath: include-configmap.xml
          readOnly: true
        - name: liberty-config
          mountPath: /etc/wlp/configmap
          readOnly: true
```

ConfigMapの利用例：server-overrides.xml（1/4）

- server-overrides.xmlを含むConfigMapを作成しコンテナーにマウントすることで、WebSphere Libertyの構成定義をオーバーライドする例です。

- この例では、コンテナー内に定義されているデータソース定義をオーバーライドします。
  - 尚、各要素に id が付与された、下図の server.xml がイメージの /config に配置されている前提としています

```
<server>
    … … … … … … … … …

    <dataSource id="SampleDS" jndiName="jdbc/sample" jdbcDriverRef="SampleJD">
        <properties serverName="dbserver01"
                    portNumber="3333"
                    databaseName="db01"
                    user="dbuser01"
                    password="{xor}LywoOw==" />
    </dataSource>
    <jdbcDriver id="SampleJD">
        <library>
            <fileset dir="${server.config.dir}/resources/mysql" includes="mysql-*.jar"/>
        </library>
    </jdbcDriver>

    … … … … … … … … …
</server>
```

- 手順は次の通りです。

ConfigMapの利用例：server-overrides.xml（2/4）

■ まず、データソース定義をオーバーライドするために、下図のような xml ファイルを作成します。

– 作成するxmlファイルの名前は適当な名前で問題ありません。
– この例では、DB 接続ユーザーとパスワードをファイル内に直接記述しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
    <dataSource id="SampleDS" jndiName="jdbc/sample" jdbcDriverRef="SampleJD">
        <properties serverName="mysql"
                    portNumber="3306"
                    databaseName="sampledb"
                    user="user"
                    password="{xor}Lz4sLCgwLTs=" />
    </dataSource>
</server>
```

• パスワードは、WebSphere Libertyが提供するsecurityUtility encodeコマンドでエンコードしたものを記載します。
  – WebSphere Libertyのコンテナー・イメージを利用してコマンドを実行する方法が簡単です。

```
docker run --rm -it ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi /opt/ibm/wlp/bin/securityUtility encode <パスワード>
```

ConfigMapの利用例：server-overrides.xml（3/4）

■ 次に、作成した xml ファイルからConfigMapを作成します。
- この例では、作成するConfigMapの名前を「ds-overrides-fix」としています。
- oc create configmapコマンドの--from-fileオプションは「 --from-file=(ConfigMap内のdata名)=(内容が格納されているローカル・ファイル)」の形式です。
- data名はHelm Chartが前提としている「server-overrides.xml」で固定です。
- 内容が格納されているローカル・ファイルのパスには、先ほど作成したxmlファイルを指定します。

```
oc create configmap ds-overrides-fix ¥
                    --from-file=server-overrides.xml=(作成したxmlファイルのパスと名前)
```

■ Helm Chartの変数ファイルを準備します。
- 作成したConfigMap内の xml ファイル(server-overrides.xml)を利用してWebSphere Libertyの構成定義がオーバーライドされるように、Helm Chartの変数ファイルで「images.serverOverridesConfigMapName」に作成したConfigMapの名前を指定します。

```
image:
  ......
  serverOverridesConfigMapName: "ds-overrides-fix"
```

■ 準備ができたので、Helm Chartを使ってデプロイします。

ConfigMapの利用例：server-overrides.xml（4/4）

■ 動作を確認します。

– WebSphere Libertyのログ(/logs/messages.log)を確認すると、ConfigMapに定義したserver-overrides.xmlが/config/configDropins/overridesに配置され、ロードされていることが確認できます。

– オーバーライドしたデータソース定義が使用されるようになります。

(messages.log抜粋)

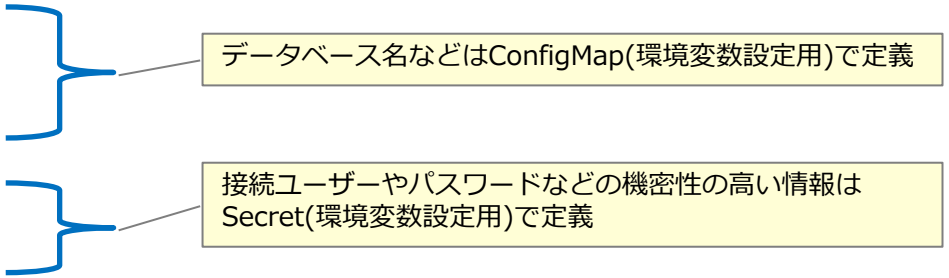
```
… … … … … … … … … … … … …
CWWKG0093A: Processing configuration drop-ins resource:
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults/keystore.xml
CWWKG0093A: Processing configuration drop-ins resource:
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/include-configmap.xml
CWWKG0028A: Processing included configuration resource: /etc/wlp/configmap/server.xml
CWWKG0093A: Processing configuration drop-ins resource:
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/server-overrides.xml
… … … … … … … … … … … … …
```

/config/は、/opt/ibm/wlp/usr/servers/defaultServerにシンボリックリンクされています。
/config/configDropins/overrides/server-overrides.xmlに、ConfigMapとして作成したserver.xmlフラグメントが反映されたことがわかります。

ConfigMapとSecretの利用例：server-overrides.xmlと環境変数（1/4）

- 前の例と同様に、server-overrides.xmlを含むConfigMapを作成しコンテナーにマウントすることで、WebSphere Libertyに定義されているデータソース定義をオーバーライドする例です。

- この例では、データソース定義に使用される以下の情報をConfigMapおよびSecretに格納し、環境変数を利用してWebSphere Libertyのコンテナーに設定します。
  - データベース名
  - ホスト名
  - ポート
  - 接続ユーザー
  - パスワード



- 前の例で示した、各要素に id が付与された server.xml を前提としています。

- 手順は次の通りです。

ConfigMapとSecretの利用例：server-overrides.xmlと環境変数（2/4）

- まず、データソース定義をオーバーライドするために、下図のような xml ファイルを作成します。
  - 作成するxmlファイルの名前は適当な名前で問題ありません。
  - 今回は、データベース名、ホスト名、ポート、接続ユーザー、および、パスワードを環境変数から取得するようになっています。

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
    <dataSource id="SampleDS" jndiName="jdbc/sample" jdbcDriverRef="SampleJD">
        <properties serverName="${env.DB_HOST}"
                    portNumber="${env.DB_PORT}"
                    databaseName="${env.DB_NAME}"
                    user="${env.DB_USER}"
                    password="${env.DB_PASSWORD}" />
    </dataSource>
</server>
```

- 作成した xml ファイルからserver-overrides.xml用のConfigMapを作成します。
  - この例では、作成するConfigMapの名前を「ds-overrides-var」としています。

```
oc create configmap ds-overrides-var ¥
                    --from-file=server-overrides.xml=(作成したxmlファイルのパスと名前)
```

ConfigMapとSecretの利用例：server-overrides.xmlと環境変数（3/4）

■ 次に、環境変数用のConfigMapとSecretを作成します。

– この例では、作成するConfigMapとSecretの名前を以下の様にしています。

ここでは例として「oc create」コマンドでリソースを直接作成していますが、
構成管理の観点からは、yamlファイルを作成し、「oc apply」する方が推奨されます。

```
oc create configmap ds-overrides-configmap ¥
  --from-literal=DB_HOST=mysql ¥
  --from-literal=DB_PORT="3306" ¥
  --from-literal=DB_NAME=sampledb
```

```
oc create secret generic ds-overrides-secret ¥
  --from-literal=DB_USER=user ¥
  --from-literal=DB_PASSWORD="{xor}Lz4sLCgwLTs="
```

• パスワードは、WebSphere Libertyが提供するsecurityUtility encodeコマンドでエンコードしたものを記載します。

– WebSphere Libertyのコンテナー・イメージを利用してコマンドを実行する方法が簡単です。

```
docker run --rm -it ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi /opt/ibm/wlp/bin/securityUtility encode <パスワード>
```

ConfigMapとSecretの利用例：server-overrides.xmlと環境変数（4/4）

- Helm Chartの変数ファイルを準備します。
  - 右記参照
  - 「images.extraEnvs」内にConfigMapおよびSecretから環境変数を設定する指定を追加します。
  - 先ほどと同様に、「images.serverOverridesConfigMapName」に作成したConfigMapの名前を指定します。
- 準備ができたら、Helm Chartを使ってデプロイします。
- 動作を確認します。
  - server-overrides.xmlが/config/configDropins/overridesにマウントされ、ConfigMapおよびSecretから環境変数が設定されていることが確認できます。
  - オーバーライドしたデータソース定義が使用されるようになります。

```
image:
  ......
  extraEnvs:
  - name: DB_HOST
    valueFrom:
      configMapKeyRef:
        name: ds-overrides-configmap
        key: DB_HOST
  - name: DB_PORT
    valueFrom:
      configMapKeyRef:
        name: ds-overrides-configmap
        key: DB_PORT
  - name: DB_NAME
    valueFrom:
      configMapKeyRef:
        name: ds-overrides-configmap
        key: DB_NAME
  - name: DB_USER
    valueFrom:
      secretKeyRef:
        name: ds-overrides-secret
        key: DB_USER
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: ds-overrides-secret
        key: DB_PASSWORD
  ......
  serverOverridesConfigMapName: "ds-overrides-var"
  ......
```

ConfigMap/Secretからvalueをとるのではなく、ここに直接valueを記載することも可能です。

Helm Chartが作成したConfigMapの利用例（1/3）

- Helm Chartが作成したConfigMapを変更することで、WebSphere Libertyの構成情報を変更する例です。
  - この例で変更した内容は、Helm Chartの変数ファイルを修正し「helm upgrade」を行っても保持されますが、 「helm uninstall」を行うと削除されます。
  - このため、恒久的な変更ではなく、一時的な変更に適した方法となります。
  - 変更できるのは、WebSphere Libertyの構成情報のみで、環境変数を変更することはできません。

- ここでは、WebSphere Libertyのトレース設定を一時的に変更する例を記載します。

- まず、リリース済みのWebSphere Libertyの情報を確認します。
  - 下記の例の場合は、infra-test-httpというリリース名のWebSphere Libertyがあり、 Helm Chartが作成したConfigMapの名前がinfra-test-http-ibm-websになっています。
    - ConfigMapの名前は、「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字になります。

