WAS Liberty on IBM Cloud Private 利用ガイド

version 2.0.0

2019/08/06

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2019年6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 19.0.0.4、19.0.0.5、19.0.0.6
  - IBM Cloud Charts, ibm-websphere-liberty v1.9.0
  - IBM Cloud Private v3.1.2.0
- 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。
  - 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。
  - 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。
  - 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。
  - 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。
  - 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。

- 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。

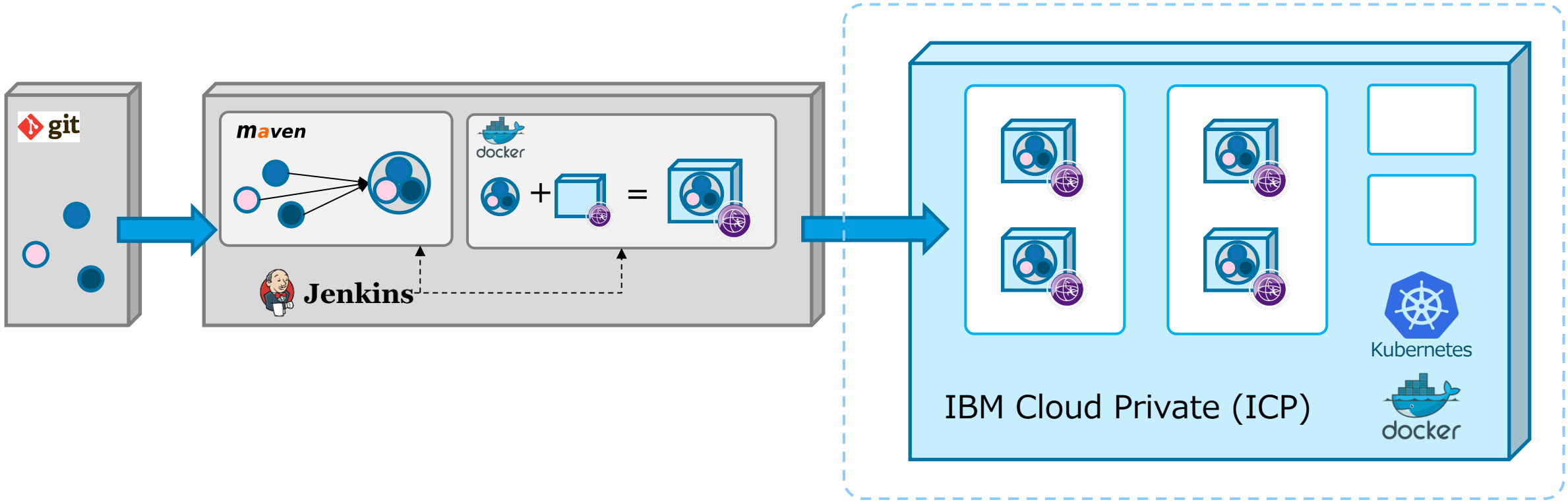
資料変更履歴

■ 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	2018/07/20	初版
2.0.0	2019/08/06	括弧内の製品バージョンに対応 (IBM Cloud Private 3.1.2、Websphere Application Server Liberty (docker) 19.0.0.4/19.0.0.5/19.0.0.6、ibm-websphere-liberty Helm v1.9.0)

このガイドについて

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



（このガイドの対象範囲外）

このガイドの対象範囲

目次

1. IBM Cloud Private (ICP)概要

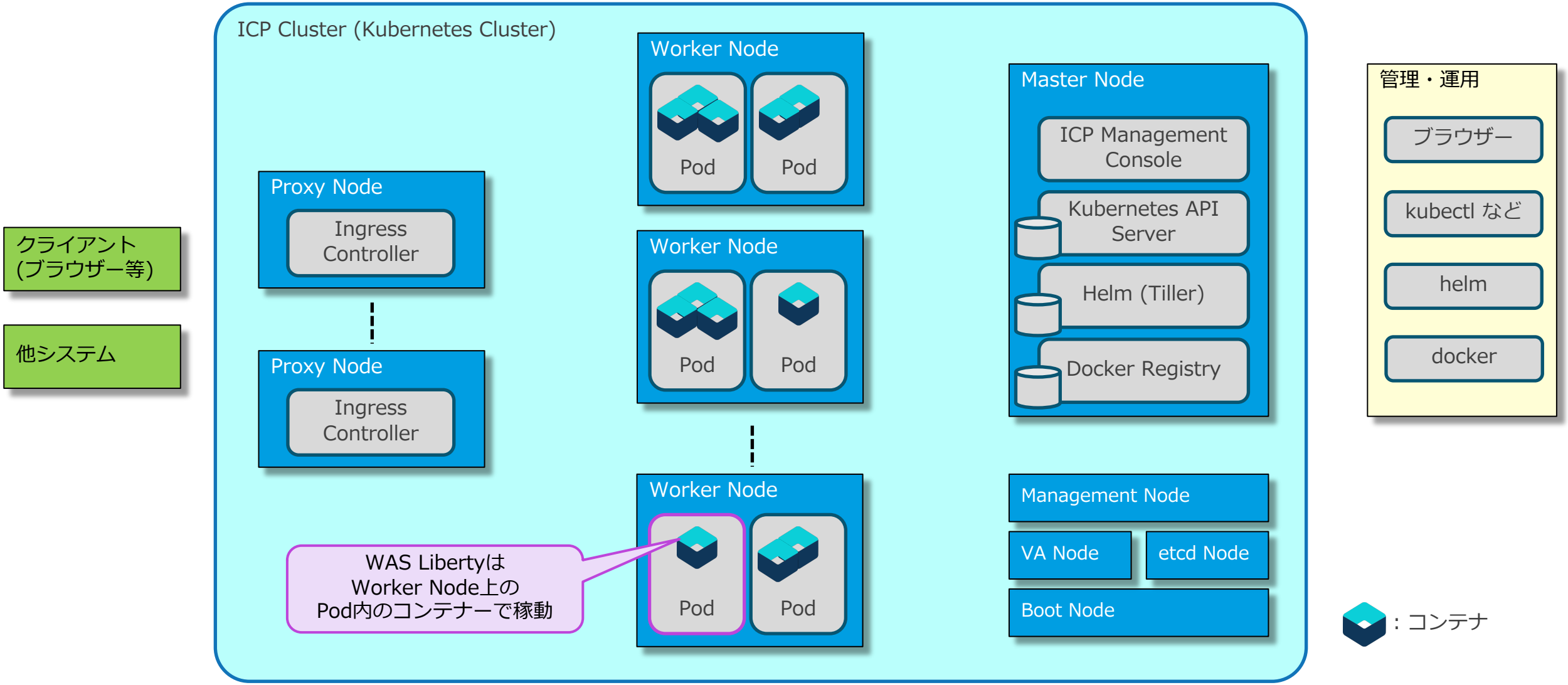
IBM Cloud Private (ICP)

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

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

IBM Cloud Private (ICP)の構成 (1)

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



IBM Cloud Private (ICP)の構成 (2)

ICP 3.1.2 Knowledge Center / アーキテクチャー
https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/getting_started/architecture.html

- ICP Cluster
  - 以下のノードで構成され、Master Nodeによって管理されます。
- Node
  - Master Node
    - クラスターを管理するノードです。複数の Master Node(3または5)を配置することで高可用性を確保できます。
  - Proxy Node
    - クライアント(ブラウザー)や他システムからの入り口となるノードです。複数のProxy Nodeを配置することで高可用性を確保できます。
    - Ingress Controllerが稼動し、リバース・プロキシーの機能を提供します。
  - Worker Node
    - アプリケーションやミドルウェアを稼動させるためのノードです。WAS Libertyはこのノードで稼動します。
    - 複数のWorker Nodeを配置することで、高可用性を確保し、処理能力を増やすことが可能です。
  - その他のノード
    - Management Nodeはモニタリングおよびロギング用のフレームワークを稼動させるノードです。
    - VA NodeはVulnerability Advisorの機能が稼動するノードで配置はオプションです。
    - etcd NodeはMaster Nodeからetcdのワークロードをオフロードするために配置されるオプションのノードです。
    - Boot Nodeはクラスターのインストール、ノード増強、クラスターの更新などで使用されるノードです。
- Pod
  - 計算リソース、ネットワーク、ストレージを共有する緊密な関係を持ったコンテナーのグループで、Kubernetesの管理単位となります。
  - WAS LibertyはPod内のコンテナーで稼動します。
  - (各ノードで稼動するICP (k8s)の機能もPodとして実装されているものがありますが、「IBM Cloud Private (ICP)の構成 (1)」の図ではその様には示されていません。)
- コンテナー
  - Dockerのコンテナ・ランタイム(インスタンス)です。

ICPの管理操作

■ ICPの管理操作は、以下のインターフェースを介して実行します。

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

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

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

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

ICPの管理操作：補足

(別バージョンのICP使用する場合は、必ずそのバージョンのICPのKnowledge Centerをご参照ください。)

- ICP Management Console へのアクセス方法や、各コマンドの利用方法に関しては、ICPのKnowledge Centerを参照してください。
  - ICP Management Consoleへのアクセス
    - ICP 3.1.2 Knowledge Center / management consoleを使用したIBM Cloud Privateクラスターへのアクセス
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/cfc_gui.html
  - Web Terminalの利用
    - ICP 3.1.2 Knowledge Center / Web 端末を使用した management console からのクラスター管理
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/webterm_gui.html
  - cloudctlコマンドの利用
    - ICP 3.1.2 Knowledge Center / IBM Cloud Private CLI (cloudctl) を使用したクラスターの管理
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/icp_cli.html
  - kubectlコマンドの利用
    - ICP 3.1.2 Knowledge Center / Kubernetes CLI (kubectl) からクラスターへのアクセス
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/cfc_cli.html
  - helmコマンドの利用
    - ICP 3.1.2 Knowledge Center / Helm CLI (helm) のインストール
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/create_helm_cli.html
    - ICP 3.1.2 Knowledge Center / Helm CLI への内部 Helm リポジトリーの追加
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/add_int_helm_repo_to_cli.html
  - イメージの操作、dockerコマンドの利用
    - ICP 3.1.2 Knowledge Center / イメージのプッシュおよびプル
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_images/using_docker_cli.html
    - ICP 3.1.2 Knowledge Center / イメージの管理
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_images/managing_images.html

基礎知識：Workload 関連

(別バージョンのICP使用する場合は、必ずそのバージョンのICPのKnowledge Centerをご参照ください。)

- Workloadを定義することで、アプリケーションやミドルウェアをICPクラスターで稼動させます。本ガイドに関連するWorkloadを以下に記載します。
  - ReplicaSet
    - テンプレートに従って同一構成の Pod を複数同時に起動し管理します。
    - Pod の名前は、ランダムなサフィックスが付与された一時的な名前となります。
    - ICP 3.1.2 Knowledge Center / ReplicaSet の管理
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/replicaSets.html
    - Kubernetes Documentation / Replicaset
    - https://kubernetes.io/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。
    - ICP 3.1.2 Knowledge Center / デプロイメントの作成
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/deploy_app.html
    - Kubernetes Documentation / Deployments
    - https://kubernetes.io/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。
    - ICP 3.1.2 Knowledge Center / StatefulSet の作成
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/stateful_apps.html
    - Kubernetes Documentation / StatefulSets
    - https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

基礎知識：ServiceとEndpoint

(別バージョンのICP使用する場合は、必ずそのバージョンのICPのKnowledge Centerをご参照ください。)

- Serviceは、ReplicaSetやStatefulSetなどが作成した論理的なPodの集まりを抽象化する機能を提供します。
  - ICP 3.1.2 Knowledge Center / サービスの作成
  - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/expose_app.html
  - Kubernetes Documentation / Service
  - https://kubernetes.io/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
    - ICPクラスター内でのみ有効なIPアドレス(ClusterIP)を使用してサービスを公開します。
    - サービスへのアクセスで必要となるCluster IPは、Service名を使用してDNSより取得できます。
  - type: NodePort
    - ClusterIPの機能を提供します。
    - さらに、各NodeのIPアドレスで、外部からアクセス可能なポート番号(NodePort)を使用してサービスを公開します。
  - Headless Service
    - ClusterIPにNoneが指定されているServiceで、Cluster IPが割り当てられません（type: ClusterIPの特殊形態です）
    - Service名を使用してDNSをlookupすると、Serviceを提供するPodのIPアドレスが返されます。
    - Serviceを構成するPodのIPアドレスが必要な場合などのように、特別な用途で利用されます。

- Endpointは、関連付けられたServiceを提供可能な、PodのIPアドレスのリストです。
  - (上述の「Kubernetes Documentation / Service」参照)

- IPアドレスのリストの内容はLiveness/Readiness Probe(次ページ参照)の結果に基づいて更新されます。

基礎知識： Probe

- Probeは、Pod内のコンテナーの稼働状況をチェックし、コンテナーに対するヘルスチェック機能を提供します。
  - Kubernetes Documentation / Configure Liveness and Readiness Probes
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/

- System Probeを以下に記載します。
  - Liveness Probe
    - コンテナーの生死状態をチェックするためのProbeです。
    - このProbeが失敗すると、コンテナーはキルされ、Podのリスタート・ポリシーに従って処理されます。
  - Readiness Probe
    - コンテナーのReady状態をチェックするためのProbeです。
    - このProbeが失敗すると、リクエストが該当の Pod へ割り振られなくなります。
    - (該当のPodのIPアドレスがEndpointから除去されます。)

基礎知識：ConfigMapとSecret

(別バージョンのICP使用する場合は、必ずそのバージョンのICPのKnowledge Centerをご参照ください。)

- 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。
  - ICP 3.1.2 Knowledge Center / ConfigMap の作成
  - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/create_config_map.html
  - Kubernetes Documentation / Configure a Pod to Use a ConfigMap
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/
  - ICP 3.1.2 Knowledge Center / 秘密の管理
  - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_applications/create_secrets.html
  - Kubernetes Documentation / Secrets
  - 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Dockerイメージに含める必要がなくなるので、環境毎にイメージをビルドする必要がなくなります。)イメージの再利用性・可搬性が高まります。)
  - 機密性の高い情報をDockerイメージに含める必要がなくなります。

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

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

基礎知識：Storage関連

(別バージョンのICP使用する場合は、必ずそのバージョンのICPのKnowledge Centerをご参照ください。)

- Persistent Volumeは、Podに永続領域を提供します。
  - ICP 3.1.2 Knowledge Center / Kubernetes ストレージの知識
  - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/kub_storage.html
  - Kubernetes Documentation / Persistent Volumes
  - https://kubernetes.io/docs/concepts/storage/persistent-volumes/

- Podが使用する永続領域(Persistent Volume)は２つの方法で準備できます。
  - Static Provisioning
    - 予め準備されている領域を、PersistentVolumeとして定義しておく方法です。
    - PodのPersistentVolumeClaimsにマッチするPersistentVolumeがPod に割り当てられます。
  - Dynamic Provisioning
    - StorageClassで定義されているprovisionerから動的に領域を準備する方法です。
    - PodのPersistentVolumeClaimsで指定されたStorageClassから動的に領域が準備され、Pod に割り当てられます。

- 永続領域(Persistent Volume)のマウント方法には以下のものがあります。
  - ReadWriteOnce:　単一のPodからのみread-write可能
  - ReadOnlyMany:　複数のPodからreadのみ可能
  - ReadWriteMany:　複数のPodからread-write可能

2. ICP環境でのWAS Liberty概要

ICP環境でのWAS Libertyの可用性と拡張性

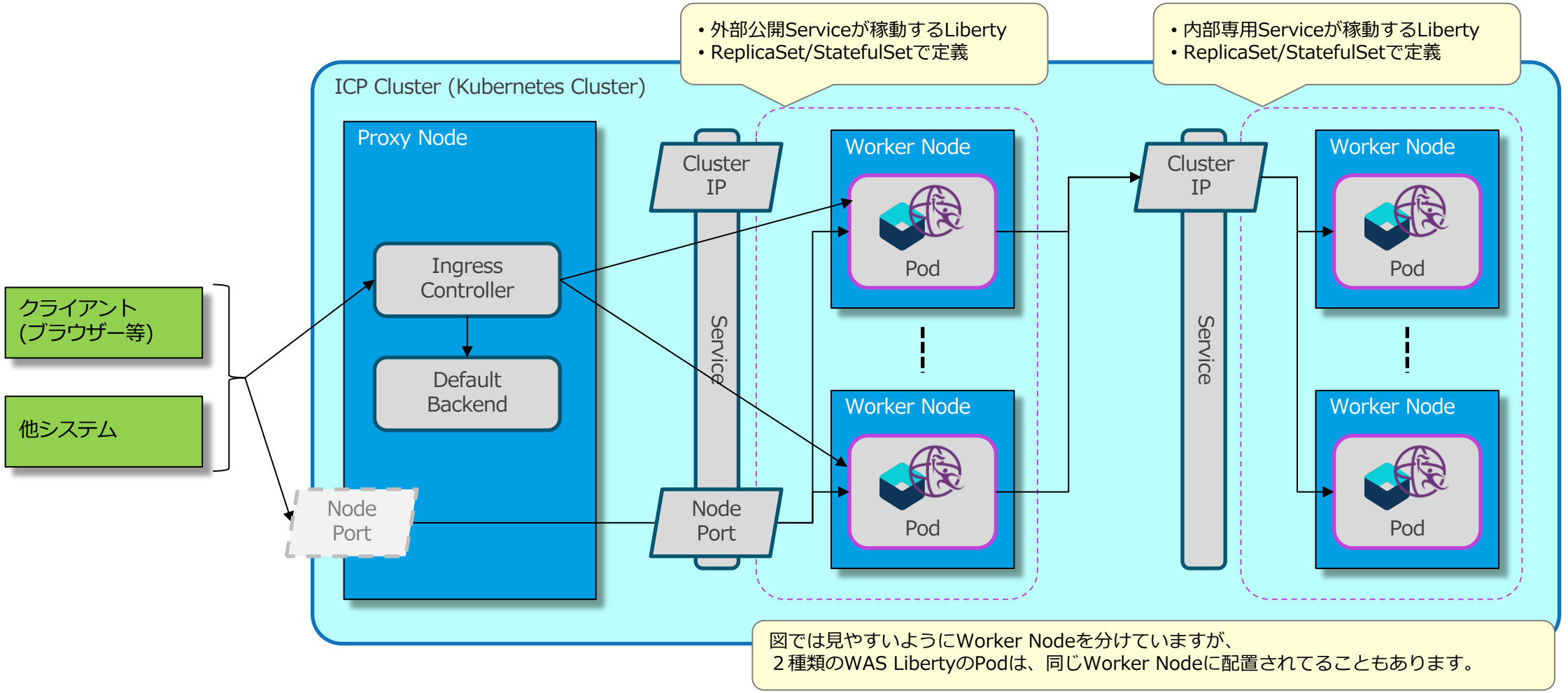
- WAS LibertyはWorker Node上のPod内のコンテナーで稼動します。

- WAS Libertyの可用性・拡張性はICPの機能で確保します。
  - WAS LibertyのNetwork Deployment版が提供するCollective等の機能は使用しません。

- 以下の機能はICPの機能で実現します。
  - 割振制御・再起動
    - Readiness Probeによってヘルスチェックが行われ、異常がある場合は割振り先(Endpoint)から除去されます。
    - Liveness Probeを設定することで、異常がある場合は自動で再起動させることが可能です。
  - WAS Libertyのクラスタリング
    - テンプレートに従ってPodを複数作成することで、目的数のPodを稼動させ、クラスタリングします。
    - 負荷分散は ClusterIP/NodePort/Ingressによって行われます。(後述)
  - オート・スケーリング
    - 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、LibertyをDeploymentとしてインストールした場合(ログを永続化していない場合)のみです。

ICP環境で稼動するWAS Libertyへのアクセス：概要

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



ICP環境で稼動するWAS Libertyへのアクセス：外部からのアクセス

■ ICPクラスター外からのアクセスには２つの方法があります。

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

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

ICP環境で稼動するWAS Libertyへのアクセス：内部からのアクセス

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

3. 基本的なインストール

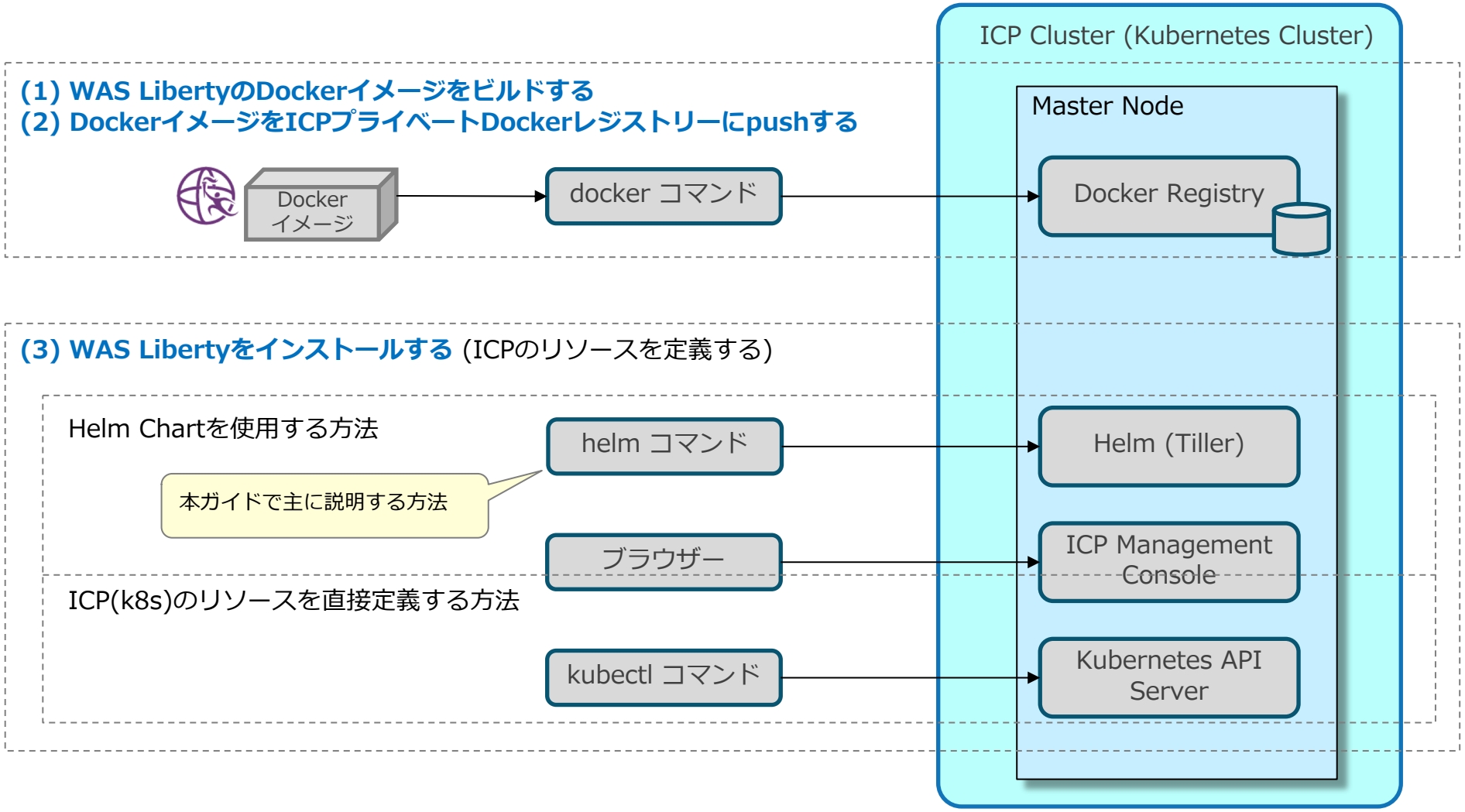
目次(3章)

- ICP環境でのWAS Libertyのインストールの流れ
- ICP環境でのWAS Libertyのインストール
  - (1) WAS LibertyのDockerイメージをビルドする
  - (2) DockerイメージをICPプライベートDockerレジストリーへpushする
  - (3) WAS Libertyをインストールする(helmコマンド)
- IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの
- (補足)ICP Management ConsoleによるWAS LibertyのHelm Chartのインストール
- WAS LibertyのHelm Chartの運用
  - Helm Chartが作成したリソースの確認
  - (補足)Helm Chartによって作成されるICP(Kubernetes)リソース一覧
  - Helm Chartを使用した運用操作
  - (補足)helmコマンドリファレンス
- (補足)ICP環境でのWAS Libertyのserver.xmlの設定方法
  - server.xmlとserver.xmlフラグメント
  - ICP環境でのWAS Libertyのserver.xmlの設定方法

目次(3章)

ICP環境でのWAS Libertyのインストールの流れ

■ WAS Libertyのインストールの流れを下図に示します。



ICP環境でのWAS Libertyのインストールの流れ

■ WAS Libertyのインストールの流れは以下のようになります。

– **(1) WAS LibertyのDockerイメージをビルドする**

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

– **(2) DockerイメージをICPプライベートDockerレジストリーにpushする**

- (1)でビルドしたDockerイメージをICPのプライベートDockerレジストリーに登録(push)します。

– **(3) WAS Libertyをインストールする**

- 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などを使用してインストールします。
  - インストール操作は、helmコマンドで行うか、ICP Management Consoleから行います。
- 2. ICPのリソースを直接定義する方法
  - ICPリソース(ReplicaSet や Serviceなど)の定義情報をyamlまたは json 形式で記述したファイルを使用して、kubectl applyコマンドでインストールします。
  - ICPリソースの定義情報をICP Management Consoleの画面で入力することでインストールすることも可能です。

本ガイドで主に説明する方法

■ 本ガイドの後述の章では、IBM提供のWAS LibertyのHelm Chartを利用して、helmコマンドでインストールする方法を主に説明していきます。

目次(3章)

(1) WAS LibertyのDockerイメージをビルドする

■ WAS LibertyのDockerイメージの作成方法は2つあります。

– IBM提供のDockerイメージから作成する方法

• (例)IBM提供の19.0.0.6-kernelのtagのDockerイメージには以下の設定が施されています。

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

(補足) DockerHub / websphere-liberty(Docker Official Images)で、使用したいDockerイメージのTagをクリックすると該当イメージをビルドする際に使用されたDockerfileが表示されます。
DockerHub / websphere-liberty(Docker Official Images)
https://hub.docker.com/_/websphere-liberty

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

– 独自のDockerイメージを作成する方法

• IBM提供のHelm Chartは、IBM提供のDockerイメージと組み合わせて利用することを前提として作成されています。
  – （WAS Libertyのログのディレクトリなどが/logsとなっていることが前提、など）
• 独自のDockerイメージをIBM提供のHelm Chartと組み合わせて使用する場合は、以下の指定をDockerfileに組み込む必要があります。

(Dockerfileの例)

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

WAS Liberty Knowledge Center / Liberty Helm chart
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_icp_helm.html

(1) WAS LibertyのDockerイメージをビルドする

■ IBM提供のDockerイメージから作成する方法

– 本ガイドでは基本的にIBM提供のWAS LibertyのDockerイメージをベースイメージとして使用します。

- DockerHub / websphere-liberty(Official Images)
- https://hub.docker.com/_/websphere-liberty

IBM提供イメージ
- websphere-liberty

（2019年7月現在）

指定できるtag
- 19.0.0.6-kernel
- 19.0.0.6-javaee8
- 19.0.0.6-webProfile8
- 19.0.0.6-microProfile1
- 19.0.0.6-microProfile2
- 19.0.0.6-springBoot2
- 19.0.0.6-springBoot1
- 19.0.0.6-webProfile7
- 19.0.0.6-javaee7
- など

– 要件に応じたDockerイメージを作成するために、以下のステップなどをDockerfileに記述してビルドします。

- WAS Libertyの構成ファイル(server.xml など)やアプリケーションをイメージの/configにコピー
- アプリケーションの稼動に必要となるその他のファイルをイメージにコピー
  – JDBCドライバー、リソース・アダプター、共有ライブラリー用のjarファイルなど
- WAS Libertyのフィーチャーの追加(必要な場合)

（Dockerfileの例1）

```
FROM websphere-liberty:19.0.0.6-javaee8
COPY --chown=1001:0  Sample.war /config/dropins/
COPY --chown=1001:0  server.xml /config/
RUN installUtility install --acceptLicense defaultServer
```

この例では、サンプルアプリを/config/dropins/にコピーしています。
また、WAS Libertyの構成ファイル(server.xml)を/config/にコピーしています。
この場合はDockerfile内に明示的に不足しているフィーチャーを記載しなくても「installUtility install –acceptLicense defaultServer」コマンドでserver.xmlに定義されているフィーチャーは自動でインストールされます。

（Dockerfileの例2）

```
FROM websphere-liberty:19.0.0.6-javaee8
COPY --chown=1001:0  Sample.war /config/dropins/
RUN installUtility install --acceptLicense audit-1.0
```

この例では、WAS Libertyの構成ファイル(server.xml)はコピーしていません。
ベースイメージとして使用している「19.0.0.6-javaee8」に不足している「audit-1.0」を明示的に追加したい場合は明示的にinstallUtilityでインストールします。

(1) WAS LibertyのDockerイメージをビルドする

■ IBM提供のDockerイメージから作成する方法

– 手順概要 (server.xmlとアプリケーションをイメージにコピーする場合の例)

• (1) 作業ディレクトリでDockerfileを作成します。

(例)Dockerfileの例

```
FROM websphere-liberty:19.0.0.6-javaee8
COPY --chown=1001:0 Sum.war /config/dropins/
COPY --chown=1001:0 server.xml /config/
RUN installUtility install --acceptLicense defaultServer
```

IBM提供のWAS LibertyのDockerイメージ(tag: 19.0.0.6-javaee8)をベースイメージとして使用する場合

• (2) Dockerfileが作成されたことを確認する。(コピーするserver.xmlとアプリケーションも同じディレクトリに配置します。)

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

• (3) 「docker build -f Dockerfile –t <image name>:<tag name> .」コマンドを入力し、Dockerイメージをビルドします。

```
$ 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
(略)
```

- -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Docker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Dockerfileのパスまたは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。

• (4) 作成されたDockerイメージを確認します。

```
$ docker images | grep myliberty
mylibertyapp                                    1.0                 709ce7a8c295        6 minutes ago       520MB
(略)
```

(2) DockerイメージをICPプライベートDockerレジストリーにpushする

■ 作成したDockerイメージをICPのプライベートDockerレジストリーにpushします。

– 以下のリンク先の手順に従ってICPのプライベートDockerレジストリーにログインしてからDockerイメージを登録(docker push)します。

- ICP 3.1.2 Knowledge Center / イメージのプッシュおよびプル
- https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_images/using_docker_cli.html

– 手順概要

- (1) 以下のリンク先を参照し、作業端末のOS毎に必要な前提設定を実施します。
  – ICP 3.1.2 Knowledge Center / Docker CLI の認証の構成
  – https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_images/configuring_docker_cli.html

- (2) (1)のリンク先の最後の手順として「docker login <cluster_CA_domain>:8500」コマンドでICPのプライベートDockerレジストリーにログインします。
  – <cluster_CA_domain>は、ICPのインストール時にconfig.yamlで設定したCAドメイン名を指定します。
  – (何も設定していない場合のデフォルト値は「mycluster.icp」です。)

```
$ docker login mycluster.icp:8500
Username: admin
Password:
Login Succeeded
$
```

ICPの管理者ユーザーのユーザー名(admin)とパスワード入力します。
（以前に一度docker loginしていた場合は、UsernameとPasswordは再度聞かれません。）

(2) DockerイメージをICPプライベートDockerレジストリーにpushする

- (3) 作業端末上で、「docker tag <source_image> <target_image>」コマンドで、アップロードするDockerイメージ名をICPのプライベートDockerレジストリーに適したものに変更します。

引数	入力値	備考
<source_image>	<image name>:<tag name>	この例では、前述の手順(1)で作成した「mylibertyapp:1.0」を指定します。
<target_image>	<cluster_CA_domain>:8500/<namespace name>/<image name>:<tag name>	この例では名前空間(handson)が事前に作成されているものとします。 例として「mycluster.icp:8500/handson/mylibertyapp:1.0」を指定します。

```
$ docker tag mylibertyapp:1.0 mycluster.icp:8500/handson/mylibertyapp:1.0
$
```

- (4) 「docker images」コマンドで、別名が付与されたイメージを確認します。

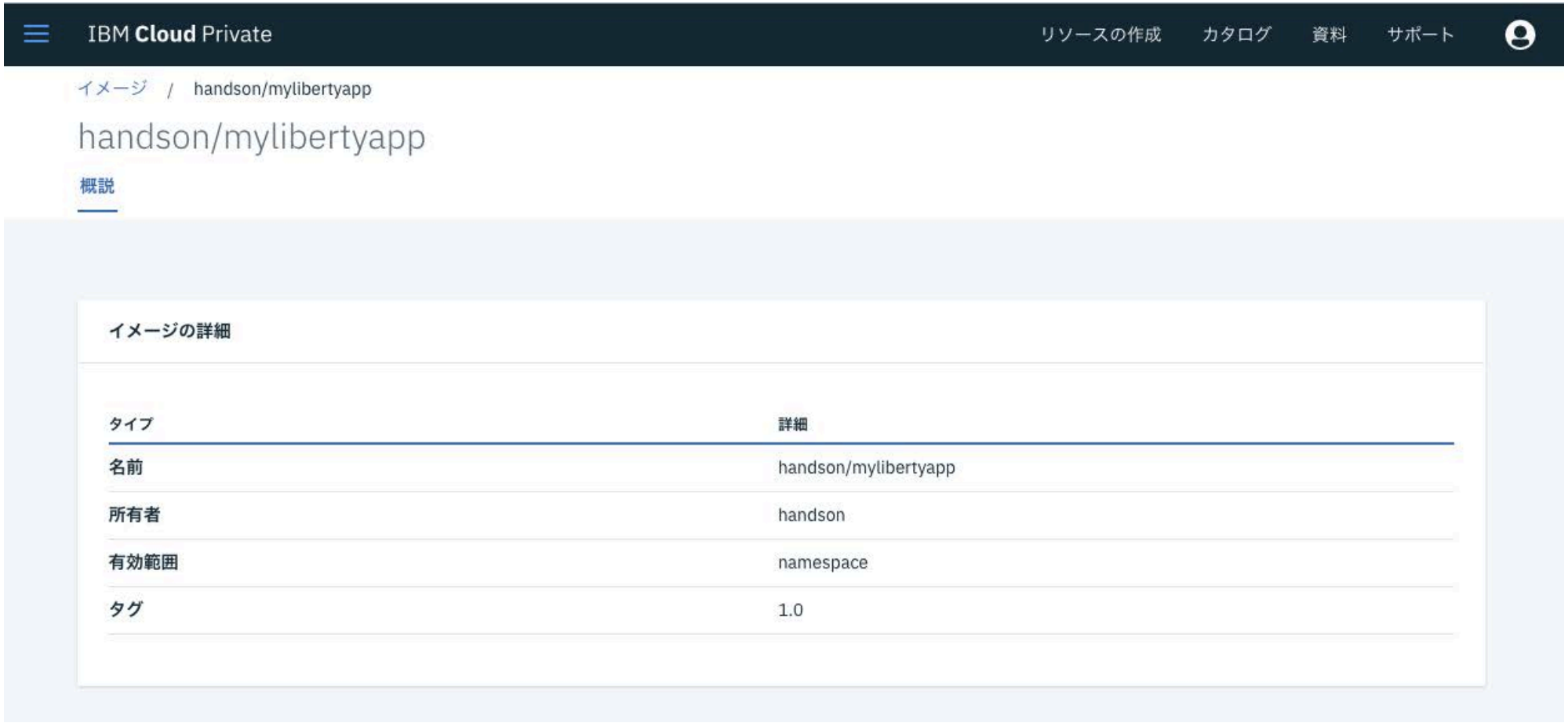
```
$ docker images | grep mylibertyapp
mylibertyapp                                  1.0                 709ce7a8c295        About an hour ago   520MB
mycluster.icp:8500/handson/mylibertyapp       1.0                 709ce7a8c295        About an hour ago   520MB
```

- (5) 「docker push <image name>:<tag name>」コマンドで、ICPのプライベートDockerレジストリーにDockerイメージを登録(push)します。

```
$ docker push mycluster.icp:8500/handson/mylibertyapp:1.0
The push refers to repository [mycluster.icp:8500/handson/mylibertyapp]
81bfea1502fb: Pushed
(略)
```

(2) DockerイメージをICPプライベートDockerレジストリーにpushする

- (6) pushされたイメージをICPコンソールから確認します。ICP Management Consoleにログインし、ナビゲーション・メニューから、「コンテナー・イメージ」を選択します。名前がhandson/mylibertyappのエントリーがあることを確認します。
  - (Tagまで確認するには、表示されたhandson/mylibertyappのエントリーを選択して表示される「イメージの詳細」画面の「タグ」欄を参照します。)



(3) WAS Libertyをインストールする(helmコマンド)

■ helmコマンドによるHelm Chartのインストール/アップグレード方法

– helmコマンドを使用したインストール/アップグレード操作では、使用するHelm Chartを以下の方法で指定できます。

- Helm Documentation / HELM INSTALL
- https://helm.sh/docs/helm/#helm-install

Chart指定方法	helmコマンド	備考
Chart参照	helm install stable/mariadb	• helm repo listで登録されているHelmリポジトリーのHelm Chartを指定してインストールする方法です。 • (ICPのインターナルHelmリポジトリを参照してインストールする場合などは、事前にhelm repo addコマンドでICPのインターナルHelmリポジトリを登録する必要があります。 • ICP 3.1.2 Knowledge Center / Helm CLI への内部 Helm リポジトリーの追加 • https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/add_int_helm_repo_to_cli.html
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を指定してインストールする方法です。
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/ 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。

本ガイドで主に説明する方法

– 本ガイドの後続の章では、ダウンロードしたIBM提供のWAS LibertyのHelm Chart(ibm-websphere-liberty-1.9.0.tgz)を展開しないでそのまま指定してインストールする方法(「パッケージングされたChartファイル」)を使用します。

(3) WAS Libertyをインストールする(helmコマンド)

■ パッケージされたChartファイル

– 当ガイドでは、ダウンロードしたIBM提供のWAS LibertyのHelm Chart(ibm-websphere-liberty-1.9.0.tgz)を指定してインストールする方法を主に説明します。

– 手順の流れは以下のようになります。

– 前提事項

- (0) 以下の手順に従ってhelmコマンドのインストールと初期設定をします。
  – ICP 3.1.2 Knowledge Center / Helm CLI (helm) のインストール
  – https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/create_helm_cli.html

– 手順概要

- (1) IBM提供のWAS LibertyのHelm Chartをダウンロードします。
  – GitHubの「IBM Cloud Charts Helm Repository」のrepo/stableにパッケージングされたHelm Chartがあります。
  – このリンク先からWAS LibertyのHelm Chart（ibm-websphere-liberty-1.9.0.tgz）をダウンロードします。(最新版がある場合はそちらをダウンロード)
  – GitHub / IBM/charts/repo/stable
  – https://github.com/IBM/charts/tree/master/repo/stable

(3) WAS Libertyをインストールする(helmコマンド)

- (2) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-mylibertyapp01.yaml
```

- (3) 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。
  - **変更するパラメーターの詳細は後続の「IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの」、または4章以降を参照してください。**
  - 例として、先ほどビルドしてICPのプライベートDockerレジストリーにpushしたイメージを使用してデプロイするようにパラメーターを編集します。

(例)values-mylibertyapp01.yaml

```
(略)
image:
  repository: mycluster.icp:8500/handson/mylibertyapp
  tag: "1.0"
  pullPolicy: IfNotPresent
  license: "accept"
(略)
```

前述の手順でICPのプライベートDockerレジストリーにpushしたDockerイメージの名前とtagを使用するように修正します。
ライセンスは"accept"と入力します。

- (4) 「cloudctl login」コマンドでICPクラスターにログインします。
  - (ログインするIPアドレスはMaster NodeのIPアドレスを指定します。)
  - (Master NodeのHA構成の場合はcluster_vipなど適宜Master Nodeにアクセス可能なIPアドレスを指定してください。)
  - (以下の例は、Master NodeのIPアドレスを作業端末の/etc/hostsに記載してmycluster.icpで名前解決できるようにしている場合の例となります。)

```
$ cloudctl login -a https://mycluster.icp:8443 -u admin -p <password> -n handson
```

この例では名前空間はhandsonを指定しています。

  - (後続の章ではICPクラスターにログインしていることを暗黙の前提として、手順に記載していない場合がありますのでご注意ください。)

(3) WAS Libertyをインストールする(helmコマンド)

– 「helm install」コマンドの場合

- (5) (オプション)修正した変数ファイルでHelm Chartをインストールした場合に登録されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。

```
$ helm install --dry-run --name mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
NAME:   mylibertyapp01
$
$ helm install --dry-run --debug --name mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
(略)
# Source: ibm-websphere-liberty/templates/shared.yaml
# SLT: 'slt.deployment' from templates/_deployment.tpl
apiVersion: apps/v1
kind: Deployment

metadata:
  name: mylibertyapp01-ibm-websp
(略)
```

「--dry-run」オプションをつけて実行すると、作成されるHelmリソース名が表示されます。

「--dry-run --debug」オプションをつけて実行すると、Helm Chartをインストールする際に、どのようなマニフェストファイルでICP(k8s)リソースを作成するか事前に詳細を確認することができます。

- (6) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm install --name mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

「helm install」でHelm Chartをインストールします。

「--name」でHelmリソース名を指定します。

「-f <Helmの変数ファイル名>」で修正した変数ファイルを指定します。

使用するWAS LibertyのHelm Chartのパッケージファイルを指定します。

ICPの場合は「--tls」を指定してTLS(SSL)で通信する必要があります。

(3) WAS Libertyをインストールする(helmコマンド)

– (補足)「helm install」コマンドの代わりに「helm upgrade --install」コマンドを利用することもできます。
  • Helm Documentation / HELM UPGRADE
  • https://helm.sh/docs/helm/#helm-upgrade

– 「helm upgrade --install」コマンドは以下のような動作をするため、初期インストール時、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リソースがインストールされていない場合は、新規インストールする。
  • 同じ名前のHelmリソースがインストールされている場合は、Helmの変数ファイルの更新部分のみpatchする。

– 前ページの「helm install」コマンドによるインストール手順を「helm upgrade --install」コマンドに置き換えた場合は次ページのようになります。
  • （本ガイドの後続の章では、「helm install」と「helm upgrade --install」コマンドが混在していますが、初期インストール時はどちらも同じ結果となりますので、適宜読み替えて実行してください。）

(3) WAS Libertyをインストールする(helmコマンド)

– 「helm upgrade --install」コマンドの場合

- (5) (オプション)修正した変数ファイルでHelm Chartをインストールした場合に登録されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。

```
$ helm upgrade --install --dry-run mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
Release "mylibertyapp01" does not exist. Installing it now.
NAME:   mylibertyapp01
$
$ helm upgrade --install --dry-run --debug mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-
1.9.0.tgz –tls
(略)
# Source: ibm-websphere-liberty/templates/shared.yaml
# SLT: 'slt.deployment' from templates/_deployment.tpl
apiVersion: apps/v1
kind: Deployment

metadata:
  name: mylibertyapp01-ibm-websp
(略)
```

「--dry-run」オプションをつけて実行すると、作成されるHelmリソース名が表示されます。

指定した名前のHelmリソースがあるか確認し、存在しない場合はインストールをする旨が表示されています。

「--dry-run --debug」オプションをつけて実行すると、Helm Chartをインストールする際に、どのようなマニフェストファイルでICP(k8s)リソースを作成するか事前に詳細を確認することができます。

- (6) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm upgrade --install mylibertyapp01 -f ./values-mylibertyapp01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

「helm upgrade --install」でHelm Chartをインストールします。

「helm install」の時とは異なり、「--name」を使用せずに、Helmリソース名を指定します。

「-f <Helmの変数ファイル名>」で修正した変数ファイルを指定します。

使用するWAS LibertyのHelm Chartのパッケージファイルを指定します。

ICPの場合は「--tls」を指定してTLS(SSL)で通信する必要があります。

(3) WAS Libertyをインストールする(helmコマンド)

- (7) 正常にインストールできたか確認します。
  - 「helm list --namespace <namespace name> --tls」コマンドで、インストールされたHelmリソースの名前を確認できます。

```
$ helm list --namespace handson --tls
NAME            REVISION        UPDATED                         STATUS          CHART                           NAMESPACE
mylibertyapp01  1               Wed Jul  3 18:02:26 2019        DEPLOYED        ibm-websphere-liberty-1.9.0     handson
```

  - 「helm status <Helmリソース名>--tls」コマンドで、作成されたICP(k8s)リソースの一覧と状況が確認できます。

```
$ helm status mylibertyapp01 --tls
LAST DEPLOYED: Wed Jul  3 18:02:26 2019
NAMESPACE: handson
STATUS: DEPLOYED

RESOURCES:
==> v1/Deployment
NAME                      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
mylibertyapp01-ibm-websp  1        1        1           1          11m

==> v1/Pod(related)
NAME                                      READY  STATUS   RESTARTS  AGE
mylibertyapp01-ibm-websp-86ccbf4448-j2fmh  1/1    Running  0         11m

==> v1/Secret
NAME                          TYPE    DATA  AGE
mylibertyapp01-ibm-websp-tls  Opaque  2     11m

