第1章:
ICP Kubernetes環境のシステム像、および、その基本機能の概略

章内目次

第1章 第1節:
ICP製品によるコンテナー運用支援機能とそのシステム像

コンテナー運用を支援するためのICP製品機能

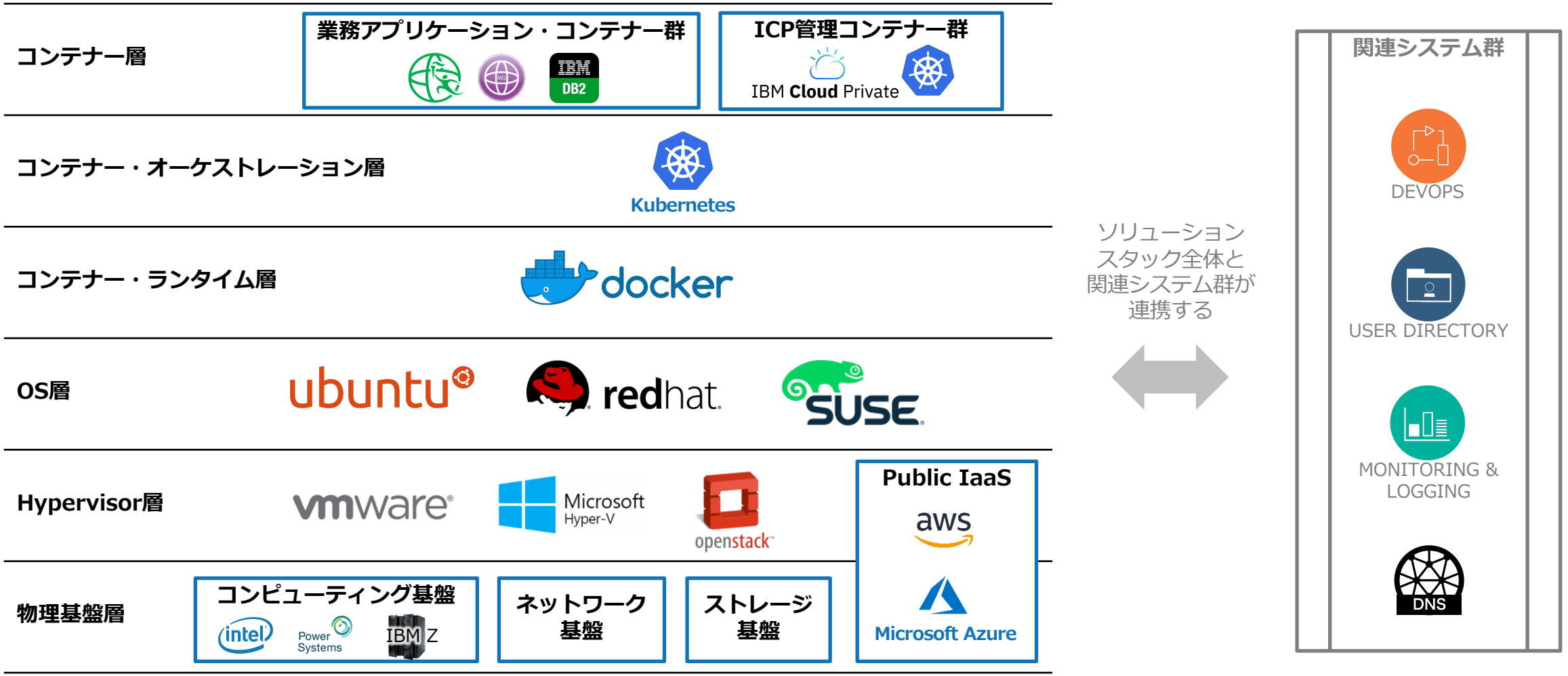
■ ICP製品が提供する、コンテナー運用を支援するための主要機能を下表に示す

機能項目	機能の概略	機能項目の実装の概略（※）
コンテナー稼働の維持・管理	• コンテナーの稼働状態を維持・管理するための機能	• Kubernetes製品の「Workload」区分に属するリソースを中心として実装される（1.2節でその概略を解説）
ネットワーク・アクセスの維持・制御	• コンテナー（上で稼働するサービス）へのネットワーク・アクセスを維持・制御するための機能	• Kubernetes製品の「Service」区分に属するリソースを中心として実装される（1.3節でその概略を解説）
永続ストレージ、および環境変数の提供	• コンテナーによる永続データ、および環境変数の保管・利用を支援するための機能	• Kubernetes製品の「Config and Storage」区分に属するリソースを中心として実装される（1.4節でその概略を解説）
認証・認可	• Kubernetes環境での認証・認可を実現するための機能	• Kubernetes製品の「Cluster」区分に属するリソース、および、それらリソースと連携するICP製品独自機能の組み合わせにより実装される
リポジトリー	• Kubernetes環境へのアプリケーション展開の際に用いる各種の雛形を保管・公開するための機能	• ICP製品に組み込まれた各種のオープンソース・ソフトウェアの組み合わせを中心として実装される（5.2節で解説）
ロギング・モニタリング	• Kubernetes環境でのロギング・モニタリングを支援するための機能	• ICP製品に組み込まれた各種のオープンソース・ソフトウェアの組み合わせを中心として実装される
管理インターフェース（GUI、CLI）	• Kubernetes環境への管理運用操作を実現するための機能	• ICP製品に組み込まれたオープンソース・ソフトウェア製品が提供するCLI、および、ICP製品独自のGUI/CLIの組み合わせとして実装される（1.5節でその概略を解説）

※: 表中でのKubernetes製品のリソース区分（Workload / Service / Config and Storage / Cluster）はAPI ReferenceでのAPI区分に基づく形で定義した。

コンテナー運用を実現するためのソリューション・スタック構成

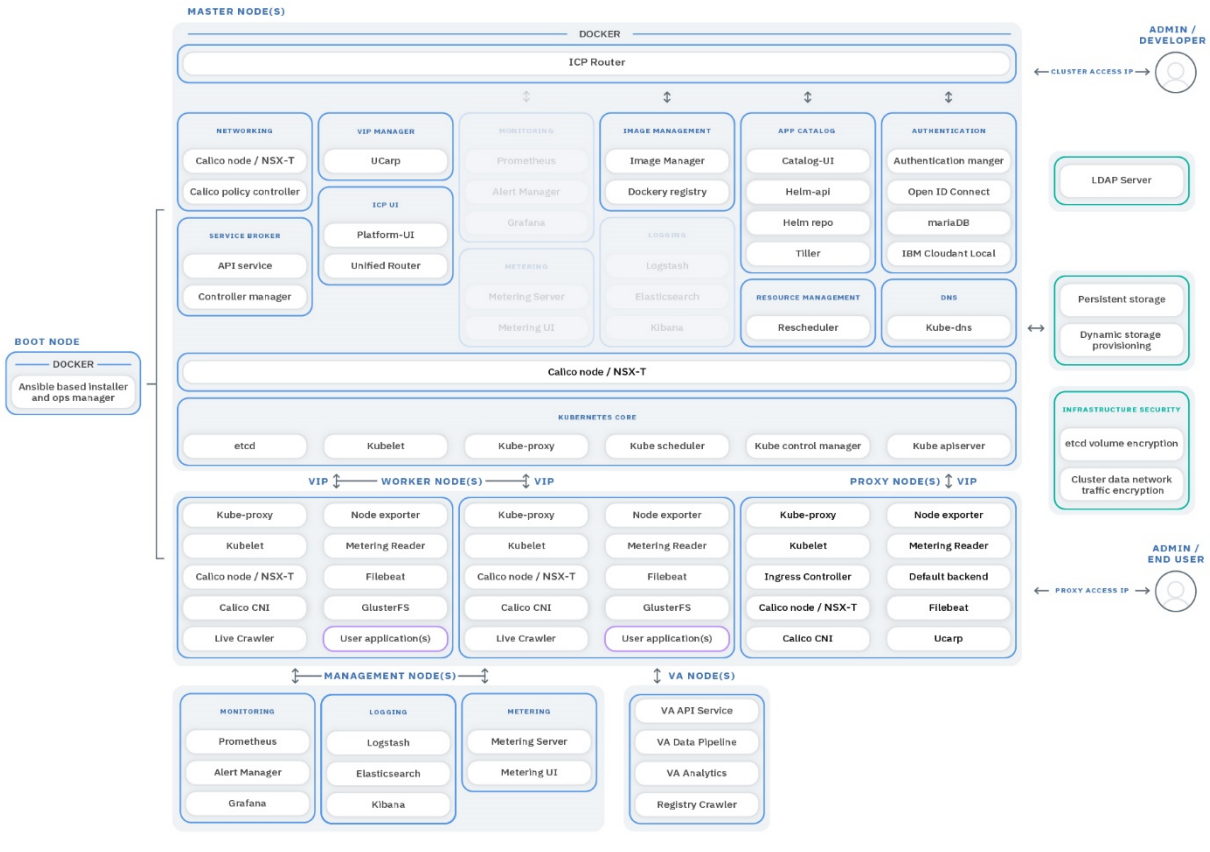
■ ICP製品はKubernetes製品を中核としたソリューション・スタック構成を採ることにより、コンテナー運用を実現するための包括的な機能を提供する