```
> helm list
NAME              NAMESPACE       REVISION        STATUS          CHART                           APP VERSION
infra-test-http   wlp-sample      1               deployed        ibm-websphere-liberty-1.10.0    19.0.0.6

> oc get configmap
NAME                        DATA   AGE
infra-test-http-ibm-webs    2      24s
```

Helm Chartが作成したConfigMapの利用例（2/3）

- 次に、Helm Chartが作成したConfigMapを変更します。
  - 以下のコマンドを実行します。

```
> oc edit configmap infra-test-http-ibm-webs
```

  - 表示されたエディターでserver.xml内に以下のようなトレース指定を追加し、保管します。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  include-configmap.xml: |-
    <server>
      <include optional="true" location="/etc/wlp/configmap/server.xml"/>
      <include optional="true" location="/etc/wlp/configmap/cluster-ssl.xml"/>
    </server>
  server.xml: |-
    <server>
      <!-- Customize the running configuration. -->
      <logging  traceSpecification="com.ibm.ws.webcontainer*=all"/>
    </server>
kind: ConfigMap
.............
```

Helm Chartが作成したConfigMapの利用例（3/3）

■ 変更内容が稼働中のPodに反映され、トレース指定が有効になります。

```
> oc exec infra-test-http-ibm-webs-6c44d8d976-ln28z -it -- bash

bash-4.4$ cat /etc/wlp/configmap/server.xml
<server>
  <!-- Customize the running configuration. -->
  <logging  traceSpecification="com.ibm.ws.webcontainer*=all"/>
</server>bash-4.4$

bash-4.4$ tail /logs/messages.log
...... Starting server configuration update.
...... Processing configuration drop-ins resource: /opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults/keystore.xml
...... Processing configuration drop-ins resource: /opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/include-configmap.xml
...... Processing included configuration resource: /etc/wlp/configmap/server.xml
...... Processing configuration drop-ins resource: /opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/server-overrides.xml
...... The trace state has been changed. The new trace state is *=info:com.ibm.ws.webcontainer*=all.
...... The server configuration was successfully updated in 2.838 seconds.

bash-4.4$ ls -al /logs/
total 988
drwxrwxr-x. 1 default root   4096 Mar  6 02:40 .
drwxr-xr-x. 1 root    root   4096 Mar  6 02:26 ..
-rw-r-----. 1 default root  15929 Mar  6 02:41 http_access.log
-rw-r-----. 1 default root  15768 Mar  6 02:40 messages.log
-rw-r--r--. 1 default root   2114 Mar  6 00:22 messages_20.03.06_02.26.24.0.log
-rw-r-----. 1 default root 950713 Mar  6 02:41 trace.log
```

9. DBを使用したセッション・パーシスタンス

目次(9章)

- WebSphere Libertyのセッション・パーシスタンス機能
- DBを使用したセッション・パーシスタンスの構成例
  - 概要と前提
  - 手順
  - 確認

WebSphere 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

– WebSphere 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。

– WebSphere Libertyの障害時に、別のWebSphere 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

– WebSphere 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
- JCache
- WebSphere eXtreme Scale (WXS)

– 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。

■ 本章で、データベースをしたセッション・パーシスタンスの構成例を、次章で、JCacheを使用したセッション・パーシスタンスの構成例を説明します。

DBを使用したセッション・パーシスタンスの構成例：概要と前提

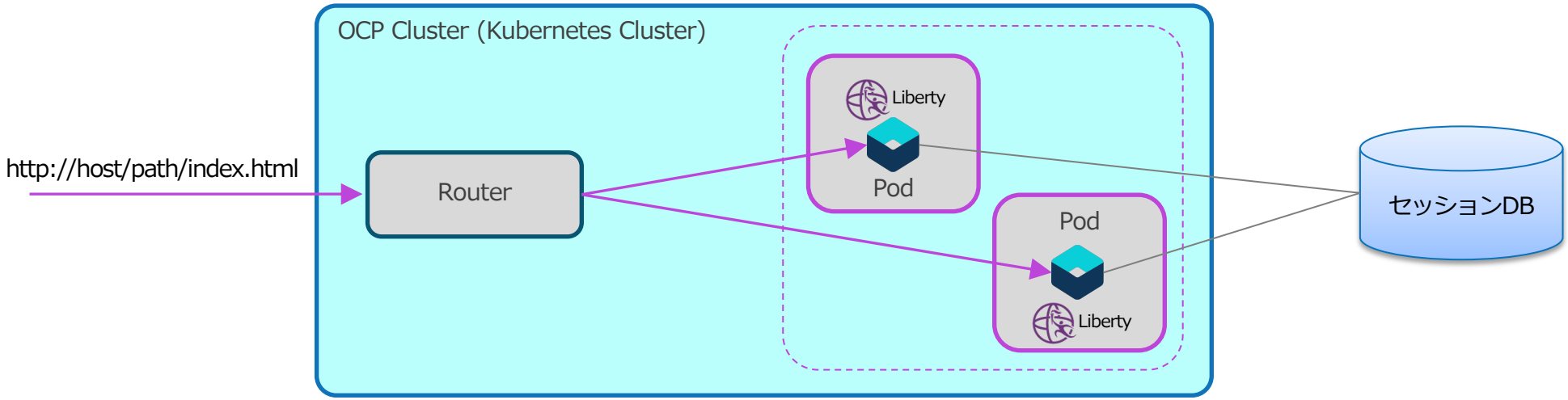
■ 概要

– OCP環境でも、従来環境と同様の構成定義を行うことで、DBを使用したセッション・パーシスタンスが実現可能です。
  • WebSphere Liberty Knowledge Center / データベースへの Liberty 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

– セッション・パーシスタンスが構成されていないWebSphere Libertyのイメージに対して、セッションDBの構成定義を記述したserver.xmlフラグメントをConfigMapを使用して注入する方法で、セッション・パーシスタンスを構成します。

■ 前提

– WebSphere Libertyのイメージに「sessionDatabase-1.0」フィーチャーが含まれている必要があります。
– WebSphere LibertyのイメージにJDBCドライバーが含まれている必要があります。



DBを使用したセッション・パーシスタンスの構成例：手順(1/2)

■ ConfigMapの準備

– 「server-overrides.xml」というファイル名で、右図のようなserver.xmlフラグメントを作成します。
- DB(この例ではmysession)をあらかじめ作成し、接続ユーザー(この例ではuser)から書き込みが可能なように設定しておく必要があります。
- 接続情報を環境変数化することも可能ですが、ここでは環境変数化しない例を示します。
- パスワードは「securityUtility encode」コマンドでエンコードしたものを記載します。

– 作成したserver.xmlフラグメントからConfigMapを作成します。下記のocコマンドを実行します。

```
oc create configmap session-db-configmap ¥
  --from-file=server-overrides.xml=./server-overrides.xml
```

```
<server>

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

    <jdbcDriver id="MySQLDriver">
        <library>
            <fileset dir="${server.config.dir}/resources/mysql"
                     includes="mysql-*.jar" />
        </library>
    </jdbcDriver>

    <dataSource id="SessionDS"
                jndiName="jdbc/mydb"
                jdbcDriverRef="MySQLDriver">
        <properties serverName="mysql"
                    portNumber="3306"
                    databaseName="mysession"
                    user="user"
                    password="{xor}MzY9Oi0rJg==" />
    </dataSource>

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

    <httpSession storageRef="SessionDB"/>

</server>
```

DBを使用したセッション・パーシスタンスの構成例：手順(2/2)

■ デプロイ

– Helm Chartの変数ファイルに以下の指定を追加します。

- image.serverOverridesConfigMapNameの指定を追加(変更)します。
- その他の指定は、セッション・パーシスタンスを構成していない場合と同じです。

```
image:
    ......
    serverOverridesConfigMapName: "session-db-configmap"
    ......
```

– 修正した変数ファイルを使用して、Helm installまたはupgradeを実行します。

- helm install/upgradeの実行方法は、セッション・パーシスタンスを構成していない場合と同じです。

DBを使用したセッション・パーシスタンスの構成例：確認

■ セッション・パーシスタンスの稼働状況の確認

- WebSphere Libertyが起動すると、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 mysession
Database changed
mysql> show tables;
+---------------------+
| Tables_in_mysession |
+---------------------+
| sessions            |
+---------------------+
1 row in set (0.03 sec)