==> v1/ConfigMap
NAME                      DATA  AGE
mylibertyapp01-ibm-websp  2     11m
(略)
```

作成された「Deployment」の名前と状況

作成された「Pod」の名前と状況

作成された「Secret」の名前

作成された「ConfigMap」の名前

目次(3章)

- ICP環境でのWAS Libertyのインストールの流れ
- ICP環境でのWAS Libertyのインストール
  - (1) WAS LibertyのDockerイメージをビルドする
  - (2) DockerイメージをICPプライベートDockerレジストリーへpushする
  - (3) WAS Libertyをインストールする(helmコマンド)
- IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの
- (補足)ICP Management ConsoleによるWAS LibertyのHelm Chartのインストール
- WAS LibertyのHelm Chartの運用
  - Helm Chartが作成したリソースの確認
  - (補足)Helm Chartによって作成されるICP(Kubernetes)リソース一覧
  - Helm Chartを使用した運用操作
  - (補足)helmコマンドリファレンス
- (補足)ICP環境でのWAS Libertyのserver.xmlの設定方法
  - server.xmlとserver.xmlフラグメント
  - ICP環境でのWAS Libertyのserver.xmlの設定方法

IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの

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

(*) ibm-websphere-liberty v1.9.0の場合

Qualifier	Parameter	デフォルト値	説明
image	repository	websphere-liberty	Docker イメージの名前を指定します。 ここで指定した Docker イメージが Pod 内のコンテナーで稼動することになります。アプリなどを組み込み、ビルドした Docker イメージを指定します。 通常、「<クラスター名>:<Docker レジストーのポート番号>/<ネームスペース>/<名前>」の形式になります。 例：mycluster.icp:8500/default/my-app
	tag	javaee8	Dockerイメージを識別するタグを指定します。
	pullPolicy	IfNotPresent	Dockerイメージをリポジトリーから pullするポリシーを指定します。 使用するDocker イメージはプライベート・レジストーに登録済みなので、通常は、「IfNotPresent」を指定します。
	license	""	WAS Liberty のライセンスを所有している場合は、「accept」を指定します。 指定しないと、開発ライセンスでの使用となります。
	readinessProbe	{}	コンテナからのリクエストを受け付ける状態かどうかを確認します。 デフォルトでは、MicroProfile Healthが有効になっている場合は、パスが/または/healthに設定されたHTTP probeを使用してチェックされます。
	livenessProbe	{}	コンテナが起動しているかどうかを確認します。デフォルトはreadinessProbeと同じ設定となります。
	extraEnvs	{}	コンテナ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	""	事前にserver-overrides.xmlファイルからConfigMapを作成し、このserver-overrides.xmlをserver.xmlフラグメントとして配置したい場合に、このConfigMap名を指定します。 (server-overrides.xmlはコンテナ内の/config/configDropins/overrides/server-overrides.xmlに配置されます。)
	extraVolumeMounts	[]	追加でPodにVolumeをマウントしたい場合指定します。

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

IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの

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

(*) ibm-websphere-liberty v1.9.0の場合

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。 ICPで現在指定できるのは、「ClusterIP」または「NodePort」です。 詳細は2章の「ICP環境で稼動するWAS Libertyへのアクセス」を参照してください。
	name	""	Service名を指定します。(DNSのAレコードになります。)
	port	9443	WAS Liberty のポート番号を指定します。デフォルトの場合、9080 (HTTP) または 9443 (HTTPS) となります。
	targetPort	9443	PodがWAS Libertyのポートを公開するために使用するポート番号を指定します。 Cluster IPとこのポート番号を使用して、ICP クラスター内からアクセスできるようになります。
	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

IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの

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

(*) ibm-websphere-liberty v1.9.0の場合

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)を有効にす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。(詳細は7章参照)
	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。（詳細は7章参照）
ingress	enabled	false	Ingress Controller(Proxy Node上で稼動するリバース・プロキシー)経由でのアクセス有無を指定します。 詳細は2章の「ICP環境で稼動するWAS Libertyへのアクセス」を参照してください。
	rewriteTarget	"/"	以下のpathで指定するパスの、書き換え先のパスを指定します。 WAS Libertyへのリクエストは、ここで指定したパスに書き換えられて転送されることになります。
	path	"/"	Ingress Controllerは、このパス配下へのアクセスをWAS Libertyに転送します。
	host	""	IngressでProxy ServerのIPアドレスの代わりに解決されるFQDNを指定することができます。
	secretName	""	HTTPS接続で使用するTLS証明書や鍵のSecretを指定します。 指定されていない場合は、Helmデプロイ時に自己署名の証明書と鍵を作成してSecretとして保管したものを使用します。
	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

IBM提供のWAS LibertyのHelm Chartのパラメーター：基本的なもの

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

(*) ibm-websphere-liberty v1.9.0の場合

Qualifier	Parameter	デフォルト値	説明
microprofile	health.enabled	false	trueを指定すると、WAS Liberty のコンテナーに対するReadiness Probeのパスが/から/healthに変わります。 主に、MicroProfile Healthに対応したマイクロサービスをWAS Libertyで稼動させる場合にtrueを指定します。
monitoring	---	---	後述(6章)
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	---	---	後述(5章)
logs	---	---	後述(5章)
autoscaling	---	---	後述(8章)
resources	---	---	後述(8章)
env	jvmArgs	""	Libertyランタイムに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)をインストールする設定です。 namespaceでRBACを使用している場合はtrue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

目次(3章)

(補足)ICP Management ConsoleによるWAS LibertyのHelm Chartのインストール

- ICP Management ConsoleでHelm Chartを扱うためには、ICPのHelm Chartのカタログに Helm Chartを追加する必要があります。

- 新しいHelm Chartや最新のバージョンのHelm ChartをICP Management Consoleのカタログから利用する場合は、以下の手順でICPのHelm Chartのカタログに取り込む必要があります。
  - リモートのリポジトリーに登録されているHelm Chartを追加・最新化する場合
    - ICP 3.1.2 Knowledge Center / Helm リポジトリーの管理
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/manage_helm_repo.html
  - 作成したHelm Chartを利用する場合
    - ICP 3.1.2 Knowledge Center / カスタム・アプリケーションの追加
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/add_package.html
  - 製品提供のHelm Chartをオフライン環境のICPで利用する場合
    - ICP 3.1.2 Knowledge Center / インターネット接続がないクラスターへのフィーチャー・アプリケーションの追加
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/add_package_offline.html
  - Helm ChartをICPのHelm Chartのカタログに登録するコマンド
    - ICP 3.1.2 Knowledge Center / IBM Cloud Private CLI のカタログ・コマンド (catalog) / cloudctl catalog load-chart
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/manage_cluster/cli_catalog_commands.html#load-chart

    - 次のページでダウンロードしてきたWAS LibertyのHelm Chartをオフライン環境のICPのHelm Chartのカタログに登録する手順を補足として記載します。

(補足)ICP Management ConsoleによるWAS LibertyのHelm Chartのインストール

■ WAS LibertyのHelm Chartをオフライン環境のICPのHelm Chartのカタログに登録する場合

– 手順概要

- (1) 前述の「(3) WAS Libertyをインストールする(helmコマンド)」の手順(1)の手順に従って、IBM提供のWAS LibertyのHelm Chartをダウンロードします。
  – ここでは例として、「ibm-websphere-liberty-1.9.0.tgz」をダウンロードしました。

- (2) 「cloudctl login」コマンドでICPクラスターにログインします。
  – (ログインするIPアドレスはMaster NodeのIPアドレスを指定します。)
  – (Master NodeのHA構成の場合はcluster_vipなど適宜Master Nodeにアクセス可能なIPアドレスを指定してください。)
  – (以下の例は、Master NodeのIPアドレスを作業端末の/etc/hostsに記載してmycluster.icpで名前解決できるようにしている場合の例となります。)

```
$ cloudctl login -a https://mycluster.icp:8443 --skip-ssl-validation
```

- (3) ダウンロードしたWAS LibertyのHelm Chartを「cloudctl catalog load-chart --archive <path-to-archive-file>」コマンドでICPのインターナルHelmリポジトリに登録します。

```
$ cloudctl catalog load-chart --archive ibm-websphere-liberty-1.9.0.tgz
```

- (4) ICPのインターナルHelmレポジトリ(local-charts)にHelm Chartが登録されたことを確認します。

```
$ cloudctl catalog charts --repo local-charts
名前                    バージョン    リポジトリー    説明
ibm-websphere-liberty   1.9.0         local-charts   WebSphere Liberty is a fast, dynamic, and easy-to-use Java application
                                                     server.
```

- (5) ICPコンソールから確認します。ICP Management Consoleにログインし、右上のメニューから、「カタログ」を選択します。名前がibm-webshere-libertyのエントリーがあることを確認します。

目次(3章)

Helm Chartが作成したリソースの確認

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

– Helm Chartをインストールした際に作成されるICP(k8s)リソースの一覧と状況の表示

```
# helm status <Helmリソース名> --tls
```

インストールされているHelmリリース名は「helm list –namespace <namespace名> --tls」コマンドで確認できます。

– コマンド出力例

```
$ helm status wlplocal02 --tls
LAST DEPLOYED: Fri Mar  8 16:13:03 2019
NAMESPACE: yama-ns
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/Ingress
NAME                       HOSTS  ADDRESS         PORTS  AGE
wlplocal02-ibm-websphere   *      9.188.124.218   80     2d

==> v1/Pod(related)
NAME                                       READY  STATUS   RESTARTS  AGE
wlplocal02-ibm-websphere-6b5d5f99b9-7d89h  1/1    Running  0         2d

(略)

==> v1/Service
NAME                       TYPE       CLUSTER-IP   EXTERNAL-IP  PORT(S)   AGE
wlplocal02-ibm-websphere   ClusterIP  10.0.99.127  <none>       9080/TCP  2d

==> v1/Deployment
NAME                       DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
wlplocal02-ibm-websphere   1        1        1           1          2d
```

作成された「Ingresst」の名前と状況

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

Helm Chartが作成したリソースの確認

■ 作成されたリソースのマニフェストファイルは、「helm get」コマンドで確認できます。

– Helm Chartをインストールした際に使用されたICP(k8s)リソースのマニフェストファイルの表示

```
# helm get <Helmリソース名> --tls
```

インストールされているHelmリリース名は「helm list –namespace <namespace名> --tls」コマンドで確認できます。

– コマンド出力例

```
$ helm get mylibertyapp01 --tls
REVISION: 1
RELEASED: Wed Jul  3 18:40:57 2019
CHART: ibm-websphere-liberty-1.9.0
USER-SUPPLIED VALUES:
(略)
deployment:
  annotations: {}
  labels: {}
(略)
---
# Source: ibm-websphere-liberty/templates/shared.yaml
# SLT: 'slt.deployment' from templates/_deployment.tpl
apiVersion: apps/v1
kind: Deployment

metadata:
  name: mylibertyapp01-ibm-websp
  labels:
    chart: "ibm-websphere-liberty-1.9.0"
    app: mylibertyapp01-ibm-websp
    release: "mylibertyapp01"
    heritage: "Tiller"
(略)
```

Helm Chartのインストール時にHelm変数ファイルで設定したパラメーターの設定値と、Helm Chart内に定義されていたテンプレートファイルにHelmの変数ファイルで設定したパラメーターの設定値をマージして実際にICP(k8s)リソースを作成する際に適用されたマニフェストファイルを確認することができます。

Helm Chartのパラメーターの設定値を表示

適用された「Deployment」のマニフェストファイルを表示

Helm Chartが作成したリソースの確認

■ 作成されたリソースのパラメーターの設定値は、「helm get values」コマンドで確認できます。

– Helm Chartをインストールした際に使用されたHelm Chartのパラメーターの設定値の表示

```
# helm get values <Helmリソース名> --tls
```

インストールされているHelmリリース名は「helm list –namespace <namespace名> --tls」コマンドで確認できます。

– コマンド出力例

```
$ helm get values mylibertyapp01 --tls
(略)
deployment:
  annotations: {}
  labels: {}
(略)
image:
  extraEnvs: []
  extraVolumeMounts: []
  license: accept
  lifecycle: {}
  livenessProbe: {}
  pullPolicy: IfNotPresent
  readinessProbe: {}
  repository: mycluster.icp:8500/handson/mylibertyapp
  security: {}
  serverOverridesConfigMapName: ""
  tag: "1.0"
ingress:
  annotations: {}
  enabled: false
(略)
```

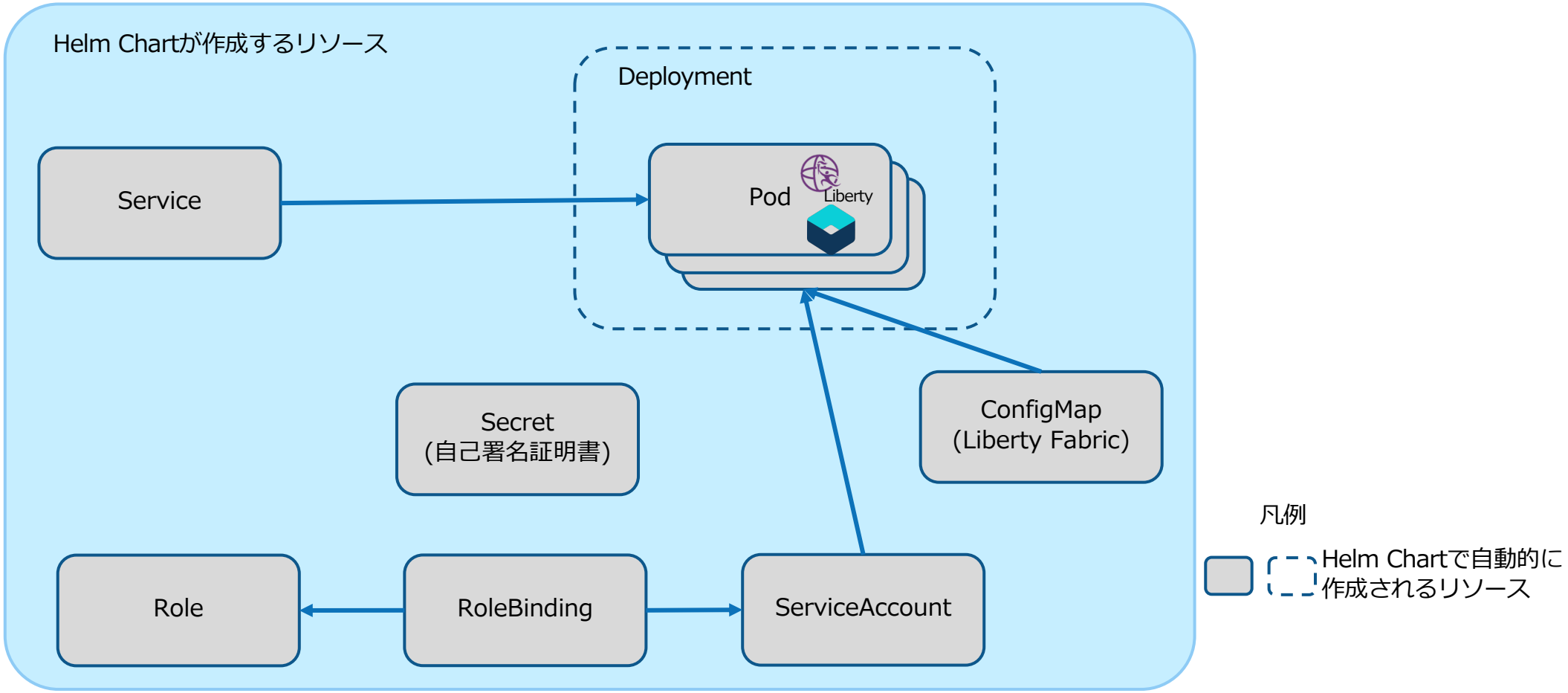
Helm Chartのインストール時に適用したHelm変数ファイルで設定したパラメーターの設定値確認することができます。

Helm Chartのパラメーターの設定値を表示

(補足)Helm Chartによって作成されるKubernetesリソース一覧

■ Helm Chart概要図

– WAS LibertyのHelm Chartをデフォルトの設定値でインストールした場合に作成されるKubernetesリソース



(補足)Helm Chartによって作成されるKubernetesリソース一覧

■ リソース一覧

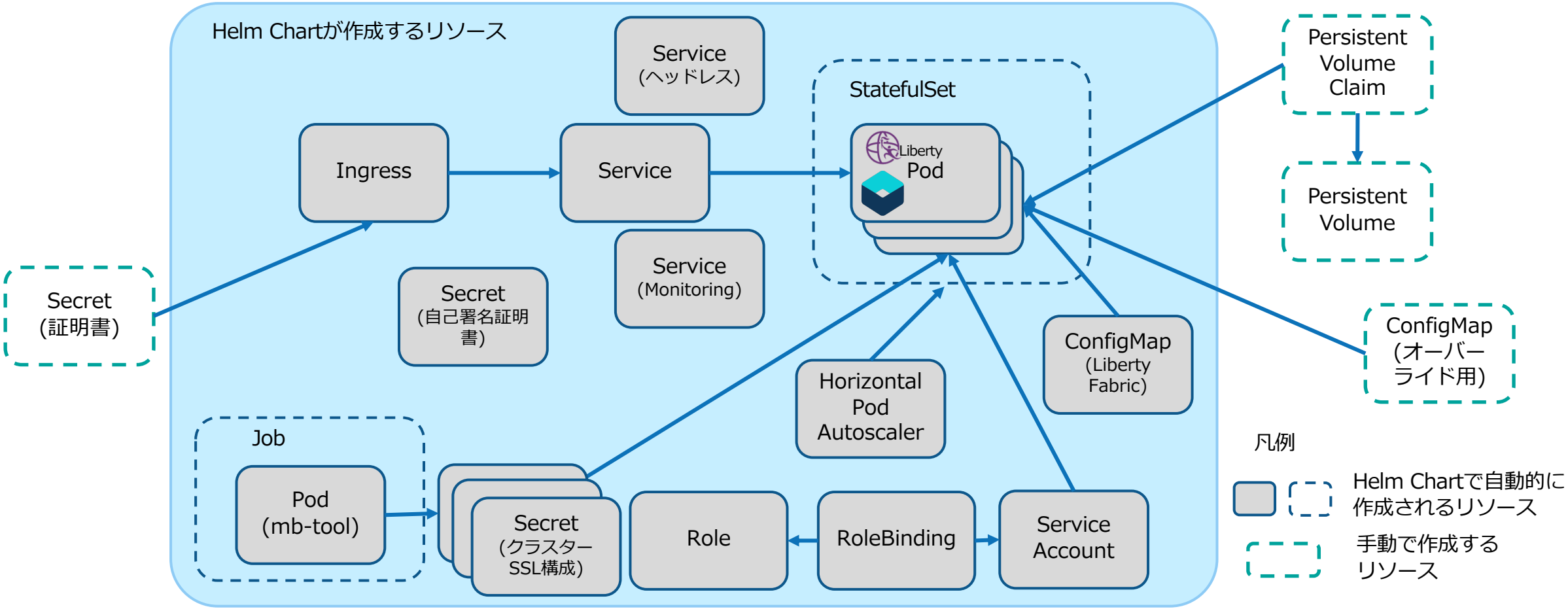
– WAS LibertyのHelm Chartをデフォルトの設定値でインストールした場合に作成されるKubernetesリソース

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WAS LibertyはDeploymentとして実行される。
Service	service.enable=true	WAS Libertyへのアクセスを負荷分散するServiceリソース。Ingressを利用しない場合はNodePortでのアクセスとなる。
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 (詳細は第11章を参照)。
Secret(自己署名証明書)	ssl.enabled=true	SSLを利用する場合に使用可能な自己署名証明書と秘密鍵を格納したSecret。Ingressを利用しない場合は利用されない。
ServiceAccount	rbac.install=true	WAS LibertyのPodを実行するServiceAccount。
Role	rbac.install=true	WAS 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Kubernetesリソース一覧

■ Helm Chart概要図

– WAS LibertyのHelm Chartを設定値を変更してインストールした場合に作成されるKubernetesリソースの例



(補足)Helm Chartによって作成されるKubernetesリソース一覧

■ リソース一覧

– WAS LibertyのHelm Chartを設定値を変更してインストールした場合に作成されるKubernetes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、WAS LibertyはStatefulSetとして実行される。
Service(ヘッドレス)		StatefulSetの場合、ヘッドレスサービス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WAS 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WAS LibertyへのIngress Controller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)		複数の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	使用状況に応じてPodの数を増やす場合に設定するHPAリソース。なお、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ICP(Kubernetes)リソース一覧

■ リソース一覧

– WAS Libertyの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ICP(Kubernetes)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を格納するためのリソース。
PrsistentVolumeClaim	n/a	永続ログ等で特定のPV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。(詳細は後述の「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)
Secret(証明書)	n/a	自ら取得した証明書を利用する場合、Secretに格納しIngressに設定する。

Helm Chartを使用した運用操作

- helmコマンドを使用してPodの運用ができます。
  - (以下の例では変更内容を示すために「--set」で値を変更していますが、変更内容を残す必要がある場合は、Helmの変数ファイルのパラメータの設定値を更新して「helm upgrade」コマンドを実行する方法をお勧めします。)

helm upgrade で変更できる内容には制限があります

  - レプリカ数の増減

```
# helm upgrade wlplocal02 -f ./valueswlp.yaml --set replicaCount=3 ./ibm-websphere-liberty --tls
```

  - 全Podの停止 (レプリカ数を「0」に設定)

```
# helm upgrade wlplocal02 -f ./ valueswlp.yaml --set replicaCount=0 ./ibm-websphere-liberty --tls
```

  - Dockerイメージの更新

```
# helm upgrade wlplocal02 -f ./ valueswlp.yaml --set image.tag=v200 ./ibm-websphere-liberty --tls
```

- Dockerイメージの名前やtagを変更すると、strategy/updateStrategyに従ってPodが更新されます。(strategy/updateStrategyの変更に関しては後述(4章)参照)
  - RollingUpdate: ローリング・アップデートが行われ、すぐにイメージが更新されます。
  - OnDelete: Podが削除されるまで古いイメージのまま動き続けます。

Deployment

```
spec:
 (略)
 "strategy": {
      "type": "RollingUpdate",
      "rollingUpdate": {
        "maxUnavailable": "25%",
        "maxSurge": "25%"
      }
```

StatefulSet

```
spec:
  (略)
  updateStrategy:
    type: OnDelete
```

Helm Chartを使用した運用操作

– リリースされている内容の表示

- Helm Chartのインストール時に指定したHelmのパラメーターの設定値と適用されたICP(k8s)リソースのマニフェストファイルの表示

```
# helm get wlplocal02 --tls
```

- Helm Chartのインストール時に指定したHelmのパラメーターの設定値の表示

```
# helm get values wlplocal02 --tls
```

– 更新履歴の確認

```
# helm history wlplocal02 --tls
```

– 履歴内のリビジョンを復元

```
# helm rollback wlplocal02 2 --tls
```

復元するリビジョンの番号(左記の例では2)を指定して実行する

– アンインストール

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

```
# helm delete wlplocal02 --tls
```

履歴も残して削除する場合

- 履歴も含めて完全に削除する場合は「--purge」オプションをつけて「helm delete」する必要があります。

```
# helm delete --purge wlplocal02 --tls
```

履歴も含めて完全に削除する場合

(補足)helmコマンドリファレンス

- helmコマンドの利用方法、オプションは以下のリンク先をご参照ください。
  - Helm Documentation / HELM
  - https://helm.sh/docs/helm/

helmコマンド	説明	参考URL
helm create	指定された名前で新しいHelm Chartを作成する	https://helm.sh/docs/helm/#helm-create
helm delete	対象のHelmリリースを削除する	https://helm.sh/docs/helm/#helm-delete
helm get	対象のリリースのHelmのパラメーターとマニフェストファイルの表示	https://helm.sh/docs/helm/#helm-get
helm history	インストールの更新履歴の表示	https://helm.sh/docs/helm/#helm-history
helm init	helmコマンドの初期化	https://helm.sh/docs/helm/#helm-init
helm inspect values	Helm Chartからパラメーターとデフォルト設定値を入手する	https://helm.sh/docs/helm/#helm-inspect-values
helm install	Helm Chartのインストール	https://helm.sh/docs/helm/#helm-install
helm lint	Helm Chartのファイルをパッケージ化する際のコードの確認	https://helm.sh/docs/helm/#helm-lint
helm list	インストールされたHelm Chartの一覧表示	https://helm.sh/docs/helm/#helm-list
helm package	Helm Chartファイルのパッケージ化	https://helm.sh/docs/helm/#helm-package
helm repo list	Helmのリポジトリーのリスト表示	https://helm.sh/docs/helm/#helm-repo-list
helm rollback	インストールしたリリース一覧の表示	https://helm.sh/docs/helm/#helm-rollback
helm search	Helmのリポジトリーに登録されているHelm Chartの検索	https://helm.sh/docs/helm/#helm-search
helm status	インストールしたHelm Chartの状況の表示	https://helm.sh/docs/helm/#helm-status
helm upgrade	対象Helm Chartの新しいリリースのアップグレード	https://helm.sh/docs/helm/#helm-upgrade

目次(3章)

server.xmlとserver.xmlフラグメント

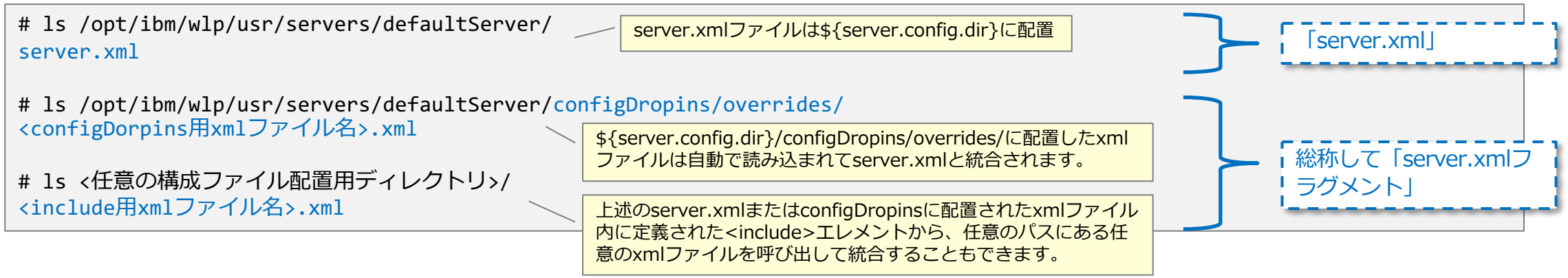
WAS Liberty Knowledge Center / Liberty の手動による管理
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_env.html
WAS Liberty Knowledge Center / 構成ファイル内での include エレメントの使用
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_includes.html
WAS Liberty Knowledge Center / 構成ドロップイン (dropins) フォルダーを使用したサーバー構成の指定
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_setup_dropins.html

■ server.xmlの構成

– server.xmlはWAS Libertyの構成ファイルです。

– WAS Libertyの構成や設定を一つのserver.xml内に全て定義することもできますし、「configDropins/overrides」ディレクトリや「include」エレメントを使用して複数のxmlファイルから構成情報を統合して定義することもできます。

• (例) ${server.config.dir}が「/opt/ibm/wlp/usr/servers/defaultServer/」の場合



– 本ガイドでは、元々の「server.xml」と区別して、「configDropins/overrides」ディレクトリや「include」エレメントで読み込まれるxmlファイルを総称して「server.xmlフラグメント」と呼びます。

• 後続の章で、「server.xmlにxxxの設定を記載」のような記述がある場合は、server.xmlまたはserver.xmlフラグメントのいずれかにその設定を定義することを意味します。

ICP環境でのWAS Libertyのserver.xmlの設定方法

■ ICP環境でWAS Libertyのserver.xmlまたはserver.xmlフラグメントの設定方法

– ICP環境では、以下のような箇所でserver.xmlまたはserver.xmlフラグメントを設定することができます。

- IBM提供のWAS LibertyのDockerイメージでは、「/opt/ibm/wlp/usr/servers/defaultServer/」が「/config」にリンクされています。
- （server.xmlなどの構成ファイルの配置ディレクトリは「/config/」となります。）

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(0)	Dockerイメージ	IBM提供のWAS LibertyのDockerイメージに含まれるserver.xmlファイル	server.xml	• /config/server.xml
(1)		作成したserver.xmlをコピーしてDockerイメージをビルド	server.xml	• /config/server.xml
(2)		WAS LibertyのDockerイメージが提供する機能(変数ARGとシェル)を利用してserver.xmlフラグメントをコピーしてDockerイメージをビルド	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/mp-monitoring.xml など
(3)	Helm ChartとConfigMap	server.xmlフラグメントをConfigMapとして事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/server-overrides.xml
(4)		(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント）

本ガイドの後続の章では主にこの3つのいずれかの方法を採用しています。

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(0)	Dockerイメージ	IBM提供のWAS LibertyのDockerイメージに含まれるserver.xmlファイル	server.xml	• /config/server.xml

– IBM提供のWAS LibertyのDockerイメージに含まれるserver.xmlファイルは以下の箇所で確認できます。
  • DockerHub / websphere-liberty(Official Images)で、使用したいDockerイメージのTagをクリックすると該当イメージをビルドする際に使用されたDockerfileが表示されます。
  • 表示された画面で一つ上のディレクトリを選択すると含まれているserver.xmlがあります。
    – DockerHub / websphere-liberty(Official Images)
    – https://hub.docker.com/_/websphere-liberty

  • IBM提供のWAS LibertyのDockerイメージでは、「/opt/ibm/wlp/usr/servers/defaultServer/」が「/config」にリンクされています。
  • 従って、server.xmlは「/config/server.xml」に配置されています。

(例)コンテナ内のserver.xmlのパス

```
default@mylibertyapp01-ibm-websp-86ccbf4448-bdrbt:/$ ls -l /config/server.xml
-rw-r--r--. 1 root root 756 Dec 21  2018 /config/server.xml
default@mylibertyapp01-ibm-websp-86ccbf4448-bdrbt:/$
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(1)	Dockerイメージ	作成したserver.xmlをコピーしてDockerイメージをビルド	server.xml	• /config/server.xml

– Dockerイメージビルド時に自分の作成したserver.xmlをコピーします。

(例)Dockerfileの例

```
FROM websphere-liberty:19.0.0.6-javaee8
COPY --chown=1001:0 Sum.war /config/dropins/
COPY --chown=1001:0 server.xml /config/
RUN installUtility install --acceptLicense defaultServer
```

IBM提供のWAS LibertyのDockerイメージ(tag: 19.0.0.6-javaee8)をベースイメージとして使用する場合

– Helm Chartをインストール後、コンテナ内で確認するとコピーしたserver.xmlに設定が置き換わっています。

(例)コンテナ内のserver.xmlのパス

```
default@mylibertyapp01-ibm-websp-86ccbf4448-bdrbt:/$ ls -l /config/server.xml
-rw-r--r--. 1 root root 756 Dec 21  2018 /config/server.xml
default@mylibertyapp01-ibm-websp-86ccbf4448-bdrbt:/$
default@mylibertyapp01-ibm-websp-86ccbf4448-bdrbt:/$ cat /config/server.xml
<自分の作成したserver.xmlに定義していた内容が表示>
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(2)	Dockerイメージ	WAS LibertyのDockerイメージが提供する機能(変数ARGとシェル)を利用してserver.xmlフラグメントをコピーしてDockerイメージをビルド	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/mp-monitoring.xml など

– IBM提供のWAS LibertyのDockerイメージが提供するEnterprise Function機能を利用すると、事前定義されたserver.xmlフラグメント(snippet)が自動でイメージに組み込まれてビルドされます。
  • GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Enterprise Functionality
  • https://github.com/WASdev/ci.docker#enterprise-functionality

Enterprise Functionality

This section describes the optional enterprise functionality that can be enabled via the Dockerfile during `build` time, by setting particular argument (`ARG`) or environment variable (`ENV`) and calling `RUN configure.sh` . Each of these options trigger the inclusion of specific configuration via XML snippets, described below:

- `HTTP_ENDPOINT`
  - Decription: Add configuration properties for an HTTP endpoint.
  - XML Snippet Location: http-ssl-endpoint.xml when SSL is enabled. Otherwise http-endpoint.xml
- `MP_HEALTH_CHECK`
  - Decription: Check the health of the environment using Liberty feature `mpHealth-1.0` (implements MicroProfile Health).
  - XML Snippet Location: mp-health-check.xml
- `MP_MONITORING`
  - Decription: Monitor the server runtime environment and application metrics by using Liberty features `mpMetrics-1.1` (implements Microprofile Metrics) and `monitor-1.0` .
  - XML Snippet Location: mp-monitoring.xml
  - Note: With this option, `/metrics` endpoint is configured without authentication to support the environments that do not yet support scraping secured endpoints.

(例)MP_MONITORINGを有効化した場合に組み込まれるserver.xmlフラグメント(mp-monitoring.xml)の内容

10 lines (8 sloc) | 227 Bytes

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <server>
3      <featureManager>
4          <feature>mpMetrics-1.1</feature>
5          <feature>monitor-1.0</feature>
6      </featureManager>
7
8      <mpMetrics authentication="false" />
9  </server>
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(2)	Dockerイメージ	WAS LibertyのDockerイメージが提供する機能(変数ARGとシェル)を利用してserver.xmlフラグメントをコピーしてDockerイメージをビルド	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/mp-monitoring.xml など

– (例) MP_MONITORINGを有効化する場合

• Dockerfile内のARG変数で「ARG MP_MONITORING=true」と設定し、「RUN configure.sh」でシェルを実行させると、「mpMetrics-1.1」と「monitor-1.0」フィーチャーの追加と構成が事前定義されたserver.xmlフラグメント（mp-monitoring.xml）が自動でイメージに組み込まれてビルドされます。

（Dockerfileの例）

```
FROM websphere-liberty:19.0.0.6-webProfile8
COPY --chown=1001:0  Sample.war /config/dropins/

# Optional functionality
ARG MP_MONITORING=true

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

「MP_MONITORING=true」とすることで、事前定義されたmp-monitoring.xmlが自動でイメージに組み込まれます。

configure.shでは、mp-monitoring.xmlの組み込みと、「installUtility install –acceptLicense defaultServer」コマンドを実施しフィーチャーのインストールも実施します。
(configure.shはtag:kernelのイメージに組み込まれているため、記載するだけで実行できます。)

• (補足)server.xmlフラグメント(mp-monitoring.xml)はコンテナ内の「configDropins/overrides」のディレクトリに配置されます。
  – /config/configDropins/overrides/mp-monitoring.xml

– 上述リンク先を参照し、該当する機能がある場合はこの方法を取ることもできます。

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(3)	Helm ChartとConfigMap	server.xmlフラグメントをConfigMapとして事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/server-overrides.xml

– 更新したい設定を追加したserver.xmlフラグメントをconfigMapとして事前に作成し、Helmの変数ファイルの「image.serverOverridesConfigMapName」パラメーターの設定値にこのconfigMapを指定する場合
  • ビルド時にserver.xmlをコピーせず、Helm Chart側でconfigmapを使用してserver.xmlフラグメントを配置する場合は、不足しているフィーチャーはDockerfileに明示的に記載してインストールしておく必要があります。

– 手順概要
  • (1) 更新したい設定を追加したserver.xmlフラグメントを作成します。ファイル名は「server-overrides.xml」の必要があります。

```
$ cat server-overrides.xml
<server>
    <featureManager>
        <!-- Enable monitoring -->
        <feature>mpMetrics-1.1</feature>
(略)
</server>
```

  • (2) 作成したserver.xmlフラグメントからConfigMapを作成します。

```
$ kubectl create configmap --save-config custom-server-overrides --from-file=./server-overrides.xml -n handson
configmap/custom-server-overrides created
$
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(3)	Helm ChartとConfigMap	server.xmlフラグメントをConfigMapとして事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/server-overrides.xml

- (3) Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-monitor-test01.yaml
```

- (4) 抽出した変数ファイルの「image.serverOverridesConfigMapname」に先ほど作成したConfigMapの名前を設定します。

```
images:
(略)
  serverOverridesConfigMapName: "custom-server-overrides"
```

server-overrides.xmlから作成したconfigmap名を指定します。

- (5) 編集した変数ファイルを使用して、Helm Chartをデプロイします。

```
helm upgrade --install monitor-ingress-ssl -f ./values-monitor-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (6) ConfigMapに定義したserver.xmlフラグメント(server-overrides.xm.)はコンテナ内の「configDropins/overrides」のディレクトリに配置されます。

(例)コンテナ内の/config/confitDropins/overrides/server-overrides.xml

```
default@monitor-ingress-ssl-ibm-6dc7668cb5-7jlql:/$ cat /config/configDropins/overrides/server-overrides.xml
<server-overrides.xmlに定義していた内容が表示>
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(4)	Helm ChartとConfigMap	(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント）

– IBM提供のWAS LibertyのHelm Chartをデプロイすると、自動でinclude用のserver.xmlフラグメントが作成されます。
  • /config/configDropins/overrides/include-configmap.xml

```
default@monitor-ingress-ssl-ibm-6dc7668cb5-7jlql:/$ cat /config/configDropins/overrides/include-configmap.xml
<server>
  <include optional="true" location="/etc/wlp/configmap/server.xml"/>
  <include optional="true" location="/etc/wlp/configmap/cluster-ssl.xml"/>
</server>default@monitor-ingress-ssl-ibm-6dc7668cb5-7jlql:~$
```

– includeされている、server.xmlフラグメントは、最初は何も設定されていません。
  • /etc/wlp/configmap/server.xml

```
default@monitor-ingress-ssl-ibm-6dc7668cb5-7jlql:/$ cat /etc/wlp/configmap/server.xml
<server>
  <!-- Customize the running configuration. -->
</server>
```

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(4)	Helm ChartとConfigMap	(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント)

– includeされているetc/wlp/configmap/server.xmlは、Helm Chartインストール時に自動で作成されているConfigMapに定義されています。このConfigMapのData部分の「server.xml」を編集することで設定が反映されます。
  • ConfigMapの名前は、「<Helmリリース名>-ibm」です。

```
Mac: lab75 $ kubectl describe configmaps monitor-ingress-ssl-ibm
Name:         monitor-ingress-ssl-ibm
(略)
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. -->
</server>
Events:  <none>
```

「<Helmリリース名>-ibm」という名前のConfigMapのData部分のserver.xml部分を「kubectl edit」コマンドで修正することで、コンテナ内の/etc/wlp/configmap/server.xmlが更新されます。

ICP環境でのWAS Libertyのserver.xmlの設定方法

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(4)	Helm ChartとConfigMap	(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント)

– ConfigMapを修正するとincludeされているserver.xmlフラグメントの設定も更新されます。
  • /etc/wlp/configmap/server.xml

```
default@monitor-ingress-ssl-ibm-6dc7668cb5-7jlql:/$ cat /etc/wlp/configmap/server.xml
<server>
  (ConfigMapで更新した内容が反映)
</server>
```

– (補足)この方法では、変更内容をHelm Chartの変数ファイルやConfigMapのマニフェストファイルなどのようにコードで管理でき無くなります。あくまでもテスト用の一時的な設定の反映のみで使用されることをお勧めします。

4. Helm Chartの利用

目次(4章)

- Helm Chartの利用
- Helm Chartの変数ファイルによる設定変更
  - Readiness Probeの設定値の変更
  - Liveness Probeの設定値の変更
  - Deploymentの設定値の変更
  - Serviceの設定値の変更
  - Ingressの設定値の変更
    - 仮想PATHの利用(HTTP)
    - 仮想PATHの利用(HTTPS)
    - 名前ベースの仮想ホストの利用(HTTP)
- (補足)Helm Chartの変更
  - Helm Chartの変更
  - WAS LibertyのHelm Chartの構造
  - Helm Chartの変更手順
  - Helm Chartの変更
    - Deploymentの変更
    - StatefulSetの変更

目次(4章)

Helm Chartの利用

- IBM提供のWAS LibertyのHelm Chartのインストールを方法の例を説明します。

- 要件やユースケースに応じてHelm Chartの変数ファイルのパラメーターの設定値を変更してHelm Chartをインストールします。
  - 4章では、主に共通でよく設定をする項目についていくつか取り上げ具体的な設定手順を説明します。
  - 個々の要件に応じたより詳細な設定手順などは、5章以降で説明します。
    - (ログ、モニタリング、セッション・パーシステンスなど)

- 基本的にはHelm Chartの提供する機能だけで主な要件は実現することができます。
  - ただし、一部、Helm Chart自体を修正する必要がある項目もあります。
  - 4章の後半では、補足として、Helm Chartを修正する必要がある項目の修正方法について説明します。

目次(4章)

Readiness Probeの設定値の変更

- WAS LibertyのHelm Chartの変数ファイルで、Readiness Probeに関する設定値を変更する手順を示します。

– 設定概要
  - Readiness Probeを設定することで、Podがトラフィックを受け入れられる状態かどうかを確認することができます。
  - トラフィックを受けられない状態のPodがある場合は、サービスのエンドポイントから外し、トラフィックを流さないようにします。
  - Podは再起動されません。

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

– 前提作業
  - 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
  - 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

  - 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

  - cloudctlコマンドでICPにログインしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

Readiness Probeの設定値の変更

– 手順概要

- (1) Helm Chartから変数ファイルを抽出します。

```
# helm inspect values ibm-websphere-liberty-1.8.0.tgz > readinessProbeTest.yml
```

- (2) 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。
  – 以下の青字部分を変更します。

(例)readinessProbeTest.yml

```
(略)
image:
  repository: mycluster.icp:8500/default/websphere-liberty        # 使用するイメージ名
  tag: javaee8                                                    # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                               # Licenseの承認
(略)
readinessProbe:
    httpGet:
      path: /
      port: 9443
      scheme: HTTPS
    initialDelaySeconds: 20                 # コンテナが起動してから監視を始めるまでの時間
    timeoutSeconds: 1                       # 監視のレスポンスを待つ時間
    periodSeconds: 5                        # 監視を走らせる間隔
    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割り振り対象から削除するか
(略)
```

Readiness Probeの設定値の変更

- (3) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
# helm install --name wlp-test-readiness -f ./readinessProbeTest.yml ./ibm-websphere-liberty-1.8.0.tgz --tls
```

- (4) 正常にインストールできたか確認します。

```
# helm status wlp-test-readiness –tls
LAST DEPLOYED: Fri Mar  1 15:11:48 2019
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/Pod(related)
NAME                                        READY  STATUS   RESTARTS  AGE
wlp-test-readiness-ibm-w-856b9b6475-v8rkr   1/1    Running  0         32m
==> v1/Secret
NAME                            TYPE    DATA  AGE
wlp-test-readiness-ibm-w-tls    Opaque  2     32m
==> v1/ConfigMap
NAME                        DATA  AGE
wlp-test-readiness-ibm-w    2     32m
==> v1/RoleBinding
NAME                        AGE
wlp-test-readiness-ibm-w    32m
==> v1/Service
NAME                        TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
wlp-test-readiness-ibm-w    NodePort   10.0.85.176   <none>       9443:30881/TCP   32m
==> v1/Deployment
NAME                        DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
wlp-test-readiness-ibm-w    1        1        1           1          32m
(略)
```

DeploymentのAVAILABLEが0でないことを確認

Readiness Probeの設定値の変更

- (5) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get deployment <deployment name> -o yaml」コマンドでDeploymentの設定内容を確認します。

```
# kubectl get deployment wlp-test-readiness -o yaml
  (略)
    readinessProbe:
          failureThreshold: 5
          httpGet:
            path: /
            port: 9443
            scheme: HTTPS
          initialDelaySeconds: 20
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 1
  (略)
```

Readiness Probeとして設定した値が反映していることを確認します。

- (6) Webブラウザで表示されることを確認します。
  - ICP Management Consoleの「メニュー」→「ワークロード」→「Helmリリース」でインストールしたHelmリリース名をクリックし、右上の「起動」ボタンを押します。
  - もしくは、Webブラウザに直接以下URLで表示されることを確認します。
  - "https://<Proxy NodeのIPアドレス>:<Serviceで表示されたPORT(3xxxx)>" (NodePortの場合)
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

目次(4章)

Liveness Probeの設定値の変更

■ WAS LibertyのHelm Chartの変数ファイルで、Liveness Probeに関する設定値を変更する手順を示します。

– 設定概要
- Liveness Probeを設定することで、Podが正常に稼働していることを確認できます。
- Podが正常に稼働していない場合は、Podを自動で再起動されます。
- 但し、再起動によりコンテナー内に出力されたログなどは消失します。（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。）

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

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

Liveness Probeの設定値の変更

– 手順概要

- (1) Helm Chartから変数ファイルを抽出します。

```
# helm inspect values ibm-websphere-liberty-1.8.0.tgz > livenessProbeTest.yml
```

- (2) 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。
  – 以下の青字部分を変更します。

(例) livenessProbeTest.yml

```
(略)
image:
  repository: mycluster.icp:8500/default/websphere-liberty          # 使用するイメージ名
  tag: javaee8                                                      # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                                 # Licenseの承認
(略)
livenessProbe:
  # Example:
  # livenessProbe:
     httpGet:
       path: /
       port: 9443
     initialDelaySeconds: 20                                # コンテナが起動してから監視を始めるまでの時間
     timeoutSeconds: 1                                      # 監視のレスポンスを待つ時間
     periodSeconds: 5                                       # 監視を走らせる間隔
     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
(略)
```

Liveness Probeの設定値の変更

- (3) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
# helm install --name wlp-test-liveness -f ./livenessProbeTest.yml ./ibm-websphere-liberty-1.8.0.tgz --tls
```

- (4) 正常にインストールできたか確認します。

```
# helm status wlp-test-liveness –tls
LAST DEPLOYED: Fri Mar  1 15:11:48 2019
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/RoleBinding
NAME                       AGE
wlp-test-liveness-ibm-we   1h
==> v1/Service
NAME                       TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
wlp-test-liveness-ibm-we   NodePort   10.0.5.192   <none>        9443:32242/TCP   1h
==> v1/Deployment
NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
wlp-test-liveness-ibm-we   1         1         1            1           1h
==> v1/Pod(related)
NAME                                       READY   STATUS    RESTARTS   AGE
wlp-test-liveness-ibm-we-fd5c9c7b8-dntvs   1/1     Running   17         1h
==> v1/Secret
NAME                           TYPE     DATA   AGE
wlp-test-liveness-ibm-we-tls   Opaque   2      1h
==> v1/ConfigMap
NAME                       DATA   AGE
wlp-test-liveness-ibm-we   2      1h
(略)
```

DeploymentのAVAILABLEが0でないことを確認

Liveness Probeの設定値の変更

- (5) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get deployment <deployment name> -o yaml」コマンドでDeploymentの設定内容を確認します。

```
# kubectl get deployment wlp-test-liveness -o yaml
  (略)
    livenessProbe:
          failureThreshold: 5
          httpGet:
            path: /
            port: 9443
            scheme: HTTP
          initialDelaySeconds: 20
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 1
  (略)
```

Liveness Probeとして設定した値が反映していることを確認します。

- (6) Webブラウザで表示されることを確認します。
  - ICP Management Consoleの「メニュー」→「ワークロード」→「Helmリリース」でインストールしたHelmリリース名をクリックし、右上の「起動」ボタンを押します。
  - もしくは、Webブラウザに直接以下URLで表示されることを確認します。
  - "https://<Proxy NodeのIPアドレス>:<Serviceで表示されたPORT(3xxxx)>" (NodePortの場合)
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

目次(4章)

Deploymentの設定値の変更

■ WAS LibertyのHelm Chartの変数ファイルで、Deploymentに関する設定値を変更する手順を示します。

– 設定概要
- WAS Libertyのログを永続化しない場合はPodは「Deployment」として作成されます。(デフォルト)(5章参照)
- ユーザーはまずWAS LibertyのDockerイメージにアプリケーションやWAS Libertyの構成ファイル(server.xml)などをコピーし新規にDockerイメージを作成後、ICPプライベートDockerレジストリーに登録します。
- そのDockerイメージのイメージ名とtag名をHelm Chartの変数ファイルで指定します。
- また例としてPodのreplica数を2とする設定方法も記載します。(デフォルトはreplica数は1)

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

Deploymentの設定値の変更

– 手順概要

- (1) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-1.9.0.tgz > deploy-test01.yml
```

- (2) 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。
  – 以下の青字部分を変更します。

(例) deploy-test01.yml

```
(略)
image:
  repository: mycluster.icp:8500/handson/mylibertyapp          # 使用するイメージ名
  tag: "1.0"                                                    # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                             # Licenseの承認
(略)
logs:
  persistLogs: false                        # ログの永続化はfalse(デフォルト)
  persistTransactionLogs: false             # トランザクション・ログの永続化はfalse(デフォルト)
(略)
replicaCount: 2                             # Podのreplica数を2に変更(デフォルトは1)
```

事前にICPプライベートDockerレジストリーに登録したDockerイメージ名とtag名を指定します。
(タグは「javaee-19.0.0.6」などのように文字列の場合は""を入れる必要はありませんが、「1.0」などのように数字だけの場合は""で囲む必要があります。)

※image.licenseはイメージの製品ライセンス・ステータスを選択します。
- 空欄：deploymentライセンス
- accept: 製品ライセンス

ログの永続化をしない場合は「Deployment」としてPodは作成されます。
(ログを永続化する場合は「Statefulset」になります。)(5章参照)

- (3) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。