ICP製品機能を実現するためのソフトウェア・コンポーネント

■ ICP製品は多種多様なソフトウェア・コンポーネントから構成される（これらコンポーネントがソリューション・スタックの「OS層」以上の階層に配置される）

- 参照: IBM Knowledge Center - IBM® Cloud Private コンポーネント
- 参照: IBM Knowledge Center – アーキテクチャー（以下に示すコンポーネント配置図が掲載されている）



多種多様なソフトウェア・コンポーネントが複数ノードに分散配置される。
これらのソフトウェア・コンポーネントにより、ICP製品機能が実装される。

※: 通常時での製品機能の利用に際しては個々のソフトウェア・コンポーネントの存在を意識することは無いものの、各種の基盤運用や、高可用性構成の検討の際には、これらコンポーネントを明確に認識する必要が生じ得る。

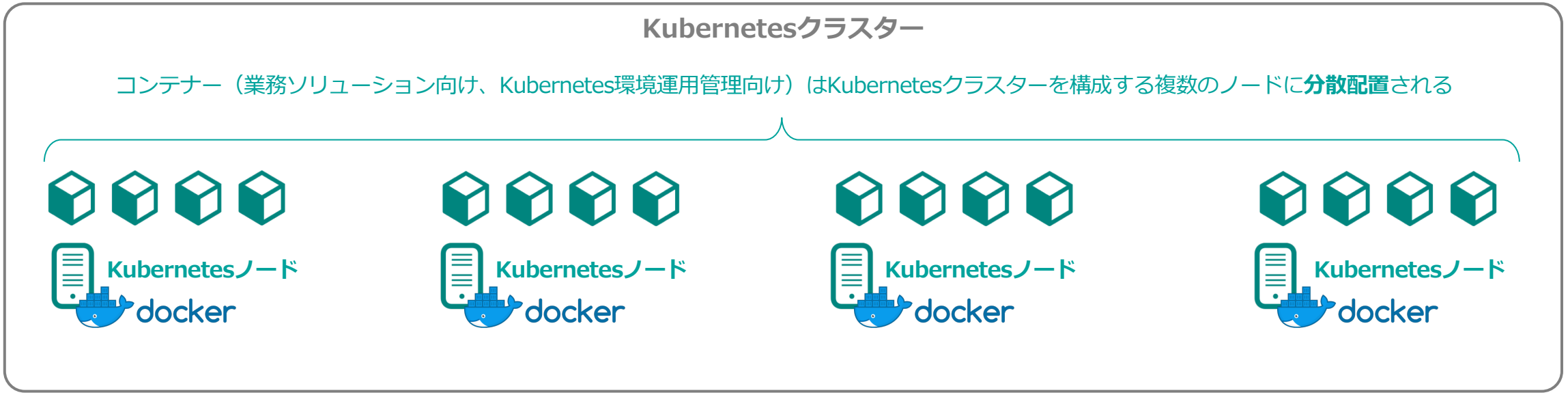
ICP製品により実現するKubernetes環境のシステム構成

■ ICP製品により実現する、コンテナー運用を支援するためのシステム環境（本ガイドでは「Kubernetes環境」と呼称する）でのシステム構成の要点は以下の3点

① OSが導入されたサーバー上に、コンテナー・ランタイム、および、コンテナー・オーケストレーションのためのソフトウェア・モジュールを導入する（このサーバーを「**ノード**」と呼ぶ）

② 複数のノードを互いに連携させることによるクラスター構成を、 Kubernetes製品機能を用いて実現する（このようなクラスター構成の単位を「**Kubernetesクラスター**」と呼称する）

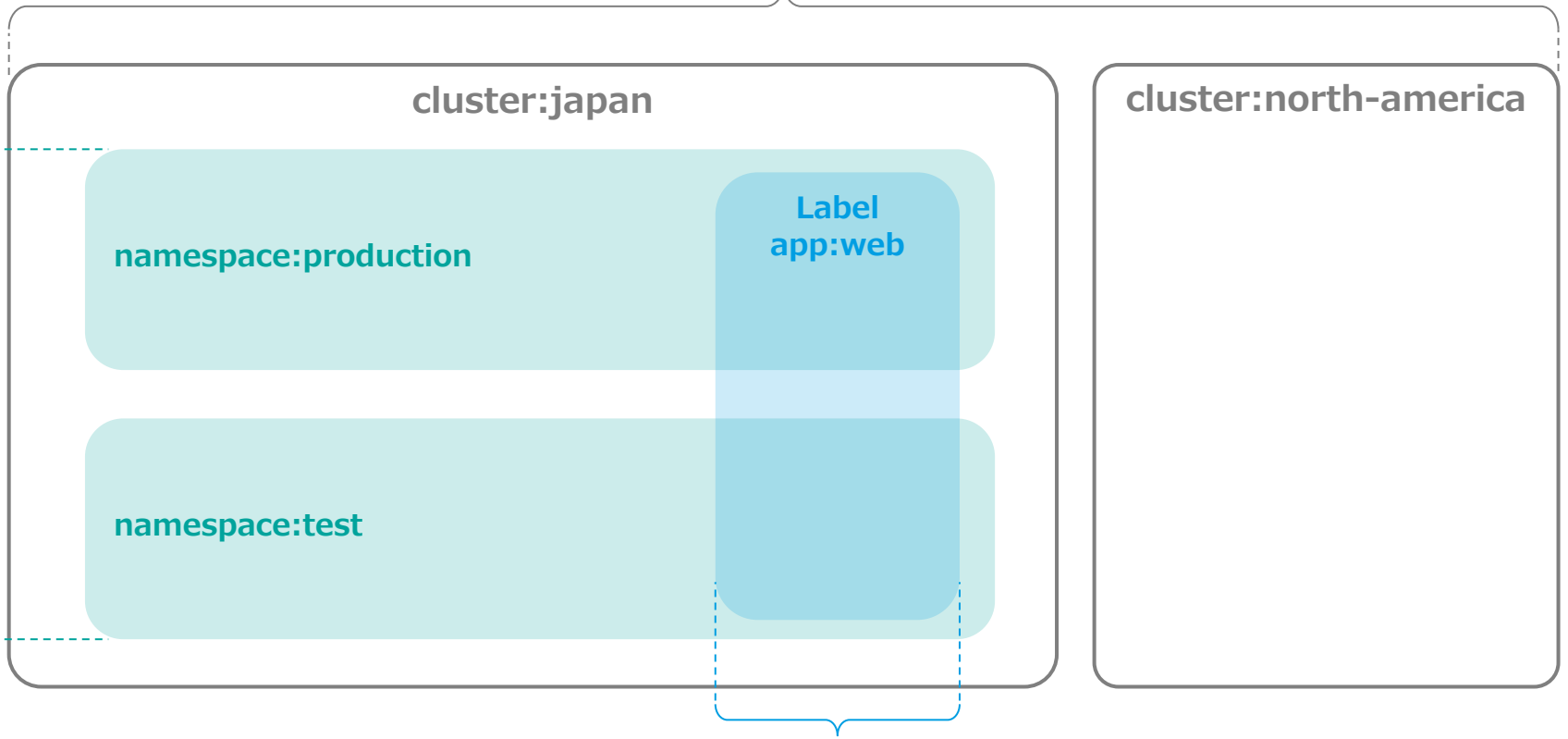
③ クラスター内の各ノード上では業務ソリューションを稼働させるためのコンテナー、および、Kubernetes環境の運用管理のために用いるコンテナーを稼働させる



Kubernetes環境内部の論理構造（基本的な概念）

■ Kubernetes環境内部での論理構造に関わる基本的な概念を下図に示す

Kubernetes環境での論理構造の最上位の単位は「Cluster（クラスター）」である。
それぞれのクラスターは、相互に依存することなく、互いに独立して機能する。
（一方で、複数のクラスターを一元管理するための「IBM Multicloud Manager」製品が提供されている）