mysql> select id, lastaccess, length(small), length(medium), length(large) from sessions;
+-------------------------+---------------+---------------+----------------+---------------+
| id                      | lastaccess    | length(small) | length(medium) | length(large) |
+-------------------------+---------------+---------------+----------------+---------------+
| default_host/InfraTest  | 1582258542749 |          NULL |           NULL |          NULL |
| xWxrCoxX7sMcM6Ax_tFBaFo | 1582258492725 |           169 |           NULL |          NULL |
+-------------------------+---------------+---------------+----------------+---------------+
2 rows in set (0.03 sec)
```

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

10. JCacheを使用した セッション・パーシスタンス

目次(10章)

- JCacheを使用したセッション・パーシスタンス
- Hazelcastを使用する場合
  - トポロジー構成
  - ディスカバリー方法
  - 設定例
- (補足) 事前定義server.xmlフラグメントの利用
  - 概要
  - 設定例

JCacheを使用したセッション・パーシスタンス

- WebSphere Liberty は JCache を使用したセッション・パーシスタンス機能を提供しています。
  - 参考情報：
    - WebSphere Liberty Knowledge Center / JCache を使用した Liberty 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_jcache.html
    - WebSphere Liberty Knowledge Center / JCache Session Persistence 1.0
      - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.liberty.autogen.nd.doc/ae/rwlp_feature_sessionCache-1.0.html
    - Open Liberty Blog / JCache session persistence
      - 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、Peer-to-Peer構成のセッション・パーシスタンスの設定例を紹介します。
  - 「Hazelcast」は上記URLで紹介されているJCache実装の１つです。

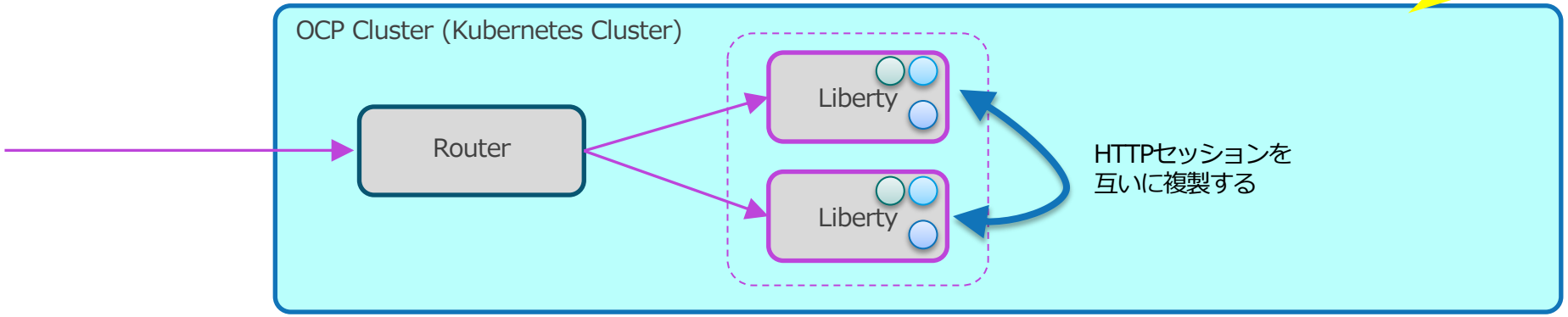
Hazelcastを使用する場合：トポロジー構成

■ 以下の2つのトポロジー構成が可能です。

– Peer-to-Peer構成

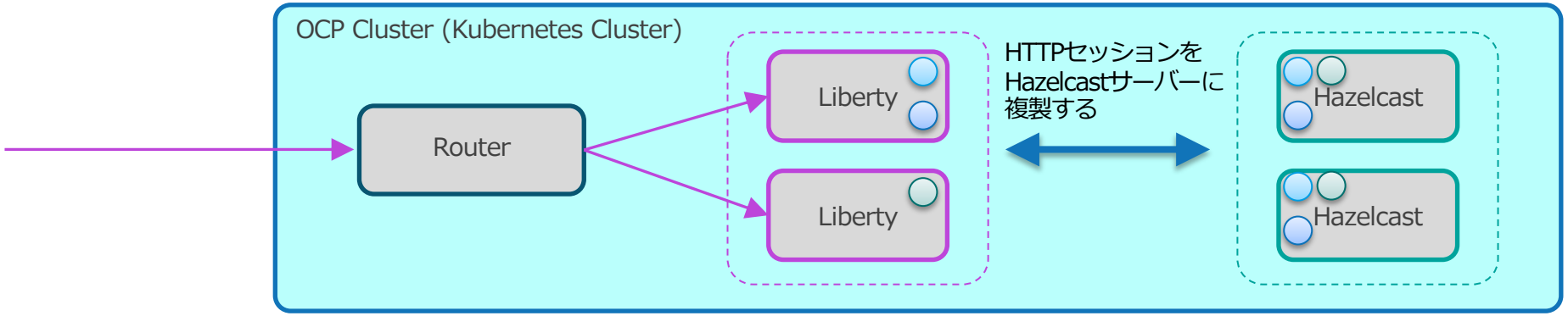
• セッションをWebSphere Libertyサーバー間で互いにレプリケーションします。

この資料では、
Peer to Peerの構成を
説明します。



– Client-Server構成

• セッションをHazelcastサーバーにレプリケーションします。



Hazelcastを使用する場合：ディスカバリー方法

- レプリケーション先となるHazelcastメンバーは以下の方法でディスカバリーします。
  - Kubernetes API
    - サービスを指定し、Kubernetes API経由でメンバーをディスカバリーします。
      - この方法を使用するにはサービス定義が必要になります。
      - さらに、Podのサービス・アカウントに対して、endpointsのget権限を付与する必要があります。
    - 新規にHazelcast用のサービス(5701ポート)を定義するか、WebSphere Libertyのサービス定義を流用し、ポート番号をHazelcast用にオーバーライドして利用します。
  - DNS Lookup
    - Headless ServiceをKubernetes DNSでlookupすることでメンバーをディスカバリーします。
      - この方法を使用する場合は、Headless Serviceが必要になります。
    - 新規にHazelcast用のHeadless Service(5701ポート)を定義するか、WebSphere LibertyのPodをStatefulSetとしてデプロイすると、StatefulSetの前提としてHeadless Serviceが定義されるので、それを利用します。
      - Helm Chartの変数ファイルで「logs.persistLogs」または「logs.persistTransactionLogs」に「true」を指定すると、StatefulSetとしてデプロイされます。(第6章「ログとダンプの永続化と転送」参照)

- 参考情報
  - レプリケーション先のHazelcastメンバーのディスカバリーに関しては、以下の情報を参考にしてください。
    - GitHub / hazelcast/hazelcast-kubernetes / Hazelcast Discovery Plugin for Kubernetes
      - https://github.com/hazelcast/hazelcast-kubernetes
    - Maven Repository / com.hazelcast » hazelcast-kubernetes
      - https://mvnrepository.com/artifact/com.hazelcast/hazelcast-kubernetes

Hazelcastを使用する場合：設定例（1/5）

- Hazelcastを使用したPeer-to-Peer構成のセッション・パーシスタンスの設定例を紹介します。
  手順は、以下の通りです。

- まず、server.xmlフラグメント(hazelcast-sessioncache.xml)を作成します。
  - WebSphere Libertyの設定を記述する方法はいくつかありますが、ここでは、以下のような内容を記述したserver.xmlフラグメント(hazelcast-sessioncache.xml)を作成します。

```
<server>

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

  <httpSessionCache libraryRef="HazelcastLib">
    <properties hazelcast.config.location="file:${shared.config.dir}/hazelcast/hazelcast.xml"/>
  </httpSessionCache>

  <library id="HazelcastLib">
    <fileset dir="${shared.resource.dir}/hazelcast/" includes="hazelcast-all-*.jar"/>
  </library>

</server>
```

「sessionCache-1.0」フィーチャーの追加

Hazelcastの構成ファイル(hazelcast.xml)のURIを指定(次ページで作成)

hazelcastのjarが必要

Hazelcastを使用する場合：設定例（2/5）

■ 次に、Hazelcastの構成ファイル(hazelcast.xml)を作成します。

- メンバーのディスカバリー方法(「DNS Lookup」または「Kubernetes API」)によって、discovery-strategyのpropertyに指定する内容が変わります。
  - <service-name>は、「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字、または、変数ファイルの「service.name」で指定した名前
  - <service-dns>は、「(service-name)-sts.(namespace). svc.cluster.local」の形式

```
<?xml version="1.0" encoding="UTF-8"?>
<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-3.12.xsd">
  <network>
    <join>
      <multicast enabled="false"/>
      <kubernetes enabled="true">

      </kubernetes>
    </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。

メンバーのディスカバリー方法によって、この部分に定義する内容が変わります。(下参照)

(「Kubernetes API」でディスカバリーする場合)

```
<namespace>wlp-sample</namespace>
<service-name>hazelcast-kubeapi-ibm-we</service-name>
<service-port>5701</service-port>
```

(「DNS Lookup」でディスカバリーする場合)

```
<service-dns>hazelcast-dnslkup-ibm-we-sts.wlp-sample.svc.cluster.local</service-dns>
<service-dns-timeout>5</service-dns-timeout>
```

Hazelcastを使用する場合：設定例（3/5）

- Dockerfileに以下の内容を追加します。
  - Hazelcast のイメージ「hazelcast/hazelcast」から、必要なライブラリーを所定のディレクトリーへコピー
  - server.xmlフラグメント(hazelcast-sessioncache.xml)とHazelcastの構成ファイル(hazelcast.xml)を所定のディレクトリーへコピー
  - Javaのシステム・プロパティー hazelcast.jcache.provider.type に server を設定

(Dockerfileに追加する内容の例)

```
COPY --from=hazelcast/hazelcast:3.12.6 ¥
     --chown=1001:0 /opt/hazelcast/lib/hazelcast-all-*.jar ¥
                                              /opt/ibm/wlp/usr/shared/resources/hazelcast/
COPY --chown=1001:0 hazelcast-sessioncache.xml /config/configDropins/overrides/
COPY --chown=1001:0 hazelcast.xml              /opt/ibm/wlp/usr/shared/config/hazelcast/
ENV JAVA_TOOL_OPTIONS="-Dhazelcast.jcache.provider.type=server ${JAVA_TOOL_OPTIONS}"
```

- イメージをビルドし、レジストリーへpushします。

Hazelcastを使用する場合：設定例（4/5）

- Kubernetes APIでディスカバリーする場合は、サービス・アカウントに必要なロールを付与します。
  - RBACを無効化している場合は、 defaultサービス・アカウントに、endpointsのget権限を付与します。
    - RBACを無効化している場合＝Helm Chart の変数ファイルで「rbac.install: false」を指定している場合

```
> oc adm policy add-role-to-user view -z default
clusterrole.rbac.authorization.k8s.io/view added: "default"

> oc adm policy who-can get endpoints
（出力省略：endpointsのget権限を持つユーザー/グループがリストされます。）
```

  - RBACを有効化している場合は、 Helm Chart が作成するサービス・アカウントに必要な権限が付与されるため、上記の操作は不要です。
    - RBACを有効化している場合＝Helm Chart の変数ファイルで「rbac.install: true（デフォルト）」を指定している場合

- サービス・アカウントにSCC（ibm-websphere-liberty-scc）を付与します。（通常と同じ）
  - RBACを無効化している場合は、 defaultサービス・アカウントに付与します。

```
> oc adm policy add-scc-to-user ibm-websphere-liberty-scc -z default
```

  - RBACを有効化している場合は、 Helm Chart が作成するサービス・アカウントに付与します。
    - サービス・アカウントの名前は「(Helmのリリース名)-ibm-websphere-liberty」の先頭24文字です。

```
> oc adm policy add-scc-to-user ibm-websphere-liberty-scc -z hazelcast-dnslkup-ibm-we
```

- Helm Chart を使用してデプロイします。
  - Helm Chart の変数ファイルには、Hazelcastを使用したセッション・パーシスタンスに固有の設定はありません。
  - 但し、ディスカバリー方法として「DNS Lookup」を使用する場合は、ログを永続化し、StatefulSetとしてデプロイする必要があります。