```
$ helm install --dry-run --debug --name deploy-test01 -f ./deploy-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

Deploymentの設定値の変更

- (4) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm install --name deploy-test01 -f ./deploy-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (5) 正常にインストールできたか確認します。

```
$ helm status deploy-test01 --tls
LAST DEPLOYED: Sat Jul  6 17:10:24 2019
NAMESPACE: handson
STATUS: DEPLOYED

RESOURCES:
(略)
==> v1/Service
NAME                       TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
deploy-test01-ibm-websph   NodePort   10.0.84.151   <none>       9443:30383/TCP   39s

==> v1/Deployment
NAME                       DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
deploy-test01-ibm-websph   2        2        2           2          39s

==> v1/Pod(related)
NAME                                       READY  STATUS   RESTARTS  AGE
deploy-test01-ibm-websph-d8b695564-64t86   1/1    Running  0         39s
deploy-test01-ibm-websph-d8b695564-9xkwf   1/1    Running  0         39s
(略)
```

Deploymentが作成され、DESIREDとAVAILABLEの数が2となっていることを確認します。

replica数を2としたので、Podが2つ作成されていることを確認します。

Deploymentの設定値の変更

- (6) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get deployment <deployment name> -o yaml」コマンドでDeploymentの設定内容を確認します。

```
$ kubectl get deployment deploy-test01-ibm-websph -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
(略)
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  (略)
  template:
    (略)
    spec:
      (略)
      containers:
        (略)
        image: mycluster.icp:8500/handson/mylibertyapp:1.0
        imagePullPolicy: IfNotPresent
        livenessProbe:
(略)
status:
  availableReplicas: 2
  conditions:
  (略)
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
  updatedReplicas: 2
```

「replicas」(レプリカ数)の値が2となっていることを確認します。

指定したイメージ名とtag名が使用されていることを確認します。

Deploymentの設定値の変更

- (7) Helm ChartのデフォルトではNodePortのServiceが作成されますのでPort番号を確認します。

```
$ kubectl get service
NAME                          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
deploy-test01-ibm-websph   NodePort   10.0.84.151   <none>        9443:30383/TCP   22m
```

- (8) Webブラウザで表示されることを確認します。
  - ICP Management Consoleの「メニュー」→「ワークロード」→「Helmリリース」でインストールしたHelmリリース名をクリックし、右上の「起動」ボタンを押します。
  - もしくは、Webブラウザに直接以下URLで表示されることを確認します。
  - "https://<Proxy NodeのIPアドレス>:<Serviceで表示されたPORT(3xxxx)>" (NodePortの場合)
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

目次(4章)

- Helm Chartの利用
- Helm Chartの変数ファイルによる設定変更
  - Readiness Probeの設定値の変更
  - Liveness Probeの設定値の変更
  - Deploymentの設定値の変更
  - Serviceの設定値の変更
  - Ingressの設定値の変更
    - 仮想PATHの利用(HTTP)
    - 仮想PATHの利用(HTTPS)
    - 名前ベースの仮想ホストの利用(HTTP)
- (補足)Helm Chartの変更
  - Helm Chartの変更
  - WAS LibertyのHelm Chartの構造
  - Helm Chartの変更手順
  - Helm Chartの変更
    - Deploymentの変更
    - StatefulSetの変更

Serviceの設定値の変更

■ WAS LibertyのHelm Chartの変数ファイルでServiceに関する設定値を変更する手順を示します。

– 設定概要
- WAS LibertyのHelm Chartでインストールをすると、デフォルトでは外部接続のためにtypeがNodePortのServiceが作成されます。
- デフォルトではSSLが有効でHTTPS(9443)で接続するように設定されています。
- 該当の設定箇所を確認します。

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

Serviceの設定値の変更

– 手順概要

- (1) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-1.9.0.tgz > service-test01.yml
```

- (2) 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。
  – 以下の青字部分を変更します。

(例) service-test01.yml

```
(略)
image:
  repository: mycluster.icp:8500/handson/websphere-liberty          # 使用するイメージ名
  tag: 19.0.0.5-javaee8                                             # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                                 # Licenseの承認
(略)
service:
  enabled: true
  name: ""
  port: 9443                                                        # デフォルトは9443(HTTPの場合は9080に変更)
  targetPort: 9443                                                  # デフォルトは9443(HTTPの場合は9080に変更)
  type: NodePort                                                    # デフォルトはNodePort
(略)
ssl:
  enabled: true                                                     # デフォルトはtrue(HTTPの場合はfalseに変更)
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false
(略)
```

Serviceの設定値の変更

- (3) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します

```
$ helm install --dry-run --debug --name service-test01 -f ./service-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (4) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm install --name service-test01 -f ./service-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (5) 正常にインストールできたか確認します。

```
$ helm status service-test01 --tls
LAST DEPLOYED: Thu Jul  4 18:39:40 2019
NAMESPACE: handson
STATUS: DEPLOYED

RESOURCES:
==> v1/Deployment
NAME                      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
service-test01-ibm-websp  1        1        1           1          27s

==> v1/Pod(related)
NAME                                      READY  STATUS   RESTARTS  AGE
service-test01-ibm-websp-69f4d44c7d-jgddl  1/1    Running  0         27s

(略)

==> v1/Service
NAME                      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)         AGE
service-test01-ibm-websp  NodePort  10.0.36.195   <none>       9443:32380/TCP  27s
(略)
```

DeploymentのAVAILABLEが0でないことを確認

typeがNodePortのServiceが作成されていることを確認。

Serviceの設定値の変更

- (6) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get service」コマンドでServiceが作成されたことを確認します。

```
$ kubectl get service
NAME                          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
service-test01-ibm-websp      NodePort    10.0.36.195    <none>        9443:32380/TCP    96s
```

typeがNodePortのServiceが作成されていることを確認します。
NodePortのPort番号を確認します。

  - 「kubectl get service <service name> -o yaml」コマンドでServiceの設定内容を確認します。

```
$ kubectl get service service-test01-ibm-websp -o yaml
apiVersion: v1
kind: Service
(略)
spec:
  (略)
  ports:
  - name: https
    nodePort: 32380
    port: 9443
    protocol: TCP
    targetPort: 9443
  (略)
  type: NodePort
```

Serviceに設定した値が反映していることを確認します。

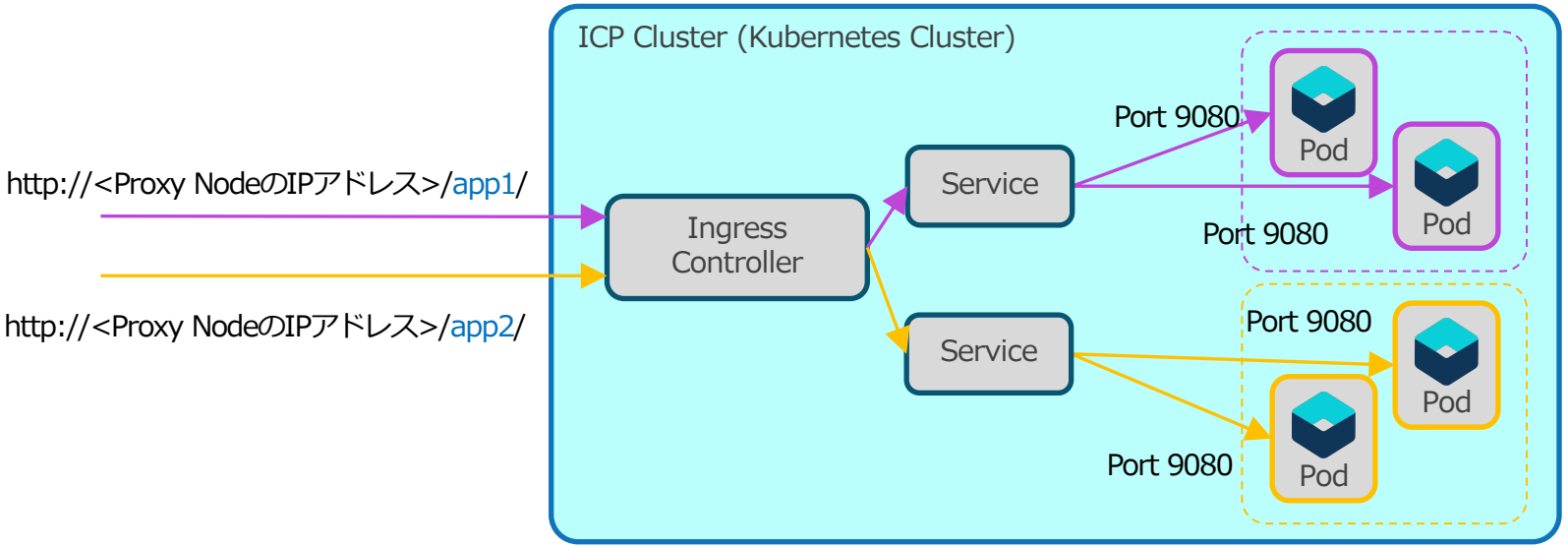
- (7) Webブラウザで表示されることを確認します。
  - ICP Management Consoleの「メニュー」→「ワークロード」→「Helmリリース」でインストールしたHelmリリース名をクリックし、右上の「起動」ボタンを押します。
  - もしくは、Webブラウザに直接以下URLで表示されることを確認します。
  - "https://<Proxy NodeのIPアドレス>:<Serviceで表示されたPORT(3xxxx)>" (NodePortの場合)
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

目次(4章)

- Helm Chartの利用
- Helm Chartの変数ファイルによる設定変更
  - Readiness Probeの設定値の変更
  - Liveness Probeの設定値の変更
  - Deploymentの設定値の変更
  - Serviceの設定値の変更
  - Ingressの設定値の変更
    - 仮想PATHの利用(HTTP)
    - 仮想PATHの利用(HTTPS)
    - 名前ベースの仮想ホストの利用(HTTP)
- (補足)Helm Chartの変更
  - Helm Chartの変更
  - WAS LibertyのHelm Chartの構造
  - Helm Chartの変更手順
  - Helm Chartの変更
    - Deploymentの変更
    - StatefulSetの変更

Ingressの設定値の変更：仮想PATHの利用(HTTP)

- WAS LibertyのHelm Chartの変数ファイルでIngressに関する設定値を変更する手順を示します。
  - 設定概要
    - ICPのIngress Controllerはコンテキストパスを変更することができます。
    - コンテキストパスを変更することで、同じパスを持った別のサービスへの処理を割り振ることが可能です。
    - Helm Chartの変数ファイルのIngressのPathのパラメーターを指定することで変更できます。
    - Ingress Controller接続では、Proxy NodeのIPアドレスを使用します。
    - Proxy NodeのIPアドレスが不明な場合は「kubectl get nodes –l proxy=true」コマンドで確認します。
    - HTTP接続、HTTPS接続どちらも可能です。まずはHTTP接続手順について記載します。
    - デフォルトで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）
    - Kubernetes Documentation / Ingress
    - https://kubernetes.io/docs/concepts/services-networking/ingress/



Ingressの設定値の変更：仮想PATHの利用(HTTP)

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

– 手順概要
- (1) Helm Chartから変数ファイルを抽出します。

```
# helm inspect values ibm-websphere-liberty-1.9.0.tgz > wlp-ingress01.yml
```

Ingressの設定値の変更：仮想PATHの利用(HTTP)

- (2) 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。
  - 以下の青字部分を変更します。

(例) wlp-ingress01.yml

```
(略)
image:
  repository: mycluster.icp:8500/default/websphere-liberty          # 使用するしたイメージ名
  tag: 19.0.0.3-javaee8                                             # 使用するしたイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                                 # Licenseの承認
(略)
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: NodePort
(略)
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: "/"
  path: "/wlp-path1"                                                # コンテキストパスを記載
  host: ""
  secretName: ""
  annotations:{}
(略)
```

Ingressの設定値の変更：仮想PATHの利用(HTTP)

- (3) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します

```
# helm install --debug --dry-run --name wlp-ingress01 -f ./wlp-ingress01.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (4) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
# helm install --name wlp-ingress01 -f ./wlp-ingress01.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (5) 正常にインストールできたか確認します。

```
# helm status wlp-ingress01 --tls
LAST DEPLOYED: Tue Apr  9 11:05:10 2019
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/Role
NAME                       AGE
wlp190-ingress01-ibm-web  1m
==> v1/RoleBinding
NAME                       AGE
wlp190-ingress01-ibm-web  1m
==> v1/Service
NAME                       TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
wlp-ingress01-ibm-web  ClusterIP  10.0.182.130  <none>       9080/TCP  1m
==> v1/Deployment
NAME                       DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
wlp-ingress01-ibm-web  1        0        0           0          1m
==> v1beta1/Ingress
NAME                       HOSTS  ADDRESS        PORTS  AGE
wlp-ingress01-ibm-web  *      9.xxx.xxx.xxx  80     1m
(略)
```

PORTが9080になっていることが確認できます。

「Ingress」が作成されています。

Ingressの設定値の変更：仮想PATHの利用(HTTP)

- (6) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get ingress <ingress name> -o yaml」コマンドでIngressの設定内容を確認します。

```
# kubectl get ingress wlp-ingress01-ibm-web -o yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    (略)
    ingress.kubernetes.io/affinity: cookie
    ingress.kubernetes.io/session-cookie-name: route
    (略)
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/session-cookie-name: route
    (略)
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: wlp190-ingress01-ibm-web
          servicePort: 9080
        path: /wlp-path01
status:
  loadBalancer:
    ingress:
    - ip: 9.xxx.xxx.xxx
```

デフォルトでActive Cookie方式のHTTPセッションのアフィニティーを維持できます。(nginxがサポートするのはCookieのみ)

外部からの接続用に設定したコンテキストパス

接続用のProxy NodeのIPアドレス

Ingressの設定値の変更：仮想PATHの利用(HTTP)

- (7) Webブラウザで表示されることを確認します。
  - Webブラウザに直接以下URLで表示されることを確認します。
  - "http://<Proxy NodeのIPアドレス>/<設定したpath名>/"
  - (例)"http://9.xxx.xxx.xxx/wlp-path01/"
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

最後の"/"(スラッシュ)を忘れずに！

目次(4章)

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

- WAS LibertyのHelm Chartの変数ファイルでIngressに関する設定値を変更する手順を示します。
  - 設定概要
    - Ingress ControllerでHTTPS(SSL)での接続設定も可能です。
    - SSLでの接続のためにサーバー証明書と鍵を利用します。
      - 証明書がすでにある場合は、TLS Secretとして登録します。
      - 証明書がない場合は、TLS Secretの指定をしないことで、自己署名証明書が自動的に作成されます。
    - サーバー証明書と鍵を予め登録する場合
      - NGINX Ingress Controller の仕様にあわせて、証明書と鍵は「PEM-encoded X.509, RSA (2048) secret」として登録します。
      - NGINX Ingress Controller / TLS/HTTPS
      - https://kubernetes.github.io/ingress-nginx/user-guide/tls/
      - TLS Secret は以下のようなコマンドで PEM ファイルから定義できます。

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

      - ここで作成したSecretを使用する場合は、Helm ChartのIngressに関するパラメーターで指定します。
    - 以降のページではSecretを指定せず、自動的に作成される自己署名証明書でHTTPS接続する設定について記載します。

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

– 手順概要
- (1) Helm Chartから変数ファイルを抽出します。

```
# helm inspect values ibm-websphere-liberty-1.9.0.tgz > wlp-ingress02.yml
```

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

- (2) 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。
  - 以下の青字部分を変更します。

(例) wlp-ingress02.yml

```
(略)
image:
  repository: mycluster.icp:8500/default/websphere-liberty          # 使用するイメージ名
  tag: 19.0.0.3-javaee8                                             # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                                 # Licenseの承認
(略)
service:
  enabled: true
  name: ""
  port: 9443                                                        # HTTPS接続なので9443を設定(デフォルト)
  targetPort: 9443                                                  # HTTPS接続なので9443を設定(デフォルト)
  type: NodePort
(略)
ssl:
  enabled: true                                                     # HTTPS接続なのでtrueを設定(デフォルト)
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false

ingress:
  enabled: true                                                     # Ingressを使用するのでtrueに変更
  rewriteTarget: "/"
  path: "/wlp-path02"                                               # コンテキストパスを記載
  host: ""
  secretName: ""                                  # Secretを作成した場合はここに記載。記載しない場合は自動的に作成される。
  annotations:{}
(略)
```

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

- (3) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します

```
# helm install --debug --dry-run --name wlp-ingress02 -f ./wlp-ingress02.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (4) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
# helm install --name wlp-ingress02 -f ./wlp-ingress02.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (5) 正常にインストールできたか確認します。

```
# helm status wlp-ingress02 --tls
LAST DEPLOYED: Tue Apr  9 13:14:33 2019
NAMESPACE: yama-ns
STATUS: DEPLOYED
RESOURCES:
==> v1/Secret
NAME                            TYPE    DATA  AGE
wlp-ingress02-ibm-web-tls  Opaque  2     2h
(略)
==> v1/Pod(related)
NAME                                      READY  STATUS   RESTARTS  AGE
wlp-ingress02-ibm-web-68d7f59cc4-6n8hc  1/1    Running  0         2h
(略)
==> v1/Service
NAME                      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
wlp-ingress02-ibm-web  NodePort  10.0.162.238  <none>       9443:32214/TCP  2h
==> v1/Deployment
NAME                      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
wlp-ingress02-ibm-web  1        1        1           1          2h
==> v1beta1/Ingress
NAME                      HOSTS  ADDRESS        PORTS    AGE
wlp-ingress02-ibm-web  *      9.xxx.xxx.xxx  80, 443  2h
```

Secretが自動で作成されたことが確認できます。

ServiceのPORTが9443となっていることを確認します。

Portに443が追加されていることが確認できます。

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

- (6) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get ingress <ingress name> -o yaml」コマンドでIngressの設定内容を確認します。

```
# kubectl get ingress wlp-ingress02-ibm-web -o yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    (略)
    ingress.kubernetes.io/affinity: cookie
    ingress.kubernetes.io/backend-protocol: HTTPS
    ingress.kubernetes.io/session-cookie-name: route
    (略)
    nginx.ingress.kubernetes.io/affinity: cookie
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
    nginx.ingress.kubernetes.io/session-cookie-name: route
    (略)
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: wlp-ingress02-ibm-web
          servicePort: 9443
        path: /wlp-pat02
  tls:
  - secretName: wlp-ingress02-ibm-web-tls
status:
  loadBalancer:
    ingress:
    - ip: 9.xxx.xxx.xxx
```

デフォルトでActive Cookie方式のHTTPセッションのアフィニティーを維持できます。(nginxがサポートするのはCookieのみ)

HTTPS接続するために必要なannotationが自動で追加されます。

外部からの接続用に設定したコンテキストパス

HTTPS接続用のSecret

接続用のProxy NodeのIPアドレス

Ingressの設定値の変更：仮想PATHの利用(HTTPS)

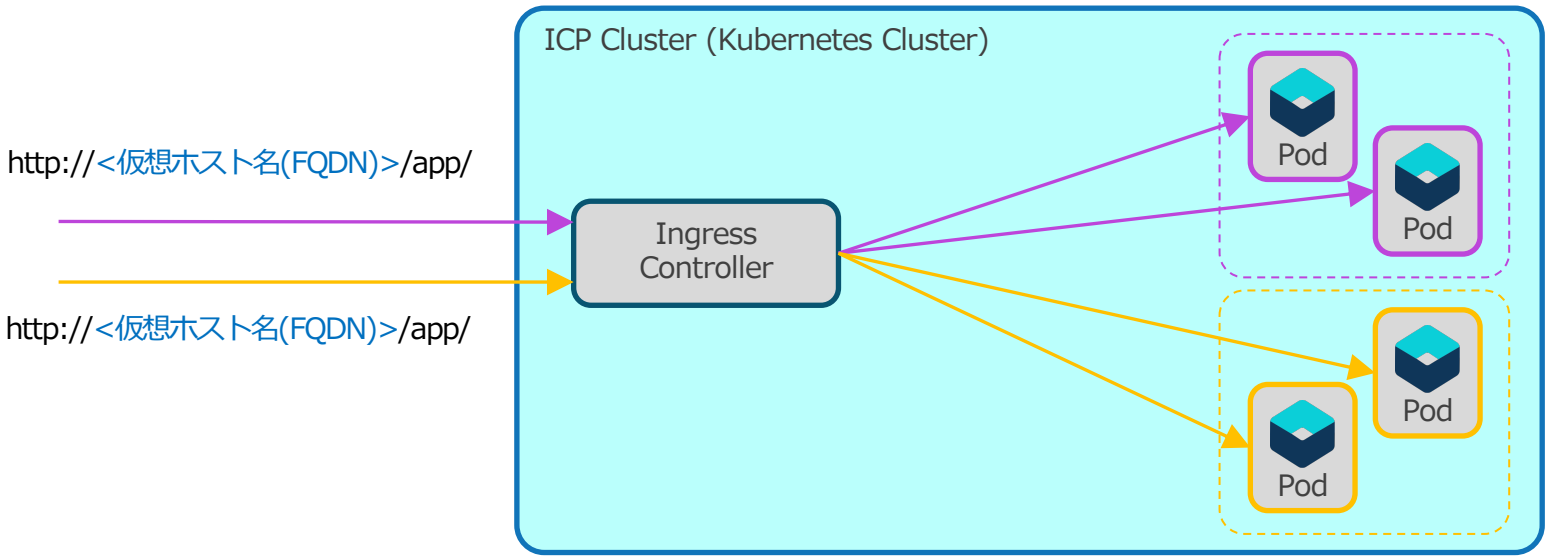
- (7) Webブラウザで表示されることを確認します。
  - Webブラウザに直接以下URLで表示されることを確認します。
  - "https://<Proxy NodeのIPアドレス>/<設定したpath名>/"
  - (例)"https://9.xxx.xxx.xxx/wlp-path02/"
  - (Proxy NodeがHA構成の場合はproxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスを指定してください。)

最後の"/"(スラッシュ)を忘れずに！

目次(4章)

Ingressの設定値の変更：名前ベースの仮想ホストの利用(HTTP)

- WAS LibertyのHelm Chartの変数ファイルでIngressに関する設定値を変更する手順を示します。
  - 設定概要
    - ICPのIngress Controllerは名前ベースの仮想ホスト(Name-based Virtual Host)に対応しています。
    - ホスト名に基づいて、同じパスを持つ複数のサービスに処理を割り振ることが可能です。
    - Helm Chartの変数ファイルのパラメーターingress.hostにホスト名を指定することで、名前ベースの仮想ホストが利用可能になります。
    - ただし、指定の仮想ホスト名はFQDNである必要があり、Proxy NodeのIPアドレスで名前解決可能なドメイン名が必要です。
    - GitHub / IBM/charts/stable/ibm-websphere-liberty / Ingress configuration
    - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#ingress-configuration
    - Kubernetes Documentation / Ingress
    - https://kubernetes.io/docs/concepts/services-networking/ingress/



HTTPS (SSL) 通信で名前ベースの仮想ホストを利用することも可能です。HTTPSの設定については、「Ingressの設定値の変更：仮想PATHの利用(HTTPS)」の部分と同様ですのでそちらをご参照ください。

Ingressの設定値の変更：名前ベースの仮想ホストの利用(HTTP)

– 前提作業
- 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
- 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

- 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

- cloudctlコマンドでICPにログインしておきます。
- 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

– 手順概要
- (1) Helm Chartから変数ファイルを抽出します。

```
# helm inspect values ibm-websphere-liberty-1.9.0.tgz > wlp-test03.yml
```

Ingressの設定値の変更：名前ベースの仮想ホストの利用(HTTP)

- (2) 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。
  - 以下の青字部分を変更します。

(例) wlp-test03.yml

```
(略)
image:
  repository: mycluster.icp:8500/handson/websphere-liberty          # 使用するイメージ名
  tag: 19.0.0.6-javaee8                                             # 使用するイメージのtag名
  pullPolicy: IfNotPresent
  license: "accept"                                                 # Licenseの承認
(略)
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: NodePort
(略)
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: "/"
  path: "/wlp-path01"                                               # コンテキストパスを変更する場合はこちらに記載
  host: "host01.example.com"                                        # 割り振る仮想ホスト名(FQDN)
  secretName: ""
  annotations:{}
(略)
```

Ingressの設定値の変更：名前ベースの仮想ホストの利用(HTTP)

- (3) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します

```
# helm install --debug --dry-run --name wlp-test03 -f ./wlp-test03.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (4) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
# helm install --name wlp-test03 -f ./wlp-test03.yml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (5) 正常にインストールできたか確認します。

```
# helm status wlp-test03 --tls
LAST DEPLOYED: Fri Aug  2 17:04:41 2019
NAMESPACE: handson
STATUS: DEPLOYED

RESOURCES:
(略)
==> v1/Service
NAME                        TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)   AGE
wlp-test03-ibm-websphere    ClusterIP   10.0.39.123   <none>       9080/TCP  18m

==> v1/Deployment
NAME                        DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
wlp-test03-ibm-websphere    1        1        1           1          18m

==> v1beta1/Ingress
NAME                        HOSTS                ADDRESS        PORTS  AGE
wlp-test03-ibm-websphere    host01.example.com   9.xxx.xxx.xxx  80     18m
(略)
```

IngressのHOSTSに指定した仮想ホスト名(FQDN)が入っていることが確認できます。

Ingressの設定値の変更：名前ベースの仮想ホストの利用(HTTP)

- (6) 変数ファイルで設定した設定値が正しく反映していることを確認します。
  - 「kubectl get ingress <ingress name> -o yaml」コマンドでIngressの設定内容を確認します。

```
# kubectl get ingress wlp-test03-ibm-websphe -o yaml
apiVersion: extensions/v1beta1
kind: Ingress
(略)
spec:
  rules:
  - host: host01.example.com
    http:
      paths:
      - backend:
          serviceName: wlp-test03-ibm-websphere
          servicePort: 9080
        path: /wlp-path01
status:
  loadBalancer:
    ingress:
    - ip: 9.xxx.xxx.xxx
```

外部からの接続用に設定した仮想ホスト名(FQDN)

外部からの接続用に設定したコンテキストパス

接続用のProxy NodeのIPアドレス

- (7) Webブラウザで表示されることを確認します。
  - Webブラウザに直接以下URLで表示されることを確認します。
  - "http://<仮想ホスト名(FQDN)>/<設定したpath名>/"
  - (例)"http://host01.example.com/wlp-path01/"

最後の"/"(スラッシュ)を忘れずに！

  - (仮想ホスト名(FQDN)でProxy NodeのIPアドレスが解決できる必要があります(DNSレコード(もしくは/etc/hosts)に登録)。)
  - (Proxy NodeがHA構成の場合は仮想ホスト名(FQDN)が proxy_vipなど適宜Proxy Nodeにアクセス可能なIPアドレスが解決できるようにしてください。)

目次(4章)

Helm Chartの変更

- 基本的にはIBM提供のWAS LibertyのHelm Chartの機能だけで主な要件は実現することができます。
  - WAS Libertyの変数ファイルのパラメーターを設定して主な要件は実現することができます。

- ただし、一部の機能や要件は、Helm Chartの変数ファイルのパラメーターだけでは設定できない項目があります。
  - その場合は提供されているWAS LibertyのHelm Chart自体を修正して、要件にあった別のHelm Chartを作成するか、一から新規に独自のHelm Chartを作成する必要があります。
    - その他、Helm Chartを使用せず、Kubernetesのマニフェストファイルで直接リソースを定義する方法もありますが、本ガイドでは扱いません。
  - ここでは、WAS LibertyのHelm Chart自体を修正して、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。
  - 参考リンク
    - ICP 3.1.2 Knowledge Center / カスタム・アプリケーションの追加
    - https://www.ibm.com/support/knowledgecenter/ja/SSBS6K_3.1.2/app_center/add_package.html

目次(4章)

- Helm Chartの利用
- Helm Chartの変数ファイルによる設定変更
  - Readiness Probeの設定値の変更
  - Liveness Probeの設定値の変更
  - Deploymentの設定値の変更
  - Serviceの設定値の変更
  - Ingressの設定値の変更
    - 仮想PATHの利用(HTTP)
    - 仮想PATHの利用(HTTPS)
    - 名前ベースの仮想ホストの利用(HTTP)
- (補足)Helm Chartの変更
  - Helm Chartの変更
  - WAS LibertyのHelm Chartの構造
  - Helm Chartの変更手順
  - Helm Chartの変更
    - Deploymentの変更
    - StatefulSetの変更

WAS LibertyのHelm Chartの構造

■ WAS LibertyのHelm ChartとOpen Liberty

– 一般的に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

– WAS 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。
  • 依存関係はWAS Libertyの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: 1.9.0
alias: slt
```

– 「ibm-shared-liberty」subchartは、Open LibertyとWAS Libertyの共通部分を抜き出して作成されています。WAS LibertyのHelm Chartでは、「ibm-shared-liberty」 subchartにWAS 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。

– 「ibm-shared-liberty」subchartは、WAS LibertyのHelm Chartのディレクトリ内にパッケージ化されて取り込まれています。
  • Helm Documentation / Subcharts and Global Values
  • https://helm.sh/docs/chart_template_guide/#subcharts-and-global-values

WAS LibertyのHelm Chartの構造

■ WAS LibertyのHelm Chartの構造

WAS 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-1.9.0.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       # chartのインストール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    # ICPのカタログの構成時に表示される情報
└── values.yaml                  # 設定のデフォルト値
```

tgzを展開

WAS LibertyのHelm Chart内に含まれる「ibm-shared-liberty」 subchart

```
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
```

目次(4章)

Helm Chartの変更手順

- WAS 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。
  - 手順の流れ
    - (1) WAS LiberyのHelm Chartを展開します。
      - WAS LibertyのHelm Chart(ibm-websphere-liberty-1.X.0.tgz)と、「ibm-shared-liberty」subchart(ibm-shared-liberty-1.X.0.tgz)を展開します。
    - (2) Helm ChartのChart名を変更します。
      - Helm Chartを同じ名前にすると、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、ここではHelm Chart名を変更します。Helm Chart内にある「Chart.yaml」のnameとディレクトリ名を新しい名前に変更します。
    - (3) Helm Chart内の「ibm-shared-liberty」subchartのテンプレートを修正します。
      - 「ibm-shared-liberty」subchart内の変更が必要なテンプレートを修正します。
    - (4) 変更したHelm Chartの文法が正しいか確認をします。
      - Helm Chartの文法チェックを「helm lint」コマンドで行います。
    - (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。

    - (以降の手順は前述のWAS LibertyのHelm Chartのインストールの手順と同様です。)

    - (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) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。
    - (9) 編集した変数ファイルを使用して、Helm Chartをインストールします。
    - (10) 正常にインストールできたか確認します。

目次(4章)

Helm Chartの変更：Deploymentの変更

- WAS LibertyのHelm Chartを修正する手順を示します。
  - 設定概要
    - WAS Libertyはログを永続化しない場合はPodはDeploymentとしてデプロイされます。(5章参照)
    - Deploymentの設定の中で、「spec.strategy.rollingUpdate」は、WAS LibertyのHelm Chartの変数ファイルのパラメーターでは設定値を変更することがができません。
      - maxSurge: 更新時に、指定されたレプリカ数を超えて生成することができる Pod の数(割合)
      - maxUnavailable: 更新時に、使用不可状態にできる Pod の最大数(割合)
      - （Deploymentのデフォルト値は「maxSurge=25%」および「maxUnavailable=25%」です。）
    - この設定値を変更する場合は、Helm Chart自体の修正を行う必要があります。
      - 「ibm-shared-liberty」subchartのDeployment用のテンプレートの「_deployment.tpl 」内に定義されていないため、明示的に追記する必要があります。）

(変更前のWAS LibertyのHelm Chartでインストールした際のDeploymentの設定値抜粋)

```
$ kubectl get deployment <Deployment名> -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
(略)
spec:
  (略)
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
(略)
```

Deploymentのデフォルト値が適用され、Helm Chartの変数ファイルでは変更することができません。

Helm Chartの変更：Deploymentの変更

– 前提作業
  - 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
  - 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

  - 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

  - cloudctlコマンドでICPにログインしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

– 手順概要
  - (1) WAS LiberyのHelm Chartを展開します。
    – WAS LibertyのHelm Chart(ibm-websphere-liberty-1.x.0.tgz)と、「ibm-shared-liberty」subchart(ibm-shared-liberty-1.x.0.tgz)を展開します。

Helm Chartの変更：Deploymentの変更

- (2) Helm ChartのChart名を変更します。
  - Helm Chartを同じ名前にすると、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、ここではHelm 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。

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

ディレクトリを任意の名前に変更します。
(ここではrollingupdateを表すruをつけてibm-websphere-liberty-ruとしてみました。)

(例) ibm-websphere-liberty-ru/Chart.ymlの変更

```
(略)
maintainers:
- name: WebSphere Liberty
name: ibm-websphere-liberty-ru
(略)
version: 1.9.0
```

nameにディレクトリ名と同じ名前を設定します。

- (3) Helm Chart内の「ibm-shared-liberty」subchartのテンプレートを修正します。
  - 「ibm-shared-liberty」subchart内の変更が必要なテンプレートを修正します。今回は以下の青字部分を追記します。

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

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

spec.strategy.rollingUpdate.maxSurgeとspec.strategy.rollingUpdate.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。)

Helm Chartの変更：Deploymentの変更

- (4) 変更したHelm Chartの文法が正しいか確認をします。
  - Helm Chartの文法チェックを「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
Lint OK

1 chart(s) linted, no failures
```

- (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、「ibm-shared-liberty」subchartの「ibm-shared-liberty-1.x.0.tgz」ファイルが残っている場合はこのファイルは削除して置いてください。(subchartは展開して一部のテンプレートを編集した状態で、元の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-1.9.0.tgz)が作成されます。

```
$ helm package ibm-websphere-liberty-ru
Successfully packaged chart and saved it to: /Users/ayu/work/lab/lab75/ibm-websphere-liberty-ru-1.9.0.tgz
```

Helm Chartの変更：Deploymentの変更

- (6) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-ru-1.9.0.tgz > ru-test01.yaml
```

- (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。
  - 抽出したHelm Chartの変数ファイルから、必要な部分を適宜修正します。(設定値は3章で前述の「Deploymentの設定値の変更」参照)

- (8) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。

```
$ helm install --dry-run --debug --name ru-test01 -f ./ru-test01.yaml ./ibm-websphere-liberty-ru-1.9.0.tgz --tls
```

(出力例)

```
---
# Source: ibm-websphere-liberty-ru/templates/shared.yaml
# SLT: 'slt.deployment' from templates/_deployment.tpl
apiVersion: apps/v1
kind: Deployment

metadata:
  name: ru-test01-ibm-websphere
  labels:
    chart: "ibm-websphere-liberty-ru-1.9.0"
    (略)
spec:
  strategy:
    rollingUpdate:
      maxSurge: 50%
      maxUnavailable: 0%
    type: RollingUpdate
  (略)
```

手順(3)でsubchart内のテンプレートに直接修正したspec.strategy.rollingUpdate.maxSurgeとspec.strategy.rollingUpdate.maxUnavailableの値が反映していることが確認できます。

Helm Chartの変更：Deploymentの変更

- (9) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm install --name ru-test01 -f ./ru-test01.yaml ./ibm-websphere-liberty-ru-1.9.0.tgz --tls
```

- (10) 正常にインストールできたか確認します。

```
$ kubectl get deployment
NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
ru-test01-ibm-websphere    1         1         1            1           34s
$
$ kubectl get deployment ru-test01-ibm-websphere -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
(略)
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: ru-test01-ibm-websphere
  strategy:
    rollingUpdate:
      maxSurge: 50%
      maxUnavailable: 0%
    type: RollingUpdate
  (略)
```

手順(3)でsubchart内のテンプレートに直接修正したspec.strategy.rollingUpdate.maxSurgeとspec.strategy.rollingUpdate.maxUnavailableの値が反映していることが確認できます。

目次(4章)

Helm Chartの変更：StatefulSetの変更

■ WAS LibertyのHelm Chartを修正する手順を示します。

Kubernetes Documentation / StatefulSets
https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

– 設定概要

- WAS Libertyは、ログを永続化する場合はPodはStatefulSetとしてデプロイされます。(5章参照)
- StatefulSetの設定の中で、「spec.podManagementPolicy」と「spec.updateStrategy.type」については、Helm Chartの変数ファイルで指定することができません。この設定値を変更する場合は、Helm Chart自体の修正を行う必要があります。
- 「spec.podManagement」の指定
  - 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。
  - デフォルトでは「OrderedReady」となり、Helm Chartの変数ファイルでは変更することができません。ここでは「Parallel」に変更してみます。
- 「spec.updateStrategy.type」の指定
  - 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。
  - デフォルトでは「RollingUpdate」となり、Helm Chartの変数ファイルでは変更することができません。ここでは「OnDelete」に変更してみます。

(変更前のWAS LibertyのHelm Chartでログを永続化してインストールした際のStatefulSetの設定値抜粋)

```
$ kubectl get statefulset <StatefulSet名> -o yaml
apiVersion: apps/v1
kind: StatefulSet
(略)
spec:
  podManagementPolicy: OrderedReady
  (略)
  updateStrategy:
    rollingUpdate:
      partition: 0
    type: RollingUpdate
(略)
```

StatefulSetのデフォルト値が適用され、Helm Chartの変数ファイルでは、spec.podManagementPolicyとspec.updateStrategy.typeは変更することができません。

Helm Chartの変更：StatefulSetの変更

– 前提作業
  - 事前にDockerイメージをICPのプライベートDockerレポジトリーにpushしておきます。
  - 手順は3章の「(2) DockerイメージをICPプライベートDockerレジストリーへpushする」を参照してください。

  - 事前にWAS LibertyのHelm Chartをダウンロードしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

  - cloudctlコマンドでICPにログインしておきます。
  - 手順は3章の「(3) WAS Libertyをインストールする(helmコマンド)」を参照してください。

– 手順概要
  - (1) WAS LibertyのHelm Chartを展開します。
    – WAS LibertyのHelm Chart(ibm-websphere-liberty-1.x.0.tgz)と、「ibm-shared-liberty」subchart(ibm-shared-liberty-1.x.0.tgz)を展開します。

Helm Chartの変更：StatefulSetの変更

- (2) Helm ChartのChart名を変更します。
  - Helm Chartを同じ名前にすると、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、ここではHelm 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。

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

ディレクトリを任意の名前に変更します。(ここではOnDeleteを表すodをつけてibm-websphere-liberty-odとしてみました。)

(例) ibm-websphere-liberty-ru/Chart.ymlの変更

```
(略)
maintainers:
- name: WebSphere Liberty
name: ibm-websphere-liberty-od
(略)
version: 1.9.0
```

nameにディレクトリ名と同じ名前を設定します。

- (3) Helm Chart内の「ibm-shared-liberty」subchartのテンプレートを修正します。
  - 「ibm-shared-liberty」subchart内の変更が必要なテンプレートを修正します。今回は以下の青字部分を追記します。

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

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

「ibm-shared-liberty」subchartに含まれるDeploymentとStatefulSetのテンプレートは同一のテンプレートファイルになります。テンプレート内のif文でStatefulSetの場合の設定箇所に設定内容を追記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。

Helm Chartの変更：StatefulSetの変更

- (4) 変更したHelm Chartの文法が正しいか確認をします。
  - Helm Chartの文法チェックを「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
Lint OK

1 chart(s) linted, no failures
```

- (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、「ibm-shared-liberty」subchartの「ibm-shared-liberty-1.x.0.tgz」ファイルが残っている場合はこのファイルは削除して置いてください。(subchartは展開して一部のテンプレートを編集した状態で、元の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-1.9.0.tgz)が作成されます。

```
$ helm package ibm-websphere-liberty-od
Successfully packaged chart and saved it to: /Users/ayu/work/lab/lab75/ibm-websphere-liberty-od-1.9.0.tgz
```

Helm Chartの変更：StatefulSetの変更

- (6) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-od-1.9.0.tgz > od-test01.yaml
```

- (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。
  - 抽出したHelm Chartの変数ファイルから、必要な部分を適宜修正します。(ここではログを永続化をするように設定します。(設定内容は5章参照))

- (8) (オプション) 作成されるICP(k8s)のリソースを「--dry-run」または「--dry-run --debug」オプションで事前に確認します。

```
$ helm install --dry-run --debug --name od-test01 -f ./od-test01.yaml ./ibm-websphere-liberty-od-1.9.0.tgz --tls
```

(出力例)

```
---
# Source: ibm-websphere-liberty-od/templates/shared.yaml
# SLT: 'slt.deployment' from templates/_deployment.tpl
apiVersion: apps/v1
kind: StatefulSet

metadata:
  name: od-test01-ibm-websphere
  labels:
    chart: "ibm-websphere-liberty-od-1.9.0"
    (略)
spec:

  serviceName: od-test01-ibm-websphere
  podManagementPolicy: Parallel
  updateStrategy:
    type: OnDelete
  (略)
```

手順(3)でsubchart内のテンプレートに直接追記したspec.podManagementPolicyとspec.updateStrategy.typeの値が反映していることが確認できます。

Helm Chartの変更：StatefulSetの変更

- (9) 編集した変数ファイルを使用して、Helm Chartをインストールします。

```
$ helm install --name od-test01 -f ./od-test01.yaml ./ibm-websphere-liberty-od-1.9.0.tgz --tls
```

- (10) 正常にインストールできたか確認します。

```
$ kubectl get statefulset
NAME                       DESIRED   CURRENT   AGE
od-test01-ibm-websphere    2         2         59s
$
$ kubectl get statefulset od-test01-ibm-websphere -o yaml
apiVersion: apps/v1
kind: StatefulSet
(略)
spec:
  podManagementPolicy: Parallel
  replicas: 2
  (略)
  updateStrategy:
    type: OnDelete
  (略)
```

手順(3)でsubchart内のテンプレートに直接追記したspec.podManagementPolicyとspec.updateStrategy.typeの値が反映していることが確認できます。

5. ログとダンプの永続化と転送

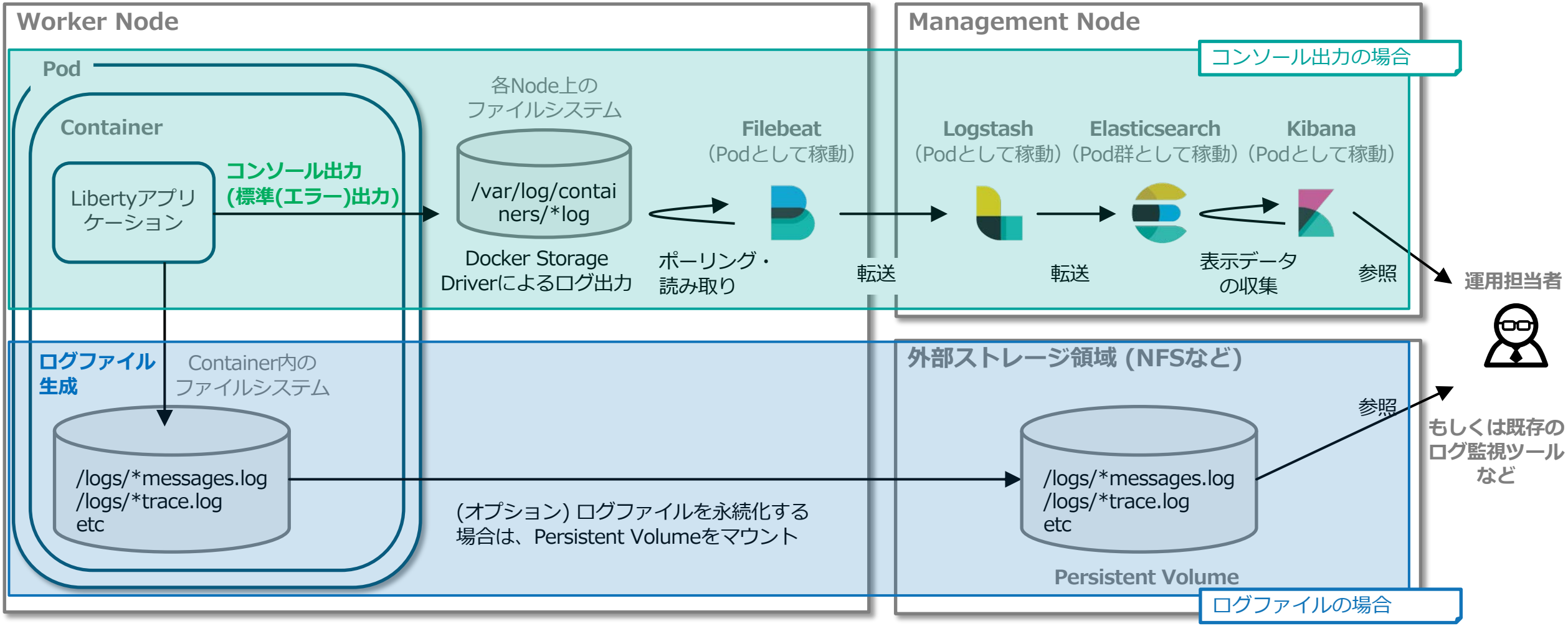
目次(5章)

- ICP環境でのWAS Libertyのログの概要
  - 「コンソール出力」
  - 「ログファイル」
- WAS Libertyが出力するログとダンプ
- IBM提供のlibertyのHelm Chartのパラメーター：ログ関連
  - 「ログファイルの永続化」に関するパラメーター
  - 「コンソール出力」に関するパラメーター
- ログファイルの永続化
  - ログファイルとダンプの永続化が必要になるケース
  - ログファイルとダンプの永続化方法
  - 永続領域(Persistent Volume)の準備
  - トランザクション・ログの永続化
  - Helmデプロイ
  - デプロイ後の運用(Helmリソース(Pod))
  - デプロイ後の運用(Helmリソース(Service))
  - デプロイ後の運用(永続領域の管理)
- コンソール出力
  - ICPのELK(Elasticsearch、Logstash、Kibana)への連携
  - コンソール出力の設定
  - Kibanaの設定
  - Kibanaの表示例
  - サンプル・ダッシュボードの利用(IBM Certified Cloud Pak)
  - サンプル・ダッシュボードの表示例
    - Liberty Problems Dashboard
    - Liberty Traffic Dashboard
    - Liberty Audit * Dashboard

目次(5章)

ICP環境でのWAS Libertyのログの概要

- WAS Libertyのログの確認経路は、大まかに「コンソール出力」と「ログファイル」の場合で異なります。「ログファイル」を永続化したい場合はHelm以外の別途構成も必要です。



目次(5章)

WAS Libertyが出力するログとダンプ

■ WAS Libertyが出力するログとダンプの一覧と、出力先を以下に示します。

– 以下のURLの情報も参照してください。

- developerWorks: WAS V9 Liberty基盤設計セミナー資料 ー「8. パフォーマンス/問題判別」
- https://www.ibm.com/developerworks/jp/websphere/library/was/liberty_v9_infradesign/