クラスター内部を論理的に分割するために「Namespace」機能が用いられる。

「Namespace」によって、クラスターが「漏れなくダブりなく」分割される。

「Namespace」を活用することにより、クラスター内部でのマルチテナンシーが実現する（Namespaceを用いたマルチテナンシー構成について第7章で解説する）。

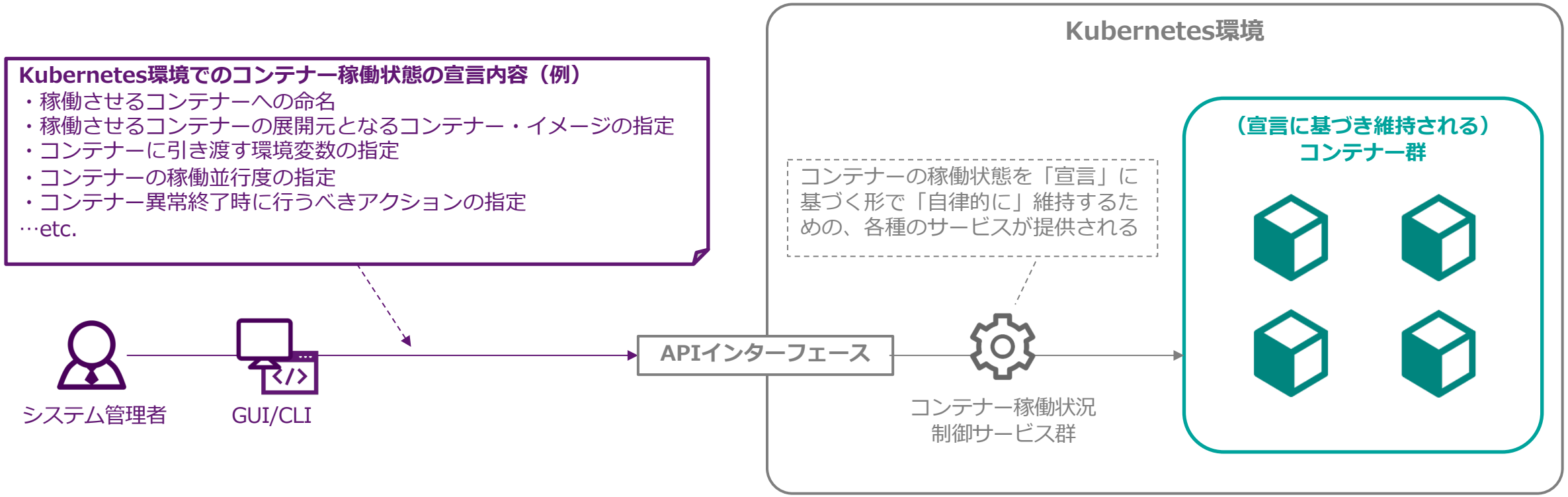
クラスター内の管理対象要素には「Label」（key-value型の識別子）を付与することが出来る。
共通する特性（用途、機能、性能…etc.）を持つ管理対象（群）をKubernetesクラスターから識別・選択するような操作が、管理対象の付与した「Label」を用いることにより実現する。

第1章 第2節:
コンテナー稼働状態の維持機能

「宣言」に基づくコンテナー稼働状態の自律的な維持

■ Kubernetes環境においては、コンテナーの「**あるべき稼働状態**」を「**宣言**」することを通じて、コンテナーの稼働状態が「**自律的に**」維持される

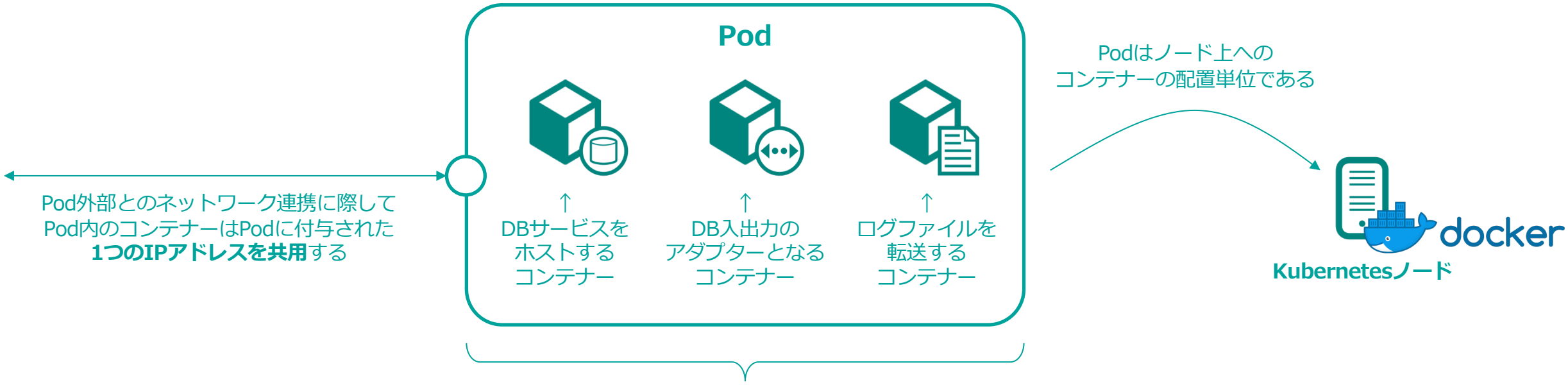
–この「宣言」を指して「**マニフェスト**」と通称される



「Pod」を単位としたコンテナーのグルーピング

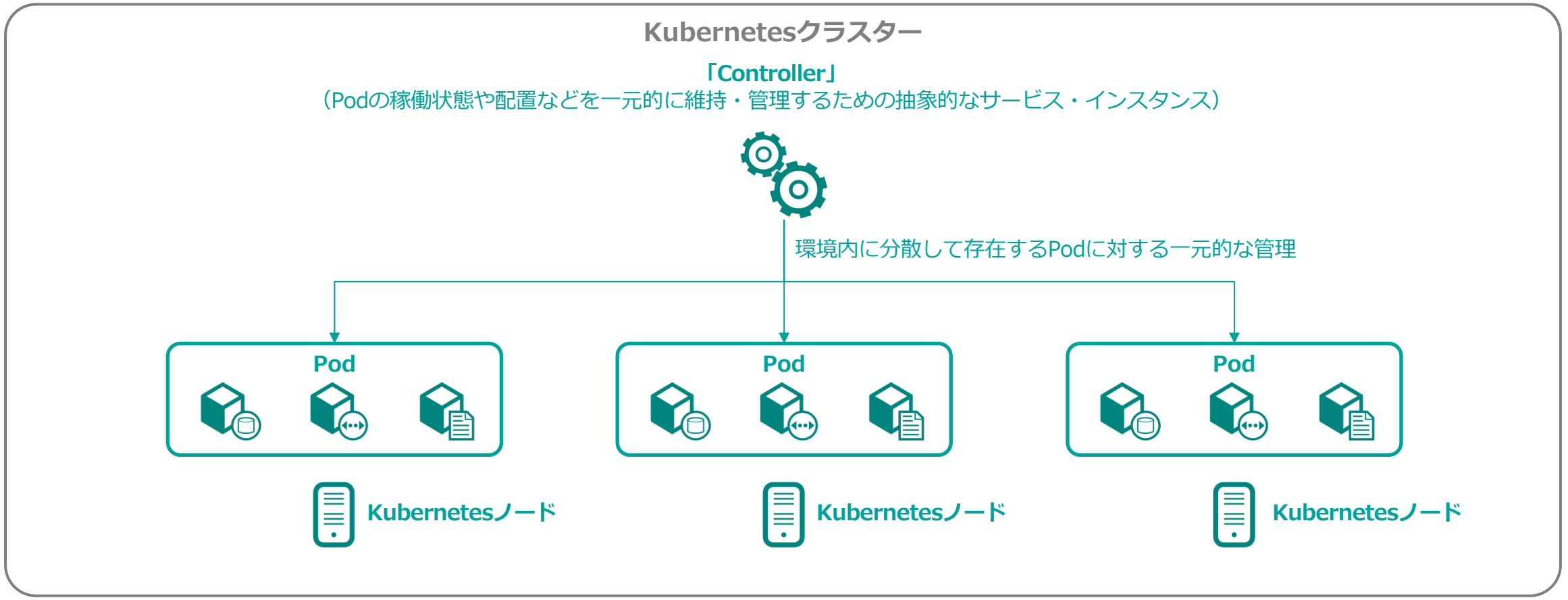
■ Kubernetesクラスターでは、複数のコンテナーをグルーピングした「Pod」がコンテナー配置（稼働）の最小単位として扱われる