Hazelcastを使用する場合：設定例（5/5）

■ 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。

```
... ... ... ...
I [172.30.150.231]:5701 [dev] [3.12.6] Kubernetes Discovery properties: { service-dns: hazelcast-sample-ibm-web-sts, service-dns-timeout: 5, service-name: null, service-port: 0,
service-label: null, service-label-value: true, namespace: wlp-sample, pod-label: null, pod-label-value: null, resolve-not-ready-addresses: false, use-node-name-as-external-address:
false, kubernetes-api-retries: 3, kubernetes-master: https://kubernetes.default.svc}
I [172.30.150.231]:5701 [dev] [3.12.6] Kubernetes Discovery activated with mode: DNS_LOOKUP
I [172.30.150.231]:5701 [dev] [3.12.6] Activating Discovery SPI Joiner
... ... ... ...
I [172.30.150.231]:5701 [dev] [3.12.6]

Members {size:2, ver:2} [
        Member [172.30.150.193]:5701 - b4d84f8f-4da2-4b0f-9ffe-bf53693a03b2
        Member [172.30.150.206]:5701 - 83d7c663-7388-4343-ba57-296aef5d64d0 this
]
... ... ... ...
```

（「DNS Lookup」の場合の/logs/messages.logの出力例）

```
... ... ... ...
I [172.30.150.249]:5701 [dev] [3.12.6] Kubernetes Discovery properties: { service-dns: null, service-dns-timeout: 5, service-name: hazelcast-kubeapi-ibm-we, service-port: 5701, service-
label: null, service-label-value: true, namespace: wlp-sample, pod-label: null, pod-label-value: null, resolve-not-ready-addresses: false, use-node-name-as-external-address: false,
kubernetes-api-retries: 3, kubernetes-master: https://kubernetes.default.svc}
I [172.30.150.249]:5701 [dev] [3.12.6] Kubernetes Discovery activated with mode: KUBERNETES_API
I [172.30.150.249]:5701 [dev] [3.12.6] Activating Discovery SPI Joiner
... ... ... ...
[INFO    ] [10.1.47.6]:5701 [dev] [3.12.1]

Members {size:2, ver:2} [
        Member [172.30.150.248]:5701 - 82d68986-4b72-42f0-9385-95097d28be4b
        Member [172.30.150.249]:5701 - 8d52d606-f062-42ef-9490-075b3a25c29b this
]
... ... ... ...
```

（「Kubernetes API」の場合の/logs/messages.logの出力例）

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

(補足) 事前定義server.xmlフラグメントの利用：概要

- IBM提供のWebSphere Libertyのイメージには、Hazelcastを使用したセッション・パーシスタンスを簡単に利用するための事前定義されたserver.xmlフラグメント(snippet)が予め用意されており、イメージのビルド時に有効にすることができます。
  - 参考：GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Session Caching
    - https://github.com/WASdev/ci.docker#session-caching

  - 前述の方法（手動で構成する方法）では、以下のファイルを準備しましたが、この方法を使用する場合は準備する必要がありません。
    - server.xmlフラグメント(hazelcast-sessioncache.xml)
    - Hazelcastの構成ファイル(hazelcast.xml)

- メンバーのディスカバリー方法は、Kubernetes APIを使用したディスカバリーになります。

  - Peer-to-Peer構成の場合、レプリケーション対象が指定されていないため、ネームスペース内の全てのPodがレプリケーション対象となり、接続を試みます。 WebSphere Libertyが稼働するネームスペースにレプリケーション対象となるWebSphere Liberty以外のPodが存在する場合は、この方法/構成を使用しないでください。

(補足) 事前定義server.xmlフラグメントの利用：設定例（1/2）

- 事前定義server.xmlフラグメントを使用したPeer-to-Peer構成のセッション・パーシスタンスの設定例を紹介します。手順は、以下の通りです。

- まず、Dockerfileに以下の内容を追加します。

(Dockerfileに追加する内容の例)

```
### Hazelcast Session Caching ###
# Copy the Hazelcast libraries from the Hazelcast Docker image
COPY --from=hazelcast/hazelcast --chown=1001:0 /opt/hazelcast/lib/*.jar /opt/ibm/wlp/usr/shared/resources/hazelcast/

# Instruct configure.sh to copy the client topology hazelcast.xml
# ARG HZ_SESSION_CACHE=client

# Default setting for the verbose option
ARG VERBOSE=false

# Instruct configure.sh to copy the embedded topology Hazelcast.xml and set the required system property
ARG HZ_SESSION_CACHE=embedded
ENV JAVA_TOOL_OPTIONS="-Dhazelcast.jcache.provider.type=server ${JAVA_TOOL_OPTIONS}"

## This script will add the requested XML snippets and grow image to be fit-for-purpose
RUN configure.sh
```

Hazelcastのライブラリを「hazelcast/hazelcast」のイメージからコピーします。

Hazelcastを利用するための変数「HZ_SESSION_CACHE」を宣言し(ビルド時のみ有効)、値としてトポロジー(この例では「embedded」)を指定します。

configure.shでは、変数宣言に応じて事前定義server.xmlフラグメント(snippet)が/config/configDropins/overrides/にコピーされます。

- イメージをビルドし、レジストリーへpushします。

(補足) 事前定義server.xmlフラグメントの利用：設定例（2/2）

- サービス・アカウントに必要なロールを付与します。
  - 「Hazelcastを使用する場合：設定例」と同様です。
- サービス・アカウントにSCC（ibm-websphere-liberty-scc）を付与します。（通常と同じ）
  - 「Hazelcastを使用する場合：設定例」と同様です。
- Helm Chart を使用してデプロイします。
  - Helm Chart の変数ファイルには、Hazelcastを使用したセッション・パーシスタンスに固有の設定はありません。

11. オートスケーリングとリソース調整

目次(11章)

オートスケーリングとリソース調整の概要

- WebSphere LibertyのHelm Chartでは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。
- また「Horizontal Pod Autoscaler (HPA)」を使用したオートスケーリングの定義も可能です。
- 「Horizontal Pod Autoscaler (HPA)」は以下の条件の場合のみ使用することができます。
  - リソース制約を有効化している場合(「resources.constraints.enabled」が「true」)
  - PodがDeploymentとしてデプロイされている場合(WebSphere Libertyのログを永続化しない場合

WebSphere LibertyのHelm Chartのパラメーター：オートスケーリング関連

■ 「オートスケーリング」に関連するパラメーター

Qualifier	Parameter	デフォルト値	説明
replicaCount	-	1	オートスケーリングが無効の場合の、Podのレプリカ数を指定します。 (オートスケーリング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。)
autoscaling	enabled	false	オートスケーリングを有効にするか否かを指定します。
	minReplicas	1	オートスケーリングを有効にした場合のPodのレプリカ数の最小値を指定します。
	maxReplicas	10	オートスケーリングを有効にした場合のPodのレプリカ数の最大値を指定します。
	targetCPUUtilizationPercentage	50	CPU使用率の目標値(*)をパーセントで指定します。(1-100の間の整数を指定します。) (*)CPUの要求量に対する割合で、全Podの平均値です。

■ 「リソース調整」に関するパラメーター

Qualifier	Parameter	デフォルト値	説明
resources	constraints.enabled	false	リソース制約を有効にするか否かを指定します。
	requests.cpu	500m	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。 CPUの指定は1vCPU(仮想CPU)を1000millicores(m)とする単位で指定します。 メモリは1G=1024Miとして指定します。
	requests.memory	512Mi	
	limits.cpu	500m	CPUとメモリーの上限を指定します。 指定単位は上記resources.requests.cpu、resources.requests.memoryと同じです。
	limits.memory	512Mi	

GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

オートスケーリングの動作例（1/2）

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

```
# oc get hpa
NAME                   REFERENCE                      TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
wlp-re01-liberty-hpa   Deployment/wlp-re01-liberty    <unknown>/50%   1         10        1          30s

# oc get hpa wlp-re01-liberty-hpa -w
NAME                        REFERENCE                             TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
....
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           2%/50%         1         10        1          22m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           2%/50%         1         10        1          22m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        1          23m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        2          23m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           99%/50%        1         10        2          24m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           52%/50%        1         10        2          25m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           51%/50%        1         10        2          26m
....
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        2          48m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        2          48m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        1          48m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        1          49m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        1          51m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        2          51m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        2          52m
....
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           52%/50%        1         10        2          55m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           100%/50%       1         10        2          56m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           93%/50%        1         10        2          56m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           93%/50%        1         10        4          56m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           79%/50%        1         10        4          57m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           27%/50%        1         10        4          58m
....
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           26%/50%        1         10        4          63m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           34%/50%        1         10        3          63m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           34%/50%        1         10        3          64m
....
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        3          99m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        3          100m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        1          100m
wlp-re01-liberty-hpa        Deployment/wlp-re01-liberty           1%/50%         1         10        1          101m
```

(例)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。

TARGETS欄の負荷が高くなると、REPLICAS欄のレプリカ数が自動で増えていることが確認できます。

TARGETS欄の負荷が低くなると、REPLICAS欄のレプリカ数が自動で減っていることが確認できます。

オートスケーリングの動作例（2/2）

- HPAの状態は「oc describe <HPA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。

```
# oc describe hpa wlp-re01-liberty-hpa
Name:                                                  wlp-re01-liberty-hpa
Namespace:                                             wlp-sample
Labels:                                                app=wlp-re01-liberty
                                                       chart=ibm-websphere-liberty-1.10.0
                                                       heritage=Helm
                                                       release=wlp-re01
Annotations:                                           <none>
CreationTimestamp:                                     Fri, 06 Mar 2020 14:32:35 +0900
Reference:                                             Deployment/wlp-re01-liberty
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  1% (7m) / 50%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  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
  ----     ------               ----             ----                       -------
  ......   ....
  Normal   SuccessfulRescale    10m              horizontal-pod-autoscaler  New size: 3; reason: cpu resource utilization (percentage of request) above target
  Normal   SuccessfulRescale    97s              horizontal-pod-autoscaler  New size: 2; reason: All metrics below target
  Normal   SuccessfulRescale    21s              horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
```

(例)負荷を増減させた後に状態を確認したところ

「Conditions」欄と「Events」欄で、リソース負荷の増減によってHPAがPodを増減させたことがメッセージから確認できます。

12. 静的コンテンツの扱い

目次(12章)