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

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

目次(5章)

IBM提供のWAS LibertyのHelm Chartのパラメーター：ログ関連

■ 「ログファイルの永続化」に関連するパラメーター

(*) ibm-websphere-liberty v1.9.0の場合

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。 （Libertyコンテナはデフォルトではdefault(uid=1001)というユーザーで起動し、このユーザーはroot(gid=0)というグループに所属しており、生成されるファイル・ディレクトリのパーミッションは664または755となります。） （もし起動ユーザーの設定を変更した場合に、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。）
	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。） （Static Provisioning の場合に、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	**WAS Liberty のログ出力領域**を永続領域上に割り当てる場合は true を選択します。 true を指定すると、**Docker イメージの /logs** に永続領域内の logs ディレクトリーがマウントされます。
	persistTransactionLogs	false	**WAS Liberty のトランザクション・ログ出力領域**を永続領域上に割り当てる場合は true を選択します。 true を指定すると、**Docker イメージの /output/tranlog** に永続領域内の tranlog ディレクトリーがマウントされます。

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

IBM提供のWAS 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

(*) ibm-websphere-liberty v1.9.0の場合

Qualifier	Parameter	デフォルト値	説明
logs	consoleFormat	json	WAS Liberty のコンソール・ログのフォーマットを指定します。 json 形式(Helm Chart のデフォルト)、または、従来からの出力形式である basic 形式が指定できます。
	consoleLogLevel	info	WAS Liberty のコンソール・ログの出力レベルを指定します。 info (Helm Chart のデフォルト), audit, warning, error または off の何れかを指定します。
	consoleSource	message,trace,accessLog,ffdc	WAS 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、WAS 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
WAS Liberty Knowledge Center / IBM Cloud Private での Liberty メッセージの分析
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_json_logging.html

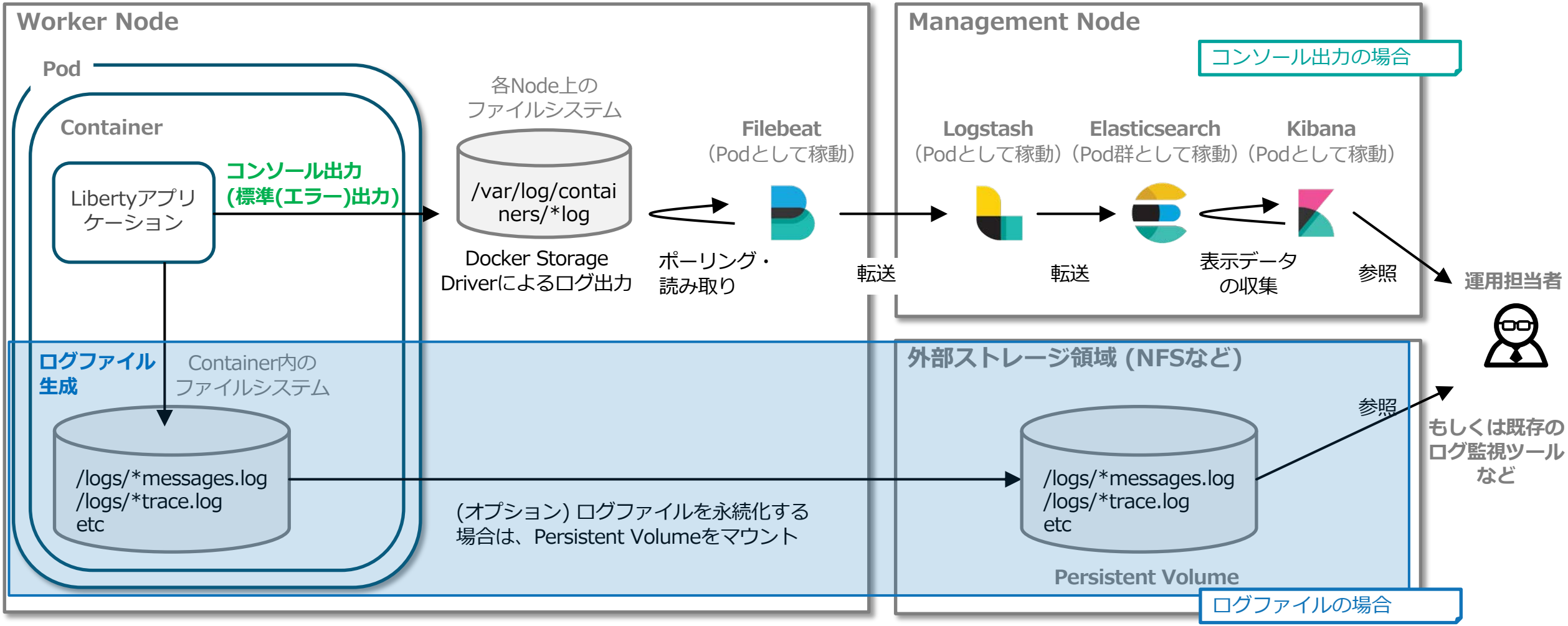
目次(5章)

ログファイルとダンプの永続化が必要になるケース

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

ログファイルとダンプの永続化方法

■ 「ログファイル」を永続化する場合はHelm Chartのパラメーターで永続化の設定をすることと、永続領域（Persistent Volume）の構成が必要です。



ログファイルとダンプの永続化方法

■ ログファイルの永続化方法

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

– ログファイル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

– 以下に示すログファイルは、出力先をログファイル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ファイルを永続化させます(/logs) 。また、ダンプを「/logs」に出力するように WAS Liberty を構成します。
  – WAS 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
- ダンプによって、ログの永続化領域（「logs」）が圧迫される可能性があります。

– 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。

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
  - ICP クラスターに定義されているStorage Classを確認します。(詳細はICPクラスターの管理者に確認)
- Static Provisioningを使用する場合
  - 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。
  - ICP Management Consoleで定義する場合は、[Menu > プラットフォーム > ストレージ]と選択して[ストレージ]画面に進み、[PersistentVolumeの作成]ボタンを押下して作成します。(定義内容はyamlファイルの例を参照)
  - kubectl applyコマンドで作成する場合は、以下のようなyamlファイルを使用します。
    - kubectlコマンドのセットアップに関しては、「1. IBM Cloud Private（ICP）概要」の「ICP の管理操作：補足」に示した情報を参照してください。

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。Dynamic Provisioningの場合は必須です。
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。
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

トランザクション・ログの永続化

■ WAS Libertyのトランザクション・ログを永続化する場合は以下の考慮が必要です。

– server.xmlに以下の指定を追加します。

- WAS Liberty の起動時にインダウトの解消が開始され、完了するまでアプリが始動されなくなります。
- GitHub / IBM/charts/stable/ibm-websphere-liberty / Transaction logs 参照
- https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#transaction-logs

(server.xmlまたはserver.xmlフラグメントへの反映方法は、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

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

– IBM 提供のHelm Chartは、デフォルトのログ配置を想定しています。

– transactionLogDirectoryは変更できません。

- デフォルトのトランザクション・ログの配置：　${server.output.dir}/tranlog/

– WAS Libertyのトランザクション関連の設定に関しては、以下の URL の情報も参照してください。

- developerWorks: WAS V9 Liberty基盤設計セミナー資料 ー「6. 外部連携」の「トランザクション」の部分
- https://www.ibm.com/developerworks/jp/websphere/library/was/liberty_v9_infradesign/

Helmデプロイ

- ログファイルを永続化してHelmをデプロイする手順を記載します。
  - 例として以下のログを永続化する場合の設定を記載します。
    - /logsに保管されるログファイル（message.log、trace.log、FFDC）
    - /output/tranlogに保管されるトランザクション・ログ
  - 手順の流れは以下となります。
    - (1) 事前に永続化領域(Persistent Volume)を作成します。
    - (2) 事前にトランザクション・ログを有効化する設定をserver.xmlに追加するためのConfigmapを作成します。
    - (3) Helm Chartから変数ファイルを抽出します。
    - (4) 抽出した変数ファイルのログファイルの永続化に関するパラメーターを修正します。
    - (5) Helm Chartをデプロイします。
    - (6) ログファイルが永続化されていることを確認します。

Helmデプロイ

– 手順概要
- (1) 事前に永続化領域(Persistent Volume)を作成します。
  – 例としてNFSのPersistent Volume(PV)を作成します。
  – Persistent Volume作成用のマニフェストファイルを作成します。
  – 以下は後続のHelm Chartの変数ファイルでPersistent Volume Claim(PVC)を作成する際にStatic Provisioningを使用し、LabelでPVとPVCのマッチングをする構成をとる場合の例となります。

Persistent Volume用のマニフェストファイル(例: hogehoge-pv-1.yaml)

```
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
```

  – 以下のコマンドを実行しPVを作成します。

```
$ kubectl apply -f hogehoge-pv-1.yaml
persistentvolume/hogehoge-pv-1 created
$
```

Helmデプロイ

- (2) 事前にトランザクション・ログを有効化する設定をserver.xmlに追加するためのConfigmapを作成します。
  - トランザクション・ログを有効化する設定を追加したserver.xmlフラグメントをConfigMapとして事前に作成します。
  - Helmの変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMap名を指定します。
  - (3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」(3)の方法参照)
  - （server.xmlまたはserver.xmlフラグメントを更新する方法は他にも幾つかあります。詳細は3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」を参照下さい。）

  - 以下のような設定のserver.xmlフラグメントを「server-overrides.xml」という名前で作成します。

server-overrides.xml

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

  - 作成したserver.xmlフラグメント（server-overrides.xml）からConfigMapを作成します。

```
kubectl create configmap --save-config custom-server-overrides --from-file=./server-overrides.xml -n <namespace名>
```

Helmデプロイ

- (3) Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-liberty-log-test01.yaml
```

- (4) 抽出した変数ファイルのログファイルの永続化に関するパラメーターを修正します。
  - トランザクション・ログを有効化するためのConfigMapに関する修正部分は以下となります。
  - 「image.serverOverridesConfigMapName」に先ほど作成したConfigMapを指定します。

(例)values-liberty-log-test01.yaml

```
image:
(略)
  serverOverridesConfigMapName: "custom-server-overrides"
```

  - ログの永続化に関する修正部分は以下となります。
  - 「logs.persistLogs」と「logs.persistTransactionLogs」をtrueに変更します。

(例)values-liberty-log-test01.yaml

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

Helmデプロイ

- (4) 抽出した変数ファイルのログファイルの永続化に関するパラメーターを修正します。(続き)
  - 永続化領域に関する修正部分は以下となります。
  - Persistent VolumeとしてNFSを使用しDynamic Provisioningをしないため、「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」の値をPVと同じ値を設定します。

(例)values-liberty-log-test01.yaml

```
## Persistence is disabled by default, set Enabled: true to use
persistence:
  name: "liberty-pvc"
  ## Tranlog requires a Persistence Storage volume size minimum of 1Gi
  size: "1Gi"

  useDynamicProvisioning: false

  fsGroupGid:

  ## Specify the name of the StorageClass
  ## Setting StorageClass: "" will use whatever storageClass is currently
  ## setup as the Default
  storageClassName: ""

  # if your not using dynamic provisioning, you can use selectors to
  # refine the binding process. You cannot specify a selector if your using dynamic provisioning!
  selector:
    label: "type"
    value: "hogehoge-pv"
```

Helmデプロイ

- (5) Helm Chartをデプロイします。
  - 修正した変数ファイルを指定してHelm Chartをデプロイします。

```
helm upgrade --install liberty-log-test01 -f values-liberty-log-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

- (6) ログファイルが永続化されていることを確認します。
  - NFSサーバー側で、Libertyのログファイル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)ログファイル

```
# ls -l /nfs/liberty/hogehoge-pv-1/logs/
合計 24
-rw-r-----. 1 1001 root 16800  6月 13 23:46 messages.log
-rw-r-----. 1 1001 root   977  6月 13 23:46 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 root 61  6月 13 23:46 partnerlog
drwxr-x---. 2 1001 root 61  6月 13 23:46 tranlog
#
# ls -l /nfs/liberty/hogehoge-pv-1/tranlog/tranlog/
合計 8
-rw-r-----. 1 1001 root       0  6月 13 23:46 DO NOT DELETE LOG FILES
-rw-r-----. 1 1001 root 1048576  6月 13 23:46 log1
-rw-r-----. 1 1001 root 1048576  6月 13 23:46 log2
```

デプロイ後の運用（Helmリソース（Pod））

■ ログファイルを永続化しない場合と永続化する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」としてインストールされます。

(例)Helmの変数ファイルでログを永続化しないで、Podのレプリカ数を2とした場合(replicaCount: 2)

```
$ kubectl get deployment
NAME                        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
liberty-log-test01-ibm-w    2         2         2            2           3m5s
$
$ kubectl 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」としてインストールされます。

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」としてインストールされます。

- 「Statefulset」としてインストールされた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。

(例)Helmの変数ファイルでログを永続化し、Podのレプリカ数を2とした場合(replicaCount: 2)

```
$ kubectl get statefulset
NAME                        DESIRED   CURRENT   AGE
liberty-log-test01-ibm-w    2         2         14m
$
$ kubectl 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」としてインストールされます。

Podの名前は、<Statefulset name>-index 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）

デプロイ後の運用（Helmリソース（Service））

■ ログファイルを永続化しない場合と永続化する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の変数ファイルでログを永続化しないで、Helmの変数ファイルで「Service.type: Nodeport」(dafault)と設定した場合

```
$ kubectl 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。

• ICPクラス外部からPodへのアクセスは通常の「Service」経由でアクセスします。

(例)Helmの変数ファイルでログを永続化し、Helmの変数ファイルで「Service.type: Nodeport」(dafault)と設定した場合

```
$ kubectl 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)

デプロイ後の運用（永続領域の管理）

- 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。
  - 「kubectl get pv」コマンドでPVの状態を、「kubectl get pvc」コマンドでPVCの状態を確認できます。

(例)事前にPVを3つ作成し、Helmの変数ファイルでPodのレプリカ数を2とした場合(replicaCount: 2)

- PVの状態

PVがPVCに関連付けられておらず、割り当て可能な状態

```
$ kubectl get pv
NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                                                  (略)
(略)
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
(略)
$
```

PVが右記のPVCに関連付けられている状態

- PVCの状態

```
$ kubectl get pvc –n handson
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

  - ICP Management Console の場合は、[Menu > プラットフォーム > ストレージ]と選択して[ストレージ]画面に進み、「PersistentVolume」、または「PersistentVolumeClaim」タブを選択することで、PV と PVC の状態を確認できます。

デプロイ後の運用（永続領域の管理）

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

- 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「kubectl delete pvc <pvc name> (-n <namespace name>)」で行います。

(例)Helmアンインストール後のPVCの削除

```
$ kubectl delete pvc liberty-pvc-liberty-log-test01-ibm-w-0
persistentvolumeclaim "liberty-pvc-liberty-log-test01-ibm-w-0" deleted
$
$ kubectl delete pvc liberty-pvc-liberty-log-test01-ibm-w-1
persistentvolumeclaim "liberty-pvc-liberty-log-test01-ibm-w-1" deleted
$
```

  - ICP Management Console の場合は、[Menu > プラットフォーム > ストレージ]と選択して[ストレージ]画面に進み、「PersistentVolumeClaim」タブでPVCの「削除」を行います。

デプロイ後の運用（永続領域の管理）

■ 不要になったPVは、PVを削除するか、PV内のPVCとの関連付け情報を削除する必要があります。

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

(例)PVの削除

```
$ kubectl delete pv hogehoge-pv-3
persistentvolume "hogehoge-pv-3" deleted
```

– PVを残したまま、PVCとの関連付け情報を削除するには、「kubectl edit pv <pv name>」コマンドで行います。

(例)PVを残したままPVCとの関連付け情報の削除

```
$ kubectl edit pv hogehoge-pv-2
persistentvolume/hogehoge-pv-2 edited
$
$ kubectl get pv hogehoge-pv-2
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
hogehoge-pv-2   1Gi        RWO            Retain           Available                                   37m
```

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

(例)hogehoge-pv-2のedit画面で、以下の部分を削除

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

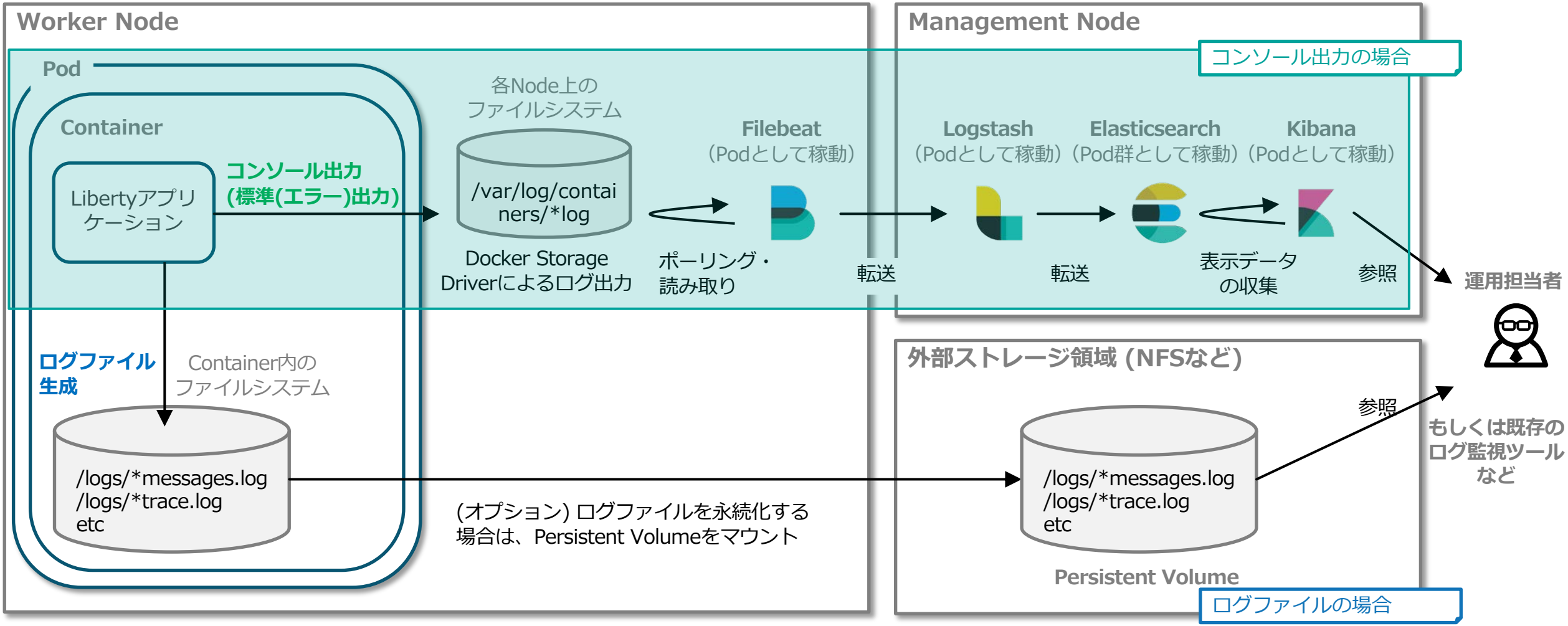
– ICP Management Console の場合は、[Menu > プラットフォーム > ストレージ]と選択して[ストレージ]画面に進み、「PersistentVolume」タブで、PVの「削除」または「編集」を行います。

目次(5章)

- ICP環境でのWAS Libertyのログの概要
  - 「コンソール出力」
  - 「ログファイル」
- WAS Libertyが出力するログとダンプ
- IBM提供のLibertyのHelm Chartのパラメーター：ログ関連
  - 「ログファイルの永続化」に関するパラメーター
  - 「コンソール出力」に関するパラメーター
- ログファイルの永続化
  - ログファイルとダンプの永続化が必要になるケース
  - ログファイルとダンプの永続化方法
  - 永続領域(Persistent Volume)の準備
  - トランザクション・ログの永続化
  - Helmデプロイ
  - デプロイ後の運用(Helmリソース(Pod))
  - デプロイ後の運用(Helmリソース(Service))
  - デプロイ後の運用(永続領域の管理)
- コンソール出力
  - ICPのELK(Elasticsearch、Logstash、Kibana)への連携
  - コンソール出力の設定
  - Kibanaの設定
  - Kibanaの表示例
  - サンプル・ダッシュボードの利用(IBM Certified Cloud Pak)
  - サンプル・ダッシュボードの表示例
    - Liberty Problems Dashboard
    - Liberty Traffic Dashboard
    - Liberty Audit * Dashboard

ICPのELK(Elasticsearch、Logstash、Kibana)への連携

■ Pod内で稼働するコンテナーの「コンソール出力」(標準出力(標準エラー出力))は、ICPのManagement Nodeで稼働するLogstashに自動的に転送され、Elasticsearchに保管されます。



コンソール出力の設定

- IBM提供のHelm Chartの「コンソール出力」に関する設定項目
  - 前述の「IBM 提供の liberty のHelm Chart のパラメーター：ログ関連」の「「コンソール出力」に関連するパラメーター」欄を参照ください。
    - 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はWAS Liberty側の設定も必要)
    - 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)

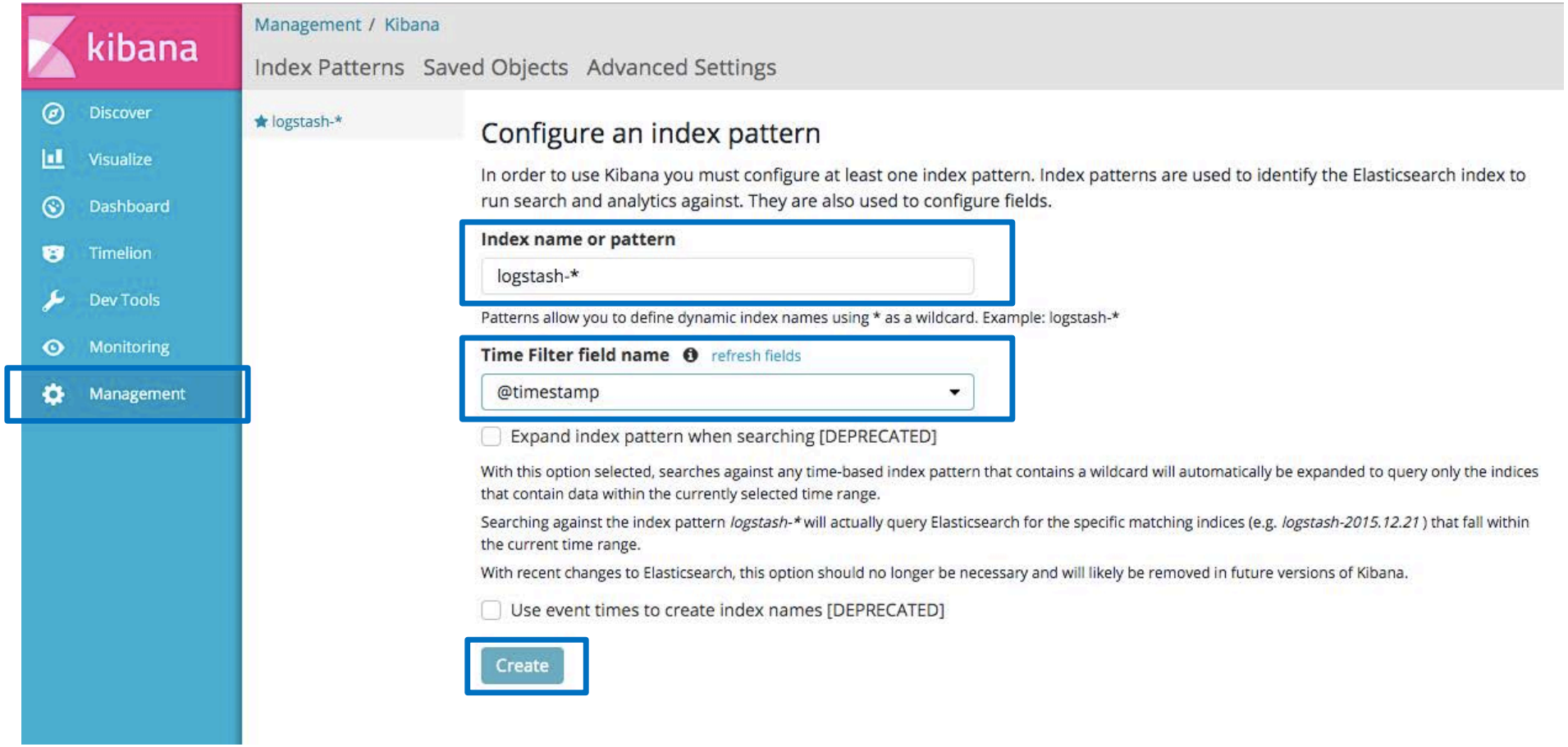
- WAS Libertyのアクセス・ログをコンソール出力させるには、WAS Libertyの設定でアクセス・ログ出力を有効にする必要があります。
  - server.xmlに以下の指定を追加します。
    - WAS Liberty Knowledge Center / HTTPアクセスのロギング
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_http_accesslogs.html
    - WAS Liberty Knowledge Center / ロギングおよびトレース
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html
    - GitHub / IBM/charts/stable/ibm-websphere-liberty / Liberty Docker image requirements
    - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#liberty-docker-image-requirements

（server.xmlへの反映方法は、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照）

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

Kibanaの設定

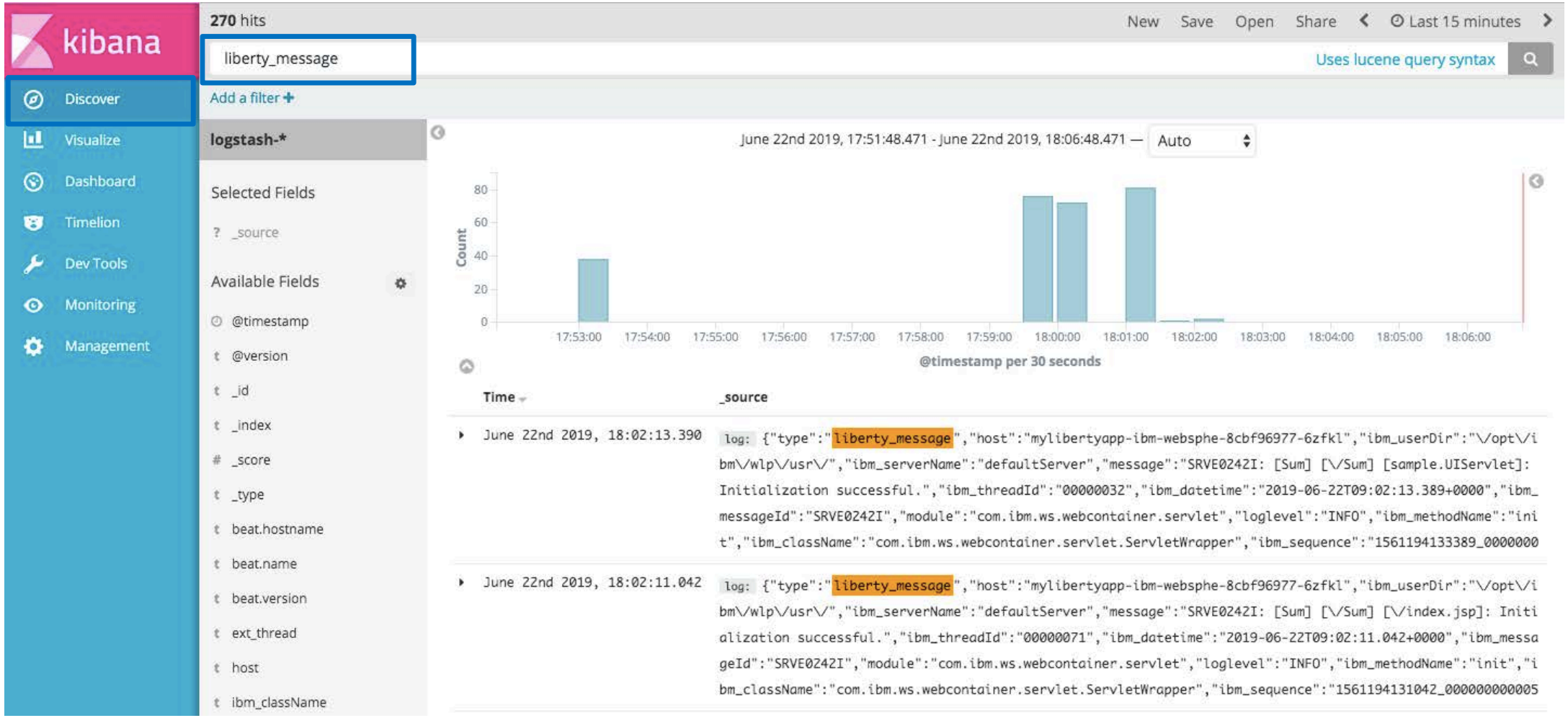
- Elasticsearchへ転送された「コンソール出力」のログはKibanaで確認することができます。
  - Kibana側で事前にインデックス・パターン「logstash-*」(time filterは「@timestamp」を指定)を作成します。
    - WAS Liberty Knowledge Center / IBM Cloud Private での Liberty メッセージの分析
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_json_logging.html



Kibanaの表示例

- Elasticsearchへ転送された「コンソール出力」のログはKibanaで確認することができます。
  - Kibanaの左側のメニューから「Discover」を選択することでログが表示されます。

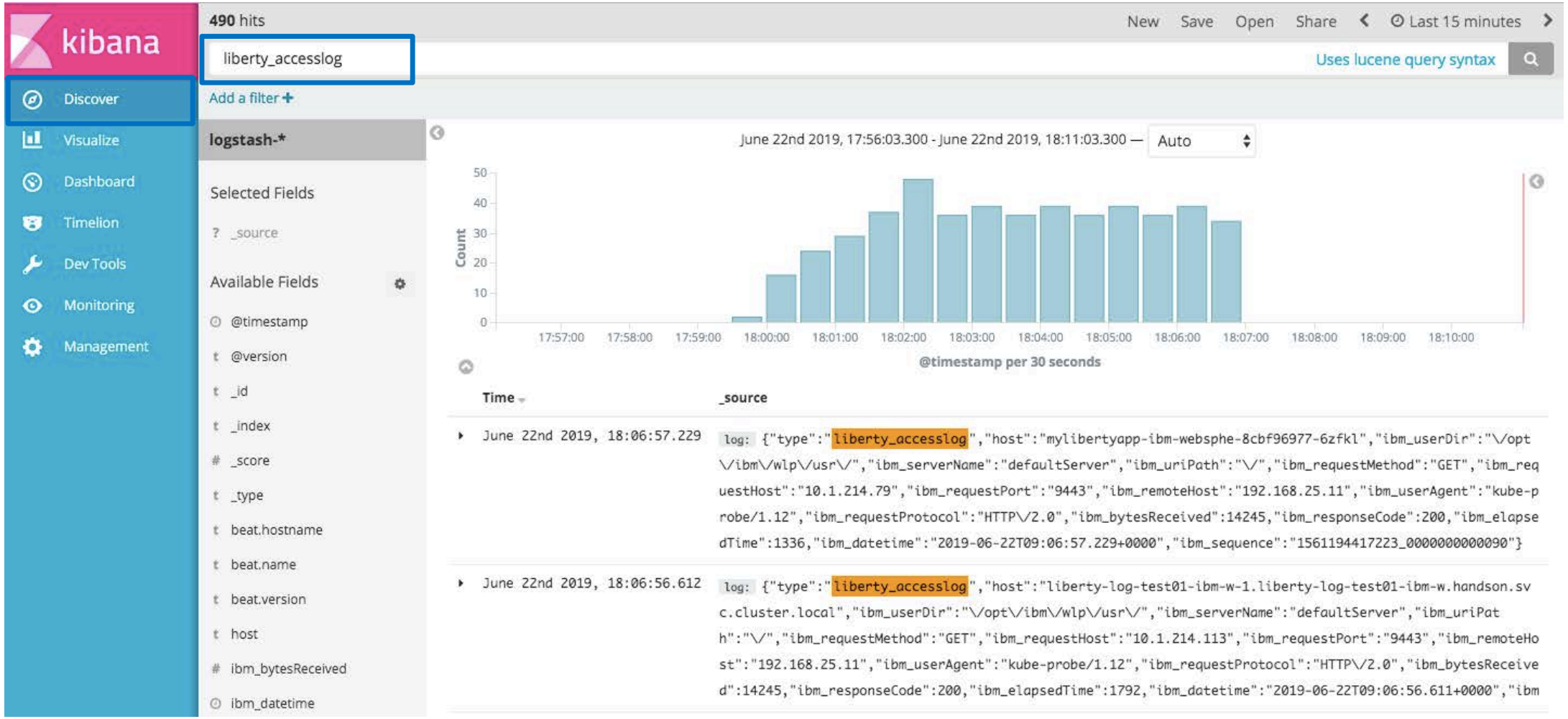
(例) 検索フィールドで「liberty_message」でフィルターした場合



Kibanaの表示例

- Elasticsearchへ転送された「コンソール出力」のログはKibanaで確認することができます。
  - Kibanaの左側のメニューから「Discover」を選択することでログが表示されます。

(例) 検索フィールドで「liberty_accesslog」でフィルターした場合



サンプル・ダッシュボードの利用（IBM Certified Cloud Pak）

■ Kibanaのサンプル・ダッシュボード

– WebSphere Libertyは「IBM Certified Cloud Pak」です。

– ロギング、モニタリングなどの運用に必要な機能も組み込まれた状態で製品が提供されています。

- WAS Liberty Knowledge Center / Liberty Cloud Pak の概要
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_cloud_pak_overview.html

– Libertyのログを目的に応じてわかりやすく表示するためのKibanaの「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

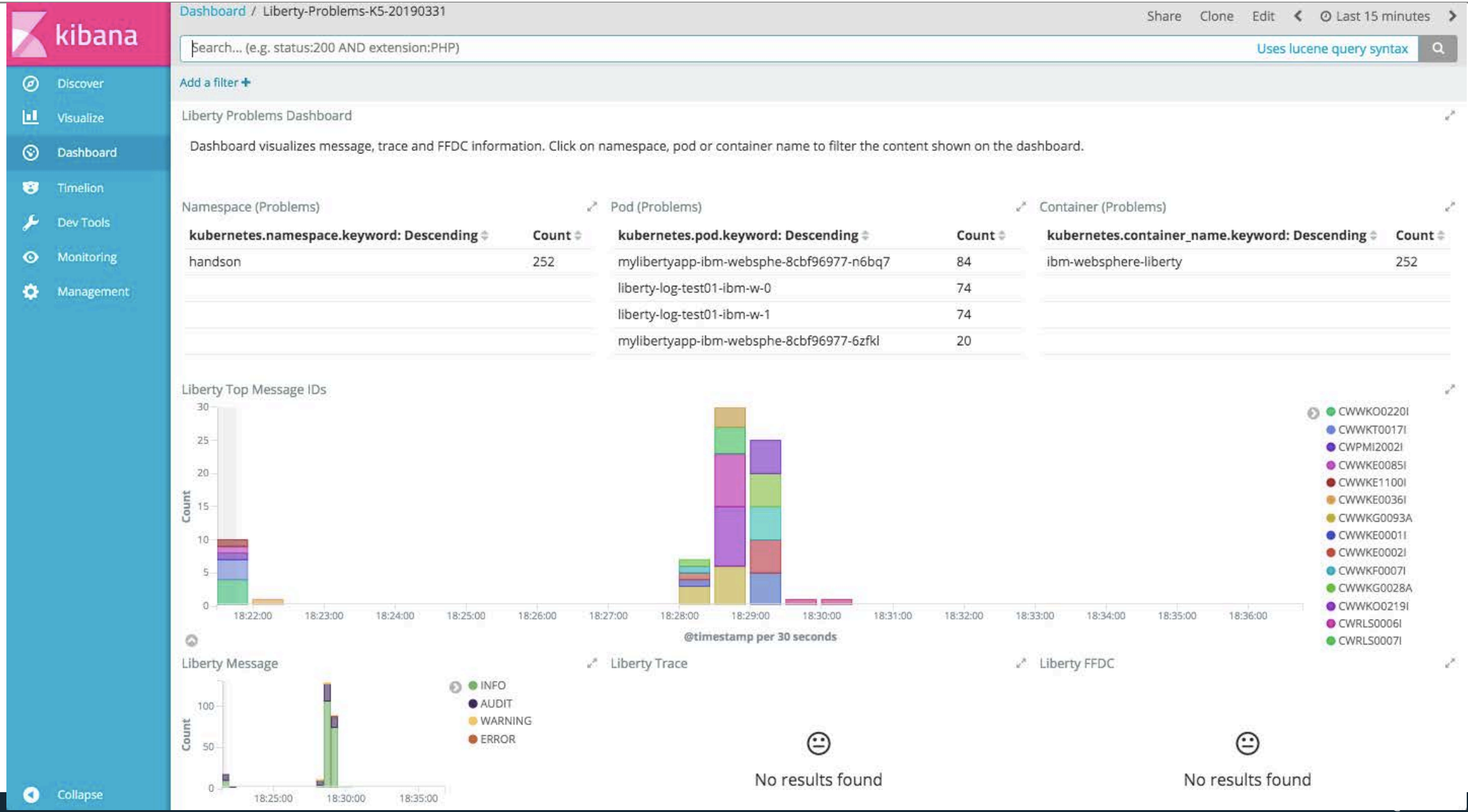
(*) ibm-websphere-liberty v1.9.0の場合

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

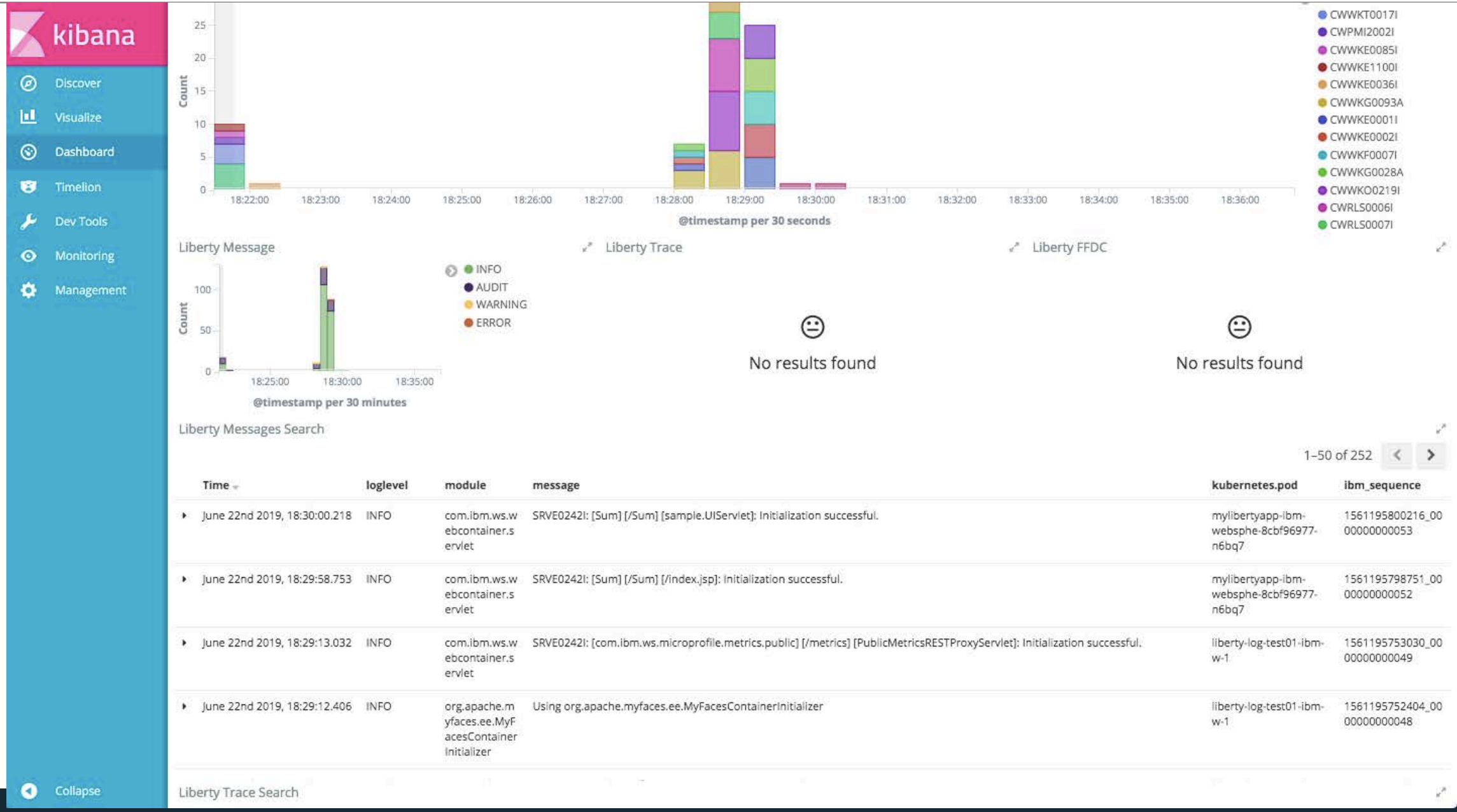
サンプル・ダッシュボードの利用（IBM Certified Cloud Pak）

- Kibanaのサンプル・ダッシュボード
  - 「サンプル・ダッシュボード」のインポート方法は以下のリンクの「次のタスク」欄を参照ください。
    - WAS Liberty Knowledge Center / IBM Cloud Private での Liberty メッセージの分析
    - https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_json_logging.html
  - （補足）LibertyのログがElasticsearchに格納されていない状態でサンプル・ダッシュボードをインポートすると、インポート時にエラー・メッセージが表示されます。
    - 例えば、アクセス・ログがElasticsearchに送られていない場合に、「Liberty Traffic Dashboard」のサンプル・ダッシュボードをインポートすると、インポート時に幾つかエラーが表示されますが、特に問題はありません。
  - （補足）サンプル・ダッシュボードがうまく表示できない場合は、Kibanaで以下の操作などを行ってください。
    - LibertyのログがKibanaで表示できることを確認する。
    - ダッシュボード定義を再インポートする。
  - 以降のページで、幾つかのサンプル・ダッシュボードの表示例を記載します。

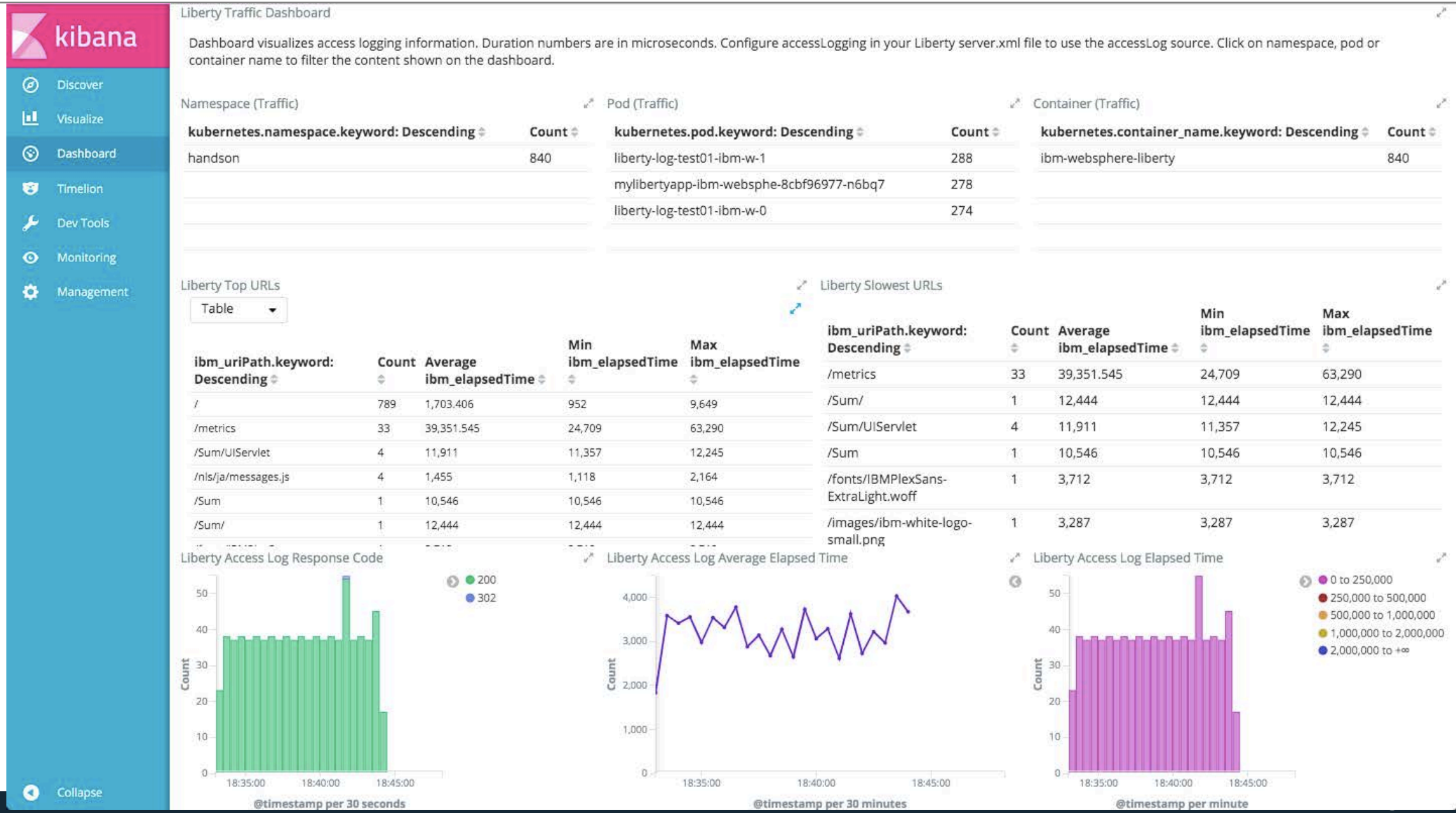
サンプル・ダッシュボードの表示例（Liberty Problems Dashboard）



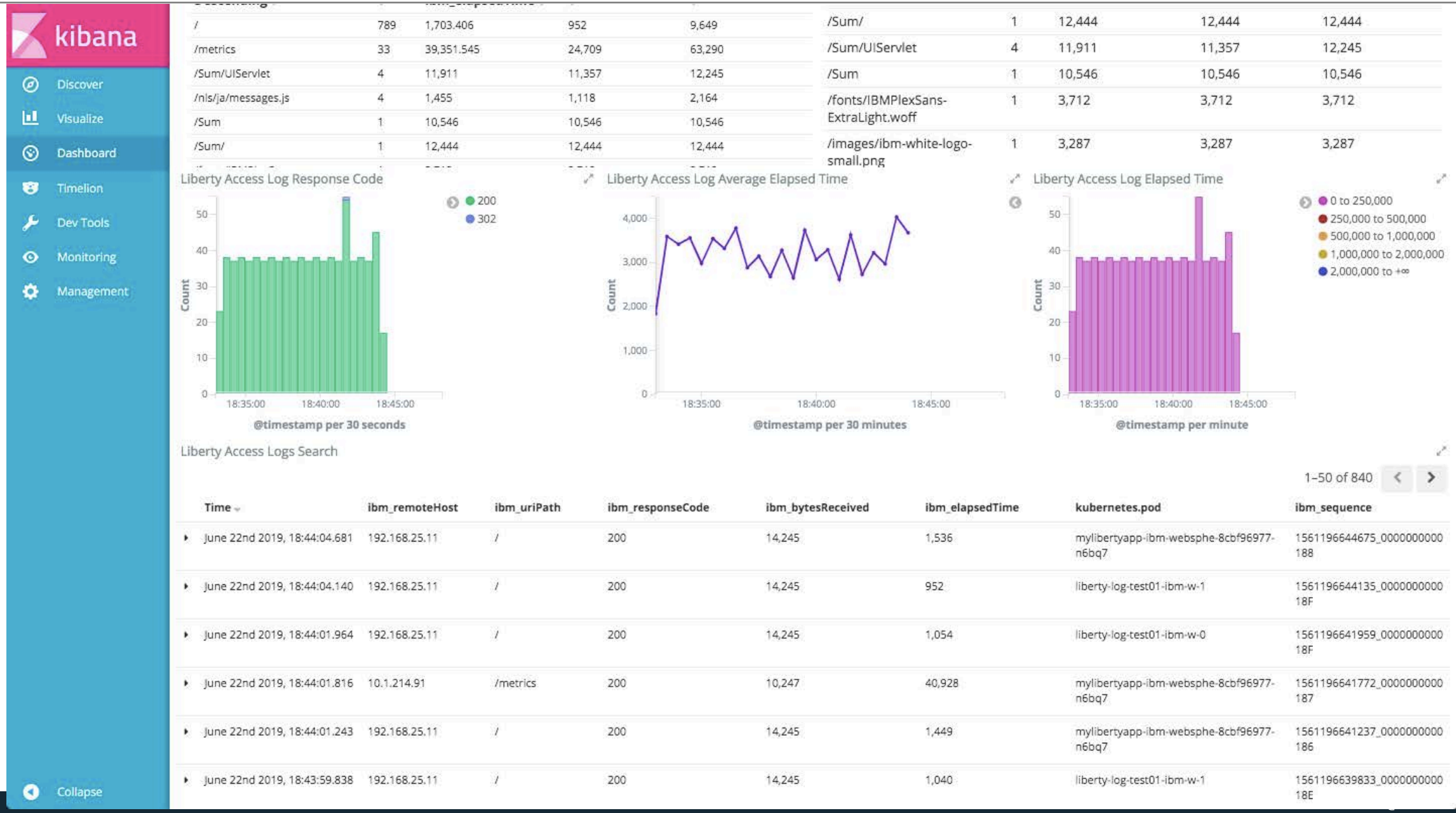
サンプル・ダッシュボードの表示例（Liberty Problems Dashboard）



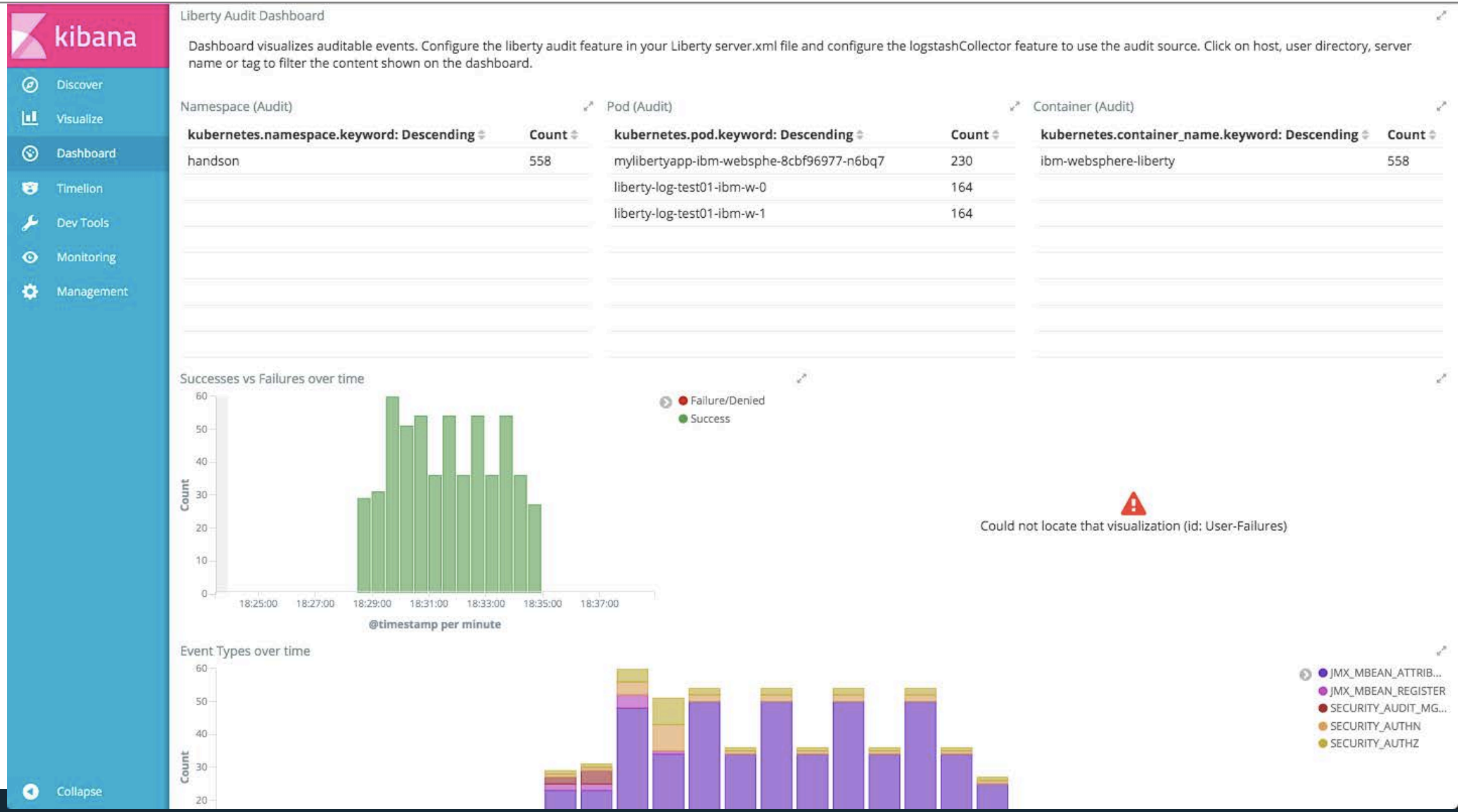
サンプル・ダッシュボードの表示例（Liberty Traffic Dashboard）



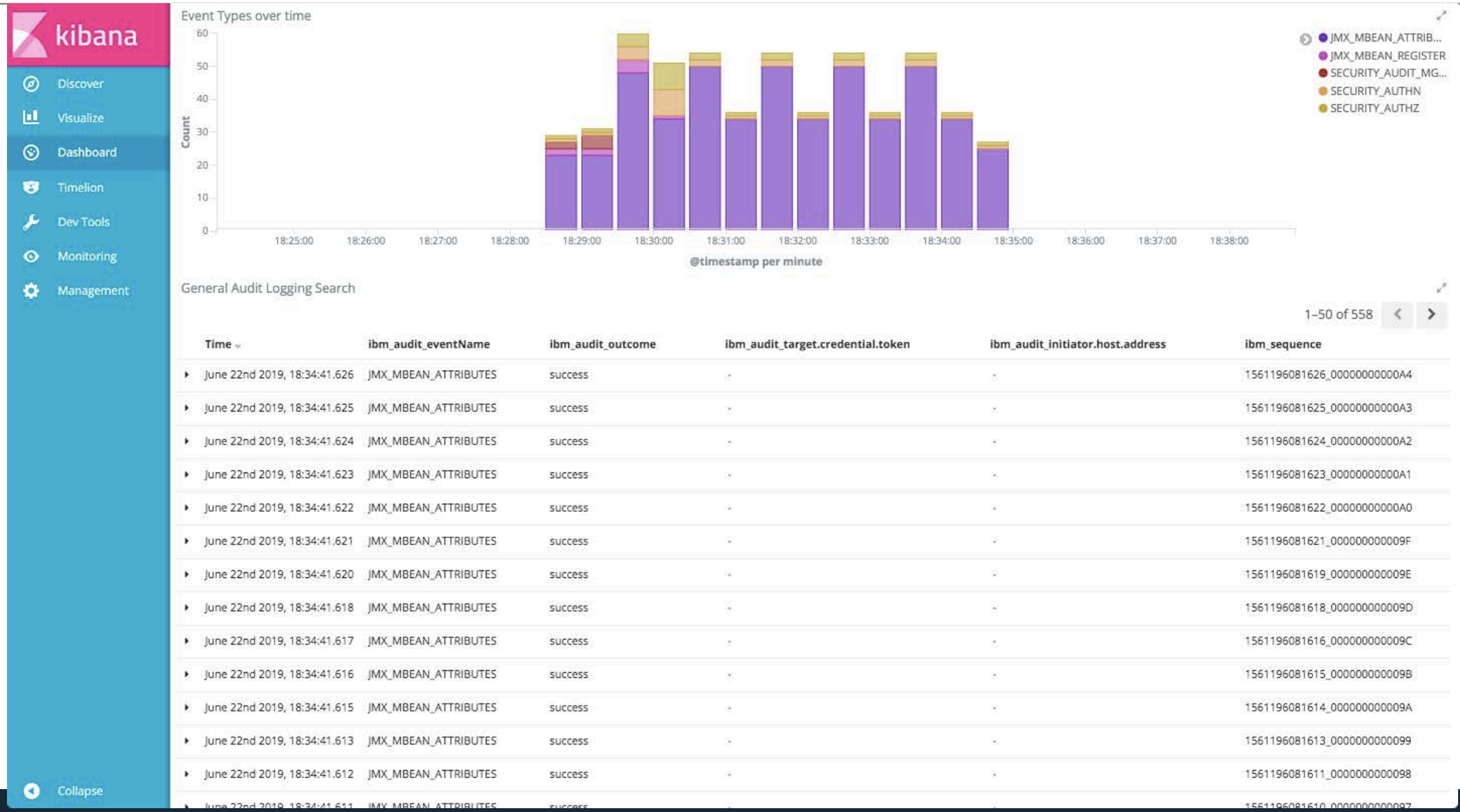
サンプル・ダッシュボードの表示例（Liberty Traffic Dashboard）



サンプル・ダッシュボードの表示例（Liberty Audit Dashboard）

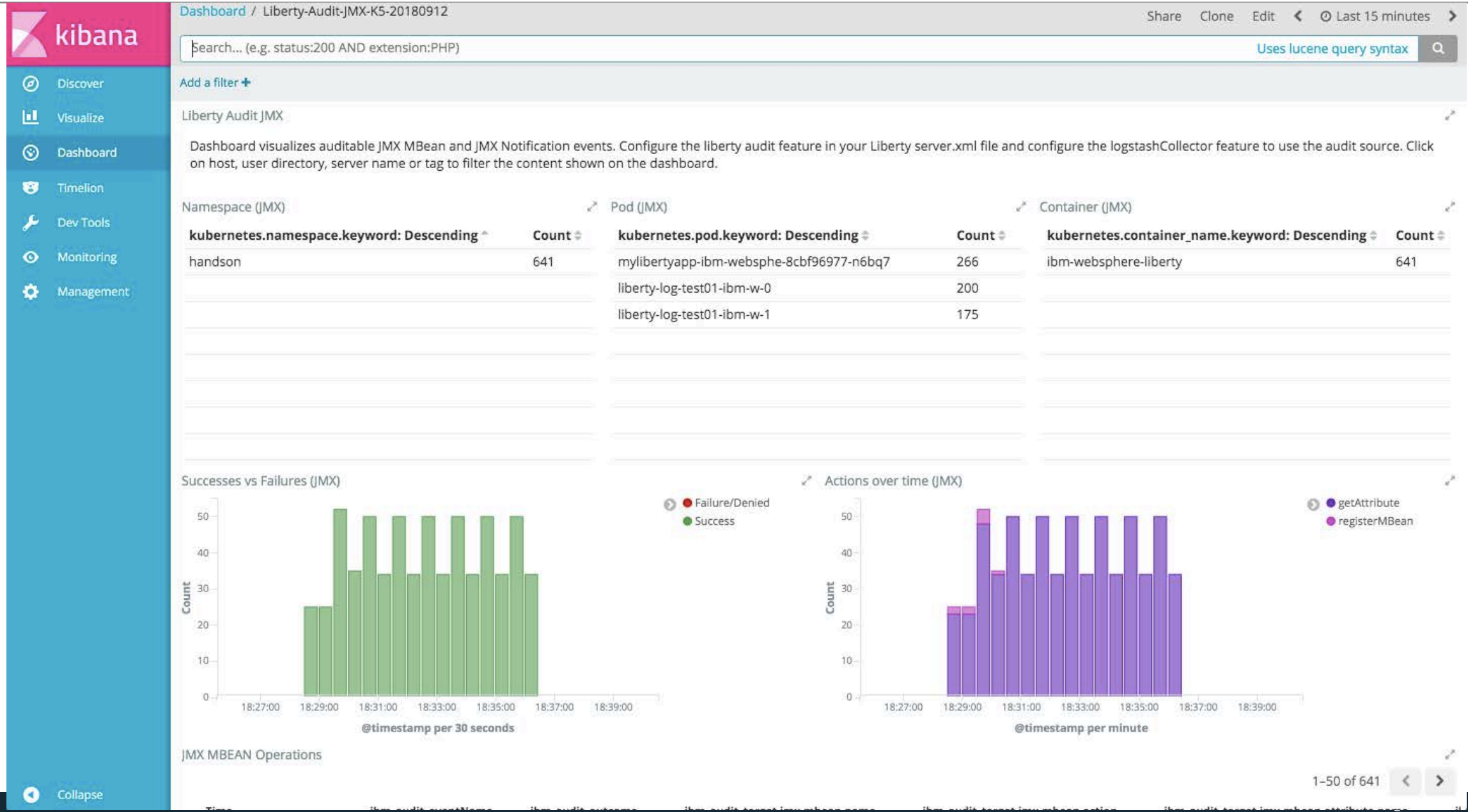


サンプル・ダッシュボードの表示例（Liberty Audit Dashboard）

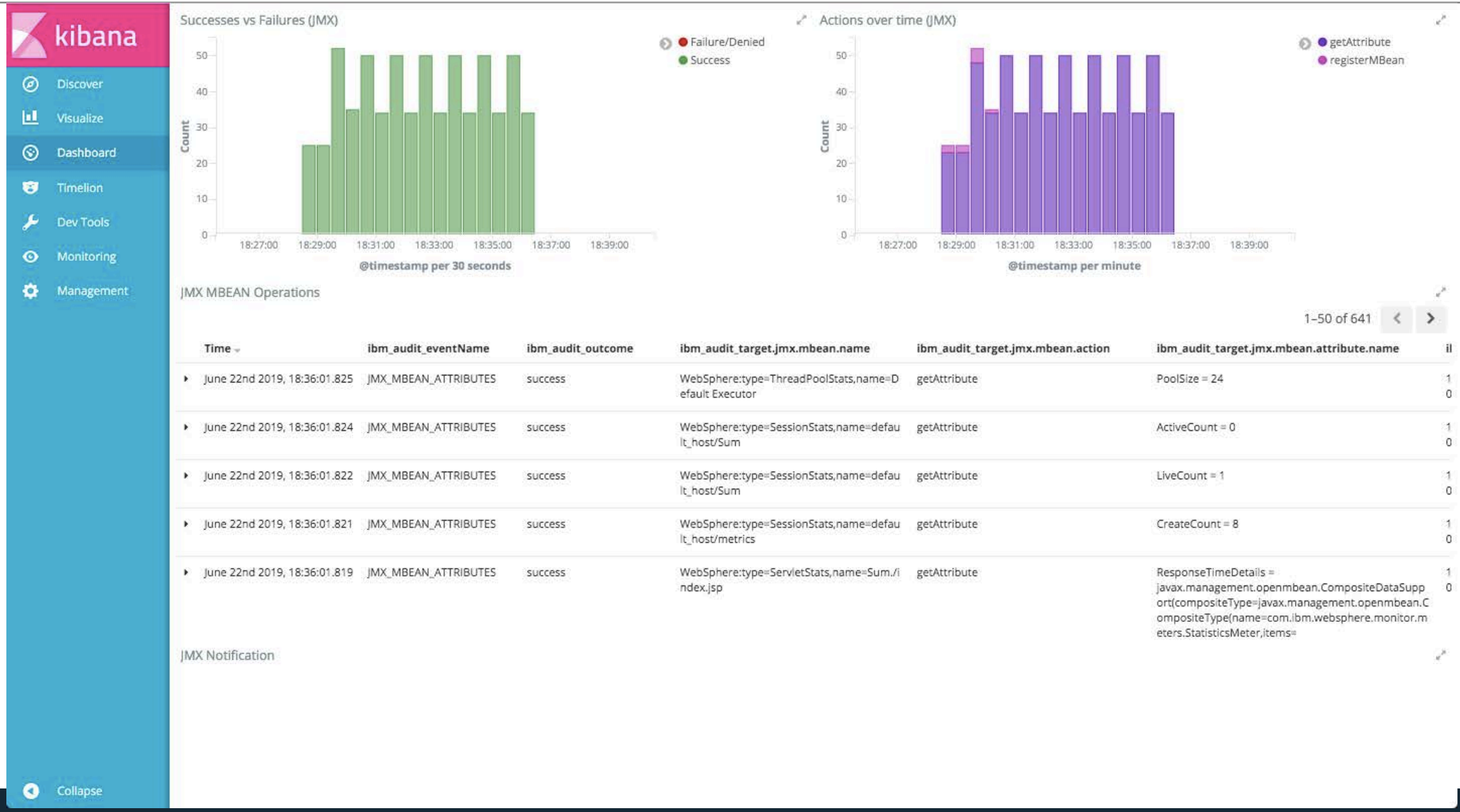


Time	ibm_audit_eventName	ibm_audit_outcome	ibm_audit_target.credential.token	ibm_audit_initiator.host.address	ibm_sequence
June 22nd 2019, 18:34:41.626	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081626_00000000000A4
June 22nd 2019, 18:34:41.625	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081625_00000000000A3
June 22nd 2019, 18:34:41.624	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081624_00000000000A2
June 22nd 2019, 18:34:41.623	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081623_00000000000A1
June 22nd 2019, 18:34:41.622	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081622_00000000000A0
June 22nd 2019, 18:34:41.621	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081621_000000000009F
June 22nd 2019, 18:34:41.620	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081619_000000000009E
June 22nd 2019, 18:34:41.618	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081618_000000000009D
June 22nd 2019, 18:34:41.617	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081616_000000000009C
June 22nd 2019, 18:34:41.616	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081615_000000000009B
June 22nd 2019, 18:34:41.615	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081614_000000000009A
June 22nd 2019, 18:34:41.613	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081613_0000000000099
June 22nd 2019, 18:34:41.612	JMX_MBEAN_ATTRIBUTES	success	-	-	1561196081611_0000000000098

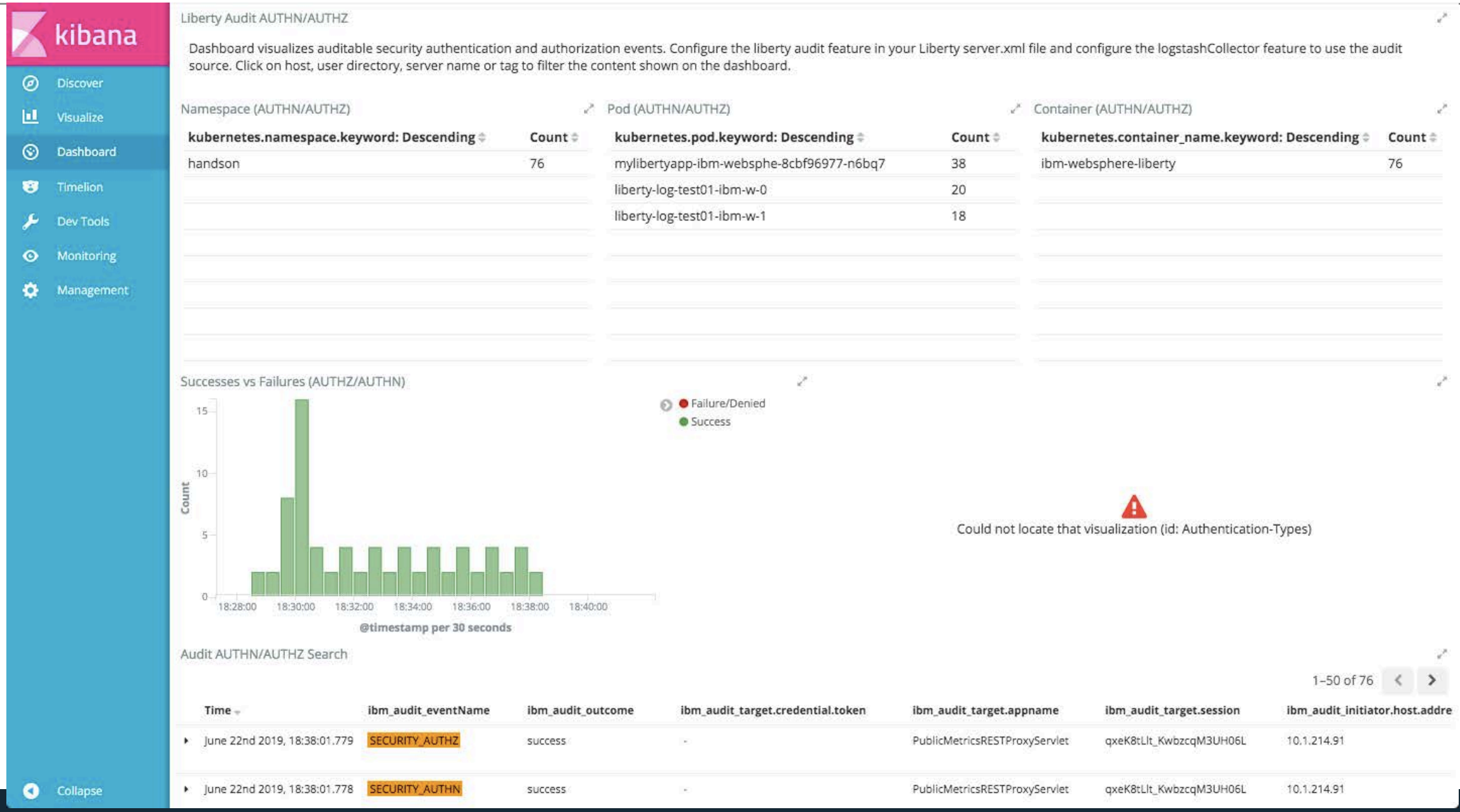
サンプル・ダッシュボードの表示例（Liberty Audit JMX Dashboard）



サンプル・ダッシュボードの表示例（Liberty Audit JMX Dashboard）



サンプル・ダッシュボードの表示例（Liberty Audit AUTHN-AUTHZ Dashboard ）



6. モニタリング

目次(6章)

目次(6章)

ICP環境のWAS Libertyのモニタリングの概要

■ WAS Libertyのモニタリングの概要

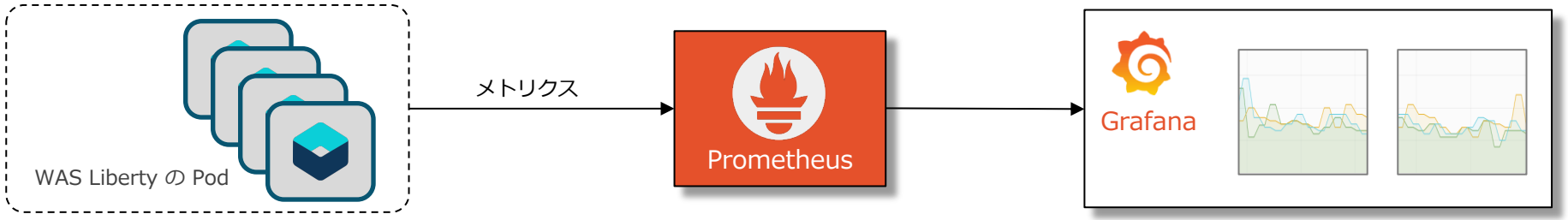
– WAS Libertyは「IBM Certified Cloud Pak」です。

– ロギング、モニタリングなどの運用に必要な機能も組み込まれた状態で製品が提供されています。

- WAS Liberty Knowledge Center / Liberty Cloud Pak の概要
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_cloud_pak_overview.html

– WAS Libertyで「mpMetrics-1.1」および「monitor-1.0」フィーチャーを有効にし、Helm Chartの変数ファイルで「monitoring.enabled」のオプションを有効にすることでPrometheusでモニタリングすることが可能になります。

- WAS Liberty Knowledge Center / IBM Cloud Private でのモニタリング
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_monitor.html
- WAS 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
- GitHub / IBM/charts/stable/ibm-websphere-liberty / monitoring
- https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#monitoring



目次(6章)

IBM提供のWAS 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

(*) ibm-websphere-liberty v1.9.0の場合

Qualifier	Parameter	デフォルト値	説明
monitoring	enabled	false	WAS 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、WAS 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
WAS Liberty Knowledge Center / IBM Cloud Private でのモニタリング
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_monitor.html

目次(6章)

(補足)WAS Libertyで利用可能なメトリクス

- 「mpMetrics-1.1」フィーチャーはMicroProfile Metrics 1.1仕様に準拠する/metrics RESTインターフェースを提供します。
  - 「mpMetrics-1.1」featureを追加すると、/metricsからメトリクスが取得できます。
  - 「monitor-1.0」featureを追加すると、venderスコープのメトリクスも取得できます。

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**）

WAS 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
WAS 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
WAS 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
WAS 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
WAS 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)WAS 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

WAS 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)WAS Libertyで利用可能なメトリクス(baseスコープ)

- (例) ibm-websphere-liberty helm v1.9.0、websphere-liberty:19.0.0.5-javaee8の場合

(例)master node上でcurlコマンドで/metrics/baseのメトリクスを取得した場合（jsonフォーマット）