- 参考情報: Pods - Kubernetes



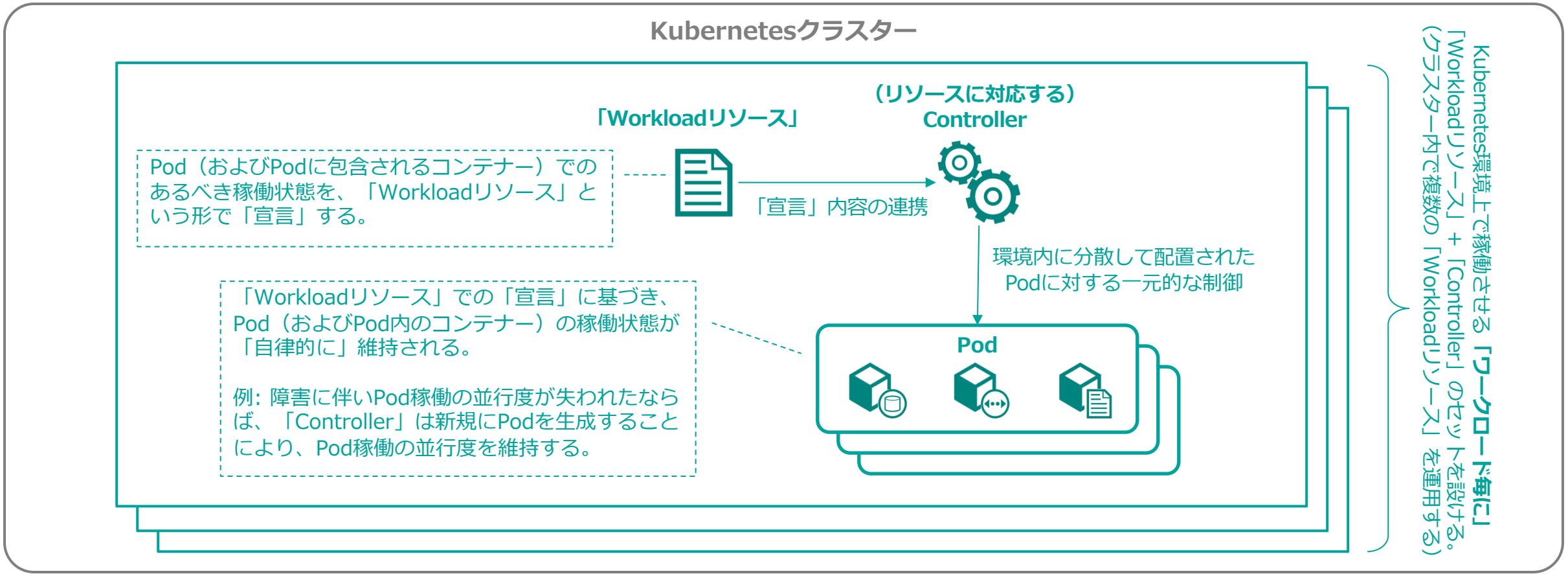
「Controller」を用いたPod稼働の制御

- Kubernetesクラスターにより、Pod（およびPodに包含されるコンテナー）の振る舞いを制御するための、各種の「Controller」が提供される
  - 参考情報: Pod Overview - Kubernetes (Pods and Controllers)



「Controller」の動作を指定するための「Workloadリソース」

- Pod（およびPodに包含されるコンテナー）の振る舞いを「Controller」を用いて制御するにあたって、各種の「Workloadリソース（※）」を用いてコンテナーのあるべき稼働状態を「宣言」する

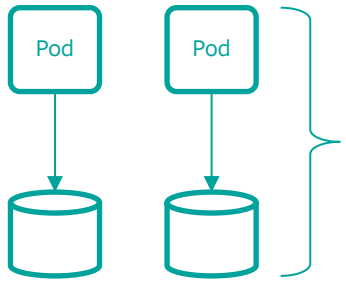
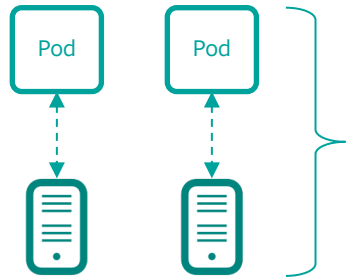


※: Pod（およびPodに包含されるコンテナー）のあるべき稼働状態を「宣言」するためのAPIリソース群を指して、本章では「Workloadリソース」と総称する

用途に応じた「Workloadリソース」種別の使い分け

■ 一般に広く利用される「Workloadリソース」の種別を下表にまとめる（※）

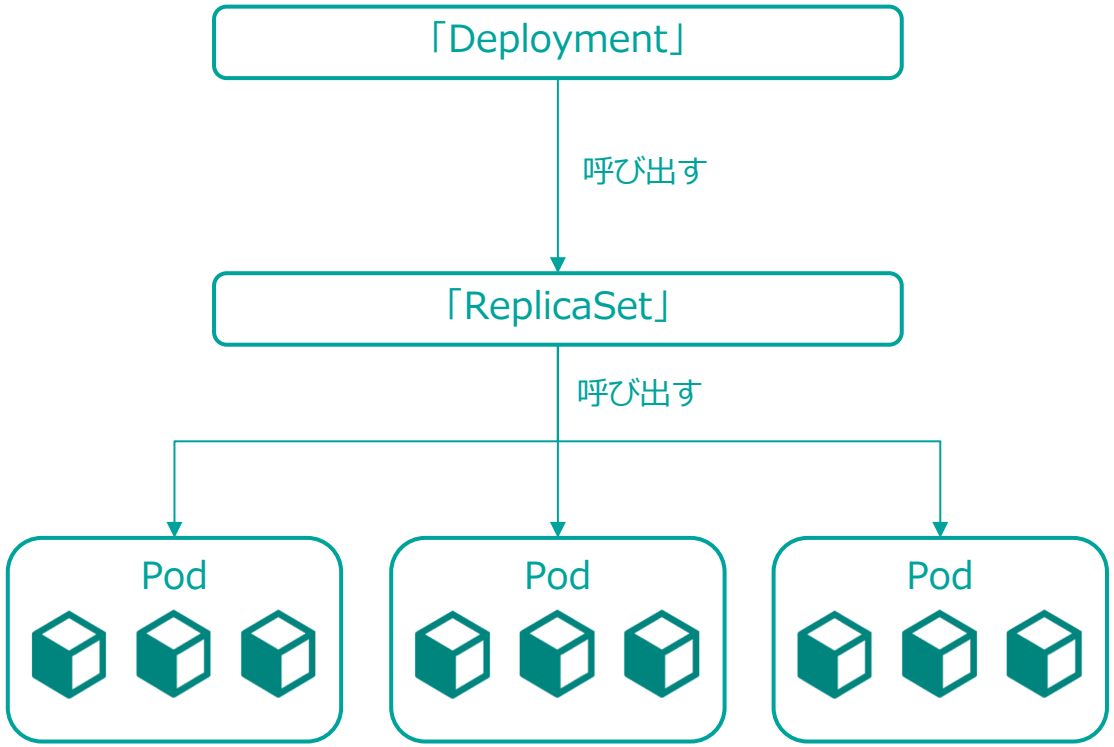
– 「Workloadリソース」種別のそれぞれが得意分野（長所）を持つことに起因して、運用するワークロードの特性に応じた「Workloadリソース」種別を使い分ける必要が生じる

Workloadリソース種別	Deployment	StatefulSet	DaemonSet	CronJob
				
得意分野（長所）	スケールアウト運用やローリングアップグレード運用への高い適正に基づく汎用性	コンテナーが扱う永続データの不整合を防止することを主眼としたコンテナーの振る舞いの制御	それぞれのノードにPodを1つずつ配置する形態でのPod配置トポロジーを簡易に実現可能	Podを用いたバッチ処理に対する簡易的なジョブスケジューリングのサポート
想定される利用用途	ステートレスなワークロード（例:Webサービス）の実行	ステートフルなワークロード（例:DBサービス）の実行	エージェント的なワークロードを各ノードに1つずつ配置する	スケジュールベースでのバッチワークロードの実行
参考情報	Deployments - Kubernetes	StatefulSets - Kubernetes	DaemonSet - Kubernetes	CronJob - Kubernetes

※: リソース種別の全量については「Overview of kubectl - Kubernetes (Resource types)」、および「Kubernetes API Reference Docs」を参照のこと
（なお、「Pod」も「Workloadリソース」の一種とみなせる）