- 静的コンテンツを扱う方法
- WebSphere LibertyのFile Serving機能の拡張文書ルートの設定方法
- 静的コンテンツを永続領域に置き拡張文書ルートで指定する場合の設定例

静的コンテンツを扱う方法

■ 静的コンテンツを扱う方法として以下の方法があります。

– WebSphere LibertyのFile Serving機能(デフォルトで有効)を使用する方法 （本章で説明）

- (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
  - 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、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。

– WebSphere Libertyの代わりに、IBM HTTP ServerなどのWebサーバーを利用する方法 （次章で説明）

- 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WebSphere 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

WebSphere LibertyのFile Serving機能の拡張文書ルートの設定方法

■ WebSphere LibertyのFile Serving機能の拡張文書ルート機能

– File Serving機能の拡張文書ルート機能はWebSphere Application Server が従来から提供している機能です。

- WebSphere 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。
  - WebSphere Application Server traditional Knowledge Center / ファイル・サービス
  - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_9.0.5/com.ibm.websphere.nd.multiplatform.doc/ae/cweb_flserv.html

– 拡張文書ルートは以下のいずれかの方法で指定することができます。

- 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>
```

- server.xml内 webApplication タグ内の web-ext タグで指定

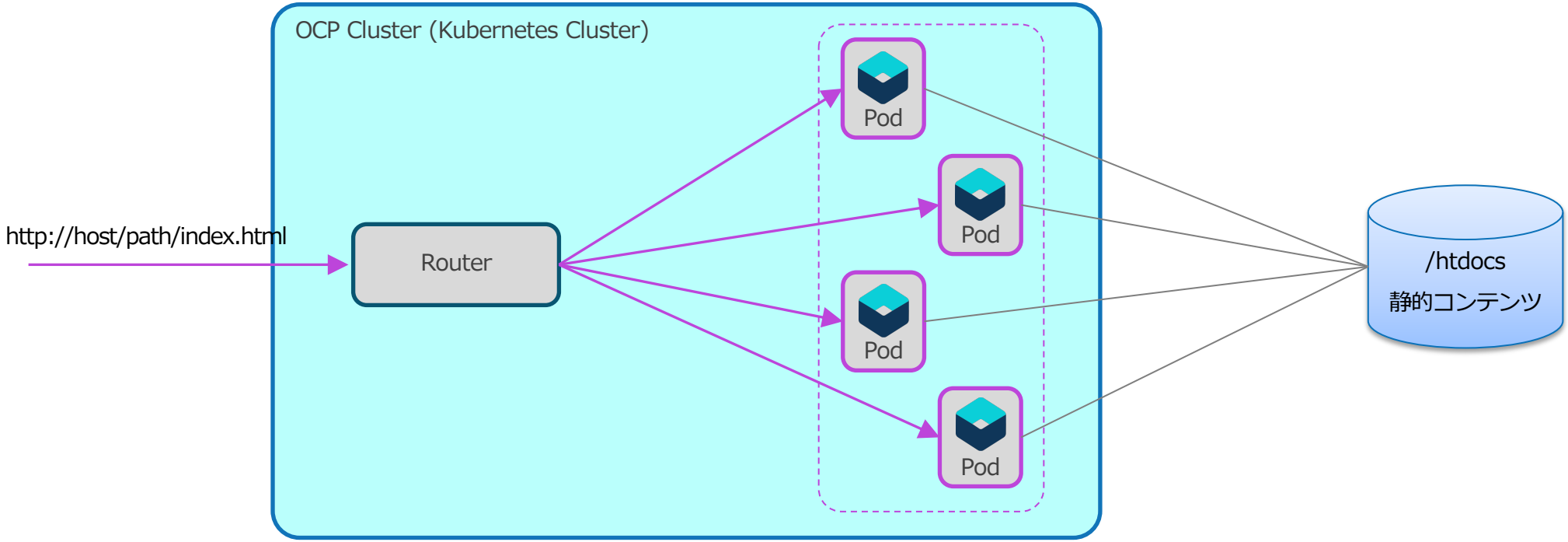
```
 ........
 <webApplication name="infra-test" location="${app.location}">
    <web-ext enable-file-serving="true">
      <file-serving-attribute name="extendedDocumentRoot" value="/htdocs"/>
    </web-ext>
 </webApplication>
 ........
```

静的コンテンツを永続領域に置き拡張文書ルートで指定する場合の設定例

■ 設定概要

– 静的コンテンツ用の永続領域(Persistent Volume)を定義し、WebSphere LibertyのFile Serving機能の拡張文書ルートとして前述の「WebSphere LibertyのFile Serving機能の拡張文書ルートの設定方法」で定義した/htdocsにマウントする例を示します。

• 永続領域に配置された静的コンテンツがWebSphere LibertyのFile Serving機能によって配信されるようになります。



静的コンテンツを永続領域に置き拡張文書ルートで指定する場合の設定例

■ 設定方法(事前準備)

– 静的コンテンツ用の「PersistentVolume」(PV)と「PersistentVolumeClaim」(PVC)を作成します。

- 例えば、以下のようなyamlファイルを作成し、「oc apply –f <マニフェストファイル名>」コマンドでPVとPVCを作成します。
- 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。
- 下記の例ではstorageClassName/accessModes/capacity.storageによって、PVとPVCが紐づけられます。

(例)PVのマニフェストファイルの例(yaml)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: htdocs-pv
spec:
  storageClassName: "htdocs-sc"
  accessModes:
  - ReadOnlyMany
  capacity:
    storage: 2Gi
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /nfs/htdocs-pv
    server: 10.20.30.40
```

(例)PVCのマニフェストファイルの例(yaml)

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: htdocs-pvc
spec:
  storageClassName: "htdocs-sc"
  accessModes: ["ReadOnlyMany"]
  resources:
    requests:
      storage: "2Gi"
```

静的コンテンツを永続領域に置き拡張文書ルートで指定する場合の設定例

■ 設定方法(Helm Chartの変数ファイル)

– Helm Chartの変数ファイルに下記の設定を追記します。

- 「image.extraVolumeMounts」に、Podにマウントするボリュームとマウントパスを指定します。
- 「pod.extraVolumes」に、そのボリュームの情報(先程指定したPVC)を指定します。

(例)Helm Chartの変数ファイルの設定例(yaml)

```
......
image:
  repository: image-registry.openshift-image-registry.svc:5000/wlp-sample/resource-eater
  tag: 19.0.0.12-full-java8-ibmjava-ubi
  pullPolicy: IfNotPresent
  license: "accept"

  ......
  extraVolumeMounts:
    - name: htdocs-vol
      mountPath: /htdocs
      readOnly: true