```
# curl -s -H "Accept: application/json" http://<podのIP addressまたはserviceのIP address>:9080/metrics/base | python -m json.tool
{
    "classloader.currentLoadedClass.count": 13658,
    "classloader.totalLoadedClass.count": 13661,
    "classloader.totalUnloadedClass.count": 3,
    "cpu.availableProcessors": 8,
    "cpu.processCpuLoad": 0.006240493828868387,
    "cpu.systemLoadAverage": 1.3,
    "gc.global.count": 10,
    "gc.global.time": 227,
    "gc.scavenge.count": 141,
    "gc.scavenge.time": 1372,
    "jvm.uptime": 517032,
    "memory.committedHeap": 107347968,
    "memory.maxHeap": 536870912,
    "memory.usedHeap": 61469656,
    "thread.count": 81,
    "thread.daemon.count": 78,
    "thread.max.count": 101
}
#
```

WAS 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

(補足)WAS 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： WAS Liberty 固有のメトリクス
(対応するLibertyのバージョンはリンク先参照のこと)

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

- (例) ibm-websphere-liberty helm v1.9.0、websphere-liberty:19.0.0.5-javaee8の場合

(例)master node上でcurlコマンドで/metrics/vendorのメトリクスを取得した場合（jsonフォーマット）

```
# curl -s -H "Accept: application/json" http://<podのIP addressまたはserviceのIP address>:9080/metrics/vendor | python -m json.tool
{
    "servlet.Sum__index_jsp.request.total": 1,
    "servlet.Sum__index_jsp.responseTime.total": 14278437.0,
    "servlet.Sum_sample_UIServlet.request.total": 1,
    "servlet.Sum_sample_UIServlet.responseTime.total": 18050686.0,
    "servlet.com_ibm_ws_microprofile_metrics_public_PublicMetricsRESTProxyServlet.request.total": 19,
    "servlet.com_ibm_ws_microprofile_metrics_public_PublicMetricsRESTProxyServlet.responseTime.total": 394287436.0,
    "session.default_host_Sum.activeSessions": 0,
    "session.default_host_Sum.create.total": 1,
    "session.default_host_Sum.invalidated.total": 0,
    "session.default_host_Sum.invalidatedbyTimeout.total": 0,
    "session.default_host_Sum.liveSessions": 1,
    "session.default_host_metrics.activeSessions": 0,
    "session.default_host_metrics.create.total": 1,
    "session.default_host_metrics.invalidated.total": 0,
    "session.default_host_metrics.invalidatedbyTimeout.total": 0,
    "session.default_host_metrics.liveSessions": 1,
    "threadpool.Default_Executor.activeThreads": 1,
    "threadpool.Default_Executor.size": 16
}
#
```

WAS 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

目次(6章)

モニタリングの設定の流れ

- ICP環境でWAS Libertyをモニタリングするには以下の設定が必要です。
  - (1)WAS Libertyの設定
    - server.xmlまたはserver.xmlフラグメントで「mpMetrics-1.1」および「monitor-1.0」フィーチャーを有効化しメトリクスが取得できるように構成します。
    - WAS LibertyのDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーをインストールします。
  - (2)Helm Chartの設定
    - Helmデプロイ時に使用するHelmの変数ファイルで「monitoring.enabled」のオプションを有効にします。
  - (3)Grafanaのサンプル・ダッシュボードの利用
    - IBM Certified Cloud Pakで提供されるGrafanaのサンプル・ダッシュボードをインポートします。

モニタリングの設定 - (1)WAS Libertyの設定

- ICP環境でWAS Libertyをモニタリングするには、WAS Liberty側で以下の設定が必要です。
  - server.xmlの構成
    - server.xmlまたはserver.xmlフラグメントで「mpMetrics-1.1」および「monitor-1.0」フィーチャーを有効化しメトリクスが取得できるように構成します。
  - Dockerイメージの構成
    - WAS LibertyのDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーをインストールします。
  - 参考リンク
    - WAS 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
    - WAS Liberty Knowledge Center / Liberty サーバー・ランタイム環境のモニター / monitor-1.0 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_monitor10.html
    - WAS 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
    - GitHub / IBM/charts/stable/ibm-websphere-liberty / monitoring
    - https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#monitoring
    - GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Enterprise Functionality
    - https://github.com/WASdev/ci.docker#enterprise-functionality

モニタリングの設定 - (1)WAS Libertyの設定

■ server.xmlの構成

– server.xmlまたはserver.xmlフラグメントで「mpMetrics-1.1」および「monitor-1.0」フィーチャー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.xmlまたはserver.xmlフラグメントへの反映方法は、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

```
<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）

- WAS 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
- GitHub / IBM/charts/stable/ibm-websphere-liberty / monitoring
- https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#monitoring
- GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Enterprise Functionality
- https://github.com/WASdev/ci.docker#enterprise-functionality

モニタリングの設定 - (1)WAS Libertyの設定

■ Dockerイメージの構成

– 使用するDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーがインストールされているか確認します。

- DockerHub / websphere-liberty(Official Images)で、使用したいDockerイメージのTagをクリックすると該当イメージをビルドする際に使用されたDockerfileが表示されます。
- 表示されたDockerfileで「monitor-1.0」および「mpMetrics-1.1」フィーチャーがインストールされているか確認します。
  – DockerHub / websphere-liberty(Official Images)
  – https://hub.docker.com/_/websphere-liberty

– (例)19.0.0.6-javaee8の場合

(19.0.0.6-javaee8のDockerfile)

```
(略)
FROM websphere-liberty:19.0.0.6-kernel
(略)
RUN if [ ! -z $REPOSITORIES_PROPERTIES ]; then mkdir /opt/ibm/wlp/etc/ ¥
  && echo $REPOSITORIES_PROPERTIES > /opt/ibm/wlp/etc/repositories.properties; fi ¥
  && installUtility install --acceptLicense ¥
    appSecurity-2.0 ldapRegistry-3.0 ¥
    localConnector-1.0 monitor-1.0 requestTiming-1.0 restConnector-2.0 sessionCache-1.0 ¥
    sessionDatabase-1.0 ssl-1.0 transportSecurity-1.0 webCache-1.0 webProfile-8.0 ¥
    appSecurityClient-1.0 javaee-8.0 javaeeClient-8.0 openidConnectClient-1.0 monitor-1.0
microProfile-2.0 microProfile-2.1 microProfile-2.2 ¥
(略)
```

monitor-1.0がインストールされています。

「mpMetrics-1.1」は明示的にインストールされていませんが、「microProfile-2.2」の前提としてインストールされます。
WAS Liberty Knowledge Center / MicroProfile 2.2
https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.liberty.autogen.nd.doc/ae/rwlp_feature_microProfile-2.2.html

モニタリングの設定 - (1)WAS Libertyの設定

■ Dockerイメージの構成

– 使用するDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーがインストールされているか確認します。

- DockerHub / websphere-liberty(Official Images)で、使用したいDockerイメージのTagをクリックすると該当イメージをビルドする際に使用されたDockerfileが表示されます。
- 表示されたDockerfileで「monitor-1.0」および「mpMetrics-1.1」フィーチャーがインストールされているか確認します。
  – DockerHub / websphere-liberty(Official Images)
  – https://hub.docker.com/_/websphere-liberty

– (例)19.0.0.6-webProfile8の場合

（19.0.0.6-webProfile8のDockerfile）

```
(略)
FROM websphere-liberty:19.0.0.6-kernel
(略)
RUN if [ ! -z $REPOSITORIES_PROPERTIES ]; then mkdir /opt/ibm/wlp/etc/ ¥
  && echo $REPOSITORIES_PROPERTIES > /opt/ibm/wlp/etc/repositories.properties; fi ¥
  && installUtility install --acceptLicense ¥
    appSecurity-2.0 ldapRegistry-3.0 ¥
    localConnector-1.0 monitor-1.0 requestTiming-1.0 restConnector-2.0 sessionCache-1.0 ¥
    sessionDatabase-1.0 ssl-1.0 transportSecurity-1.0 webCache-1.0 webProfile-8.0 ¥
(略)
```

monitor-1.0がインストールされています。

「mpMetrics-1.1」がインストールされていないため、自分でDockerfileを作成し、明示的に「mpMetrics-1.1」をインストールしてビルドする必要があります。

モニタリングの設定 - (1)WAS Libertyの設定

■ Dockerイメージの構成

– 使用するDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーがインストールされていない場合はDockerイメージのビルド時に該当フィーチャーをインストールする必要があります。

- server.xmlまたはserver.xmlフラグメントのどこにフィーチャーを反映するかによってイメージのビルドの仕方が変わります。

– (方法1)「mpMetrics-1.1」と「monitor-1.0」フィーチャーの追加と構成をしたserver.xml（ファイルの内容は前述の「server.xmlの構成」の定義を参照）を手動で作成し、ビルド時にコピーしてイメージに組み込む場合

- Dockerfile内に明示的にフィーチャーを記載しなくても「installUtility install –acceptLicense defaultServer」コマンドでserver.xmlに定義されているフィーチャーは自動でインストールされます。

（Dockerfileの例）

```
FROM websphere-liberty:19.0.0.6-webProfile8
COPY --chown=1001:0  Sample.war /config/dropins/
COPY --chown=1001:0  server.xml /config/
RUN installUtility install --acceptLicense defaultServer
```

- (補足)server.xmlはコンテナ内の以下のディレクトリに配置されます。
  – /config/server.xml

モニタリングの設定 - (1)WAS Libertyの設定

■ Dockerイメージの構成

– 使用するDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーがインストールされていない場合はDockerイメージのビルド時に該当フィーチャーをインストールする必要があります。
  • server.xmlまたはserver.xmlフラグメントのどこにフィーチャーを反映するかによってイメージのビルドの仕方が変わります。

– (方法2) websphere-libertyのDockerイメージが提供する機能(変数ARGとシェル)を利用する場合
  • Dockerfile内のARG変数で「ARG MP_MONITORING=true」と設定し、「RUN configure.sh」でシェルを実行させると、「mpMetrics-1.1」と「monitor-1.0」フィーチャーの追加と構成が事前定義されたserver.xmlフラグメント（mp-monitoring.xml）が自動でイメージに組み込まれてビルドされます。
    – GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Enterprise Functionality
    – https://github.com/WASdev/ci.docker#enterprise-functionality

（Dockerfileの例）

```
FROM websphere-liberty:19.0.0.6-webProfile8
COPY --chown=1001:0  Sample.war /config/dropins/

# Optional functionality
ARG MP_MONITORING=true

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

「MP_MONITORING=true」とすることで、事前定義されたmp-monitoring.xmlが自動でイメージに組み込まれます。

configure.shでは、mp-monitoring.xmlの組み込みと、「installUtility install –acceptLicense defaultServer」コマンドを実施しフィーチャーのインストールも実施します。
(configure.shはtag:kernelのイメージに組み込まれているため、記載するだけで実行できます。)

  • (補足)mp-monitoring.xmlはコンテナ内の以下のディレクトリに配置されます。
    – /config/configDropins/overrides/mp-monitoring.xml

モニタリングの設定 - (1)WAS Libertyの設定

■ Dockerイメージの構成

– 使用するDockerイメージに「mpMetrics-1.1」および「monitor-1.0」フィーチャーがインストールされていない場合はDockerイメージのビルド時に該当フィーチャーをインストールする必要があります。

- server.xmlまたはserver.xmlフラグメントのどこにフィーチャーを反映するかによってイメージのビルドの仕方が変わります。

– (方法3)「mpMetrics-1.1」と「monitor-1.0」フィーチャーの追加と構成をしたserver.xmlフラグメントをconfigMapとして作成し、Helmの変数ファイルの「image.serverOverridesConfigMapName」でこのconfigMapを指定する場合

- ビルド時にserver.xmlをコピーせず、Helm Chart側でconfigmapを使用してserver.xmlフラグメントを配置する場合は、不足しているフィーチャーはDockerfileに明示的に記載してインストールしておく必要があります。

（Dockerfileの例）

```
FROM websphere-liberty:19.0.0.6-webProfile8
COPY --chown=1001:0  Sample.war /config/dropins/
RUN installUtility install --acceptLicense mpMetrics-1.1
```

この例では、ベースイメージとして使用している「19.0.0.6-webProfile8」は、「monitor-1.0」はインストールされているので、不足している「mpMetrics-1.1」のみを明示的に追加でインストールします。

- (補足)server-overrides.xmlファイルをconfigmapとして作成し、Helmの変数ファイルの「image.serverOverridesConfigMapName: <configmap name>」を使用してserver.xmlフラグメントを配置する場合は、server-overrides.xmlファイルはコンテナ内の以下のディレクトリに配置されます。
  – /config/configDropins/overrides/server-overrides.xml

モニタリングの設定 - (2)Helmチャートの設定

■ IBM提供のHelm Chartの「コンソール出力」に関する設定項目

– 前述の「IBM 提供の Liberty のHelm Chart のパラメーター：モニタリング関連」を参照ください。

• デフォルトではモニタリングは無効のため、Helmの変数ファイルの「monitoring.enabled」の設定値を「true」に変更します。

– 手順概要

• (1) Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-monitor-test01.yaml
```

• (2) 抽出した変数ファイルのモニタリングに関するパラメーターを修正します。

(例)values-monitor-test01.yaml

```
(略)
monitoring:
  enabled: true
(略)
```

「monitoring.entabled」の値を「false」から「true」に変更します。

– (補足) 前述の「(1)WAS Libertyの構成」/「Dockerイメージの構成」の(方法3)の場合でConfigMapを使用してserver.xmlフラグメントを配置する場合は変数ファイルの以下の箇所も修正します。(3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」(3)の方法参照)

```
images:
(略)
  serverOverridesConfigMapName: "custom-server-overrides"
```