「Workloadリソース」の階層構造（1/2）

- 「Workloadリソース」は上位のリソースが下位のリソースを束ねる形での階層構造を採る（下図）
  - 上位のリソースに対応する「Controller」が下位のリソースを呼び出す形での制御が行われる
  - コンテナーを維持・制御するための責務が「Workloadリソース」の各階層により分割される



「Workloadリソース」の階層構造を通じて、コンテナーの動作を制御するための責務が分割される（このページでは「Deployment」リソースの場合での責務分割の例を示す）。

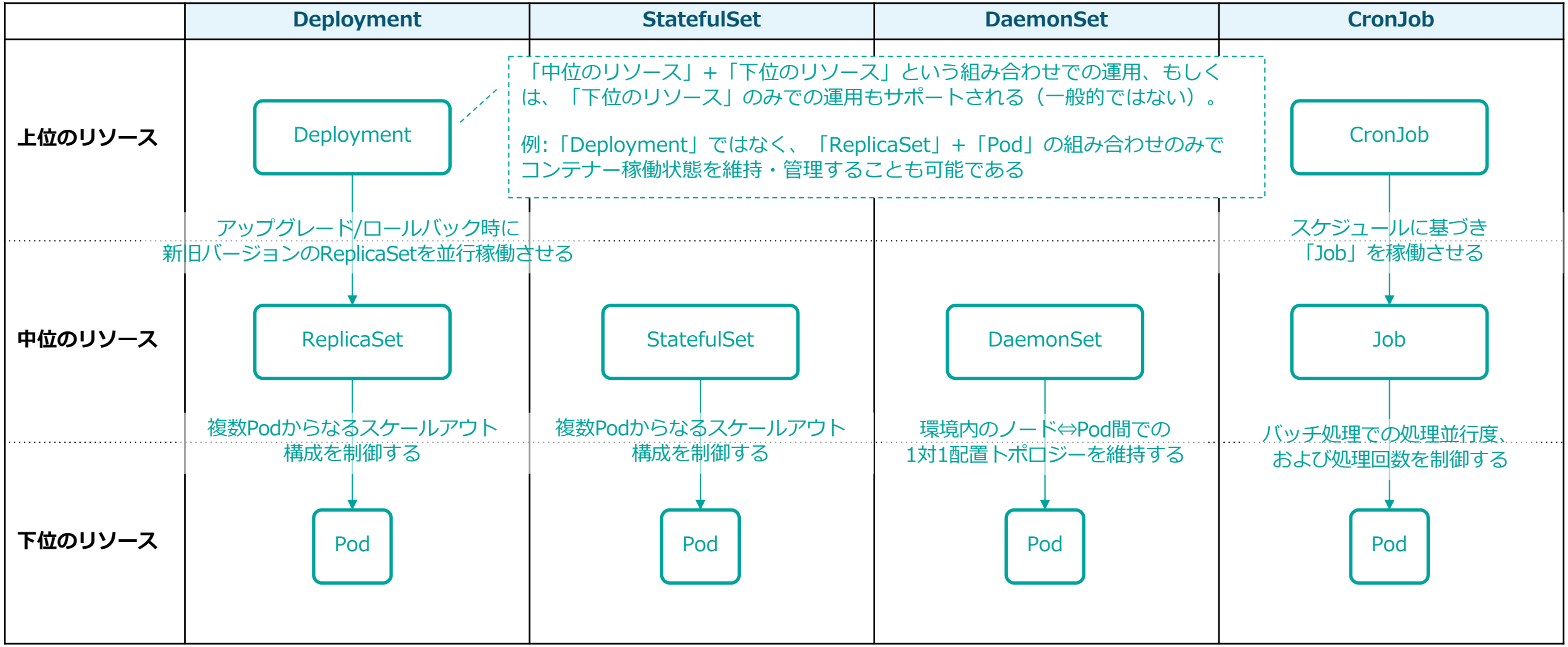
「Deployment」リソースに対応する「Controller」の主な責務:
- 新旧バージョンの「ReplicaSet」の並行稼働を制御することを通じて、円滑なローリングアップグレードやロールバックを実現すること

「ReplicaSet」リソースに対応する「Controller」の主な責務:
- 指定された並行度に基づく形でPod稼働数を維持すること

「Pod」リソースに対応する「Controller」の主な責務:
- マニフェストとして指定された「あるべき姿」に基づき、Pod内のコンテナーの振る舞いを制御すること
- Pod内のコンテナーに対するネットワーク・アクセスを提供すること

「Workloadリソース」の階層構造（2/2）

■ 主要な「Workloadリソース」での「上位」⇔「下位」関係を以下の概念図に示す



「Deployment」リソースのマニフェスト例（1/2）

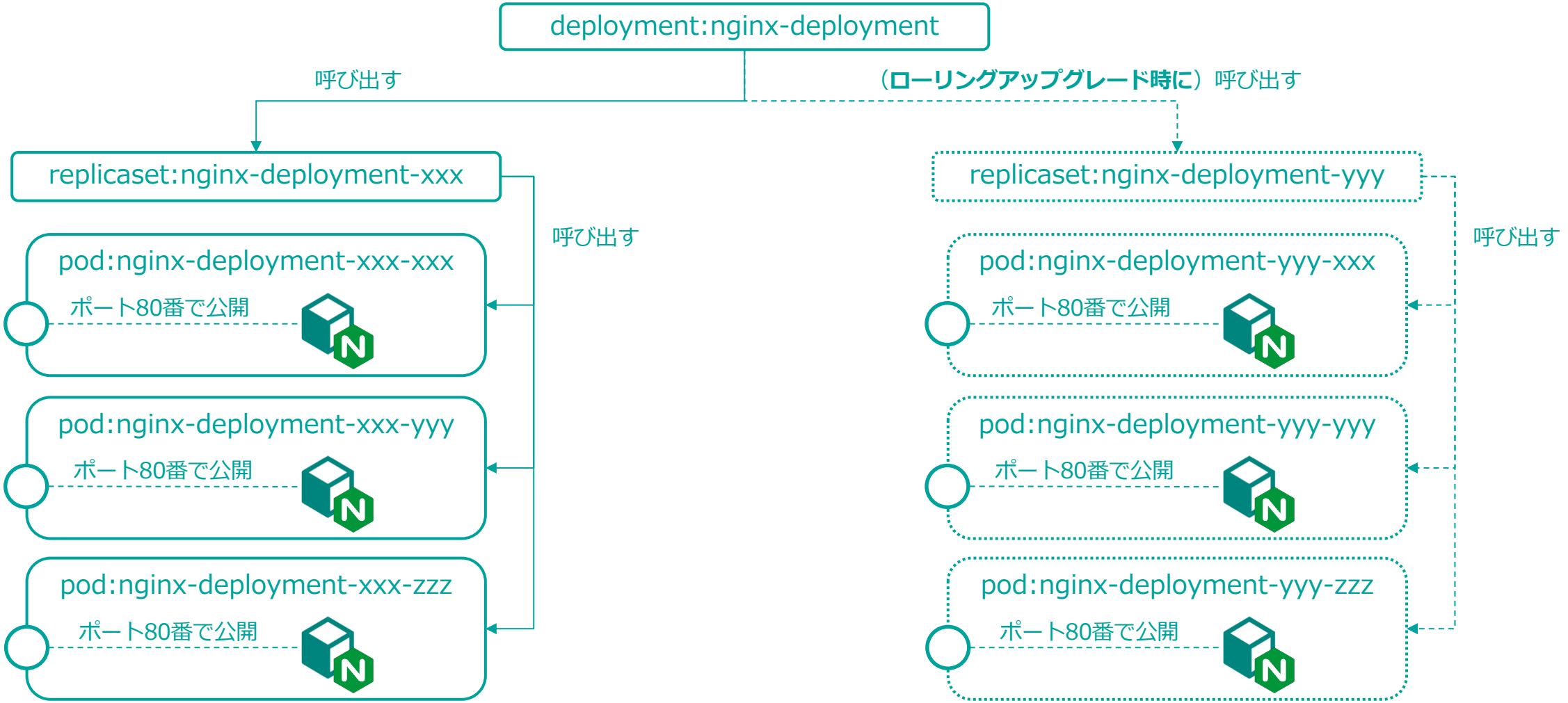
- 「Deployment」リソースのマニフェスト例を以下に示す
  - 参考情報: Deployments - Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment ← Deployment名を指定している
  labels:
    app: nginx
spec:
  replicas: 3 ← Podの稼働並列度の指定（ここでの指定内容が、Deployment Controllerが呼び出すReplicaSetリソースに反映される）
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