pod:
  extraVolumes:
    - name: htdocs-vol
      persistentVolumeClaim:
        claimName: htdocs-pvc
......
```

PVCをマウントするpathを指定します

使用するPVCを指定します。

WebSphere LibertyのHelm Chart v1.10.0 では、image.extraVolumeMountsの展開に問題があり、 変数ファイルでimage.extraVolumeMountsを指定した場合にエラーが発生することがありますが、Helm Chartを修正することで対応できます。修正方法は、第5章「Helm Chart のカスタマイズ」を参照してください。

13. IBM HTTP Server (IHS) の利用

目次(13章)

コンテナー環境でIBM HTTP Server (IHS)を使用する場合の参考情報

- IHSの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、WebサーバーPluginを含むIHSのイメージが公開されています。
  - DockerHub / ibmcom/ibm-http-server(Official Images)
    - https://hub.docker.com/r/ibmcom/ibm-http-server/
  - GitHub / WASdev/ci.docker.ibm-http-server / Dockerfiles for IBM HTTP Server
    - https://github.com/WASdev/ci.docker.ibm-http-server
  - GitHub / WASdev/ci.docker.ibm-http-server / Building an IBM HTTP Server ILAN image from binaries
    - https://github.com/WASdev/ci.docker.ibm-http-server/tree/master/ilan

- 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WebサーバーPluginのインストールが完了します。
  - IHS Knowledge Center / Installing IBM HTTP Server from an archive
    - https://www.ibm.com/support/knowledgecenter/en/SSEQTJ_9.0.5/com.ibm.websphere.ihs.doc/ihs/tihs_archive_intall.html

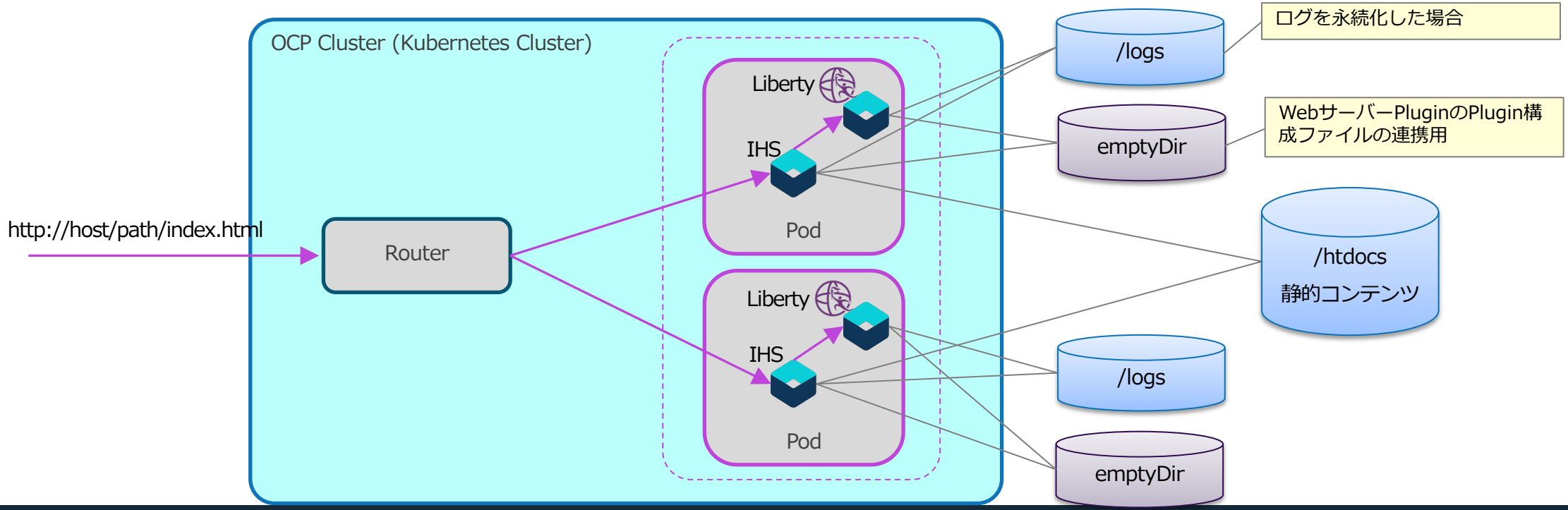
- IHSに関連するその他の情報
  - IHS Knowledge Center / Configuring IBM HTTP Server for Liberty
    - https://www.ibm.com/support/knowledgecenter/en/SSEQTJ_9.0.5/com.ibm.websphere.ihs.doc/ihs/tihs_install_config_liberty.html
  - IBM Support / PI77874: PLUGIN OFFLOAD/ONLOAD FOR SSL
    - http://www-01.ibm.com/support/docview.wss?uid=swg1PI77874

- これらの情報を参考にし、本章では、IHSのアーカイブからインストールする方法で独自のIHSのイメージを作成する方法を以降に記載します。

(補足) 本章の内容は、OCP環境でのIHSの利用をお勧めするものではありません。
既存環境からの移行等による利用を想定して、構成例や設定例を参考としてご紹介しています。

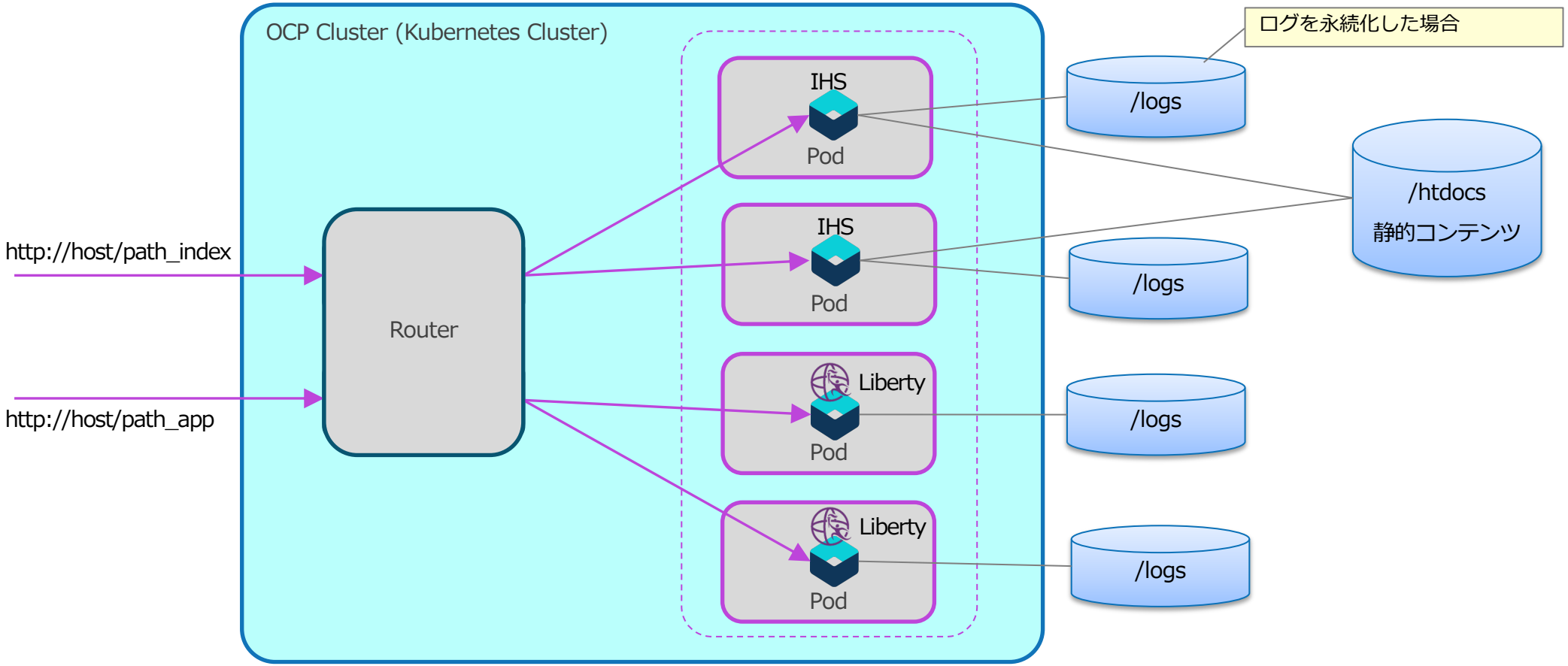
WebSphere LibertyとIHSの構成例 (IHSを「前段」に配置する構成)

- 1つのPod内でIHSのコンテナーとWebSphere Libertyのコンテナーを稼動させWebサーバーPluginで連携させます。
  - IHSとWebSphere 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、WebSphere 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。
    - WebサーバーPluginのPlugin構成ファイルは、Pod内で「emptyDir」(非永続領域)を共有することで、WebSphere LibertyのコンテナーからIHSのコンテナーへ連携します。
    - WebサーバーPluginを経由するのでServerIOTimeoutなどのWebサーバーPlugin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、ログ出力用の永続領域(/logs)をPod内で共有できます。



WebSphere LibertyとIHSの構成例 (IHSを「横並び」に配置する構成)

- IHSのPodとWebSphere LibertyのPodをそれぞれ別に作成し稼動させます。
  - IHSでは静的コンテンツを処理し、WebSphere Libertyが動的コンテンツを処理します。
  - IHSとWebSphere Liberty間でWebサーバーPluginを利用した連携はありません。
  - ログを永続化させる場合は、ログ出力用の永続領域(/logs)を、各Podに紐づけます。



IHSのイメージの作成方法

- IHSのイメージを作成するDockerfileは以下のような定義が必要になります。
  - OSの更新(dnf update)と、前提となるperlおよびunzipのインストール(dnf install –y perl unzip)
  - defaultユーザー(ID:1001)の追加
    - セキュリティの観点およびWebSphere Libertyのイメージに記載された実行ユーザーと合わせるため、defaultユーザー(ID:1001)を追加します。
  - IHSのアーカイブからIHSをインストール
    - 前述の「IHSのアーカイブからのインストールに関する情報」に従い、IHSのアーカイブからIHSをインストールします。
    - IHSのアーカイブ・ファイルを使用すると、ほぼ解凍するだけでIHSとWebサーバーPluginのインストールも行われます。
  - ディレクトリーの準備
    - WebSphere Libertyのイメージ内に定義されているディレクトリーに合わせてシンボリックリンクを作成します。
      - /htdocs　コンテンツ用　リンク先： /opt/ibm/IHS/htdocs
      - /logs　　ログ出力用　　リンク先： /opt/ibm/IHS/logs
      - /config　構成用　　　　リンク先： /opt/ibm/IHS/conf
    - Plugin構成ファイル(plugin-cfg.xml)用のディレクトリーを作成します。(WebSphere Libertyの前段に配置する時に必要)
      - /output/logs/state
  - IHS起動スクリプトの追加
    - 前述の「コンテナー環境でIBM HTTP Server (IHS)を使用する場合の参考情報」のリンク先で公開されている起動スクリプト(ihsstart.sh)をカスタマイズして追加します。（カスタマイズ例は後述参照）
  - IHS起動スクリプトの指定
    - コンテナー実行時に起動するスクリプトとしてIHS起動スクリプト(ihsstart.sh)を指定します。

  - 必要に応じて以下の操作も追加する必要があります。
    - カスタマイズしたIHSの設定ファイル(httpd.conf)を構成用のディレクトリー(/config)にコピー
    - (静的コンテンツをIHSのイメージ内に含める場合)静的コンテンツをコンテンツ用のディレクトリー(/htdocs)にコピー

IHSのイメージの作成方法：Dockerfileの例 (1/2)

■ Dockerfileの例

```
FROM registry.access.redhat.com/ubi8/ubi:latest

# Update, Install unzip, Add user 1001
RUN  dnf update  -y ¥
  && dnf install -y perl unzip ¥
  && dnf clean all  ¥
  && useradd -u 1001 -r -g 0 -s /usr/sbin/nologin default

# Install IHS (unzip IHS zip-file)
ARG  ihs_zip_file=9.0.5-WS-IHS-ARCHIVE-linux-x86_64-FP002.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  ln -s /opt/ibm/IHS/htdocs /htdocs ¥
  && ln -s /opt/ibm/IHS/logs   /logs   ¥
  && ln -s /opt/ibm/IHS/conf   /config ¥
  && mkdir -p /output/logs/state
```

(次ページへ続く)

UBI 8 をベースに作成

OSの更新、perlとunzipのインストール

defaultユーザー(ID:1001)の追加

予めダウンロードしておいた、IHSのアーカイブ・ファイルのコピー

コピーしたIHSのアーカイブ・ファイルを展開(unzip)して、
IHSとWebサーバーPluginをインストール

postinstall.shの実行

シンボリックリンクの作成
(コンテンツ用、ログ出力用、構成用のディレクトリー)

Plugin構成ファイル(plugin-cfg.xml)用のディレクトリーを作成
(WebSphere Libertyの前段に配置する時に必要)

IHSのイメージの作成方法：Dockerfileの例 (2/2)

■ Dockerfileの例（続き）

```
(続き)

# Modify httpd.conf
RUN  sed -i.org ¥
         -e "s/Listen 80/Listen 8080/" ¥
         -e "s@/opt/ibm/IHS/plugin/config/webserver1/plugin-cfg.xml@/output/logs/state/plugin-cfg.xml@" ¥
         -e "s@plugin/config/webserver1/plugin-cfg.xml@/output/logs/state/plugin-cfg.xml@" ¥
         /config/httpd.conf ¥
  && echo "SetEnv ssl-map-mode offload" >> /config/httpd.conf

# Add shell
COPY ihsstart.sh /work/

# chown
RUN  chown -R 1001:0 /opt/ibm/IHS ¥
  && chown -R 1001:0 /work

# Switch to user 1001
USER 1001

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

構成ファイル(httpd.conf)をカスタマイズ
・ポート番号を8080に変更
・Plug-in構成ファイルのパスを変更
・SSLMapModeをoffloadに設定