server-overrides.xmlから作成したconfigmap名を指定します。

• (3) 編集した変数ファイルを使用して、Helm Chartをデプロイします。

```
helm upgrade --install monitor-ingress-ssl -f ./values-monitor-test01.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

モニタリングの設定 - (3)Grafanaのサンプル・ダッシュボードの利用

■ Grafanaのサンプル・ダッシュボード

– WebSphere Libertyは「IBM Certified Cloud Pak」です。

– ロギング、モニタリングなどの運用に必要な機能も組み込まれた状態で製品が提供されています。

- WAS Liberty Knowledge Center / Liberty Cloud Pak の概要
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_cloud_pak_overview.html

– Prometheusで収集したメトリクスをわかりやすく表示するためのGrafanaの「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

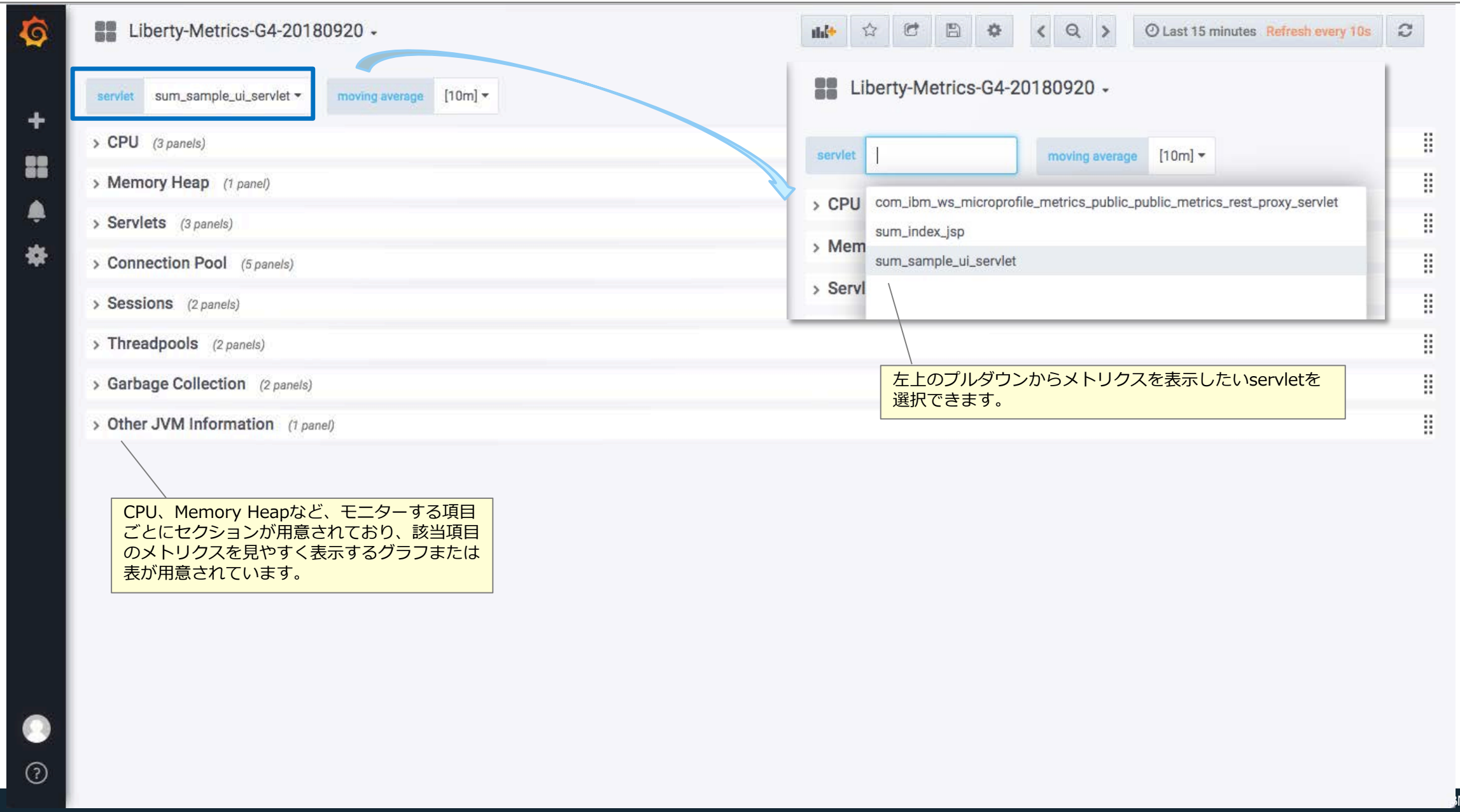
(*) ibm-websphere-liberty v1.9.0の場合

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-grafana-dashboard.json	• Liberty Metrics xx	• Prometheusで収集したメトリクスをビジュアル化するダッシュボード

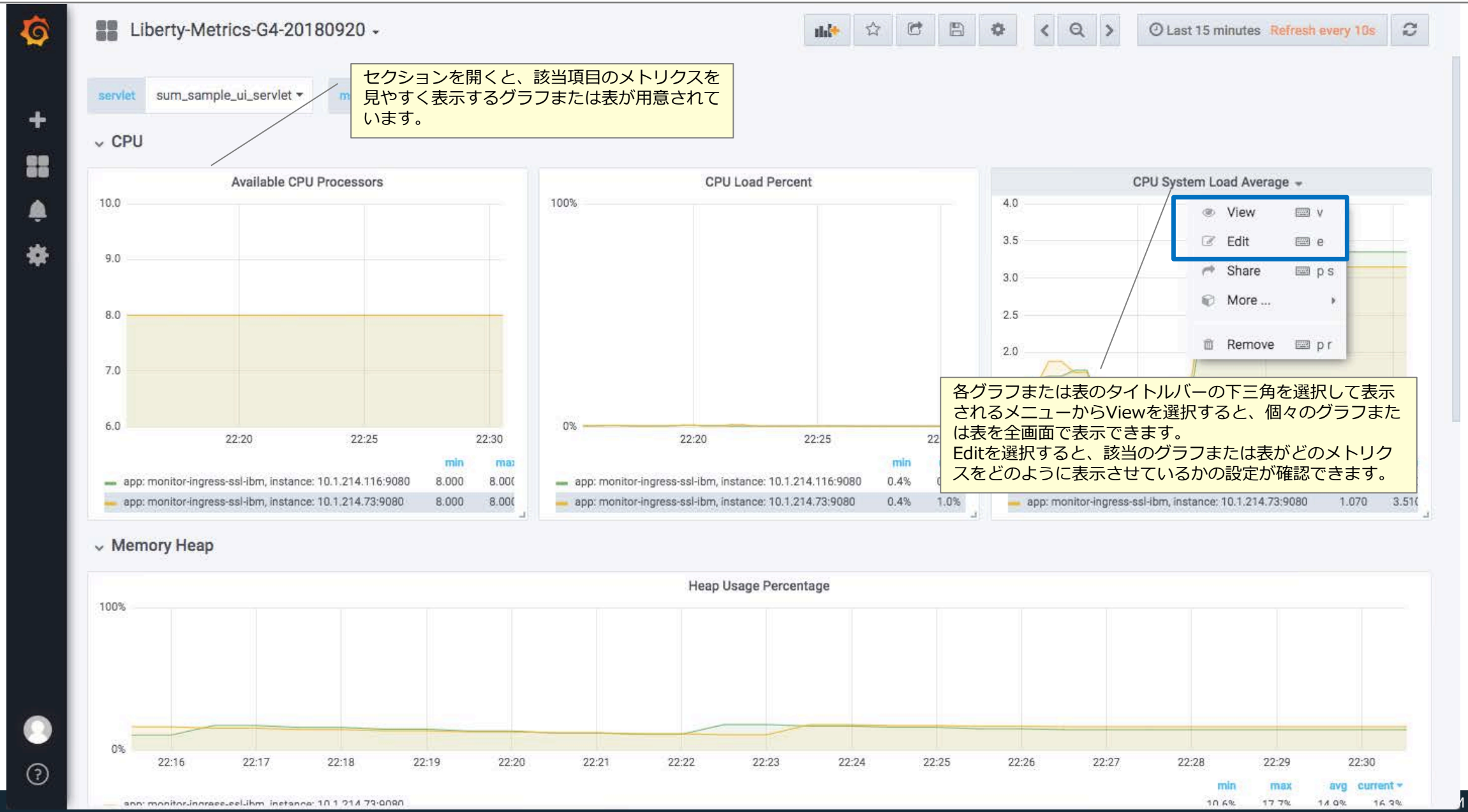
– 「サンプル・ダッシュボード」のインポート方法は以下のリンクの「手順」欄を参照ください。

- WAS Liberty Knowledge Center / IBM Cloud Private でのモニタリング
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_monitor.html

Grafanaのサンプル・ダッシュボードの表示例（全般）

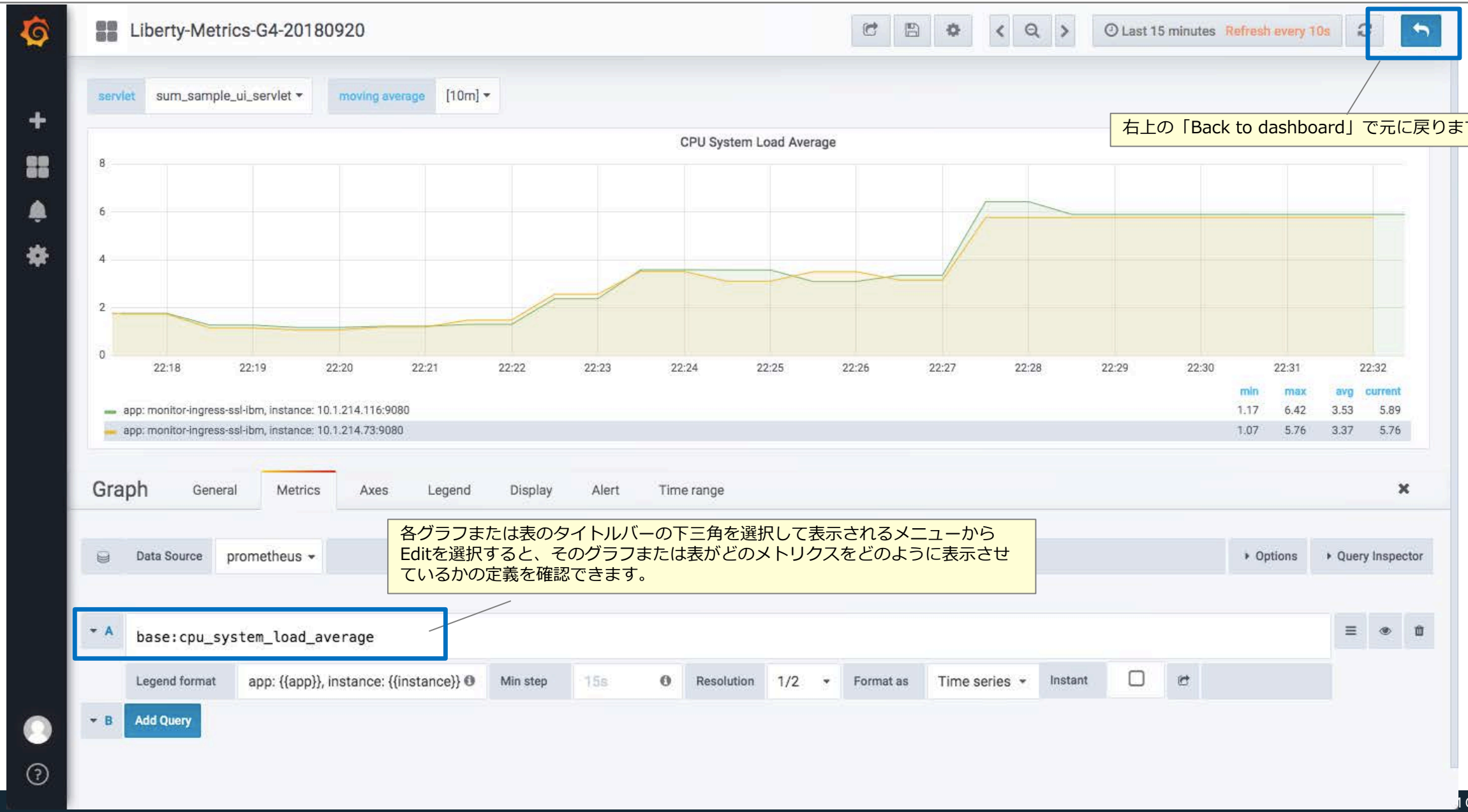


Grafanaのサンプル・ダッシュボードの表示例（全般）



Grafanaのサンプル・ダッシュボードの表示例（全般）

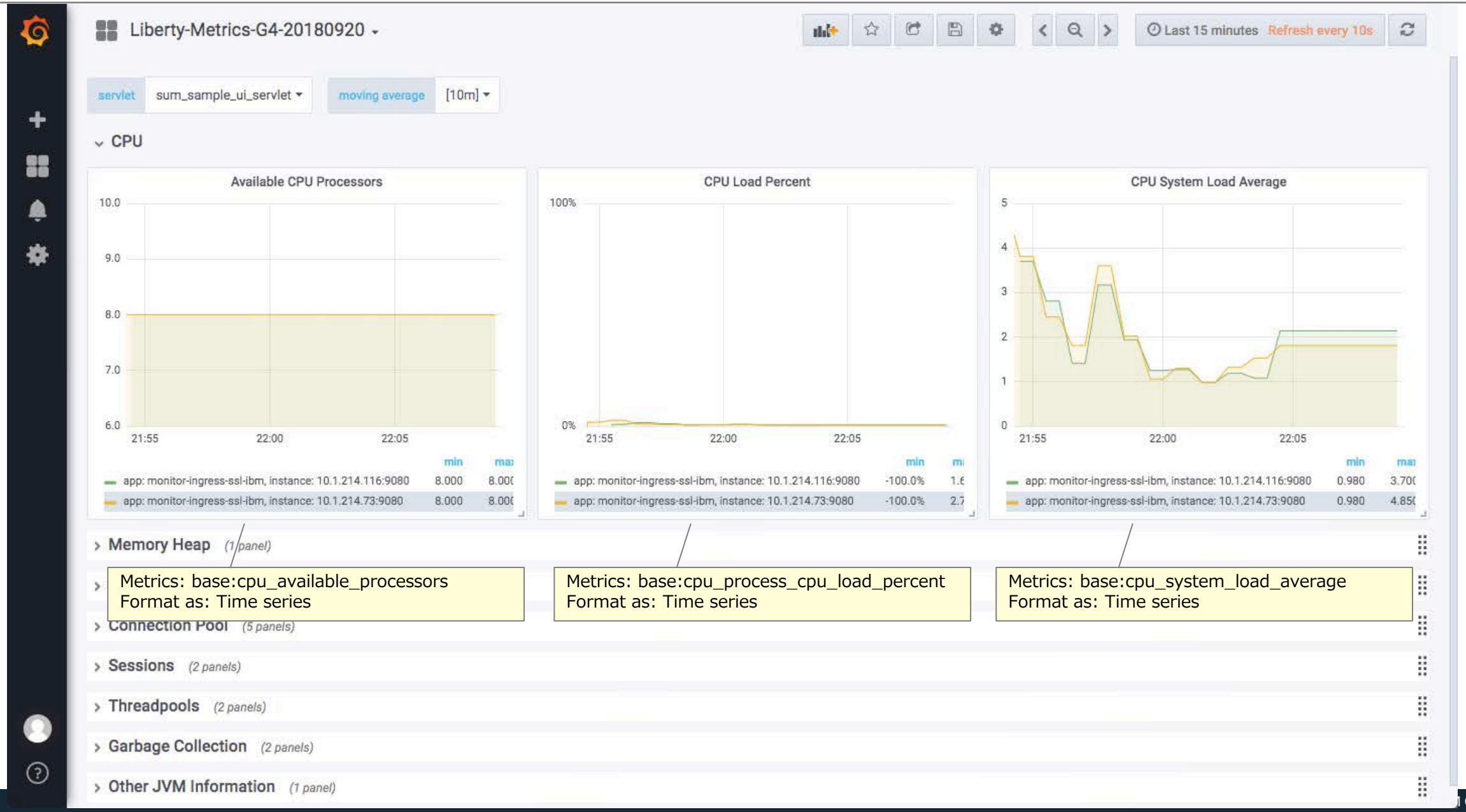
Grafanaのサンプル・ダッシュボードの表示例（全般）



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。

Grafanaのサンプル・ダッシュボードの表示例（CPU）

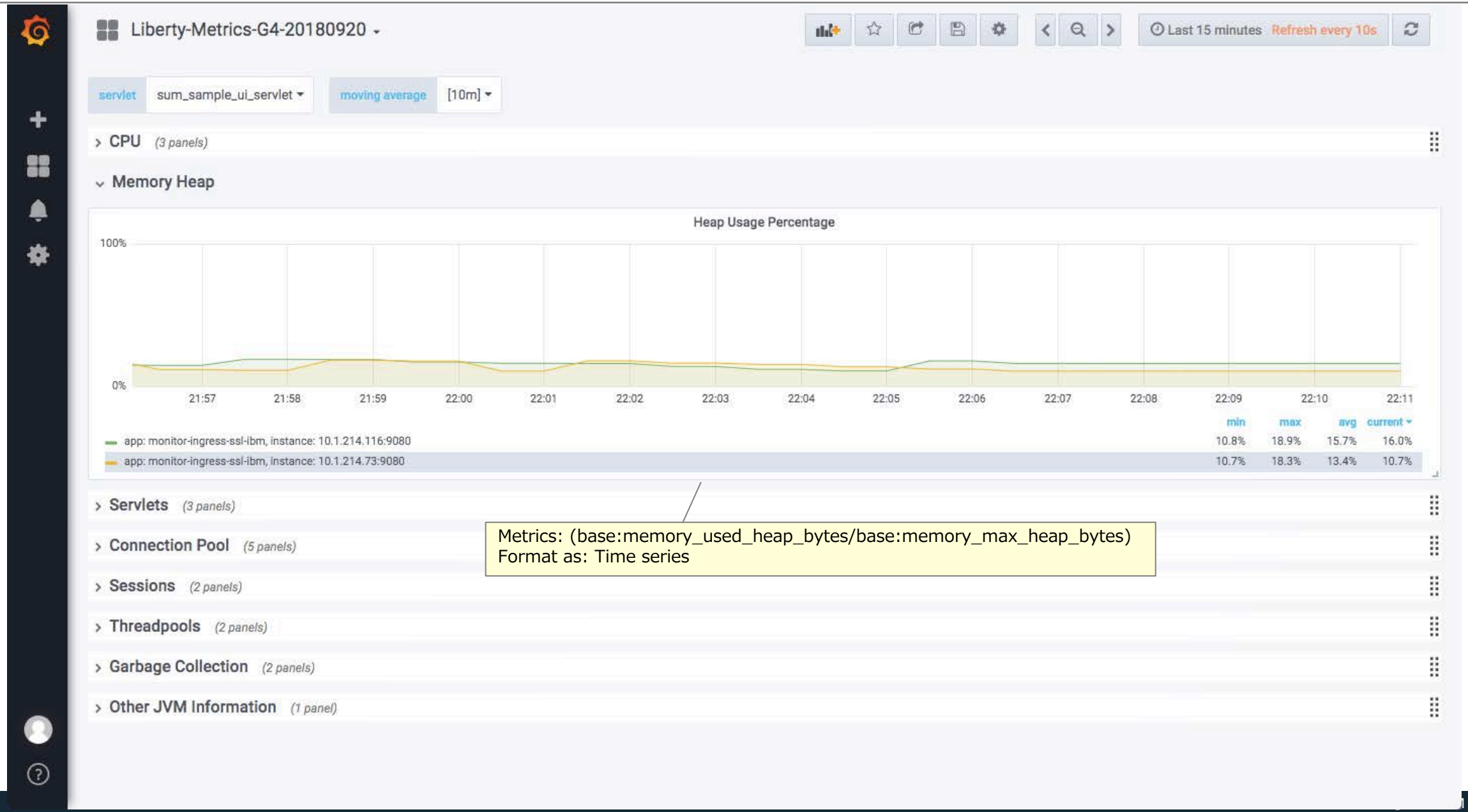


Metrics: base:cpu_available_processors
Format as: Time series

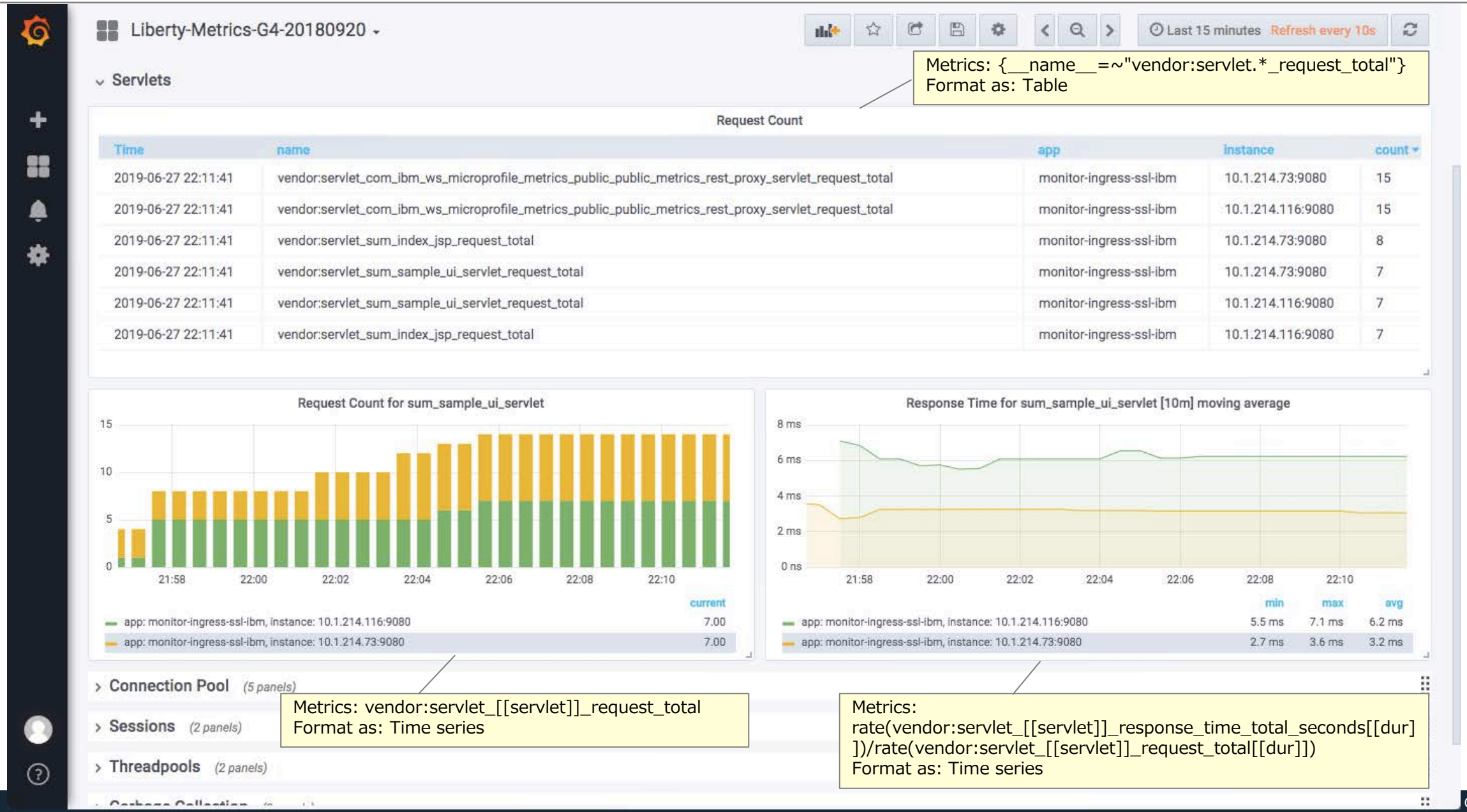
Metrics: base:cpu_process_cpu_load_percent
Format as: Time series

Metrics: base:cpu_system_load_average
Format as: Time series

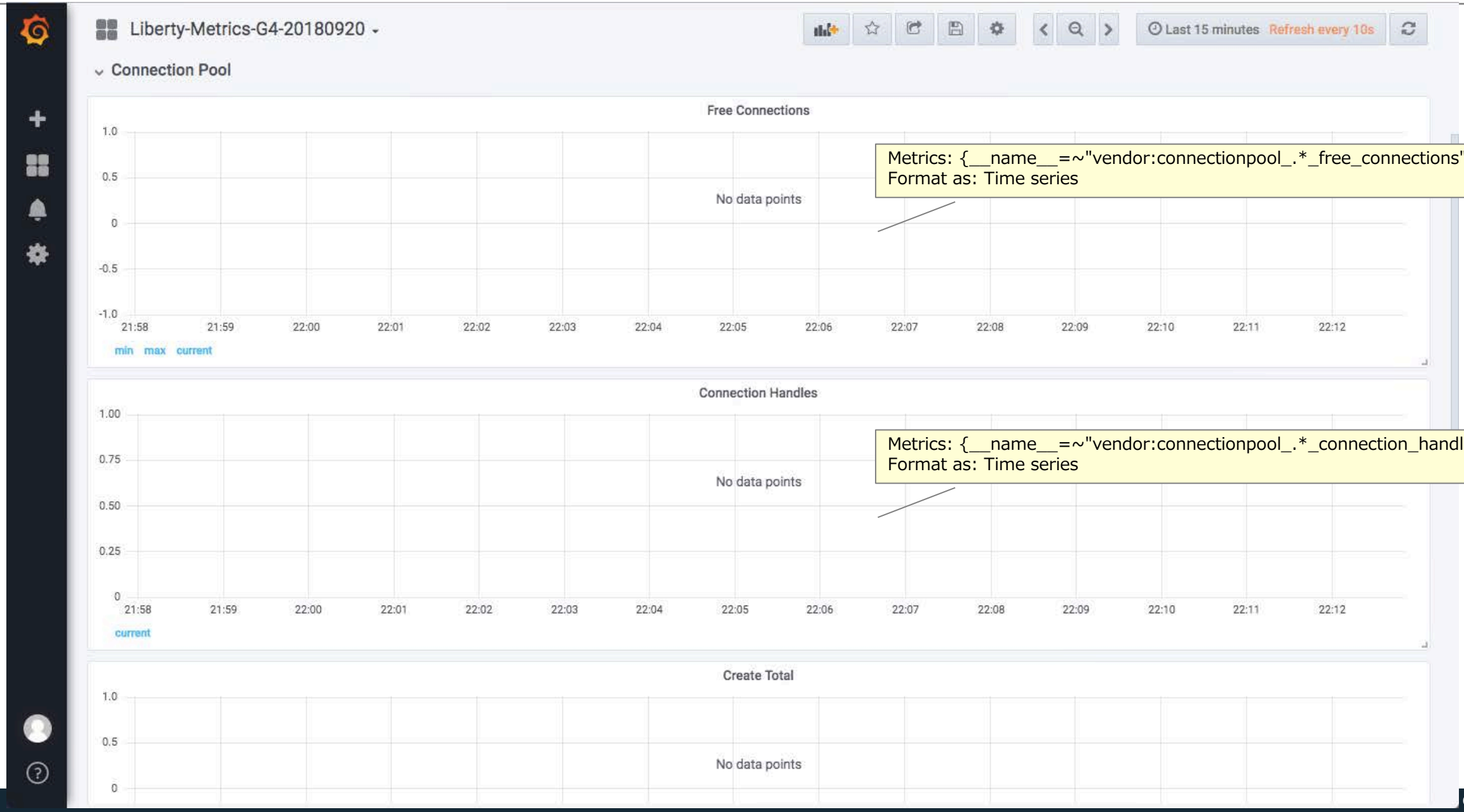
Grafanaのサンプル・ダッシュボードの表示例（Memory Heap）



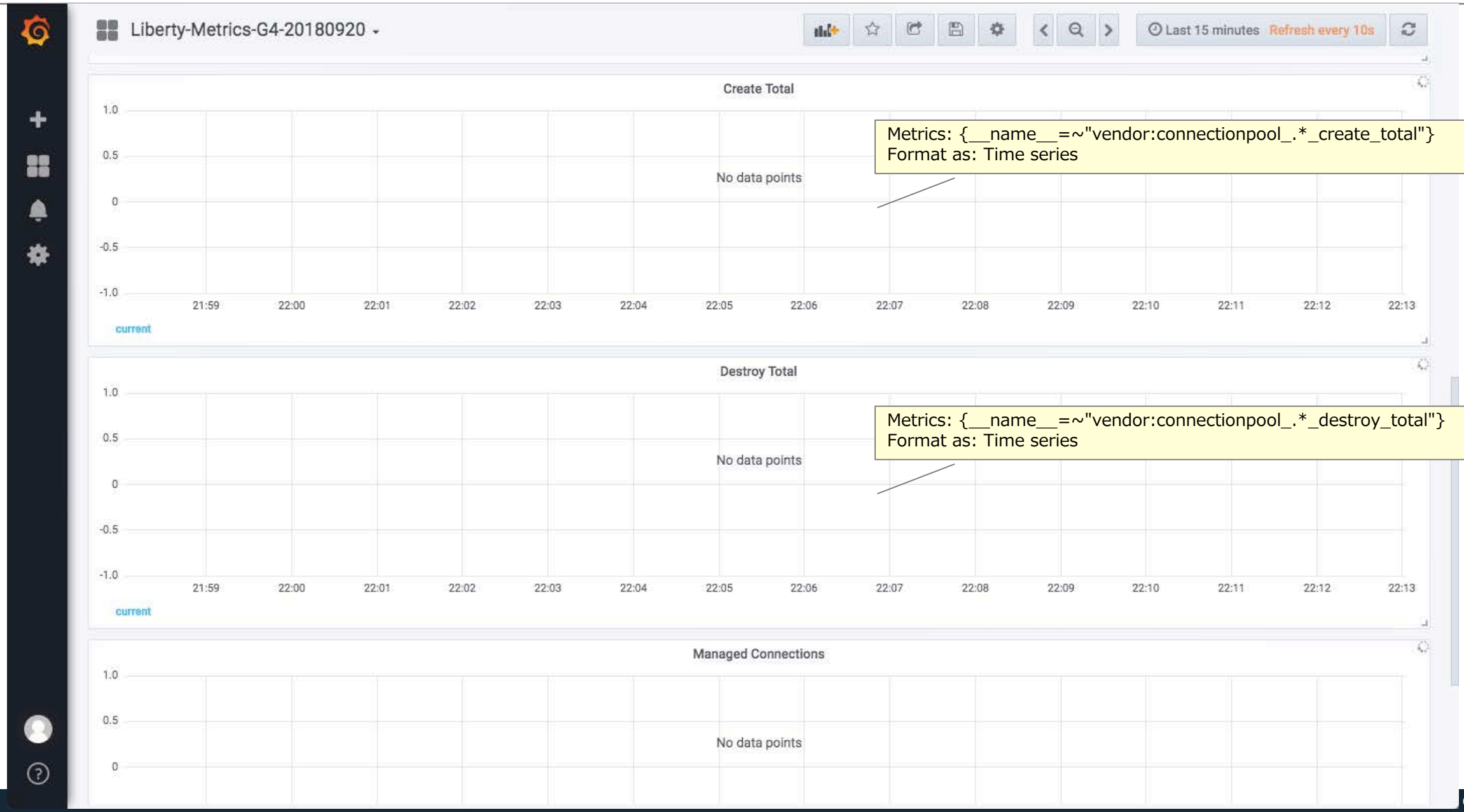
Grafanaのサンプル・ダッシュボードの表示例（Servlets）



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



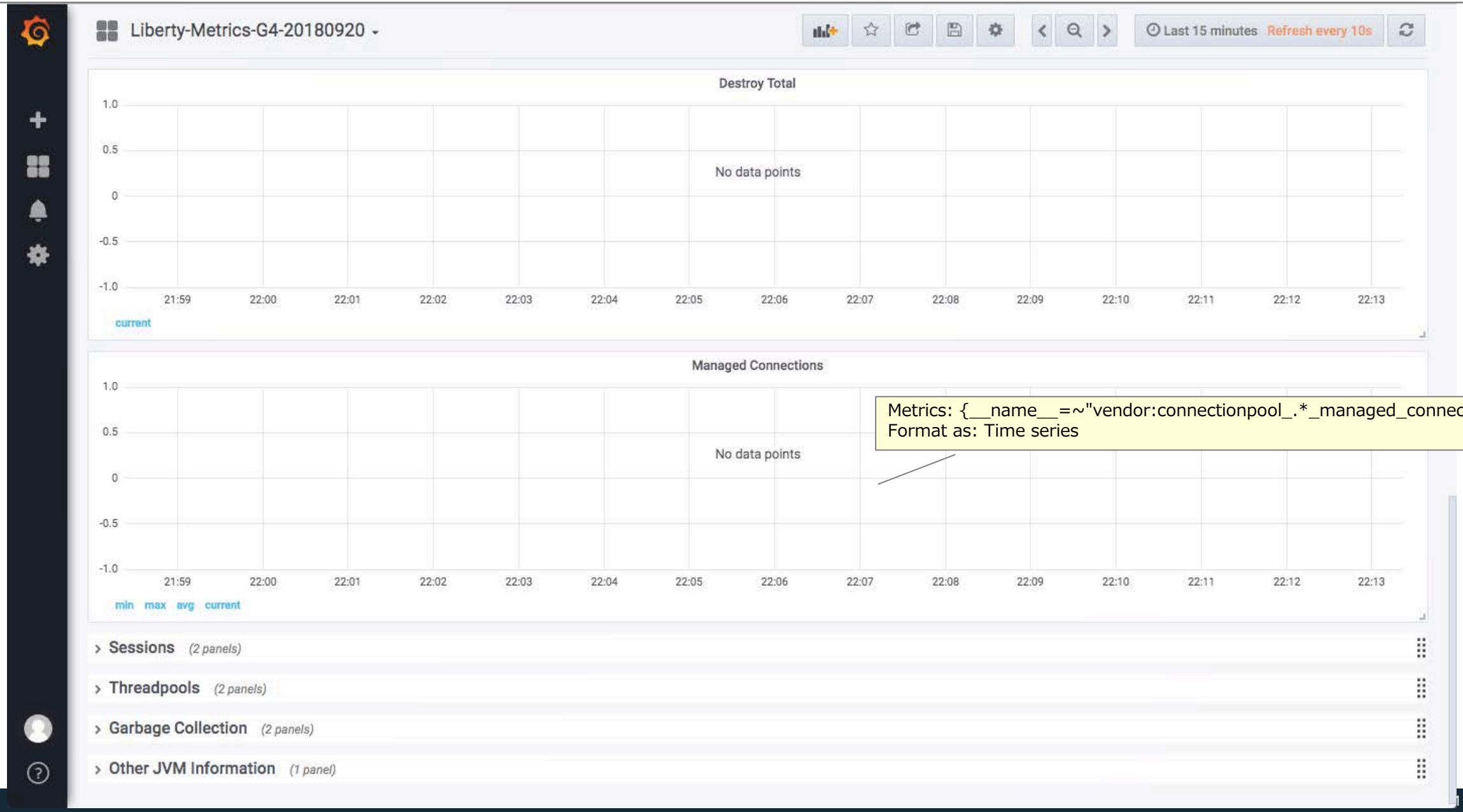
Grafanaのサンプル・ダッシュボードの表示例（Connection Pool）



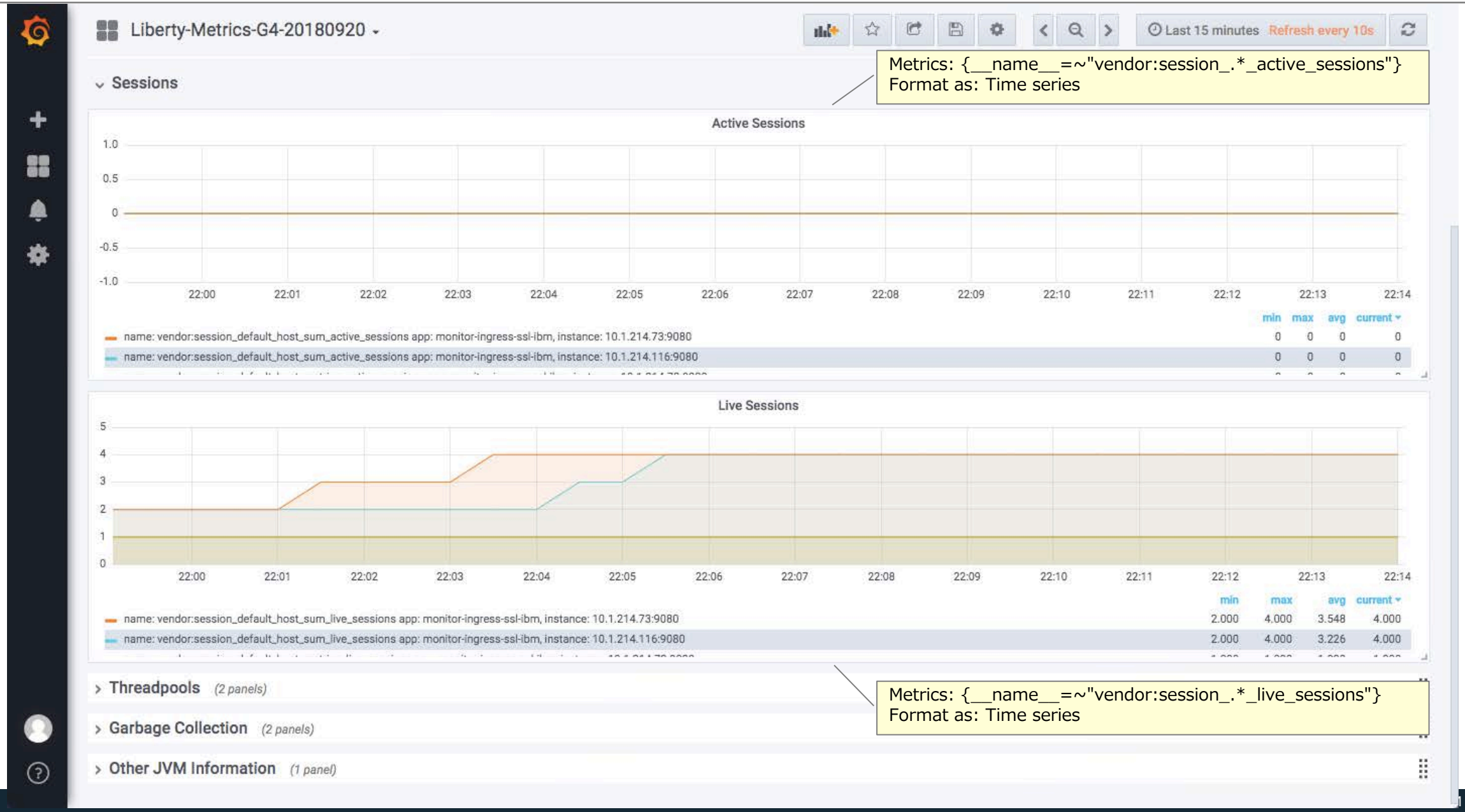
Metrics: {__name__=~"vendor:connectionpool_.*_create_total"}
Format as: Time series

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

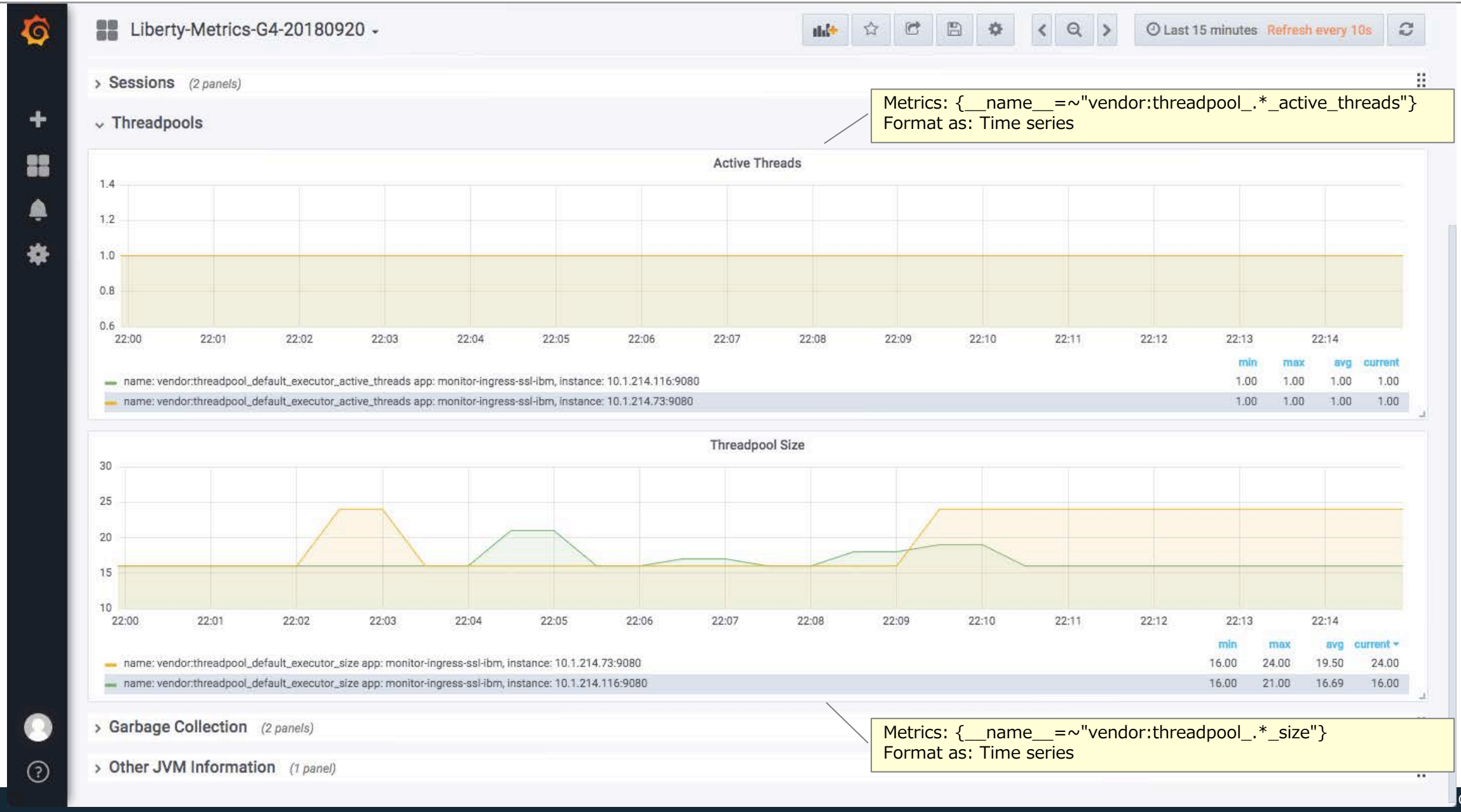
Grafanaのサンプル・ダッシュボードの表示例（Connection Pool）



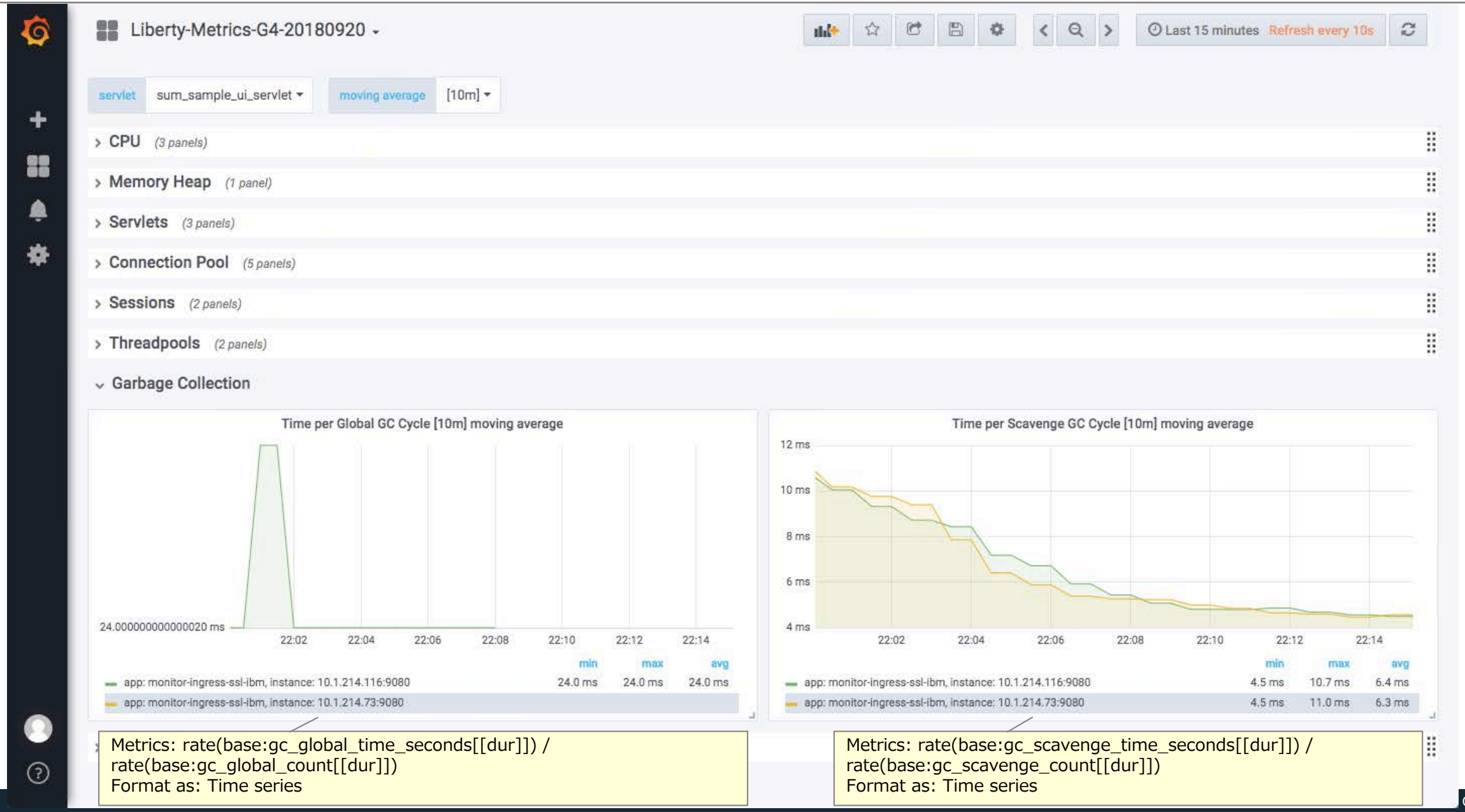
Grafanaのサンプル・ダッシュボードの表示例（Sessions）



Grafanaのサンプル・ダッシュボードの表示例（Threadpools）



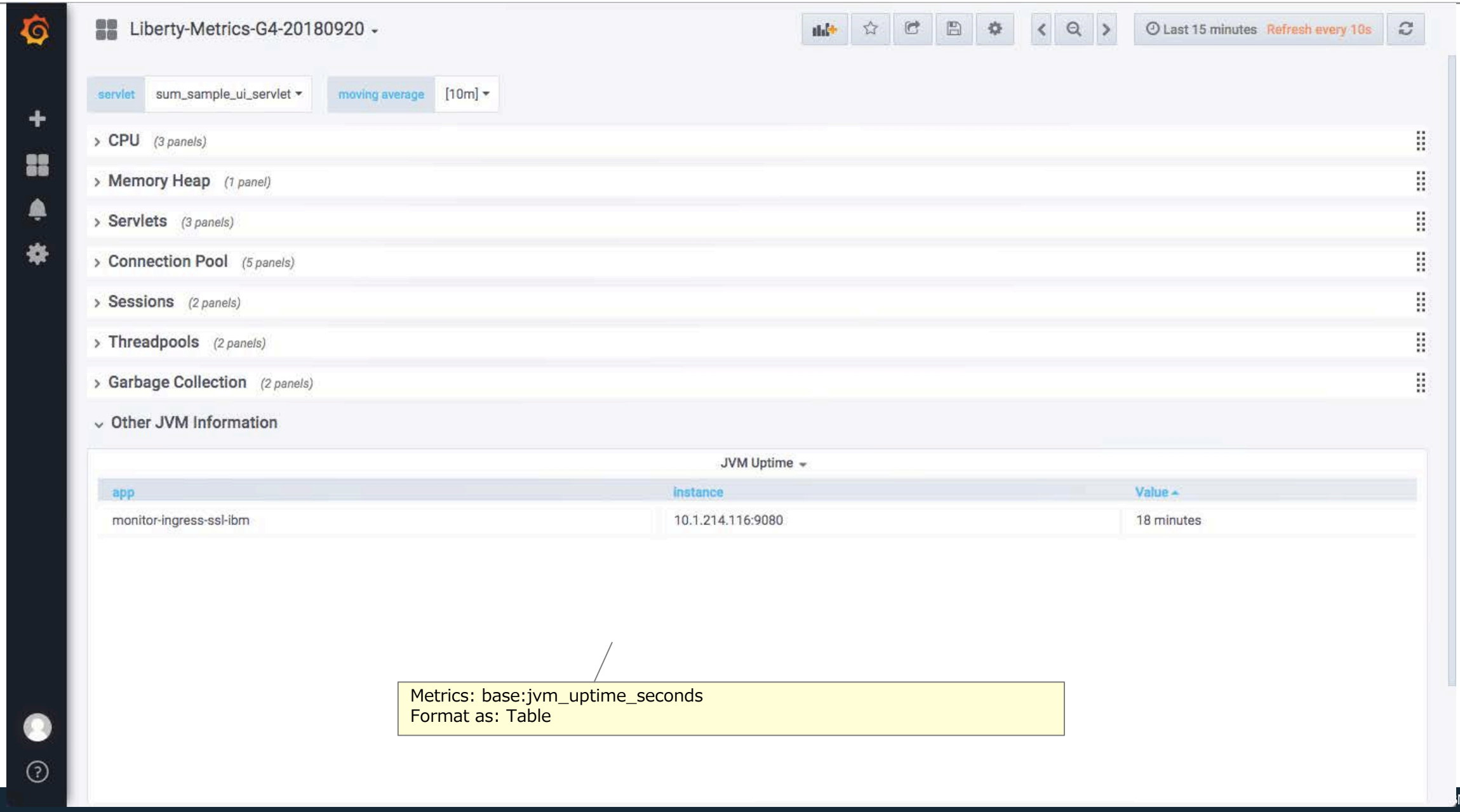
Grafanaのサンプル・ダッシュボードの表示例（Garbage Collection）



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

Grafanaのサンプル・ダッシュボードの表示例（Other JVM Information）



目次(6章)

デプロイ後の運用（Helmリソース（Service））

■ Prometheus用のServiceの構成

– Helmの変数ファイルでモニタリングを有効化（「monitoring.enabled」の値を「true」に設定）してHelm Chartをデプロイすると、PrometheusがWAS Libertyの/metrics RESTインターフェースからメトリクスを取得するためのServiceが自動で構成されます。

– Prometheus用のServiceには以下の設定が構成されます。
- /metricsアクセス用のWAS LibertyのHTTPポート(TCP:9080)の公開
- Prometheus用のannotationの付与(prometheus.io/scrape: "true")

– 実際に作成されるPrometheus用のServiceの構成は、WAS Libertyのアプリケーション接続用のServiceの構成(SSL有効/無効)によって異なります。

- (1)SSL有効の場合
  – WAS Libertyのアプリケーション接続用のServiceに加えて、Prometheus用のServiceが新規作成されます。

- (2)SSL無効の場合
  – WAS Libertyのアプリケーション接続用のServiceを兼用し、Prometheus用の設定が追加で構成されます。

– 次ページ以降でSSL有効/無効の場合の例を記載します。

デプロイ後の運用（Helmリソース（Service））

– (1)SSL有効の場合の例

• (1) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-monitor-ssl.yaml
```

• (2) 抽出した変数ファイルのパラメーターを修正します。

(例)values-monitor-ssl.yaml

```
(略)
service:
  enabled: true
  name: ""
  port: 9443
  targetPort: 9443
  type: NodePort
  annotations: {}
(略)
ssl:
  enabled: true
(略)
monitoring:
  enabled: true
(略)
```

SSLが有効のためServiceのport/targetPortは9443と設定します。(default)

SSLを有効にします。(default)

「monitoring.entabled」の値を「false」から「true」に変更します。

WAS Libertyのアプリケーション用のServiceの設定

モニタリングの設定

• (3) 編集した変数ファイルを使用して、Helm Chartをデプロイします。

```
$ helm upgrade --install monitor-ssl -f ./values-monitor-ssl.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

デプロイ後の運用（Helmリソース（Service））

- (1)SSL有効の場合の例
  - (4) Serviceが作成されていることを確認します。
    - WAS Libertyのアプリケーション用のHTTPSのService(TCP:9443)と、Prometheus用のHTTPのService(TCP:9080)が作成されます。
    - Prometheus用のService名は、「<WAS Libertyのアプリケーション用のService名>-http-clusterip」となります。

```
$ kubectl get service
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
monitor-ssl-ibm-webspher                  NodePort    10.0.85.121     <none>        9443:30776/TCP   5m40s
monitor-ssl-ibm-webspher-http-clusterip   ClusterIP   10.0.186.247    <none>        9080/TCP         5m40s
```

  - (5)Prometheus用のServiceの構成を確認します。
    - /metricsアクセス用のWAS LibertyのHTTPポート(TCP:9080)の公開
    - Prometheus用のannotationの付与(prometheus.io/scrape: "true")

```
$ kubectl get service monitor-ssl-ibm-webspher-http-clusterip -o yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
(略)
spec:
(略)
  ports:
  - name: http-service-liberty-clusterip
    port: 9080
    protocol: TCP
    targetPort: 9080
(略)
  type: ClusterIP
```

Prometheus用のannotationが付与されます。

/metricsアクセス用のWAS LibertyのHTTPポート(TCP:9080)を公開しています。

デプロイ後の運用（Helmリソース（Service））

– (2)SSL無効の場合の例

- (1) Helm Chartから変数ファイルを抽出します。

```
$ helm inspect values ibm-websphere-liberty-1.9.0.tgz > values-monitor-no-ssl.yaml
```

- (2) 抽出した変数ファイルのパラメーターを修正します。

(例)values-monitor-no-ssl.yaml

```
(略)
service:
  enabled: true
  name: ""
  port: 9080
  targetPort: 9080
  type: NodePort
  annotations: {}
(略)
ssl:
  enabled: false
(略)
monitoring:
  enabled: true
(略)
```

SSLを無効にしたためServiceのport/targetPortを9080と設定します。(9443から9080に変更)

SSLを無効にします。(「true」から「false」に変更)

「monitoring.entabled」の値を「false」から「true」に変更します。

WAS Libertyのアプリケーション用のServiceの設定

モニタリングの設定

- (3) 編集した変数ファイルを使用して、Helm Chartをデプロイします。