ReplicaSetがPodを呼び出す際に指定すべき、Pod、およびPod内のコンテナーのあるべき姿が記述されている
（例：コンテナーの雛形とすべきイメージ、コンテナーが用いるネットワーク・ポート番号…etc.）

「Deployment」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す



「StatefulSet」リソースのマニフェスト例（1/2）

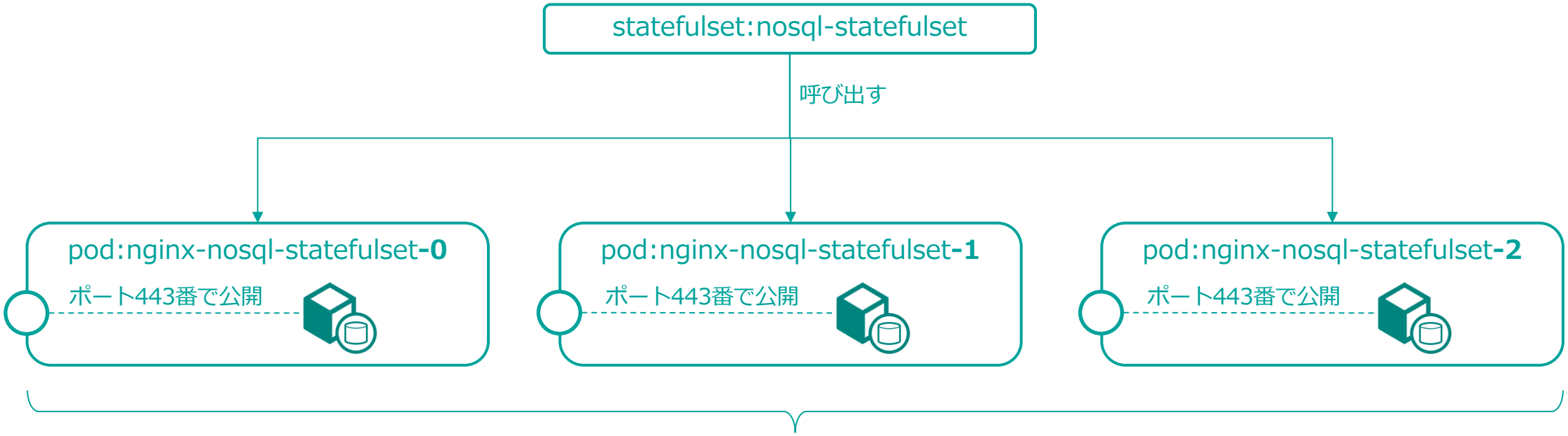
- 「StatefulSet」リソースのマニフェスト例を以下に示す
  - 参考情報: StatefulSets - Kubernetes

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nosql-statefulset ← StatefulSet名を指定している
spec:
  selector:
    matchLabels:
      app: nosql
  podManagementPolicy: OrderedReady ← 配下の複数のPodを同時に生成/削除することを認めない（既定値）
  serviceName: nosql
  replicas: 3 ← 3並列での稼働を指定している
  template:
    metadata:
      labels:
        app: nosql
    spec:
      containers:
      - name: nosql
        image: nosql:1.0
        ports:
        - containerPort: 443
```

StatefulSetがPodを呼び出す際に指定すべき、Pod、およびPod内のコンテナーのあるべき姿が記述されている
（例：コンテナーの雛形とすべきイメージ、コンテナーが用いるネットワーク・ポート番号…etc.）

「StatefulSet」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す



※:StatefulSetでは呼び出されるPodの名称が**連番**となる。
このStatefulSetをスケールアウトした場合においても、新規に追加されたPodは連番で命名されていく。
逆に、このStatefulSetがスケールインされる場合には、連番に基づき（老番から）Podが削除される。

マニフェストで「podManagementPolicy: OrderedReady」を指定したことにより、
スケールアウト/スケールインの際において**複数のPodが同時に生成/廃棄されることが禁じられる**
（StatefulSet内の複数Podにまたがる DBクラスタリングなどにおいて、**データ不整合が生じる事態を防止する**意図がある）。

「DaemonSet」リソースのマニフェスト例（1/2）

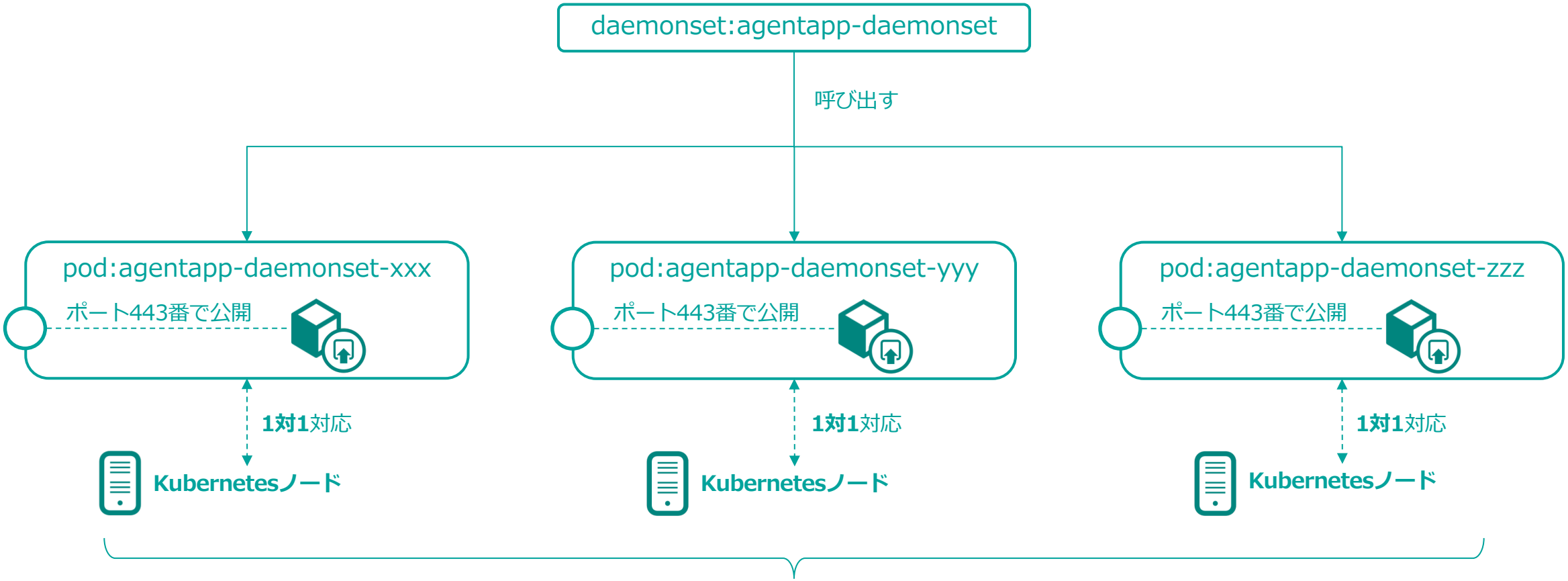
- 「DaemonSet」リソースのマニフェスト例を以下に示す
  - 参考情報: DaemonSet - Kubernetes

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: agentapp-daemonset ← DaemonSet名を指定している
spec:
  selector:
    matchLabels:
      name: agentapp
  template:
    metadata:
      labels:
        name: agentapp
    spec:
      containers:
      - name: agentapp
        image: agentapp:1.0
        ports:
        - containerPort: 443
```

DaemonSetがPodを呼び出す際に指定すべき、Pod、およびPod内のコンテナーのあるべき姿が記述されている
（例：コンテナーの雛形とすべきイメージ、コンテナーが用いるネットワーク・ポート番号…etc.）