IHS起動スクリプト(ihsstart.sh)の追加

オーナの変更

実行ユーザーをdefault(ID:1001)に変更

コンテナー実行時に起動するスクリプトとして
IHS起動スクリプト(ihsstart.sh)を指定

起動スクリプト(ihsstart.sh)のカスタマイズ例

■ 起動スクリプトの例

- WebSphere Libertyの前段に配置する場合を考慮し、Plugin構成ファイルが生成された後にIHSを起動するようにカスタマイズ
- Plugin構成ファイル(plugin-cfg.xml)の生成を待つか否かを、環境変数WAIT_PLUGIN_GENERATIONで指定

```
#! /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
}

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

(左からの続き)

```
if [ "$WAIT_PLUGIN_GENERATION" = "true" ]; then
   while [ ! -e "/output/logs/state/plugin-cfg.xml" ]; do
      echo "Waiting for plugin-cfg.xml generation"
      sleep 5
   done
fi

startServer
trap "stopServer" SIGTERM

sleep 10
tail -f /logs/error_log &

if [ "$WAIT_PLUGIN_GENERATION" = "true" ]; then
   tail -f /logs/http_plugin.log &
fi

while [ -f "/logs/httpd.pid" ]; do
   sleep 5
done
```

環境変数「WAIT_PLUGIN_GENERATION」がtrueの場合は、Plugin構成ファイルが生成されるまで待機
(環境変数はHelm Chartの変数ファイルで設定)

IHS 起動、trap 設定（変更前と同様）

10秒sleepした後、
error_logをコンソールへtail出力

http_plugin.logをコンソールへtail出力

IHS稼働中はsleep（変更前と同様）

IHSを「前段」に配置する構成の例：WebSphere Liberty側の設定

■ WebSphere Liberty側の設定

– IHS の設定に合わせて、server.xmlまたはserver.xmlフラグメントで「pluginConfiguration」エレメントを構成します。

設定項目	備考
pluginInstallRoot	• Webサーバープラグインのインストール先を指定 • 前述のIHSイメージの場合、Webサーバープラグインのインストール先は「/opt/ibm/IHS/plugin」
webserverName	• Webサーバー名を指定 • IHSをアーカイブ・インストールした場合、Webサーバー名のデフォルトは「webserver1」
logFileName	• Webサーバープラグインのログファイルの名前を指定 • 前述のIHSイメージの場合、ログ出力先ディレクトリーは「/logs」

設定例

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

– その他の設定項目に関しては、以下URLもご参照ください。

- WebSphere Liberty Knowledge Center / 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「前段」に配置する構成の例：Helm Chartの変数ファイルの設定

■ Helm Chartの変数ファイルの設定

- Plugin構成ファイル(plugin-cfg.xml)用のボリュームを追加します。
  - pod.extraVolumesに指定
- IHSコンテナー用の設定を追加します。
  - pod.extraContainers.image
    - IHSのコンテナーに関する設定を追加します
  - pod.extraContainers.env
    - Plugin構成ファイル(plugin-cfg.xml)が生成されてからIHSを起動させるために環境変数WAIT_PLUGIN_GENERATIONに"true"を設定します。
  - pod.extraContainers.volumeMounts
    - Plugin構成ファイル用のボリュームを/output/logs/stateにマウントします。
    - ログを永続化する場合は、/logsにPersistent Volume(PV)をマウントします。
    - 静的コンテンツをPVに配置する場合は、静的コンテンツ用のPVを/htdocsにマウントします。
  - pod.extraContainers.livenessProbe, pod.extraContainers.readinessProbe
    - IHSに対しるProbeを指定します。
- WebSphere LibertyのコンテナーにPlugin構成ファイル用のボリュームをマウントします。
  - imge.extraVolumeMountsに指定
- 以下の設定も変更します。
  - service.portにはIHSのポート番号を指定します。
  - ssl.enabledにはfalseを指定します。前述の「IHSのイメージの作成方法」で作成したIHSのイメージではhttp通信のみが可能です。
- 補足
  - image.livenessProbeとimage.readinessProbeはWebSphere Libertyのコンテナーに対する指定となります。
  - Helm Chartの変数ファイルのresourcesで指定できるリソース制限は、WebSphere Liberty用のコンテナーにのみ適用されます。

IHSを「前段」に配置する構成の例：Helm Chartの変数ファイルの例 (1/3)

■ 変数ファイルの例：imageの要素

```
image:
  repository: image-registry.openshift-image-registry.svc:5000/wlp-sample/resource-eater
  tag: 19.0.0.12-full-java8-ibmjava-ubi
  readinessProbe:
    httpGet:
      path: /ResourceEater
      port: 9080
      scheme: HTTP
  livenessProbe:
    httpGet:
      path: /ResourceEater
      port: 9080
      scheme: HTTP
  extraVolumeMounts:
    - name: state-vol
      mountPath: /output/logs/state

（次ページへ続く）
```

WebSphere Libertyの
イメージの指定

WebSphere Libertyに対するProbeの指定

Plugin構成ファイル(plugin-cfg.xml)を共有するための領域(emptyDir)
のマウント

WebSphere LibertyのHelm Chart v1.10.0 では、image.extraVolumeMountsの展開に問題があり、 変数ファイルでimage.extraVolumeMountsを指定した場合にエラーが発生することがありますが、Helm Chartを修正することで対応できます。
修正方法は、第5章「Helm Chart のカスタマイズ」を参照してください。

IHSを「前段」に配置する構成の例：Helm Chartの変数ファイルの例 (1/3)

■ 変数ファイルの例：podの要素

```
pod:
  extraContainers:
   - name: ihs
     image: image-registry.openshift-image-registry.svc:5000/wlp-sample/ihs:9.0.5.2-ubi
     env:
       - name: WAIT_PLUGIN_GENERATION
         value: "true"
     volumeMounts:
       - name: htdocs-vol
         mountPath: /htdocs
         readOnly: true
       - name: liberty-pvc
         mountPath: /logs
         subPath: logs
       - name: state-vol
         mountPath: /output/logs/state
     livenessProbe:
       httpGet:
         path: /
         port: 8080
         scheme: HTTP
     readinessProbe:
       httpGet:
         path: /ResourceEater
         port: 8080
         scheme: HTTP

  extraVolumes:
    - name: htdocs-vol
      persistentVolumeClaim:
        claimName: htdocs-pvc
    - name: state-vol
      emptyDir:
```

（次ページへ続く）

- IHSのイメージの指定
- Plugin構成ファイル(plugin-cfg.xml)が生成されてからIHSを起動させるための環境変数の設定(ihsstart.shがこの環境変数を利用)
- 静的コンテンツ用のPVのマウント
- ログ出力用のPVのマウント（WebSphere Libertyのログ出力用PVを共用）
- Plugin構成ファイル(plugin-cfg.xml)を共有するための領域(emptyDir)のマウント
- IHSに対するProbeの指定（readinessProbeがWebSphere Libertyに到達するように指定）

IHSコンテナーの設定

- 静的コンテンツ用のPVの指定
- Plugin構成ファイル(plugin-cfg.xml)を共有するためのボリュームとしてemptyDirを定義

IHSを「前段」に配置する構成の例：Helm Chartの変数ファイルの例 (3/3)

■ 変数ファイルの例：その他の要素

```
service:
  enabled: true
  port: 8080
  targetPort: 8080
  type: NodePort

ssl:
  enabled: false

ingress:
  enabled: true
  rewriteTarget: "/"
  path: "/"
  host: "resource-eater-ihs.·····"

persistence:
  name: "liberty-pvc"
  size: "1Gi"
  storageClassName: "wlp-log-sc"