```
$ helm upgrade --install monitor-no-ssl -f ./values-monitor-no-ssl.yaml ./ibm-websphere-liberty-1.9.0.tgz --tls
```

デプロイ後の運用（Helmリソース（Service））

– (2)SSL無効の場合の例

- (4) Serviceが作成されていることを確認します。
  – WAS Libertyのアプリケーション用のHTTPのService(TCP:9080)が作成され、Prometheus用のHTTPのServiceはこのServiceを兼用します。

```
$ kubectl get service
NAME                        TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
monitor-no-ssl-ibm-websp    NodePort   10.0.61.79    <none>        9080:32624/TCP   13s
$
```

- (5)Serviceの構成を確認します。
  – /metricsアクセス用のWAS LibertyのHTTPポート(TCP:9080)の公開
  – Prometheus用のannotationの付与(prometheus.io/scrape: "true")

```
$ kubectl get service monitor-no-ssl-ibm-websp -o yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
(略)
spec:
(略)
  ports:
  - name: http
    nodePort: 32624
    port: 9080
    protocol: TCP
    targetPort: 9080
(略)
  type: NodePort
```

WAS Libertyアプリケーション用のServiceのannotationに、Prometheus用のannotationも付与されます。

/metricsアクセス用のポートは、WAS Libertyアプリケーション接続用のポート(TCP:9080)を兼用します。

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:
(略)
 extraEnvs:
  - name: MP_METRICS_TAGS
    value: "type=web,target=cust"
(略)
```

環境変数MP_METRICS_TAGSを使用したメトリクスのタグ付け

– 確認(curlコマンドで確認)

(例)master node上でcurlコマンドで/metrics/vendorのメトリクスを取得した場合（Prometheus テキスト・フォーマット）

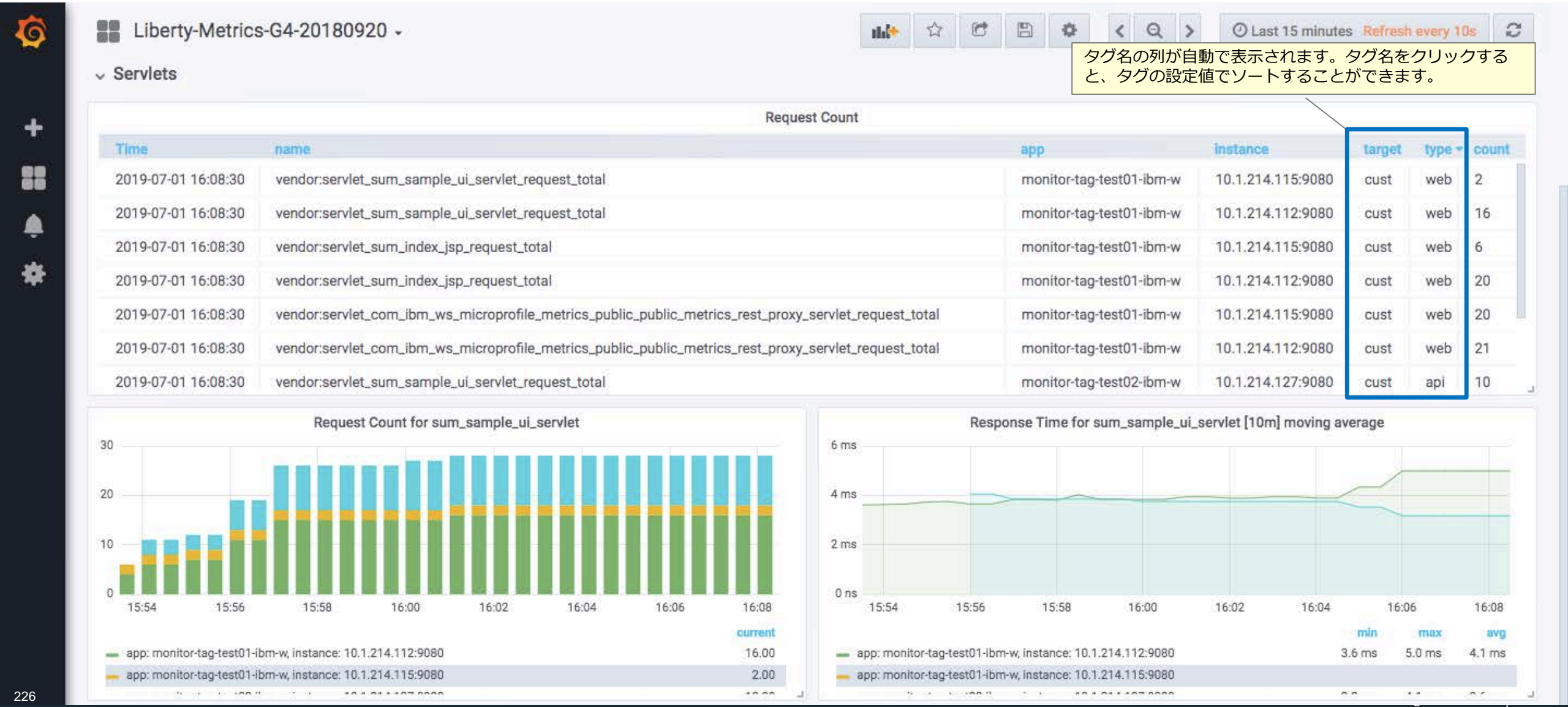
```
# curl -s http://<podのIP addressまたはserviceのIP address>:9080/metrics/vendor
# TYPE vendor:servlet_sum_sample_ui_servlet_request_total counter
# HELP vendor:servlet_sum_sample_ui_servlet_request_total The number of visits to this servlet since the start of the server.
vendor:servlet_sum_sample_ui_servlet_request_total{type="web",target="cust"} 2
# TYPE vendor:session_default_host_metrics_invalidated_total counter
# HELP vendor:session_default_host_metrics_invalidated_total The number of sessions that have logged out since this metric was
enabled.
vendor:session_default_host_metrics_invalidated_total{type="web",target="cust"} 0
# TYPE vendor:session_default_host_metrics_live_sessions gauge
# HELP vendor:session_default_host_metrics_live_sessions The number of users that are currently logged in.
vendor:session_default_host_metrics_live_sessions{type="web",target="cust"} 1
# TYPE vendor:servlet_sum_index_jsp_request_total counter
# HELP vendor:servlet_sum_index_jsp_request_total The number of visits to this servlet since the start of the server.
vendor:servlet_sum_index_jsp_request_total{type="web",target="cust"} 6
# TYPE vendor:session_default_host_sum_invalidatedby_timeout_total counter
# HELP vendor:session_default_host_sum_invalidatedby_timeout_total The number of sessions that have logged out by timeout since
this metric was enabled.
vendor:session_default_host_sum_invalidatedby_timeout_total{type="web",target="cust"} 0
# TYPE vendor:session_default_host_sum_invalidated_total counter
# HELP vendor:session_default_host_sum_invalidated_total The number of sessions that have logged out since this metric was enabled.
vendor:session_default_host_sum_invalidated_total{type="web",target="cust"} 0
(略)
```

タグが付与された状態でメトリクスが取得されます。

WAS 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

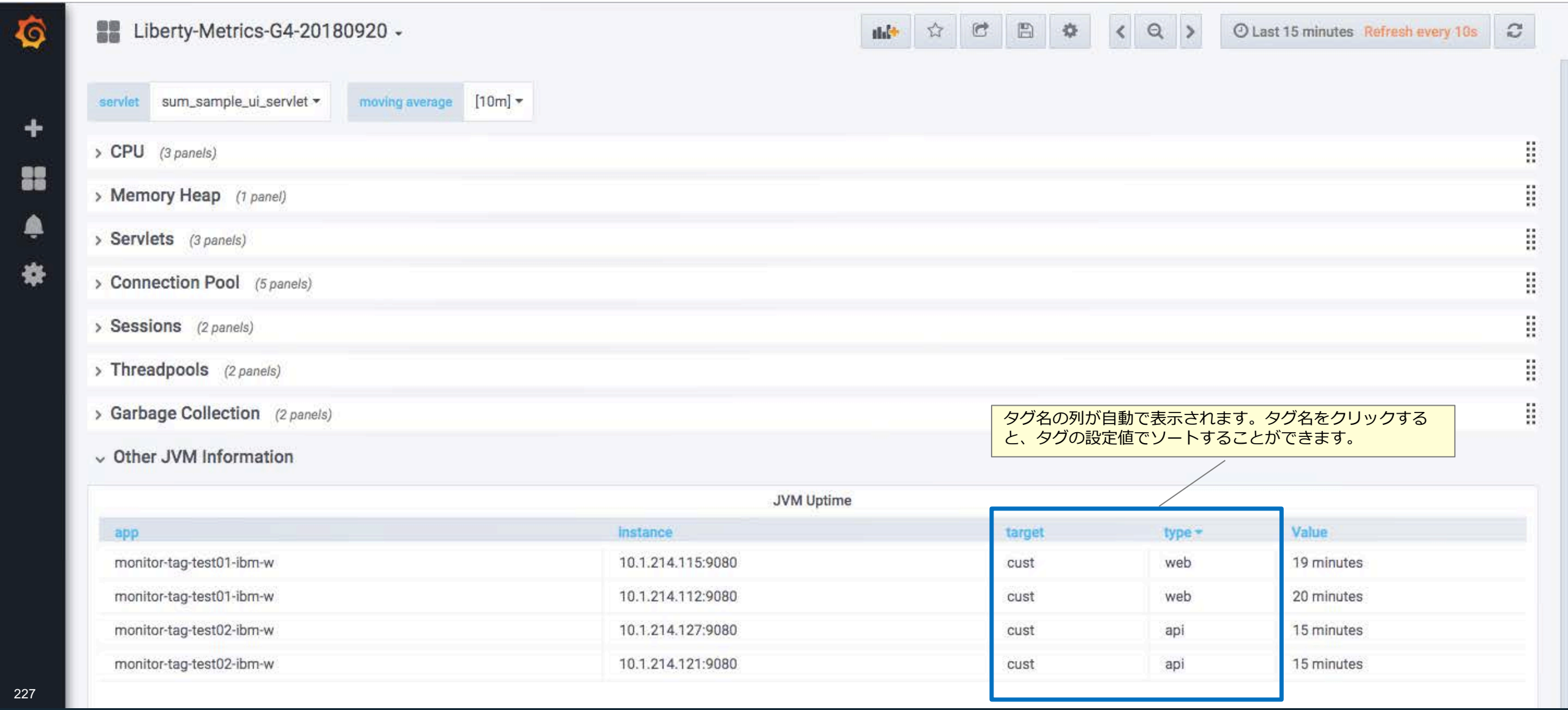
環境変数MP_METRICS_TAGSを使用したメトリクスのタグ付け

– 確認(Grafanaのサンプル・ダッシュボード(Servlets))



環境変数MP_METRICS_TAGSを使用したメトリクスのタグ付け

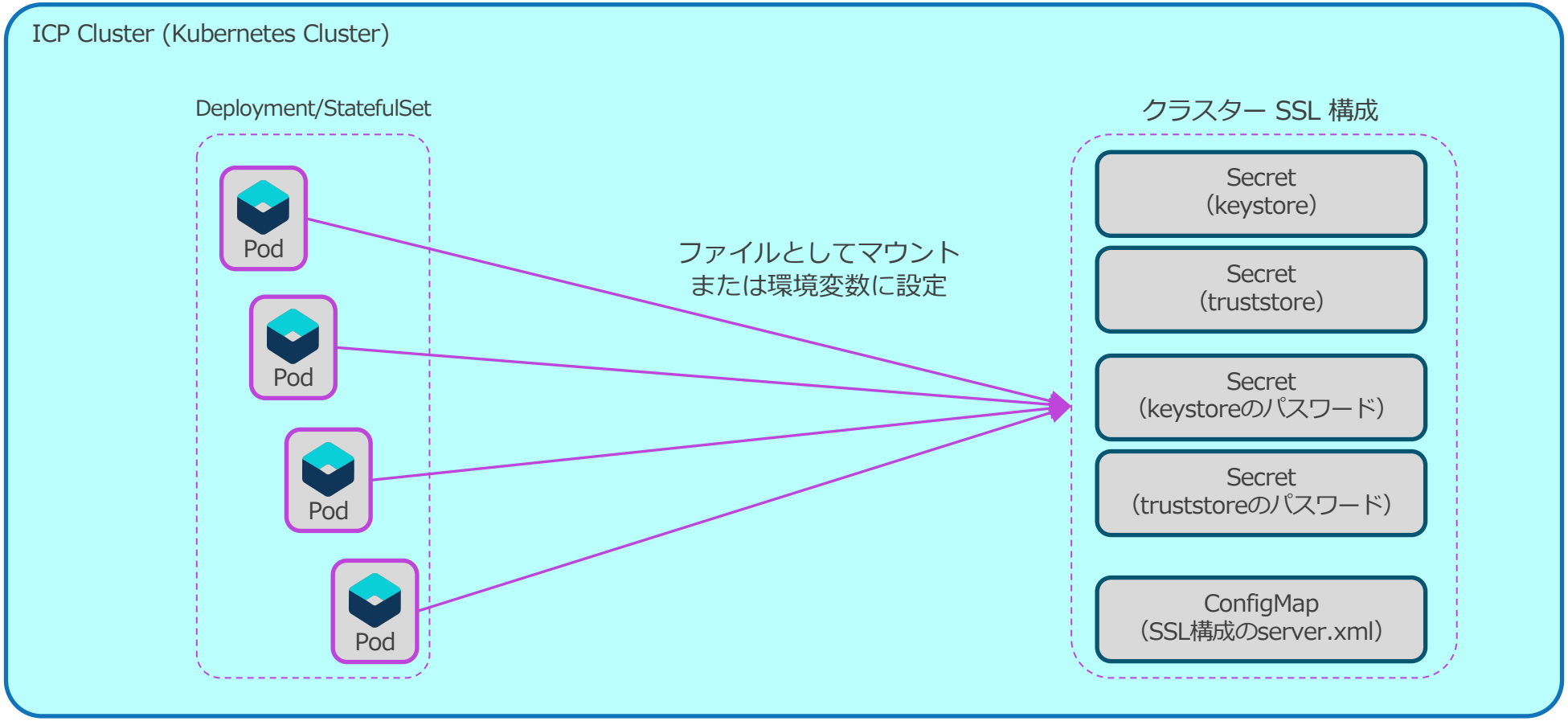
– 確認(Grafanaのサンプル・ダッシュボード(Other JVM Information))



7. クラスター SSL構成の利用

クラスターSSL構成とは

- クラスターSSL構成を利用すると、ICPクラスター内で稼動するWAS Libertyが、共通のSSL構成情報を使用できるようになります。



クラスターSSL構成の構成要素

■ 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	WAS Liberty が SSL 通信で使用する鍵が格納されているファイルです。
鍵ストアの パスワード	Secret	mb-keystore-password	password	鍵ストアのパスワードです。 WAS Liberty が提供する securityUtility encode コマンドを使用して xor 方式等でエンコードしたものを格納するようにします。
トラスト・ストア	Secret	mb-truststore	trust.jks	信頼できる署名者証明書(証明書の署名検査に使用される証明書)が格納されているファイルです。
トラスト・ストアの パスワード	Secret	mb-truststore-password	password	トラスト・ストアのパスワードです。 WAS Liberty が提供する securityUtility encode コマンドを使用して xor 方式等でエンコードしたものを格納するようにします。
WAS Liberty の 構成ファイル	ConfigMap	liberty-config	keystore.xml	鍵ストアとトラスト・ストアの情報が記述されている、WAS Liberty の構成ファイルです。 バージョン1.5.0以降のチャートでは使われません。

– 参考

- WAS 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
- WAS Liberty Knowledge Center / SSL 構成の属性
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_ssl.html

クラスターSSL構成のWAS Libertyのインストールと構成

■ クラスターSSL構成の作成

– クラスターSSL構成は、Helm Chartのパラメーター「ssl.createClusterSSLConfiguration」に「true」を指定することで、Helm Chartのインストール時に作成できます。

- liberty-secret-generator-deployのJobによってibmcom/mb-toolsのコンテナが実行され、SecretとConfigMapが作成されます。
- ibmcom/mb-toolsのコンテナはrootユーザーで実行されるためNamespaceにibm-anyuid-pspのPodSecurityPolicyを設定する必要があります。
- オフライン環境では、ibmcom/mb-toolsのコンテナ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。
- 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なので、注意が必要です。

```
$ kubectl get job
NAME                              COMPLETIONS   DURATION   AGE
liberty-secret-generator-deploy   1/1           9s         96s                    <- SecretとConfigMapを生成するJob
$ kubectl get secret
NAME                                  TYPE                                  DATA   AGE
default-token-zvkbw                   kubernetes.io/service-account-token   3      7h14m
mb-keystore                           Opaque                                1      106s <- 鍵ストアを格納するSecret
mb-keystore-password                  Opaque                                1      106s <- 鍵ストアのパスワードを格納するSecret
mb-truststore                         Opaque                                1      106s <- トラストストアを格納するSecret
mb-truststore-password                Opaque                                1      106s <- トラストストアのパスワードを格納するSecret
myliberty-ibm-websphere-tls           Opaque                                2      113s
myliberty-ibm-websphere-token-jbljg   kubernetes.io/service-account-token   3      113s
sa-liberty-guide                      kubernetes.io/dockerconfigjson        1      177m
$ kubectl get cm
NAME                      DATA   AGE
liberty-config            1      114s                                        <- keystore.xmlを格納するConfigMap
myliberty-ibm-websphere   3      2m1s
$
```

クラスターSSL構成のWAS Libertyのインストールと構成

■ クラスターSSL構成の考慮点

– クラスターSSL構成を再作成する場合は、前述のSecret、ConfigMap、および、Jobを削除してから行います。

- WAS Liberty Knowledge Center / IBM Cloud Private 用の Liberty での SSL の使用可能化
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_ssl_helm.html

– Jobが生成した鍵の使用が適さない場合は、鍵ストアとトラスト・ストアを準備し、下記のコマンドでクラスターSSL構成用のSecretを置き換えます。

- WAS Liberty Knowledge Center / サード・パーティー証明書を使用した SSL の IBM Cloud Private での使用可能化
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_icp_auto_ssl3.html
- WAS Liberty Knowledge Center / securityUtility コマンド
- https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_command_securityutil.html

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

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

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

このパスワードは環境変数を経由してWAS Libertyの構成ファイルに設定されるため、securityUtility encode 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

クラスターSSL構成のWAS Libertyのインストールと構成

■ クラスターSSL構成のWAS Libertyの構成

– Helm Chartのパラメーター「ssl.useClusterSSLConfiguration」に「true」を指定してインストールすると、WAS LibertyのICP(k8s)のマニフェストに以下の変更が加わります。

- WAS LibertyのPodがファイルとして鍵ストアとトラスト・ストアをマウントします。
  – 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

- WAS LibertyのPodの環境変数に鍵ストアとトラスト・ストアのパスワードが設定されます。
  – 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。
  – 「<Helmリリース名>-ibm-websphere」のConfigMapの各エントリは「/etc/wlp/configmap」ディレクトリにファイルとしてマウントされます。
  – 「include-configmap.xml」から「cluster-ssl.xml」がincludeされます。(11章参照)

```
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>
```

8. オートスケーリングとリソース調整

オートスケーリング概要

- IBM提供のWAS 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(WAS Libertyのログを永続化しない場合)(5章参照)

  – GitHub / IBM/charts/stable/ibm-websphere-liberty / Installing the Chart
  – https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#installing-the-chart
  – GitHub / IBM/charts/stable/ibm-websphere-liberty / Configuration
  – https://github.com/IBM/charts/tree/master/stable/ibm-websphere-liberty#configuration

IBM提供のWAS 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

(*) ibm-websphere-liberty v1.9.0の場合

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。 (オートスケーリングを有効にした場合、replicaCountに指定したレプリカ数は無視されるようになります。)
	minReplicas	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

(*) ibm-websphere-liberty v1.9.0の場合

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

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

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

- HPAの状態は「kubectl 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。

```
# kubectl describe hpa wlp-re01-liberty-hpa
Name:                                                  wlp-re01-liberty-hpa
Namespace:                                             yama-ns
Labels:                                                app=wlp-re01-liberty
                                                       chart=ibm-websphere-liberty-1.9.0
                                                       heritage=Tiller
                                                       release=wlp-re01
Annotations:                                           <none>
CreationTimestamp:                                     Tue, 14 May 2019 15:24:47 +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  42m (x2 over 70m)      horizontal-pod-autoscaler  New size: 2; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  37m                    horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal  SuccessfulRescale  30m                    horizontal-pod-autoscaler  New size: 3; reason: All metrics below target
  Normal  SuccessfulRescale  <invalid> (x2 over 45m)  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。

9. 静的コンテンツの扱い

目次(9章)

- 静的コンテンツを扱う方法
- 静的コンテンツの設定方法
  - WAS LibertyのFile Serving機能の拡張文書ルートの指定方法
  - 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。

– WAS LibertyのFile Serving機能(デフォルトで有効)を使用する方法
- (a) warモジュールに含める方法
  – アプリケーションのwarモジュール内に静的コンテンツを含めることで、簡単に対応できます。
  – 静的コンテンツを修正するには、warモジュールと Dockerイメージを再ビルドし、Pod が使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (b) コンテナー・イメージに格納し、格納先を拡張文書ルートとして指定する方法
  – Dockerイメージに静的コンテンツのディレクトリーを作り、その中に静的コンテンツを含め、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツを修正するには、Dockerイメージを再ビルドし、Podが使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (c) 永続領域に置き、そのマウント先を拡張文書ルートとして指定する方法
  – 永続領域に静的コンテンツを配置してPodにマウントし、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツの修正は、永続領域内のコンテンツの更新で行えます。また、静的コンテンツが多い/大きい場合にも適した方法となります。

– WAS Libertyの代わりに、IBM HTTP ServerなどのWebサーバーを利用する方法
- 同様に、次の２つの方法が想定されますが、メリット・デメリットはWAS Libertyの場合と同様です。
- (b) コンテナー・イメージに含める方法
- (c) 永続領域に置く方法

目次(9章)

- 静的コンテンツを扱う方法
- 静的コンテンツの設定方法
  - WAS LibertyのFile Serving機能の拡張文書ルートの指定方法
  - 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。

– WAS LibertyのFile Serving機能(デフォルトで有効)を使用する方法
- (a) warモジュールに含める方法
  – アプリケーションのwarモジュール内に静的コンテンツを含めることで、簡単に対応できます。
  – 静的コンテンツを修正するには、warモジュールと Dockerイメージを再ビルドし、Pod が使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (b) コンテナー・イメージに格納し、格納先を拡張文書ルートとして指定する方法
  – Dockerイメージに静的コンテンツのディレクトリーを作り、その中に静的コンテンツを含め、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツを修正するには、Dockerイメージを再ビルドし、Podが使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (c) 永続領域に置き、そのマウント先を拡張文書ルートとして指定する方法
  – 永続領域に静的コンテンツを配置してPodにマウントし、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツの修正は、永続領域内のコンテンツの更新で行えます。また、静的コンテンツが多い/大きい場合にも適した方法となります。

– WAS Libertyの代わりに、IBM HTTP ServerなどのWebサーバーを利用する方法
- 同様に、次の２つの方法が想定されますが、メリット・デメリットはWAS Libertyの場合と同様です。
- (b) コンテナー・イメージに含める方法
- (c) 永続領域に置く方法

File Serving機能の拡張文書ルートの機能を使用する方法を説明します。

WAS LibertyのFile Serving機能の拡張文書ルートの指定方法

■ WAS 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WASが従来から提供している機能です。

- WAS LibertyのKnowledge Center等での記載が確認できておりませんが、指定が有効に機能していることは確認済みです。
  – WAS traditional Knowledge Center / ファイル・サービス
  – https://www.ibm.com/support/knowledgecenter/ja/SSAW57_9.0.0/com.ibm.websphere.nd.multiplatform.doc/ae/cweb_flserv.html

– 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>
```

- WAS Liberty のserver.xml内の<web-ext>で指定

(server.xmlまたはserver.xmlフラグメントへの反映方法は、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

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

目次(9章)

- 静的コンテンツを扱う方法
- 静的コンテンツの設定方法
  - WAS LibertyのFile Serving機能の拡張文書ルートの指定方法
  - 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。

– WAS LibertyのFile Serving機能(デフォルトで有効)を使用する方法
- (a) warモジュールに含める方法
  – アプリケーションのwarモジュール内に静的コンテンツを含めることで、簡単に対応できます。
  – 静的コンテンツを修正するには、warモジュールと Dockerイメージを再ビルドし、Pod が使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (b) コンテナー・イメージに格納し、格納先を拡張文書ルートとして指定する方法
  – Dockerイメージに静的コンテンツのディレクトリーを作り、その中に静的コンテンツを含め、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツを修正するには、Dockerイメージを再ビルドし、Podが使用するイメージを置き換える必要があります。また、静的コンテンツが多い/大きい場合は適しません。
- (c) 永続領域に置き、そのマウント先を拡張文書ルートとして指定する方法
  – 永続領域に静的コンテンツを配置してPodにマウントし、File Serving機能の拡張文書ルートとして指定することで対応します。
  – 静的コンテンツの修正は、永続領域内のコンテンツの更新で行えます。また、静的コンテンツが多い/大きい場合にも適した方法となります。

– WAS Libertyの代わりに、IBM HTTP ServerなどのWebサーバーを利用する方法
- 同様に、次の２つの方法が想定されますが、メリット・デメリットはWAS Libertyの場合と同様です。
- (b) コンテナー・イメージに含める方法
- (c) 永続領域に置く方法

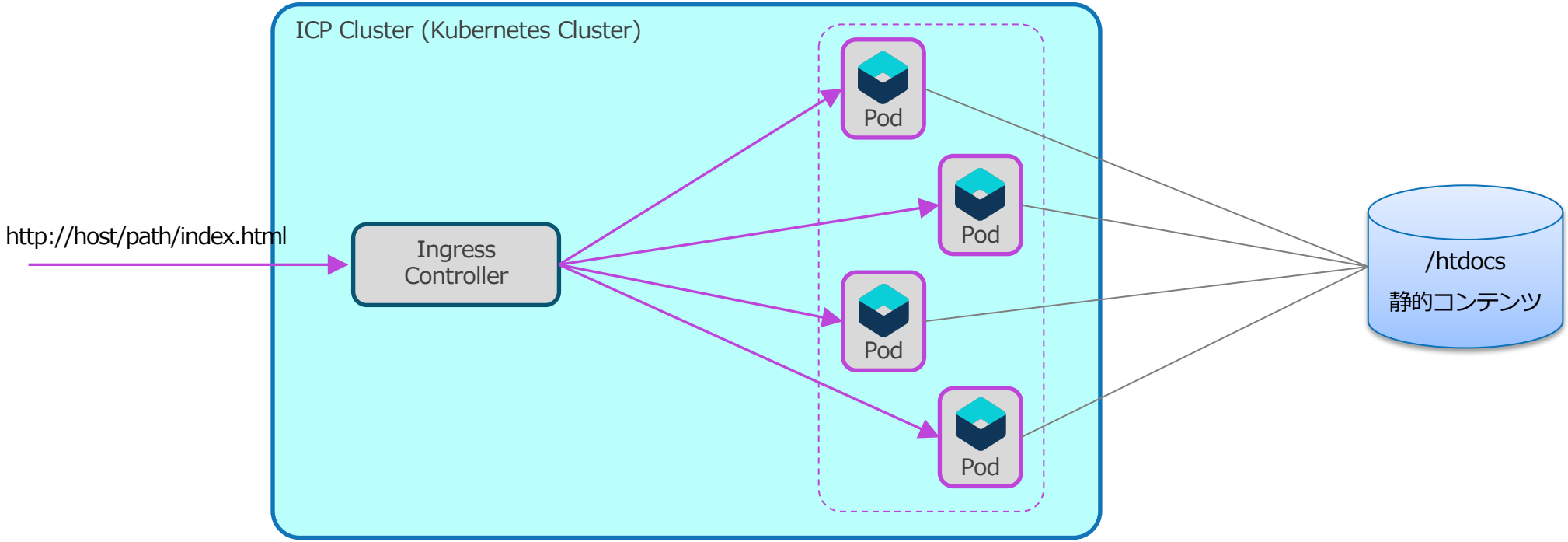
ここでは例として(c)の永続領域を使う方法を説明します。

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

■ 設定概要

– 静的コンテンツ用の永続領域(Persistent Volume)を定義し、WAS LibertyのFile Serving機能の拡張文書ルートとして前述の「WAS LibertyのFile Serving機能の拡張文書ルートの指定方法」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WAS 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。

- 例のようなyamlファイルを作成し、「kubectl 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。
  – ICP Management Consoleで定義する場合は、[Menu > プラットフォーム > ストレージ]と選択して[ストレージ]画面に進み、「PersistentVolume」および「PersistentVolumeClaim」のタブから作成します。
- 例では、label (key名: "type")をマッチング条件の１つとして指定しています。
- Podは静的コンテンツを更新する必要がないので、「accessMode」は「ReadOnlyMany」を指定しています。

(例)PVのマニフェストファイルの例(yaml)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: htdocs-pv81
spec:
  labels:
    type: htdocs-pv81
spec:
  accessModes:
  - ReadOnlyMany
  capacity:
    storage: 2Gi
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /nfs/htdocs-pv03
    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-pvc81
spec:
  storageClassName: ""
  selector:
    matchExpressions:
    - key: type
      operator: In
      values:
      - htdocs-pv81
  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。

- 「extraVolumes」に、先程指定したPVCを指定します。
- 「extraVolumeMounts」に、Podにマウントする「extraVolumes」と「マウントポイント」を指定します。

(例)Helm Chartの変数ファイルの設定例(yaml)

```
(略)
image:
  repository: mycluster.icp:8500/default/wlp-rseater
  tag: "19.0.0.4"
  pullPolicy: IfNotPresent
  license: "accept"

 (略)
  extraVolumeMounts:
   - name: htdocs
     mountPath: /htdocs
     readOnly: true

  extraVolumes:
    - name: htdocs
      persistentVolumeClaim:
      claimName: htdocs-pvc81
(略)
  serverOverridesConfigMapName: "staticdoc"
(略)
```

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。

ConfigMapとしてserver.xmlのフラグメントを追加する場合に記載します。
ConfigMapを利用したserver.xmlフラグメントの追加方法については、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」を参照してください。

10. IBM HTTP Server (IHS)の利用

目次(10章)

目次(10章)

Docker環境でIBM HTTP Server (IHS)を使用する場合の参考情報

- Docker環境でIHSを利用するための情報を以下に示します。
  - IHSのDockerイメージに関する情報
    - 以下のURLで情報が公開されており、WebサーバーPluginを含むIHSのDocker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Docker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、ICP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一時的な利用を想定して、もし使用する場合の構成例や設定例を参考としてご紹介しています。

目次(10章)

IHSのDocker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。
  - 主に以下の操作を行うように定義します。
    - OSの更新(apt-get update)と、ungipのインストール(apt-get install –y unzip)
    - defaultユーザー(ID:1001)の追加
      - セキュリティの観点およびWAS LibertyのDocker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
      - WAS LibertyのDockerイメージ内に定義されているディレクトリに合わせてディレクトリーを作成します。
      - /htdocs　　コンテンツ用
      - /logs　　ログ出力用
      - /config　　構成用
    - IHS起動スクリプトの追加
      - 前述の「Docker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。
      - (具体的な設定例は後述の「ICP環境のWAS LibertyとIHSの設定例」の「(例)IHSをWAS Libertyの「前段」に配置」参照)
    - IHS起動スクリプトの指定
      - コンテナ内実行時に起動するスクリプトとしてIHS起動スクリプトを指定します。
  - 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(/config)にコピー
    - (静的コンテンツをIHSのDocker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(/htdocs)にコピー

IHSのDockerイメージの作成方法

- IHSのイメージを作成するDockerfileの例を示します。

(例)IHSのDockerイメージ作成用のDockerfile例

```
FROM ubuntu:16.04

# Update and Install unzip, Add user 1001
RUN apt-get update && apt-get install -y unzip wget
&& useradd -u 1001 -r -g 0 -s /usr/sbin/nologin default

# Install IHS (unzip IHS zip-file)
ARG ihs_zip_file=9.0.0-WS-IHS-ARCHIVE-linux-x86_64-FP010.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

(略)
```

OSの更新と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のインストール

postinstall.shの実行

IHSのDockerイメージの作成方法

■ IHSのイメージを作成するDockerfileの例を示します。

(例)IHSのDockerイメージ作成用のDockerfile例 (続き)

```
(略)
# 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

# 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 ¥
&& chmod 744 /work/ihsstart.sh

# Switch to user 1001
USER 1001

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

ディレクトリーの準備(コンテンツ用、ログ出力用、構成用のディレクトリの作成)

WAS Libertyとの連携で使用するディレクトリの作成(青字部分)
(後述の「ICP環境のWAS LibertyとIHSの構成例」の「IHSをWAS Libertyの「前段」に配置」の場合)

WAS Libertyとの連携に関するIHSの設定ファイル(httpd.conf)の修正
(Plugin構成ファイルの場の所変更と、WAS LibertyとIHSが常にhttpで通信するように、SetEnvを追加)
(後述の「ICP環境のWAS LibertyとIHSの構成例」の「IHSをWAS Libertyの「前段」に配置」の場合)

IHS起動スクリプト(ihsstart.sh)の追加と実行権限の付与

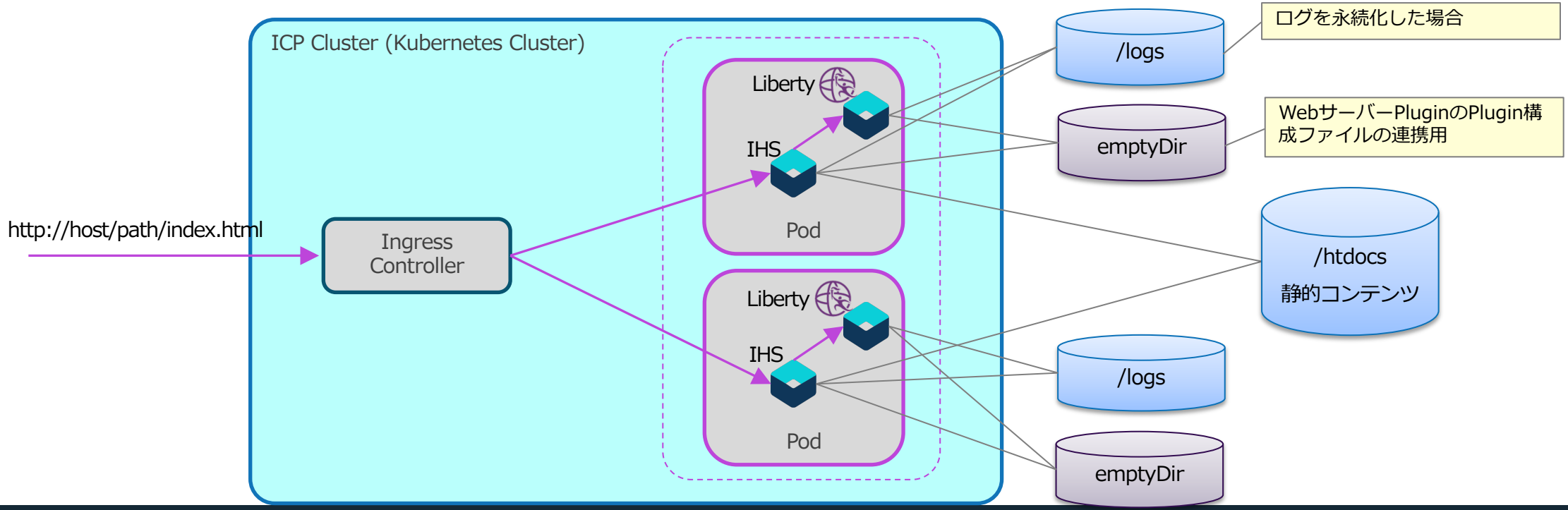
実行ユーザーをdefault(ID:1001)に変更

コンテナ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

目次(10章)

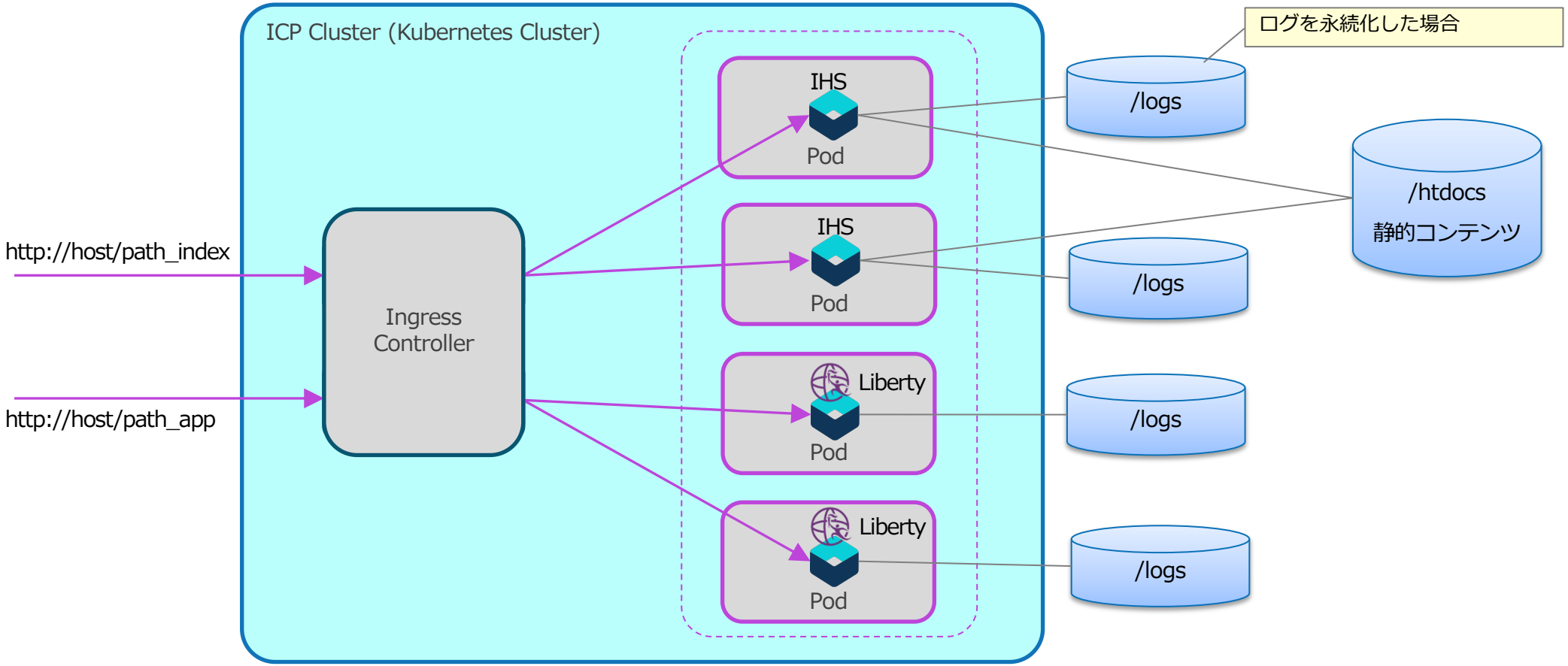
ICP環境のWAS LibertyとIHSの構成例 (IHSをWAS Libertyの「前段」に配置する構成)

- 1つのPod内でIHSのコンテナーとWAS 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WAS Liberty間は、Pod内に閉じた、１対１構成となります。
  - IHSが静的コンテンツを処理し、WAS Libertyが動的コンテンツを処理する、従来構造のWebシステムとなります。
    - 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、WAS 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。



ICP環境のWAS LibertyとIHSの構成例 (IHSをWAS Libertyの「横並び」に配置する構成)

- IHSのPodとWAS 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、WAS Libertyが動的コンテンツを処理します。
  - IHSとWAS 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。



目次(10章)

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

■ 設定概要

– 例として前述の「IHSをWAS Libertyの「前段」に配置する構成」の設定方法を記載します。

■ 設定の流れ

– 設定の流れは以下のようになります。

– (1) IHS側の設定(IHSのDockerイメージの作成)
– (2) WAS Liberty側の設定(server.xmlの構成)
– (3) Helm Chartの変数ファイルの設定

■ 設定内容

– 次ページ以降で、上述の各項目毎の設定内容例を説明します。

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

– (1) IHS側の設定(IHSのDockerイメージの作成)
  • WAS Libertyが作成するPlugin構成ファイルが生成された後にIHSが起動するようにIHSのDockerイメージに含めるIHS起動スクリプト(ihsstart.sh)をカスタマイズする必要があります。（前述の「IHSのDockerイメージの作成方法」参照）

(例)IHS起動スクリプト(ihsstart.sh)の設定例

```
#! /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」が設定されていれば、Plugin構成ファイルが作成されるまで待ちます。(この環境変数はHelm Chartの変数ファイルで設定します。)

IHS 起動、trap 設定、sleep（変更前と同様）

ログをコンソールへtail出力

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

– (2) WAS Liberty側の設定(server.xmlの構成)

- server.xmlまたはserver.xmlフラグメントの「pluginConfiguration」エレメントの設定をIHS側の構成に合わせて設定します。

設定項目	備考
pluginInstallRoot	• WebサーバーPluginのインストール先を指定 • IHSをアーカイブ・インストールした場合のWebサーバーPluginのインストール先「/opt/ibm/IHS/plugin」を設定
webserverName	• IHSをアーカイブ・インストールした場合にデフォルトで作成されるWebサーバー名「webserver1」を設定
logFileName	• IHSのログ出力用ディレクトリー(/logs)内に作成するように設定(例: /logs/http_plugin.log)

(server.xmlまたはserver.xmlフラグメントへの反映方法は、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

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

「pluginConfiguration」エレメント内に上述の定義を設定します。

- その他の設定項目に関しては、以下URLもご参照ください。
  – WAS 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

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

– (3) Helm Chartの変数ファイルの設定

- WAS LibertyのコンテナとIHSのコンテナを1つのPodとして起動させます。IHSのコンテナに関する設定は「pod.extraContainers」に設定することで、WAS LibertyのHelm ChartでIHSも同一Pod内の別コンテナとしてインストールできます。

- 前述の「IHSのDockerイメージの作成方法」で作成したIHSのDockerイメージは、WAS Libertyと同様のディレクトリー構成をしています。以下の永続領域の指定は、「pod.extraContainers.volumeMounts」で指定します。
  - /logsに出力されるログを永続化します。
  - 静的コンテンツ用の永続領域を/htdocsにマウントします。
  - WAS LibertyとIHSのコンテナでPlugin構成ファイルを共有するための領域(/output/logs/state)をマウントします。

- 以下の指定はIHSでは不要です。
  - コンソールへのログ出力を設定する環境変数
  - トランザクション・ログの永続化
  - クラスターSSL構成

- 前述の「IHSのDockerイメージの作成方法」で作成したIHSのDockerイメージではhttp通信のみが可能です。

- デフォルトの「image.livenessProbe」と「image.readinessProbe」はWAS Libertyのコンテナに対する設定となります。
  - デフォルトでは「service.port」で設定したport(今回は8080を設定)からDocumentRootの"/"に確認に行きます。
- IHSに対してProbeを設定する場合は「pod.extraContainers.livenessProbe」、「pod.extraContainers.readinessProbe」に設定してください。

- Helm Chartの変数ファイルで指定できるリソース制限は、WAS Liberty用のコンテナにのみ適用されます。

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

– (3) Helm Chartの変数ファイルの設定

(例)Helm Chartの変数ファイルの設定例(yaml)

```yaml
image:
  repository: mycluster.icp:8500/default/wlp-app
  tag: 19.0.0.4-javaee8
  pullPolicy: IfNotPresent
  license: "accept"
  serverOverridesConfigMapName: "plugincon"
  extraVolumeMounts:
    name: state03
    mountPath: /output/logs/state
  livenessProbe:
    httpGet:
      :

pod:
  extraContainers:
   - name: ihs
     image: mycluster.icp:8500/default/ihs01:v423
     env:
      - name: WAIT_PLUGIN_GENERATION
        value: "true"
     volumeMounts:
       - name: htdocs
         mountPath: /htdocs
         readOnly: true
       - name: state
         mountPath: /output/logs/state
       - name: liberty-pvc
         mountPath: /logs
         storageClassName: "wlp-logpv"
         subPath: logs
     livenessProbe:
       httpGet:
         :
```

- 使用するWAS LibertyのDockerイメージの指定
- 「(2) WAS Liberty側の設定(server.xmlの構成)」 のserver.xmlのフラグメントをConfigMapとして追加する場合の指定 (ConfigMapを利用したserver.xmlフラグメントの追加方法については、3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」を参照)
- 「(2) WAS Liberty側の設定(server.xmlの構成)」のserver.xmlフラグメントをConfigMapで指定する場合のConfigMap名の指定
- WAS LibertyのコンテナへのlivenessProbeはreadinessProbeの設定がある場合はここに追記

IHSコンテナの設定

- 使用するWAS LibertyのDockerイメージの指定
- WebサーバーPluginの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でこの環境変数を利用)
- 静的コンテンツ用のPVのマウント
- WAS LibertyとIHSのコンテナで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
- ログ出力用のPVのマウント
- IHSのコンテナへのlivenessProbeはreadinessProbeの設定がある場合はここに追記

ICP環境のWAS LibertyとIHSの設定例 (IHSをWAS Libertyの「前段」に配置する構成)

(例)Helm Chartの変数ファイルの設定例(yaml) (続き)

```
  extraVolumes:
   - name: htdocs
     persistentVolumeClaim:
       claimName: htdocs-pvc03
   - name: state03
     emptyDir:
       medium: "Memory"
service:
  enabled: true
  name: ""
  port: 8080
  targetPort: 8080
  type: NodePort

ssl:
  enabled: false
  useClusterSSLConfiguration: false
  createClusterSSLConfiguration: false

ingress:
  enabled: true
  rewriteTarget: "/"
  path: "/ihs/"
  host: ""
  secretName: ""

persistence:
  name: "liberty-pvc"
  size: "1Gi"
  useDynamicProvisioning: true
  fsGroupGid:
  storageClassName: "wlp-logpv"

logs:
  persistLogs: true
(略)
```

- 追加Volumeの設定（静的コンテンツ用のPVCの設定)
- 追加Volumeの設定（WAS LibertyとIHSのコンテナで Plugin構成ファイル(plugin-cfg.xml)を共有するための領域 (emptyDir))
- Serviceのportは、IHS用の8080に変更
- SSLは使用しない
- Ingressは利用pathを変更する
- ログ出力用のPVCの設定
- ログを永続化させる設定

11. ConfigMap と Secret の利用

目次(11章)