「DaemonSet」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す



生成されるPodとKubernetesノードとの関係が必ず**1対1**になるように、DaemonSetによるPodの維持・管理が行われる

「CronJob」リソースのマニフェスト例（1/2）

- 「CronJob」リソースのマニフェスト例を以下に示す
  - 参考情報: CronJob - Kubernetes

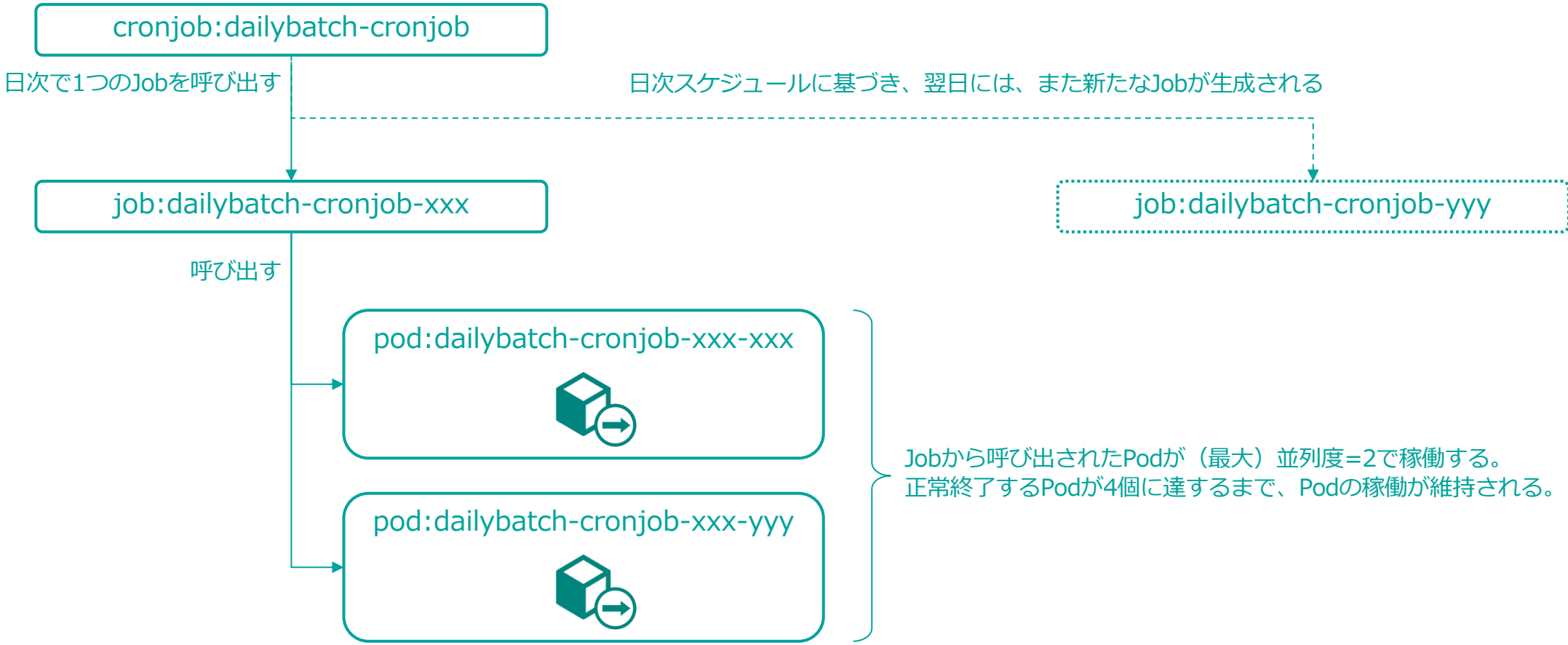
```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: dailybatch-cronjob ← CronJob名を指定している
spec:
  schedule: “@daily” ← 日次深夜でのバッチ実行を指定している
  jobTemplate:
    completions:4
    parallelism:2
    spec:
      template:
        spec:
          containers:
          - name: dailybatch
            image: dailybatch:1.0
```

CronJobがJobを呼び出す際に指定する、Jobのあるべき振る舞いが記述されている
この例では、正常終了するPodが4個に達するまで2並列で（バッチ処理を行う）Podを稼働させることを、Jobに対して指定している

CronJobにより生成されたJobがPodを呼び出す際に指定する、
PodやPodに含まれるコンテナーのあるべき姿を記述している

「CronJob」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す



第1章 第3節:
ネットワーク・アクセス維持・制御機能

ネットワーク・アクセスの維持・制御を実現するための各種機能

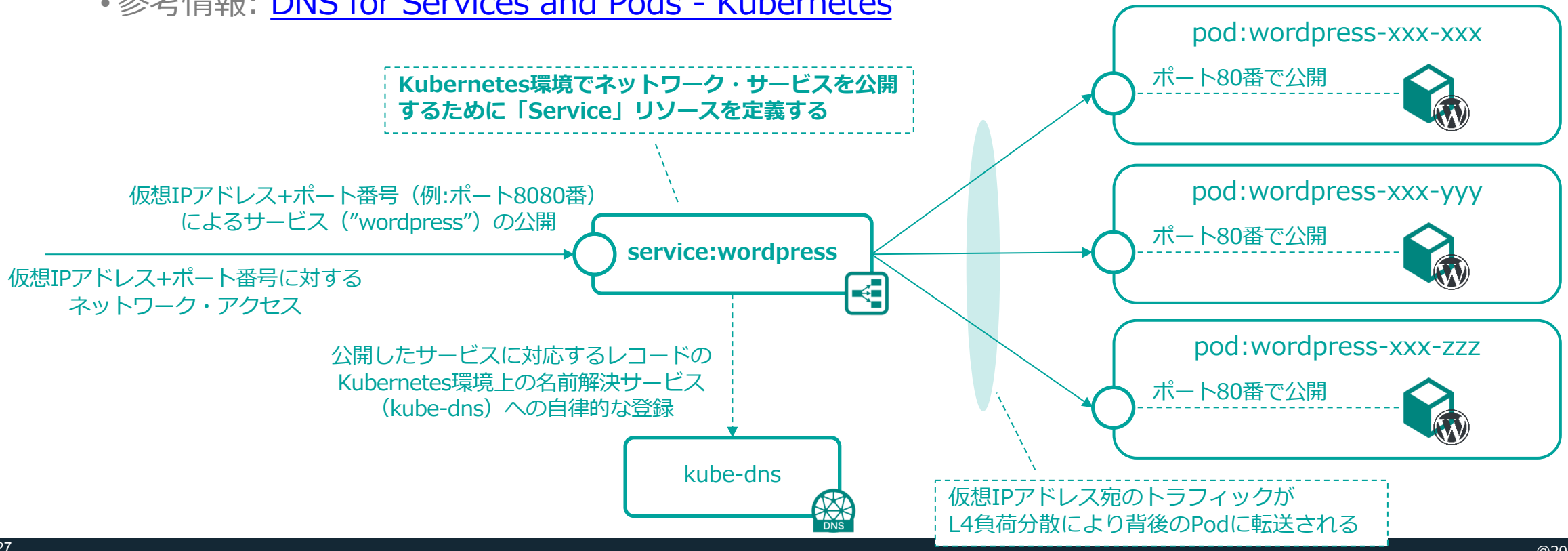
■ Kubernetes環境においては、下表に示す5種類の機能が中核となる形で、環境内で稼働するコンテナーへのネットワーク・アクセスが維持・制御される

–機能を活用するにあたって明示的な設定作業を要するという観点では、L3/L4負荷分散機能、および、L7負荷分散機能での検討上の重要度がより高い

機能項目	機能の概略	Kubernetes環境の構成・活用検討上での重要度
PodへのIPアドレスの付与	• Kubernetes環境内で稼働するPodにIPアドレスを付与する	**重要度: 低** Podに付与するためのアドレス・レンジを指定することにより、指定されたレンジからIPアドレスが自律的に払い出される。
L2/L3スイッチング機能	• Kubernetes環境内でのPod間通信、もしくは、Kubernetes環境内外の相互通信に対するL2/L3スイッチング機能（それに加えてL3 NAT機能）を提供する	**重要度: 低** ICP製品のセットアップ時に構成されるネットワーク・プラグインにより、L2/L3スイッチングが自律的に行われる。
L4負荷分散機能	• Kubernetes環境内で稼働するネットワーク・サービスに対するL4ベースでのネットワーク負荷分散機能を提供する	**重要度: 高** Kubernetes環境内で稼働するネットワーク・サービスを公開するにあたって、L4負荷分散機能利用のための設定を明示的に行われなければならない。
L7負荷分散機能	• Kubernetes環境内で稼働するWebサービスに対するL7ベースでのネットワーク負荷分散機能を提供する	**重要度: 高** Kubernetes環境内で稼働するネットワーク・サービスをL7負荷分散の背後で公開するにあたって、機能利用のための設定を明示的に行われなければならない。
名前解決機能	• Kubernetes環境内での、ネットワーク・アドレス（例:IPアドレス）に関連する各種の名前解決を実現する	**重要度: 中** 既定でDNS相当の名前解決機能が提供される。Kubernetes環境内で稼働するネットワーク・サービスをL4負荷分散機能の背後で公開することを通じて、当該のネットワーク・サービスに対応する名前解決レコードが自律的に登録される。