logs:
  persistLogs: true
  persistTransactionLogs: false

rbac:
  install: false

（終わり）
```

Serviceのportは、IHSのポート(8080)を指定

SSLは無効化（紹介したIHSのイメージではSSLは有効化していない）

Ingressの設定

ログ出力用のPVCの設定

WebSphere Libertyのログを永続化させる設定

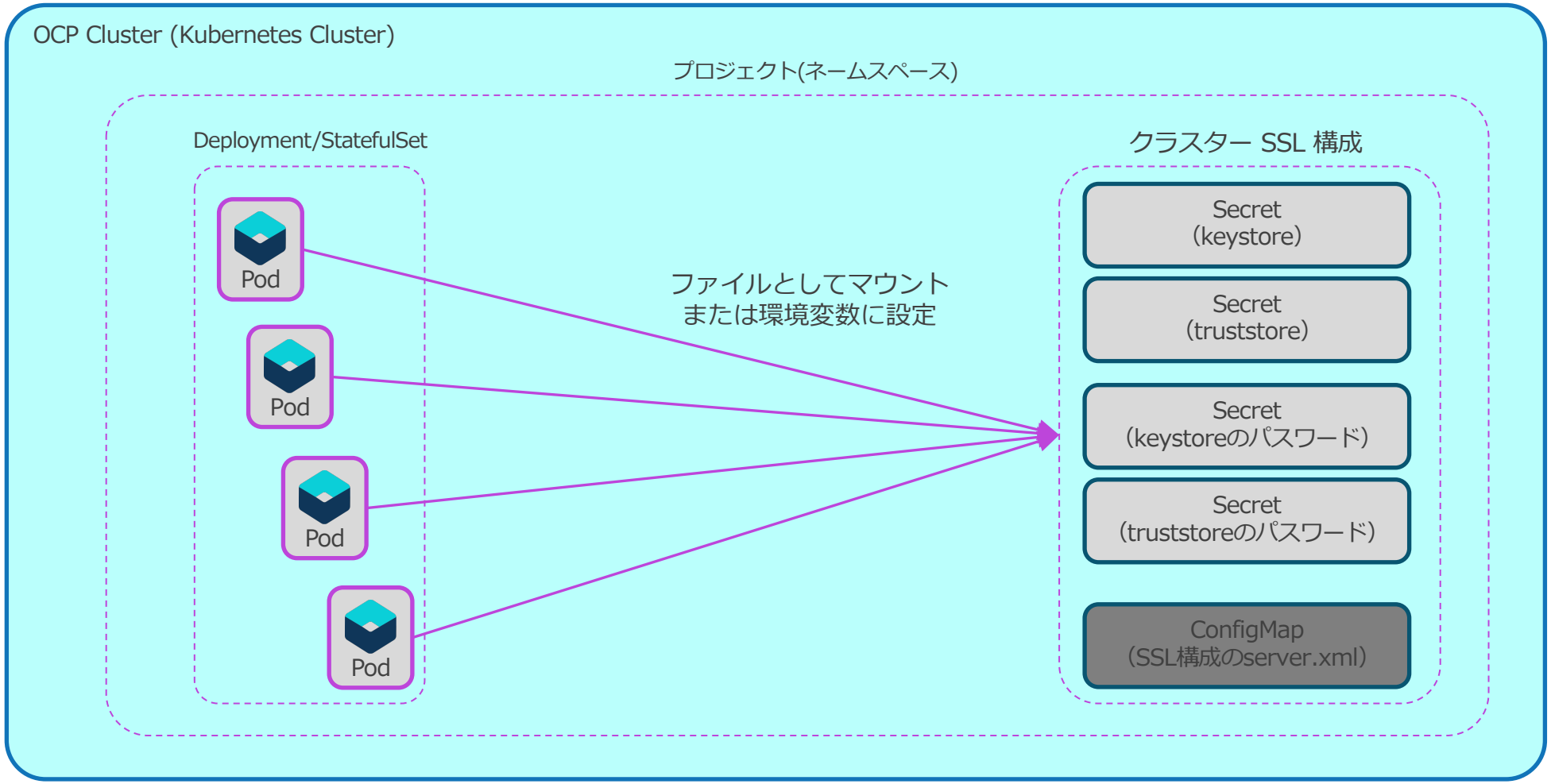
この例ではRBACを無効化

14. クラスターSSL構成の利用

目次(14章)

クラスターSSL構成とは

- クラスターSSL構成を利用すると、同じプロジェクト(ネームスペース)内で稼動するWebSphere 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構成の構成要素

■ 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	WebSphere 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。 WebSphere 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。 WebSphere 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。
WebSphere 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、WebSphere Liberty の構成ファイルです。 バージョン1.5.0以降のHelm Chartでは使われません。

– 参考
- WebSphere Liberty Knowledge Center / Liberty での 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
- WebSphere 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構成の作成：Helm Chartを利用して作成する方法

- Helm Chartの変数ファイルで「ssl.createClusterSSLConfiguration」に「true」を指定することで、デプロイ時にクラスターSSL構成を作成できます。
  - liberty-secret-generator-deployのJobによってibmcom/mb-toolsのコンテナーが実行され、前述のSecretとConfigMapが作成されます。
  - 作成後は「ssl.createClusterSSLConfiguration」に「false」を指定します。

- しかしながら、下記の考慮点があり利用に適さないので、この方法はお勧めしません。
  - この方法で作成される証明書は、有効期限が１年間の自己署名証明書になり、定期的に再作成する必要があります。
    - 有効期限を指定する方法はありません。
  - 再作成する場合は、前述のSecret、ConfigMap、および、Jobを予め手動で削除しておく必要があります。
  - オフライン環境では、ibmcom/mb-toolsのコンテナー・イメージはあらかじめ各Workerノードにロードしておく必要があります。
  - ibmcom/mb-toolsのコンテナーはrootユーザーで実行されます。

- 次に示す、手動で作成する方法をお勧めします。

クラスターSSL構成の作成：手動で作成する方法（1/2）

- 鍵ストアとトラスト・ストアを準備し、所定のSecretを定義することで、クラスターSSL構成を作成します。手順は以下の通りです。

- まず、jks (Java KeyStore) 形式の鍵ストアとトラスト・ストアを準備します。
  - 自己証明書を利用する場合は、WebSphere LibertyのsecurityUtilityコマンドでも作成できます。
    - WebSphere 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
  - securityUtilityコマンドで作成する場合の例：

```
# docker run --rm -it ¥
             -v /work/:/opt/ibm/wlp/output/defaultServer/resources/security/ ¥
             ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi ¥
             /opt/ibm/wlp/bin/securityUtility ¥
               createSSLCertificate       ¥
                 --server=defaultServer   ¥
                 --keyType=jks            ¥
                 --password=(パスワード)   ¥
                 --validity=(有効日数)     ¥
                 --subject=(サブジェクトおよび発行者の識別名)

# cp /work/key.jks /work/trust.jks
```

docker run の -v オプションで、
PCの /work ディレクトリーをコンテナーの鍵ストア出力先に
マウントする例

WebSphere Libertyのイメージを指定（タグは適時変更）

securityUtilityが生成した鍵ストアをトラストストアとして流用するためにコピーする

クラスターSSL構成の作成：手動で作成する方法（2/2）

- 鍵ストアのパスワードをWebSphere LibertyのsecurityUtilityコマンドでエンコードします。
  - WebSphere 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

```
# docker run --rm -it ¥
             ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi ¥
             /opt/ibm/wlp/bin/securityUtility  encode  (パスワード)
```

- 鍵ストア、トラスト・ストア、および、エンコードしたパスワードからクラスターSSL構成用のSecretを作成します。

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

# oc create secret generic mb-keystore-password   --from-literal=password="·····"
# oc create secret generic mb-truststore-password --from-literal=password="·····"
```

securityUtilityコマンドでエンコードしたものを指定することを推奨します

- 以上で、クラスターSSL構成の作成は完了です。

クラスターSSL構成の利用

- Helm Chartの変数ファイルで「ssl. enabled」と「ssl.useClusterSSLConfiguration」に「true」を指定してデプロイします。

```
ssl:
  enabled: true
  useClusterSSLConfiguration: true
  createClusterSSLConfiguration: false
```

- Helm Chartが作成するDeployment/StatefulSetおよびConfigMapに以下の変更が加わります。
  - 鍵ストアとトラスト・ストアが、クラスターSSL構成用のSecretからマウントされます。
    - 鍵ストアのマウント先： /etc/wlp/config/keystore/key.jks
    - トラスト・ストアのマウント先： /etc/wlp/config/truststore/trust.jks
  - 鍵ストアとトラスト・ストアのパスワードが、クラスターSSL構成用のSecretから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
  - 「<Helmリリース名>-ibm-websphere」という名前のConfigMapに「cluster-ssl.xml」が追加されます。
    - このConfigMapは/etc/wlp/configmap/cluster-ssl.xmlに配置され、/config/configDropins/overrides/include-configmap.xmlからincludeされます。

cluster-ssl.xmlの内容

```
cluster-ssl.xml: |-
  <server>
    <featureManager>
      <feature>ssl-1.0</feature>
    </featureManager>
    <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>
```

(補足) PKCS #12 形式を利用する場合：クラスターSSL構成の作成（1/2）

- PKCS #12形式の鍵ストアとトラスト・ストアを使用する場合は、Helm Chartの変更が必要になります。手順は以下の通りです。

- まず、PKCS #12 (Public Key Cryptography Standard #12) 形式の鍵ストアとトラスト・ストアを準備します。
  - 自己証明書を利用する場合は、WebSphere LibertyのsecurityUtilityコマンドでも作成できます。
    - WebSphere 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
  - securityUtilityコマンドで作成する場合の例：

```
# docker run --rm -it ¥
             -v /work/:/opt/ibm/wlp/output/defaultServer/resources/security/ ¥
             ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi ¥
             /opt/ibm/wlp/bin/securityUtility ¥
               createSSLCertificate       ¥
                 --server=defaultServer   ¥
                 --keyType=PKCS12         ¥
                 --password=(パスワード)   ¥
                 --validity=(有効日数)     ¥
                 --subject=(サブジェクトおよび発行者の識別名)

# cp /work/key.p12 /work/trust.p12
```

docker run の -v オプションで、
PCの /work ディレクトリーをコンテナーの鍵ストア出力先にマウントする例

WebSphere Libertyのイメージを指定（タグは適時変更）

securityUtilityが生成した鍵ストアをトラストストアとして流用するためにコピーする

(補足) PKCS #12 形式を利用する場合：クラスターSSL構成の作成（2/2）

- 鍵ストアのパスワードをWebSphere LibertyのsecurityUtilityコマンドでエンコードします。
  - WebSphere 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

```
# docker run --rm -it ¥
             ibmcom/websphere-liberty:19.0.0.12-full-java8-ibmjava-ubi ¥
             /opt/ibm/wlp/bin/securityUtility  encode  (パスワード)
```

- 鍵ストア、トラスト・ストア、および、エンコードしたパスワードからクラスターSSL構成用のSecretを作成します。

```
# oc create secret generic mb-keystore   --from-file=key.p12=/work/key.p12
# oc create secret generic mb-truststore --from-file=trust.p12 =/work/trust.p12

# oc create secret generic mb-keystore-password   --from-literal=password="·····"
# oc create secret generic mb-truststore-password --from-literal=password="·····"
```

securityUtilityコマンドでエンコードしたものを指定することを推奨します

- 以上で、クラスターSSL構成の作成は完了です。

(補足) PKCS #12 形式を利用する場合：クラスターSSL構成の利用

- Helm Chartを修正します。
  - Helm Chart を解凍し、ibm-websphere-liberty/charts/ibm-shared-liberty/templates/_configmap.tpl の内容を以下の様に修正します。(修正箇所は、ファイルの拡張子の部分だけです。)
  - Helm Chartの修正に関しては、第5章「Helm Chart のカスタマイズ」を参照してください。

ibm-websphere-liberty/charts/ibm-shared-liberty/templates/_configmap.tpl の47～57行目の抜粋

```
{{ if and $root.Values.ssl.enabled $root.Values.ssl.useClusterSSLConfiguration }}
  cluster-ssl.xml: |-
    <server>
      <featureManager>
        <feature>ssl-1.0</feature>
      </featureManager>
      <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" trustStoreRef="defaultTrustStore"/>
      <keyStore id="defaultKeyStore" location="/etc/wlp/config/keystore/key.p12" password="${env.MB_KEYSTORE_PASSWORD}" />
      <keyStore id="defaultTrustStore" location="/etc/wlp/config/truststore/trust.p12" password="${env.MB_TRUSTSTORE_PASSWORD}" />
    </server>
{{ end }}
```

- 修正したHelm Chartを使用してデプロイします。
  - Helm Chartの変数ファイルで「ssl. enabled」と「ssl.useClusterSSLConfiguration」に「true」を指定してデプロイします。

```
ssl:
  enabled: true
  useClusterSSLConfiguration: true
  createClusterSSLConfiguration: false
```

IBM Cloud Paks