- ConfigMapとSecretの用途
- (補足)WAS Libertyの構成情報(server.xml)と環境変数の優先順位
  - WAS Libertyの構成情報(server.xml)の優先順位
  - 環境変数の優先順位
- ConfigMapを使用したWAS Libertyの設定方法
- ConfigMapとSecretを使用したWAS Libertyの設定例
  - (例1)データソース定義を静的に定義(ConfigMap)
  - (例2)データソース定義を環境変数で定義(ConfigMapとSecret)
- (補足)WAS LibertyのHelm Chartで自動作成されるConfigMapの構成

目次(11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WAS Libertyやアプリケーションに渡すことができます。
- (2章「基礎知識：ConfigMapとSecret」参照)

目次(11章)

WAS Libertyの構成情報(server.xml)の優先順位

- WAS Libertyの構成情報(server.xmlまたはserver.xmlフラグメント)は、複数の箇所に設定できます。
  - (3章の「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

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

高．/config/configDropins/overrides内の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

環境変数の優先順位

- ConfigMapやSecretに定義されたvalueを環境変数を通じてWAS 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WAS Libertyに渡す場合、以下の優先順位に配慮する必要があります。
  - 低い優先順位で指定した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．Dockerイメージ内のserver.envでの設定

中．Helm Chartの変数ファイルでのインストール時の設定

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

低．Dockerイメージでの設定

  - 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。

目次(11章)

ConfigMapを使用したWAS Libertyの設定方法

■ ConfigMapを使用したWAS Libertyの構成情報(server.xml)の設定方法は2つの方法があります。

(3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(0)	Dockerイメージ	IBM提供のWAS LibertyのDockerイメージに含まれるserver.xmlファイル	server.xml	• /config/server.xml
(1)		作成したserver.xmlをコピーしてDockerイメージをビルド	server.xml	• /config/server.xml
(2)		WAS LibertyのDockerイメージが提供する機能(変数ARGとシェル)を利用してserver.xmlフラグメントをコピーしてDockerイメージをビルド	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/mp-monitoring.xml など
(3)	Helm ChartとConfigMap	server.xmlフラグメントをConfigMapとして事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/server-overrides.xml
(4)		(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント)

ConfigMapを使用したWAS Libertyの設定方法

■ ConfigMapを使用した(3)と(4)の設定方法の考慮点は以下のようになります。

– (3) オーバーライドしたい定義を設定したserver.xmlフラグメントを格納したConfigMapを事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定します。
  • ConfigMapの「server-overrides.xml」エントリが「/config/configDropins/overrides/server-overrides.xml」にマウントされます。
  • subPathでマウントしているため、ConfigMapを変更しても動的に反映されません。
    – (Deploymentのspec.template.spec.containers.volumeMounts.subPathで指定されます。)
  • ConfigMapのserver-overrides.xmlの変更を反映させるためには、Podを削除して再作成させる必要があります。
  • オーバーライドできるファイルは1つだけなので、全てをserver-overrides.xmlエントリに含める必要があります。

– (4) (Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを格納したConfigMap (Liberty Fabric)を修正します。
  • ConfigMapの「server.xml」エントリを直接変更します。
  • 変更は動的に反映されます。
  • Helm Chartの変数ファイルを修正し、「helm upgrade」を行っても、ConfigMap (Liberty Fabric)の「server.xml」エントリの変更内容は保持されます。
  • 一時的な変更の場合はこの方法を使用します。
    – (Helm Chartの変数ファイルや、(3)の場合のserver.xmlフラグメントなどのように、ファイルで構成情報を管理できないので、あくまでも一時的な変更の場合のみに使用することをお勧めします。)

■ 次ページの設定例では(3)の方法での設定例を紹介します。

目次(11章)

ConfigMapとSecretを使用したWAS Libertyの設定例

■ ConfigMapとSecretを使用したWAS Libertyの設定例として以下の二つの例を説明します。

– (例1)データソース定義を静的に定義(ConfigMap)
- 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を設定する例を示します。
- データソース定義の接続情報は静的に定義します。
- 設定に使用するConfigMapは以下になります。
  – ConfigMap(データソース定義を定義したserver.xmlフラグメント用)

– (例2)データソース定義を環境変数で定義(ConfigMapとSecret)
- 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を設定する例を示します。
- データソース定義の接続情報部分に「環境変数」を使用します。
- 環境変数の設定値は、環境変数設定用のConfigMapとSecretを別途作成して設定します。
- 設定に使用するConfigMapおよびSecretは以下になります。
  – ConfigMap(データソース定義を定義したserver.xmlフラグメント用)
  – ConfigMap(環境変数設定用)
  – Secret(環境変数設定用)
- ConfigMap(データソース定義を定義したserver.xmlフラグメント用)は共有して、接続情報は「環境変数」として別のConfigMapとSecretから取得することになります。

(例1)データソース定義を静的に定義(ConfigMap)

- 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を静的に設定する例を示します。

– 設定概要
  - 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を設定する例を示します。
  - データソース定義の接続情報は静的に定義します。
  - 設定に使用するConfigMapは以下になります。
    – ConfigMap(データソース定義を定義したserver.xmlフラグメント用)

– 前提作業
  - JDBCドライバーのjarファイルについては、Dockerイメージビルド時にコピー済みとします。

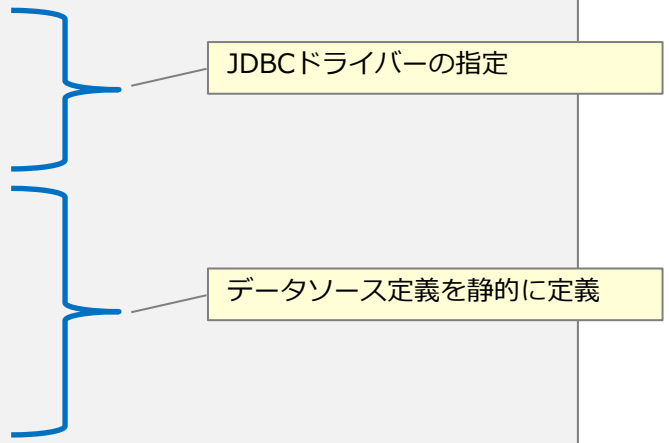
(例1)データソース定義を静的に定義(ConfigMap)

– 設定手順

- データソース定義をオーバーライドするために、以下のようなserver.xmlフラグメントを「server-overrides.xml」というファイル名で作成します。

(例)server-overrides.xml

```
<server>
    <jdbcDriver id="MySQLDriver">
        <library>
            <fileset dir="${server.config.dir}/resources/mysql" includes="mysql-*.jar" />
        </library>
    </jdbcDriver>
    <dataSource jndiName="jdbc/mydb" jdbcDriverRef="MySQLDriver">
        <properties databaseName="mydb"
            serverName="mysql"
            portNumber="3306"
            user="liberty"
            password="{xor}MzY9Oi0rJg==" />
    </dataSource>
</server>
```



- 上述「server-overrides.xml」内の「password」は、securityUtility encodeコマンドでエンコードしたものを記載します。
  – WAS Libertyのコンテナを一時的に起動してコマンドを実行する方法が簡単です。

```
docker run --rm -it websphere-liberty:19.0.0.3-kernel /opt/ibm/wlp/bin/securityUtility encode <パスワード>
```

(例1)データソース定義を静的に定義(ConfigMap)

– 設定手順

- 作成したserver.xmlフラグメントからConfigMapを作成します。

```
kubectl create configmap custom-server-configmap ¥
  --from-file=server-overrides.xml=./server-overrides.xml
```

- Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > myvalues.yaml
```

- 抽出した変数ファイルの「image.serverOverridesConfigMapname」に先ほど作成したConfigMapの名前を設定します。

(例)myvalues.yaml

```
image:
(略)
  serverOverridesConfigMapName: "custom-server-configmap"
```

- 編集した変数ファイルを使用して、Helm Chartをアップグレード/インストールします。

```
helm upgrade --install sample ibm-websphere-liberty-1.9.0.tgz ¥
  --namespace liberty-guide ¥
  --values myvalues.yaml ¥
  --tls
```

(例1)データソース定義を静的に定義(ConfigMap)

– 設定手順
  • 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。
    – 構成情報の処理状況は/logs/messages.logで確認できます。

(messages.log抜粋)

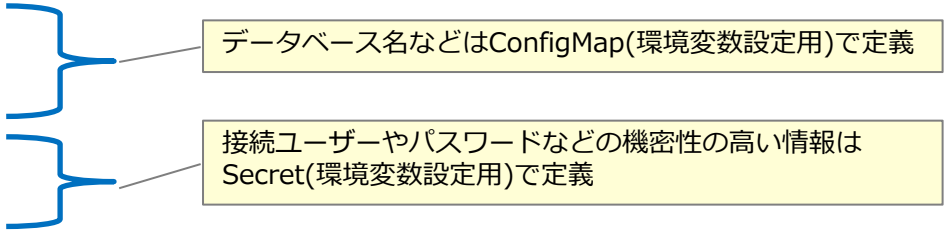
```
… … … … … … … … … … … … …
CWWKG0093A: Processing configuration drop-ins resource:
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/defaults/keystore.xml
CWWKG0093A: Processing configuration drop-ins resource:
/opt/ibm/wlp/usr/servers/defaultServer/configDropins/overrides/include-configmap.xml
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。(3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)
/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。

(例2)データソース定義を環境変数で定義(ConfigMapとSecret)

■ 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を設定する例を示します。データソース定義の接続情報部分に「環境変数」を使用します。

– 設定概要
- 前述の「ConfigMapを使用したWAS Libertyの設定方法」の(3)の方法を使用して、ConfigMapに格納するserver.xmlフラグメントにデータソース定義を設定する例を示します。
- データソース定義の接続情報部分に「環境変数」を使用します。
  – 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



- 環境変数の設定値は、環境変数設定用のConfigMapとSecretを別途作成して設定します。
- 設定に使用するConfigMapおよびSecretは以下になります。
  – ConfigMap(データソース定義を定義したserver.xmlフラグメント用)
  – ConfigMap(環境変数設定用)
  – Secret(環境変数設定用)

– 前提作業
- JDBCドライバーのjarファイルについては、Dockerイメージビルド時にコピー済みとします。

(例2)データソース定義を環境変数で定義(ConfigMapとSecret)

– 設定手順
- データソース定義をオーバーライドするために、以下のようなserver.xmlフラグメントを「server-overrides.xml」というファイル名で作成します。
- 以下の接続情報を環境変数から取得するようにします。
  – 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

(例)server-overrides.xml

```
<server>
    <jdbcDriver id="MySQLDriver">
        <library>
            <fileset dir="${server.config.dir}/resources/mysql" includes="mysql-*.jar" />
        </library>
    </jdbcDriver>
    <dataSource jndiName="jdbc/mydb" jdbcDriverRef="MySQLDriver">
        <properties databaseName="${env.DB_NAME}"
            serverName="${env.DB_HOST}"
            portNumber="${env.DB_PORT}"
            user="${env.DB_USER}"
            password="${env.DB_PASSWORD}" />
    </dataSource>
</server>
```

JDBCドライバーの指定

データソースの接続情報を環境変数として定義

(例2)データソース定義を環境変数で定義(ConfigMapとSecret)

– 設定手順

- 作成したserver.xmlフラグメントからConfigMap(データソース定義を定義したserver.xmlフラグメント用)を作成します。

```
kubectl create configmap custom-server-configmap ¥
  --from-file=server-overrides.xml=./server-overrides.xml
```

- 接続情報を格納したSecretとConfigMap (環境変数設定用)を作成します 。
  – パスワードはWAS LibertyのsecurityUtility encodeコマンドでエンコードすることを推奨します。

```
DB_PASSWORD=$(docker run --rm -it websphere-liberty:kernel /opt/ibm/wlp/bin/securityUtility encode <パスワード>)
```

  – ユーザー名とパスワードを格納したSecret(環境変数設定用)を作成します。

```
kubectl create secret generic mydb-secret ¥
  --from-literal=DB_USER=liberty ¥
  --from-literal=DB_PASSWORD=${DB_PASSWORD}
```

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

  – その他の接続情報を格納したConfigMap(環境変数設定用)を作成します。

```
kubectl create configmap mydb-config ¥
  --from-literal=DB_HOST=mysql ¥
  --from-literal=DB_PORT="3306" ¥
  --from-literal=DB_NAME=mydb
```

- Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > myvalues.yaml
```

(例2)データソース定義を環境変数で定義(ConfigMapとSecret)

– 設定手順

- 抽出した変数ファイルを以下のように指定します。
  – 「image.extraEnvs」でConfigMap(環境変数設定用)とSecret(環境変数設定用)から環境変数を指定します。
  – 「image.serverOverridesConfigMapname」に先ほど作成したConfigMap(データソース定義を定義したserver.xmlフラグメント用)の名前を指定します。

(例)myvalues.yaml

```
image:
(略)
  extraEnvs:
  - name: DB_HOST
    valueFrom:
      configMapKeyRef:
        name: mydb-config
        key: DB_HOST
  - name: DB_PORT
    valueFrom:
      configMapKeyRef:
        name: mydb-config
        key: DB_PORT
  - name: DB_NAME
    valueFrom:
      configMapKeyRef:
        name: mydb-config
        key: DB_NAME
```

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。

(例)myvalues.yaml (続き)

```
  - name: DB_USER
    valueFrom:
      secretKeyRef:
        name: mydb-secret
        key: DB_USER
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mydb-secret
        key: DB_PASSWORD
(略)
  serverOverridesConfigMapName: "custom-server-configmap"
(略)
```

- 編集した変数ファイルを使用して、Helm Chartをアップグレード/インストールします。(例1の環境変数を使用しない場合と同様です。)

```
helm upgrade --install sample ibm-websphere-liberty-1.9.0.tgz --namespace liberty-guide --values myvalues.yaml --tls
```

目次(11章)

(補足)WAS LibertyのHelm Chartで自動作成されるConfigMapの構成

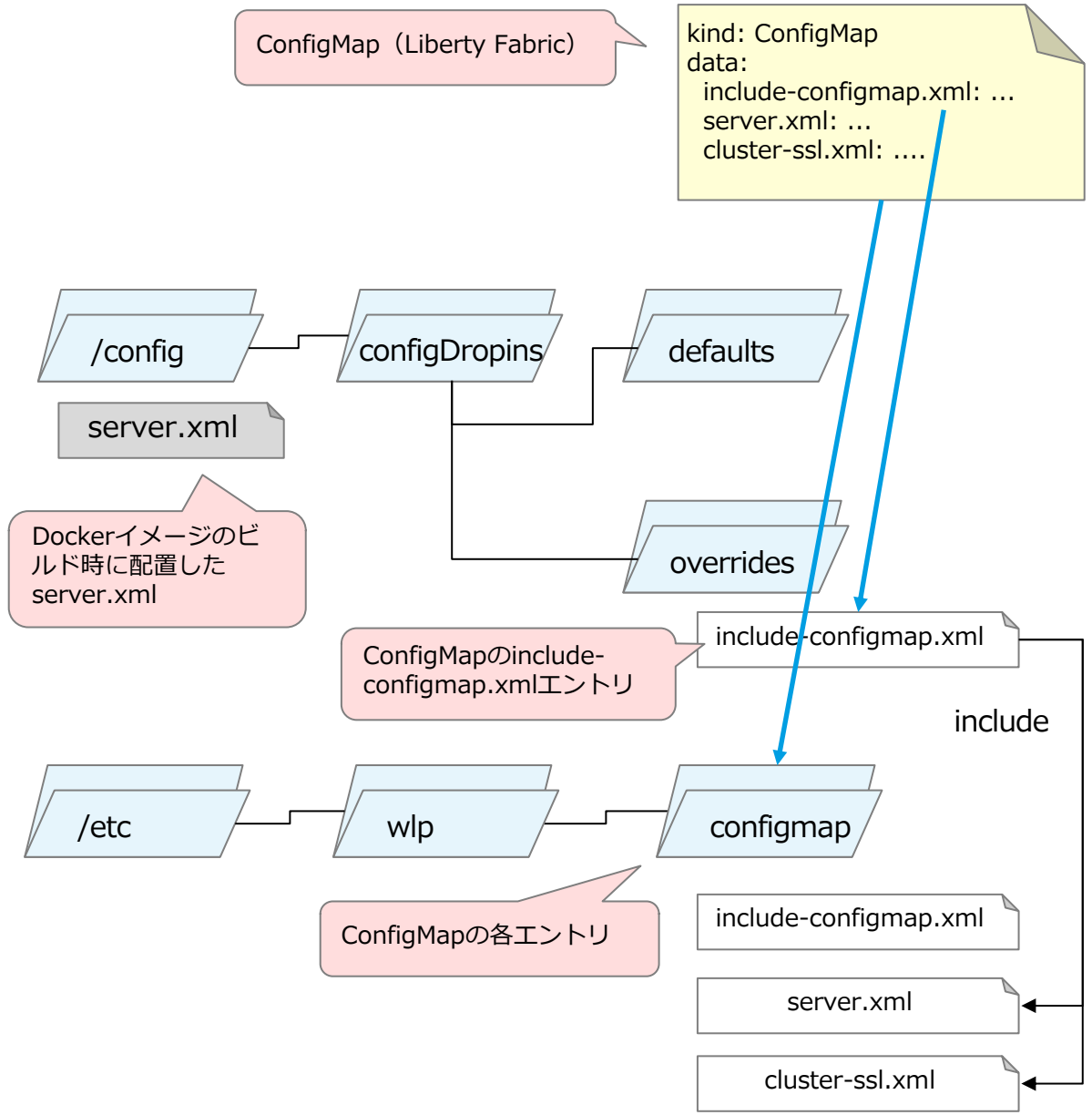
- 補足として、以下の(4)のHelm Chartインストール時に自動作成されるConfigMapによるWAS Libertyの構成ファイル(server.xmlフラグメント)の配置場所について説明します。

(3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」参照)

#	設定箇所	概要	server.xmlフラグメント	パス/ファイル名
(0)	Dockerイメージ	IBM提供のWAS LibertyのDockerイメージに含まれるserver.xmlファイル	server.xml	• /config/server.xml
(1)		作成したserver.xmlをコピーしてDockerイメージをビルド	server.xml	• /config/server.xml
(2)		WAS LibertyのDockerイメージが提供する機能(変数ARGとシェル)を利用してserver.xmlフラグメントをコピーしてDockerイメージをビルド	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/mp-monitoring.xml など
(3)	Helm ChartとConfigMap	server.xmlフラグメントをConfigMapとして事前に作成し、Helm変数ファイルの「image.serverOverridesConfigMapName」でこのConfigMapを指定	server.xmlフラグメント(configDropins)	• /config/configDropins/overrides/server-overrides.xml
(4)		(Option)Helm Chartインストール時に自動で作成されるinclude用のserver.xmlフラグメントを定義したConfigMapを修正	server.xmlフラグメント(include)	• /config/configDropins/overrides/include-configmap.xml（include用） • /etc/wlp/configmap/server.xml（include-configmap.xmlからincludeされるserver.xmlフラグメント)

(補足)WAS LibertyのHelm Chartで自動作成されるConfigMapの構成

- Helm Chartを使ってインストールした場合、構成ファイルは以下のように配置されます。
  - <Helmリソース名>-ibm-websphereという名前のConfigMap(以下Liberty Fabric)に構成ファイルが含まれます。
  - Liberty Fabricの各エントリ/etc/wlp/configmapにファイルとしてマウントされます
    - Keyがファイル名、Valueがファイルの中身となります。
    - subPath を指定しないでマウントしているため、ConfigMapの変更はコンテナ内に動的に反映されます。
  - Liberty Fabricのinclude-configmap.xmlエントリが/config/configDropins/overridesにマウントされます。
    - subPathを指定しているため、ConfigMapの変更はコンテナ内に動的には反映されません。



(補足)WAS LibertyのHelm Chartで自動作成されるConfigMapの構成

- ConfigMapと、ConfigMapのエントリをファイルとしてマウントするPodの定義の抜粋が以下になります。

```
# <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
```

12. DBを使用したセッション・パーシスタンス

WAS Libertyのセッション・パーシスタンス機能

■ セッション・パーシスタンス機能

– WAS Libertyが保持するHTTPセッション・オブジェクトを外部保管する機能です。

– WAS Libertyの障害時に、別のWAS LibertyがHTTPセッション・オブジェクトを引き継ぎ、HTTPセッションを継続することが可能になります。

– 例えば、以下のような要件を満たすために、セッション・パーシスタンスを利用します。

- HTTPセッションの高可用性を確保したい場合
- HTTPセッションのアフィニティーが十分には確保できない環境で稼動させる必要がある場合
- 無停止でサービスを提供するが、定期保守などで部分的な停止が必要な場合

– WAS Liberty がサポートするセッション・パーシスタンス方法は、以下の通りです

- データベース
- WebSphere eXtreme Scale (WXS)
- JCache (Liberty 18.0.0.2 以降で対応)

– セッション・パーシスタンスを利用するには、アプリケーション側での対応も必要になります

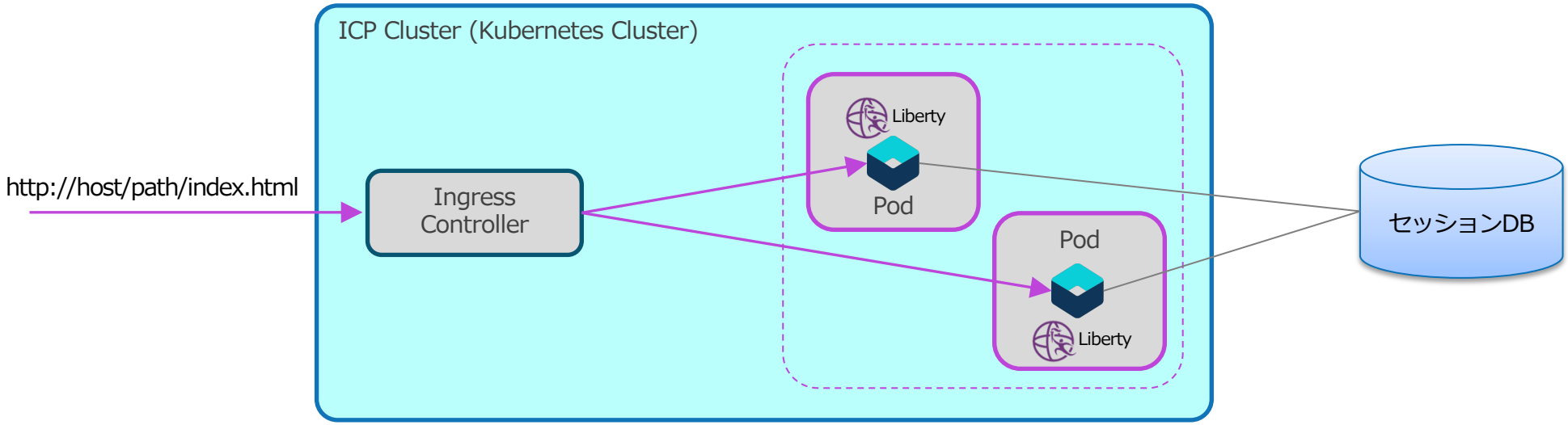
- HTTP セッションに保管するデータは Serializable なオブジェクトである必要があります。
- パフォーマンスへの影響を減らすために、HTTP セッション・データは軽量である必要があります。

■ 12章(本章)では、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、13章(次章)では、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。

セッションDBを使用したセッション・パーシスタンスの構成例

– 設定概要
  - ICP環境でも、従来環境と同様の構成定義を行うことで、データベースを使用したセッション・パーシスタンスが実現可能です。
    – WAS 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WAS Libertyのイメージに対して、セッションDBの構成定義を設定したserver.xmlフラグメントをConfigMapとして事前に作成する方法で、セッション・パーシスタンスを実装する例を記載します。
    – (3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」(3)の方法参照)

– 前提作業
  - 「sessionDatabase-1.0」フィーチャーはWAS Libertyのイメージに含まれている必要があります。
  - JDBCドライバーはあらかじめWAS Libertyのイメージに含まれている必要があります。
    – JDBCドライバーのファイルサイズが1MB以下であれば、ConfigMapまたはSecretにバイナリファイルを格納し、Helm Chartの変数ファイルの「extraVolumeMounts」/「extraVolumes」を使用してWAS Libertyコンテナにマウントさせることも可能ですが、ここでは説明しません。



セッションDBを使用したセッション・パーシスタンスの構成例

- 設定手順
  - セッションDBを構成するために、右図のようなserver.xmlフラグメントを「server-overrides.xml」というファイル名で作成します。
    - 接続先のデータベースサーバーには、セッション用のデータベース(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liberty)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、ここでは環境変数に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。

(例)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 databaseName="mysession"
            serverName="mysql"
            portNumber="3306"
            user="liberty"
            password="{xor}MzY9Oi0rJg==" />
    </dataSource>

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

</server>
```

セッションDBを使用したセッション・パーシスタンスの構成例

– 設定手順

- 作成したserver.xmlフラグメントからConfigMapを作成します。

```
kubectl create configmap custom-server-configmap ¥
  --from-file=server-overrides.xml=./server-overrides.xml
```

- Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > myvalues.yaml
```

- 抽出した変数ファイルの「image.serverOverridesConfigMapname」に先ほど作成したConfigMapの名前を設定します。

(例)myvalues.yaml

```
image:
(略)
  serverOverridesConfigMapName: "custom-server-configmap"
```

- 編集した変数ファイルを使用して、Helm Chartをアップグレード/インストールします。

```
helm upgrade --install sample ibm-websphere-liberty-1.9.0.tgz ¥
  --namespace liberty-guide ¥
  --values myvalues.yaml ¥
  --tls
```

セッションDBを使用したセッション・パーシスタンスの構成例

– 設定手順
- セッション・パーシスタンスの稼働状況を確認します。
- 起動すると、セッションDB内にセッション・パーシスタンス用のテーブルsessionsが作成されます。
- アプリケーションがHTTPセッションを作成すると、HTTPセッション・データがテーブルに保管されます。

(セッションDBの確認例)

```
mysql> use mysession;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_mysession |
+---------------------+
| sessions            |
+---------------------+
1 row in set (0.01 sec)

mysql>
```

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

13. JCacheを使用したセッション・パーシスタンス

目次(13章)

- JCacheを使用したセッション・パーシスタンス
- セッション・パーシスタンスのトポロジー
  - Peer-to-Peer構成(Embedded構成)
  - Client-Server構成
- Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法
- ICP環境でのHazelcast(Peer-to-Peer構成)の構成例
  - (1)Dockerイメージが提供する事前定義server.xmlフラグメントを使う場合
  - (2)手動で構成したserver.xmlフラグメントを使う場合

目次(13章)

JCacheを使用したセッション・パーシスタンス

- WAS Liberty 18.0.0.2（2018年6月末リリース）から、JCache を使用したセッション・パーシスタンス機能が提供されています。
  - WAS 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
  - WAS 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」を使用して、相互レプリケーション方式(Peer-to-Peer構成)のセッション・パーシスタンスの構成例を紹介します。
  - 「Hazelcast」は上記URLで紹介されているJCache実装の１つです。

目次(13章)

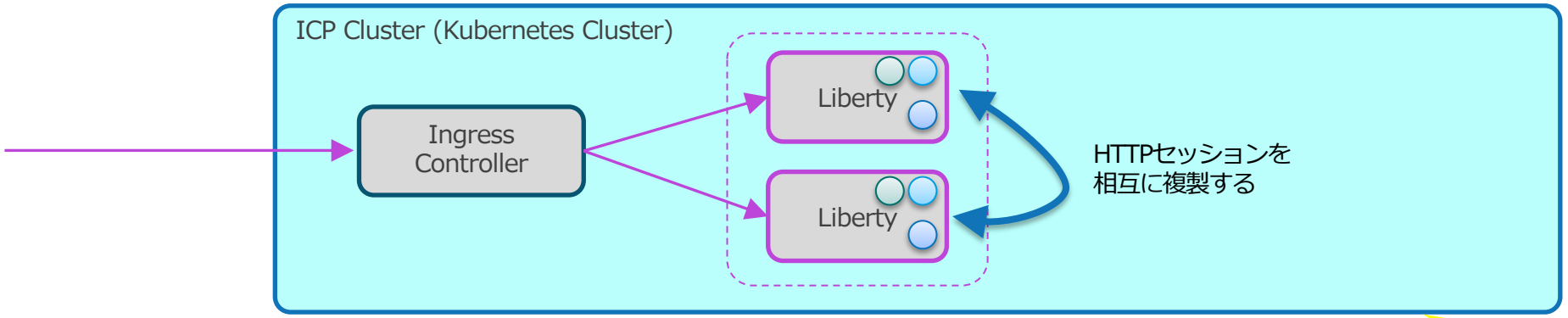
- JCacheを使用したセッション・パーシスタンス
- セッション・パーシスタンスのトポロジー
  - Peer-to-Peer構成(Embedded構成)
  - Client-Server構成
- Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法
- ICP環境でのHazelcast(Peer-to-Peer構成)の構成例
  - (1)Dockerイメージが提供する事前定義server.xmlフラグメントを使う場合
  - (2)手動で構成したserver.xmlフラグメントを使う場合

セッション・パーシスタンスのトポロジー

■ セッション・パーシスタンスは以下の2つのトポロジー構成が可能です。

– Peer-to-Peer構成(Embedded 構成)

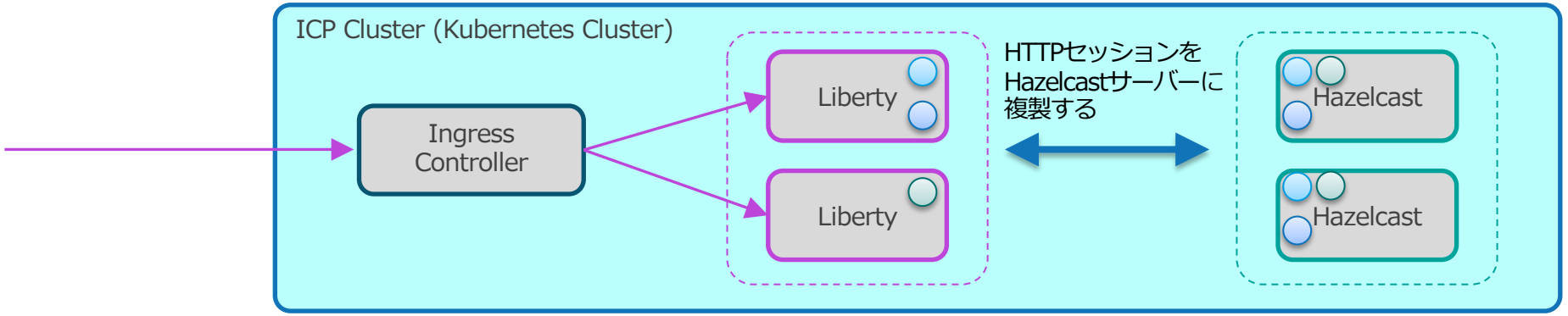
• セッションをWAS Libertyサーバー間で相互にレプリケーションします。



この資料ではこちらの
Peer to Peerの構成に
ついて説明します。

– 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。



目次(13章)

Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法

- Kubernetes環境でHazelcastを利用する場合、レプリケーションを行う宛先のHazelcastメンバーをディスカバリーする必要があり、そのためのプラグインが下記のURLで公開されています。
  - 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

- レプリケーション対象のメンバーは以下の方法で指定します。
  - Kubernetes API
    - サービスを指定し、Kubernetes API経由でメンバーを発見します。
    - この方法を使用するにはサービス定義が必要になります。
    - 新規にHazelcast用のサービス(5701ポート)を定義するか、WAS Libertyのサービス定義を利用し、ポート番号をHazelcast用にオーバーライドします。
  - DNS Lookup
    - ヘッドレス・サービスをKubernetes DNSでlookupすることでメンバーを発見します。
    - この方法を使用する場合は、ヘッドレス・サービスが必要になります。
    - 新規にHazelcast用のヘッドレス・サービス(5701ポート)を定義するか、WAS LibertyをStatefulSetとしてインストール(*)すると、StatefulSetの前提としてヘッドレス・サービスが定義されるので、それを利用できます。
      - 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としてインストールされます。(「5章:ログとダンプの永続化と転送」参照)

目次(13章)

ICP環境でのHazelcast(Peer-to-Peer構成)の構成例

- ICP環境でのHazelcast(Peer-to-Peer構成)の構成例は以下の2つの方法があります。
  - (1)Dockerイメージが提供する事前定義server.xmlフラグメントを使う場合
  - (2)手動で構成したserver.xmlを使う場合

- それぞれの構成でHazelcast(Peer-to-Peer構成)の挙動が異なります。
  - (1)Dockerイメージが提供する事前定義server.xmlフラグメントを使う場合
    - WAS LibertyのDockerイメージが提供するHazelcastの設定が事前定義されたserver.xmlフラグメント(snippet)を使用します。
      - (3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(2)の方法参照)
    - 事前定義されたserver.xmlフラグメントには、レプリケーション対象のサーバーが指定されていないため、Namespace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。
    - WAS Libertyが稼働するNamespaceにレプリケーション対象のWAS Liberty以外のPodが存在する場合は、(2)の方法を使用してください。
  - (2)手動で構成したserver.xmlフラグメントを使う場合
    - 手動でHazelcastの設定をしたserver.xmlフラグメントを作成する方法です。
      - (server.xmlまたはserver.xmlフラグメントを更新する方法は幾つかありますが、ここでは3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(1)の「作成したserver.xmlをコピーしてDockerイメージをビルド」と同様の方法で、ビルド時にserver.xmlフラグメントをコピーする方法を使用します。)
    - 前述の「Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法」に記載のレプリケーション対象のメンバーを指定する方法を使用して、レプリケーション対象を指定できます。
    - WAS Libertyが稼働するNamespaceにレプリケーション対象のWAS Liberty以外のPodが存在する場合は、この方法を使用します。

- 次ページからそれぞれの場合の構成例を記載します。

(1)Dockerイメージが提供する事前定義server.xmlフラグメントを使う場合

- IBM提供のWAS LibertyのDocker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)があらかじめ用意されており、Dockerイメージのビルド時に有効にすることができます。
  - (3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(2)の方法)
  - 「HZ_SESSION_CACHE」を有効化する場合の、Dockerfileの設定方法は以下のリンクを参照ください。
    - GitHub / WASdev/ci.docker / WebSphere Application Server Liberty Profile and Docker / Session Caching
    - https://github.com/WASdev/ci.docker#session-caching

(Dockerfile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

# 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。

- この方法で組み込まれる事前定義されたserver.xmlフラグメントには、レプリケーション対象のサーバーが指定されていないため、Namespace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。
- WAS Libertyが稼働するNamespaceにレプリケーション対象のWAS Liberty以外のPodが存在する場合は、次ページで説明する(2)の方法を使用してください。

(2)手動で構成したserver.xmlフラグメントを使う場合

■ 手動でHazelcastの設定をしたserver.xmlを作成する方法です。

– 設定概要

- 前述の「Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法」に記載のレプリケーション対象のメンバーを指定する方法を使用して、レプリケーション対象を指定できます。
- WAS Libertyが稼働するNamespaceにレプリケーション対象のWAS Liberty以外のPodが存在する場合は、この方法を使用します。

- server.xmlまたはserver.xmlフラグメントを更新する方法は幾つかありますが、ここでは3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(1)の「作成したserver.xmlをコピーしてDockerイメージをビルド」と同様の方法で、ビルド時にserver.xmlフラグメントをコピーする方法を使用します。

– 設定の流れ

- (1) 事前にHazelcastを使用するように定義したserver.xmlフラグメント(hazelcast-sessioncache.xml)を作成します。
- (2) 事前にHazelcastの構成ファイル(hazelcast.xml)を作成します。
- (3) Dockerイメージをビルドします。
- (4) Helm Chartから変数ファイルを抽出します。
- (5) 抽出した変数ファイルのパラメーターを修正します。
- (6) 編集した変数ファイルを使用して、Helm Chartをアップグレード/インストールします。
- (7) セッション・パーシスタンスの稼働状況を確認します。

(2)手動で構成したserver.xmlフラグメントを使う場合

– 設定手順
- (1) 事前にHazelcastを使用するように定義したserver.xmlフラグメント(hazelcast-sessioncache.xml)を作成します。
  – server.xmlまたはserver.xmlフラグメントを更新する方法は幾つかありますが、ここでは3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(1)の「作成したserver.xmlをコピーしてDockerイメージをビルド」と同様の方法で、ビルド時にserver.xmlフラグメントをコピーする方法を使用します。

(例)server.xmlフラグメント(例としてファイル名は「hazelcast-sessioncache.xml」で作成)

```
<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"/>
  </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-kubernetesとその依存モジュールのjarが必要

(2)手動で構成したserver.xmlフラグメントを使う場合

- (2) 事前にHazelcastの構成ファイル(hazelcast.xml)を作成します。
  - 前述の「Kubernetes環境でのHazelcast(Peer-to-Peer構成)のレプリケーション方法」に記載のレプリケーション対象のメンバーを指定する方法を使用して、レプリケーション対象を指定できます。
  - メンバーを指定する方法(「Kubernetes API」または「DNS Lookup」)によって、discovery-strategyのpropertyに指定する内容が変わります。

(例) Hazelcastの構成ファイル(例としてファイル名は「hazelcast.xml」で作成)

```
<?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。

メンバーを指定する方法(「Kubernetes API」または「DNS Lookup」)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」で発見する場合)

```
<namespace>liberty-guide</namespace>
<service-name>sample-ibm-websphere-lib</service-name>
<service-port>5701</service-port>
```

(「DNS Lookup」で発見する場合)

```
<service-dns>sample-ibm-websphere-lib-sts.liberty-guide.svc.cluster.local</service-dns>
<service-dns-timeout>5</service-dns-timeout>
```

(2)手動で構成したserver.xmlフラグメントを使う場合

- (3) Dockerイメージをビルドします。
  - 手順(1)で作成したserver.xmlフラグメント(hazelcast-sessioncache.xml)をコピーします。(Server.xmlまたはserver.xmlフラグメントを更新する方法は幾つかありますが、ここでは3章「ICP環境でのWAS Libertyのserver.xmlの設定方法」の(1)の「作成したserver.xmlをコピーしてDockerイメージをビルド」と同様の方法で、ビルド時にserver.xmlフラグメント(hazelcast-sessioncache.xml)をコピーします。)
  - 同様に手順(2)で作成したHezelcastの構成ファイル(hazelcast.xml)もコピーします。
  - また「hazelcast/hazelcast」コンテナイメージからライブラリもコピーします。(もちろんMaven等で取得したファイルをコピーすることも可能です。)

(Dockerfileの例)

```
ENV JAVA_TOOL_OPTIONS="-Dhazelcast.jcache.provider.type=server ${JAVA_TOOL_OPTIONS}"
COPY --from=hazelcast/hazelcast --chown=1001:0 /opt/hazelcast/lib/*.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/
```

  - ビルドしたイメージは、3章の「ICP環境でのWAS Libertyのインストール」の「(2) DockerイメージをICPプライベートDockerレジストリーにpushする」の手順を参照し、ICPのプライベートDockerレジストリーにpushします。

- (4) Helm Chartから変数ファイルを抽出します。

```
helm inspect values ibm-websphere-liberty-1.9.0.tgz > myvalues.yaml
```

- (5) 抽出した変数ファイルのパラメーターを修正します。
  - (3)で作成したDockerイメージ名を指定するなど、適宜必要なパラメーターを修正してください。

- (6) 編集した変数ファイルを使用して、Helm Chartをアップグレード/インストールします。

```
helm upgrade --install sample ibm-websphere-liberty-1.9.0.tgz --namespace liberty-guide --values myvalues.yaml --tls
```

(2)手動で構成したserver.xmlフラグメントを使う場合

- (7) 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。

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

```
... ... ... ...
[INFO    ] [10.1.47.6]:5701 [dev] [3.12.1] Kubernetes Discovery properties: { service-dns: null, service-dns-timeout: 5, service-name: sample-ibm-websphere-lib, service-port: 5701,
service-label: null, service-label-value: true, namespace: liberty-guide, resolve-not-ready-addresses: false, kubernetes-api-retries: 3, kubernetes-master:
https://kubernetes.default.svc}
[INFO    ] [10.1.47.6]:5701 [dev] [3.12.1] Kubernetes Discovery activated with mode: KUBERNETES_API
[INFO    ] [10.1.47.6]:5701 [dev] [3.12.1] Activating Discovery SPI Joiner
... ... ... ...
[INFO    ] [10.1.47.6]:5701 [dev] [3.12.1]

Members {size:2, ver:2} [
                Member [10.1.60.68]:5701 - 90628054-88c7-485b-8f0e-4934f731aed6
                Member [10.1.47.6]:5701 - 80750aeb-f2b0-4173-8a2d-6eb7feb0bef0 this
]
... ... ... ...
```

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

```
... ... ... ...
[INFO    ] [10.1.47.12]:5701 [dev] [3.12.1] Kubernetes Discovery properties: { service-dns: sample-ibm-websphere-lib-sts.liberty-guide.svc.cluster.local, service-dns-timeout: 5,
service-name: null, service-port: 0, service-label: null, service-label-value: true, namespace: liberty-guide, resolve-not-ready-addresses: false, kubernetes-api-retries: 3, kubernetes-
master: https://kubernetes.default.svc}
[INFO    ] [10.1.47.12]:5701 [dev] [3.12.1] Kubernetes Discovery activated with mode: DNS_LOOKUP
[INFO    ] [10.1.47.12]:5701 [dev] [3.12.1] Activating Discovery SPI Joiner
... ... ... ...
[INFO    ] [10.1.47.12]:5701 [dev] [3.12.1]

Members {size:2, ver:2} [
                Member [10.1.60.75]:5701 - a757c32f-9c11-4b88-8c73-94975b1a2a2a
                Member [10.1.47.12]:5701 - a8501d90-679a-40f7-b481-e86ab5948349 this
]
... ... ... ...
```

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

【参考】Ingress Controllerのカスタマイズ

Ingress Controllerが使用するデフォルトの証明書の変更

■ Ingress Controllerが使用するデフォルトの証明書を変更することができます。

– 設定概要
- Ingress ControllerはProxy Nodeで稼動するDaemonSetとして定義されています。
  – Ingress Controllerのネームスペースは「kube-system」です。

- Ingress ControllerのDaemonSetに定義されているnginx-ingress-controllerの起動引数に「--default-ssl-certificate=<ネームスペース名>/<TLS Secret名>」を追加することで、デフォルトの証明書を変更できます。
  – NGINX Ingress Controller / Command line arguments
  – https://kubernetes.github.io/ingress-nginx/user-guide/cli-arguments/

– 設定手順
- まず、サーバー証明書と鍵をTLS Secretとして登録します。
  – NGINX Ingress Controllerの仕様にあわせて、証明書と鍵は「PEM-encoded X.509, RSA (2048) secret」として登録します。
  – NGINX Ingress Controller / TLS/HTTPS
  – https://kubernetes.github.io/ingress-nginx/user-guide/tls/

  – TLS Secret は以下のようなコマンドでPEMファイルから定義できます。
  – （下記の例では、ネームスペース「kube-system」内に定義しています。）

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

Ingress Controllerが使用するデフォルトの証明書の変更

- 次に、Ingress Controllerの起動引数を追加します。
  - 起動引数の追加は「kubectl edit DaemonSet」コマンドで行います。
  - (ICP Management Consoleから行う場合は、[DaemonSets]画面から追加します。)

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

(例)DaemonSet「nginx-ingress-controller」のedit画面で、以下の青字部分を追加

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

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

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

エラー・ページのカスタマイズ: デフォルトのエラー・ページ

■ Ingress Controllerはデフォルトで次のエラー・ページを使用します。

– エラー・ページ

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

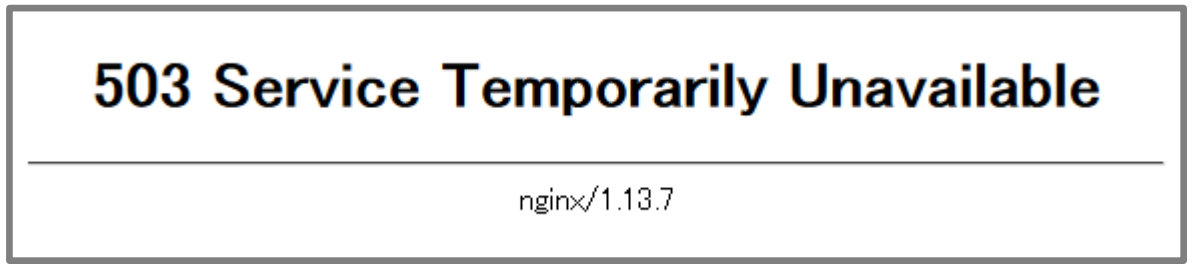
– 表示例

Default Backend が返したエラー・ページ

```
default backend - 404
```

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

Ingress Controller のデフォルトのエラー・ページ



エラー・ページのカスタマイズ: custom-http-errorsの指定

■ 「Ingress Controllerのデフォルトのエラー・ページ」の代わりに「Default Backendが生成したエラー・ページ」を使用するように構成できます。

– 設定概要
- ネームスペース「kube-system」内のConfigMap「nginx-ingress-controller」に「custom-http-errors」の指定を追加すると、指定されたHTTPステータス・コード用のエラー・ページとして「Default Backend」が生成したページが使用されるようになります。
  – エラー・ページをカスタマイズするには、「Default Backend」を差し替える必要があります。
  – 404 などのアプリケーションが返す可能性がある HTTP ステータス・コードを指定すると、アプリケーションが返したエラー・ページも 「Default Backend」が生成したページで置き換わります。
  – NGINX Ingress Controller / ConfigMap
  – https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/

– 設定手順
- 「kubectl edit ConfigMap」コマンドで追加します。
  – (ICP Management Consoleから行う場合は、[ConfigMaps]画面から追加します。)

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

- 変更を保存するとIngress Controllerが変更を検知し、自動的に反映されます。

(例)ConfigMap「nginx-ingress-controller」のedit画面で、data部分に、「custom-http-errors」の指定を追加し、保存します(以下の青字部分)。

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

エラー・ページのカスタマイズ: Default Backendの準備

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

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

エラー・ページのカスタマイズ: Default Backendの差し替え

- Ingress Controllerが使用するDefault Backendを差し替えることができます。

  - 設定概要
    - Ingress ControllerはProxy Nodeで稼動するDaemonSetとして定義されています。
      - Ingress Controllerのネームスペースは「kube-system」です。
    - Ingress ControllerのDaemonSetに定義されているnginx-ingress-controllerの起動引数の「--default-backend-service=<ネームスペース名>/<サービス名>」を変更することで、Default Backendを差し替えることができます。
      - NGINX Ingress Controller / Command line arguments
      - https://kubernetes.github.io/ingress-nginx/user-guide/cli-arguments/

エラー・ページのカスタマイズ: Default Backendの差し替え

- 設定手順
  - Ingress Controllerの起動引数を修正します。
    - 起動引数の修正は「kubectl edit DaemonSet」コマンドで行います。
    - (ICP Management Consoleから行う場合は、[DaemonSets]画面から修正します。)

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

(例)DaemonSet「nginx-ingress-controller」のedit画面で、以下の青字部分を修正

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

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

(デフォルト値は「$(POD_NAMESPACE)/default-http-backend」です。)

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

IBM Cloud Private

クラウド・ネイティブの能力をオンプレに