Kubernetes環境上でのネットワーク・サービスの公開

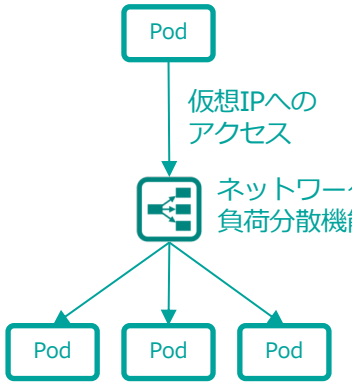
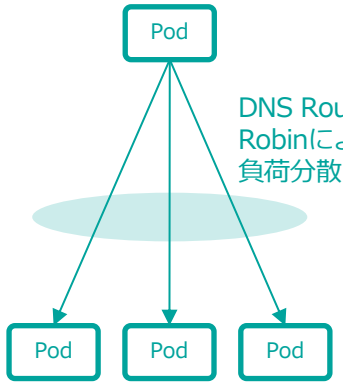
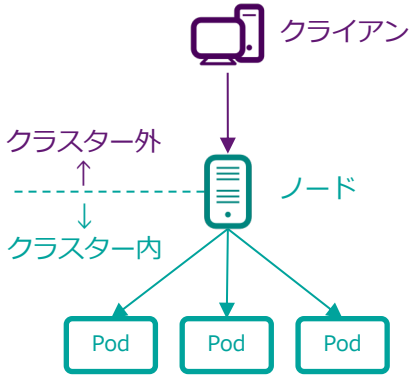
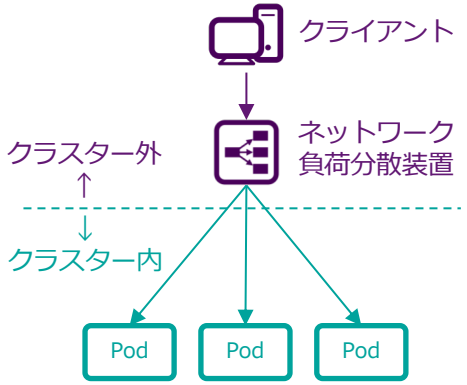
- Kubernetes環境でネットワーク・サービスを公開するために「Service」リソースを作成することにより、以下の2機能を利用するためのシステム構成が行われる
  - L4負荷分散機能（Pod宛トラフィックのネットワーク負荷分散機能）
  - 名前解決機能（DNS相当の名前解決機能）
    - 参考情報: Service - Kubernetes
    - 参考情報: DNS for Services and Pods - Kubernetes



「Service」リソース構成の主要なバリエーション

■ 「Service」リソース構成の主要なバリエーション（※1）を下表に示す

- 参考情報: Service - Kubernetes (Publishing Services)

	ClusterIP	ClusterIP (Headless)	NodePort	LoadBalancer（※2）
構成上のバリエーション				
構成の概要	Kubernetesクラスター内のみからアクセス可能な仮想IPアドレス（ClusterIP）を、クライアントからのアクセス先IPアドレスとして用いる	仮想IPアドレス方式でのネットワーク負荷分散ではなく、DNSラウンドロビン方式でのネットワーク負荷分散を行う	ノードのIPアドレスを、クライアントからのアクセス先IPアドレスとして用いる。ノードへの通信はKubernetesクラスター内のPodに転送される	Kubernetesクラスター外部の負荷分散装置が提供する仮想IPアドレスを、クライアントからのアクセス先IPアドレスとして用いる。
主な利用用途	Kubernetesクラスター内部へのステートレスなネットワーク・サービス（例:Webサービス）の公開	ステートフルなネットワーク・サービス（例:DBサービス）の公開（※3）	Kubernetesクラスター外部へのネットワーク・サービスの公開	パブリック・クラウドから調達したKubernetes環境での、Kubernetesクラスター外部へのネットワーク・サービス公開

※1: 表内では解説されない他のバリエーション種別として「Type ExternalName」や「Services without selectors」などが挙げられる。
※2: Kubernetesクラスターと連携可能なネットワーク負荷分散装置を調達する際の困難に起因して、ICP製品を用いた実装での同バリエーションの利用は一般的ではない。
※3: 「Headless」ServiceとStatefulSetと組み合わせることにより、ステートフルなサービス公開向けの名前解決を実現することが可能（構成例を後述する）。

「ClusterIP」構成の「Service」リソースのマニフェスト例（1/2）

■ 「ClusterIP」構成を採る「Service」リソースのマニフェスト例を以下に示す

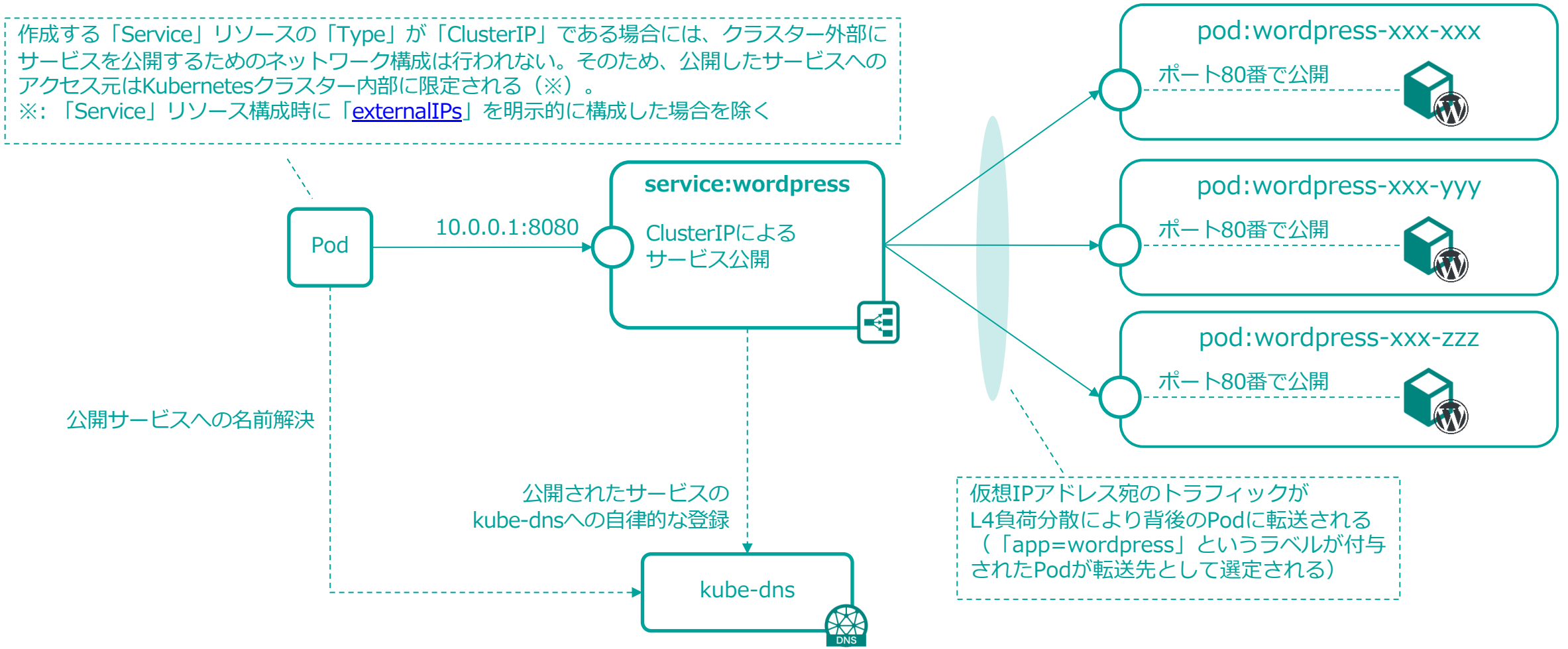
- 参考情報: Service - Kubernetes (Publishing Services)

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
spec:
  type: ClusterIP ← Serviceの「Type」として「ClsuterIP」を選択する（既定の「Type」が「ClusterIP」であることにより、省略可能）
  clusterIP: 10.0.0.1 ← クラスター内部からのサービスへのアクセス先となる仮想IPアドレスを指定する（省略時には自動採番される）
  selector:
    app: wordpress ← Podに付与されたLabel（key-valueの組み合わせ）に基づき、負荷分散の対象とするPodを識別する
  ports:
  - protocol: TCP    ┐  ネットワークアクセスに際して用いるネットワーク・ポート番号を指定する。
    port: 8080       }  「port」として、クラスター内部からのサービスのアクセス先としてのポート番号を指定する。
    targetPort: 80   ┘  「targetPort」として、ネットワーク負荷分散先で公開されるサービスのポート番号を指定する。
```

「ClusterIP」構成の「Service」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す

- 参考情報: Service - Kubernetes (Publishing Services)



「NodePort」構成の「Service」リソースのマニフェスト例（1/2）

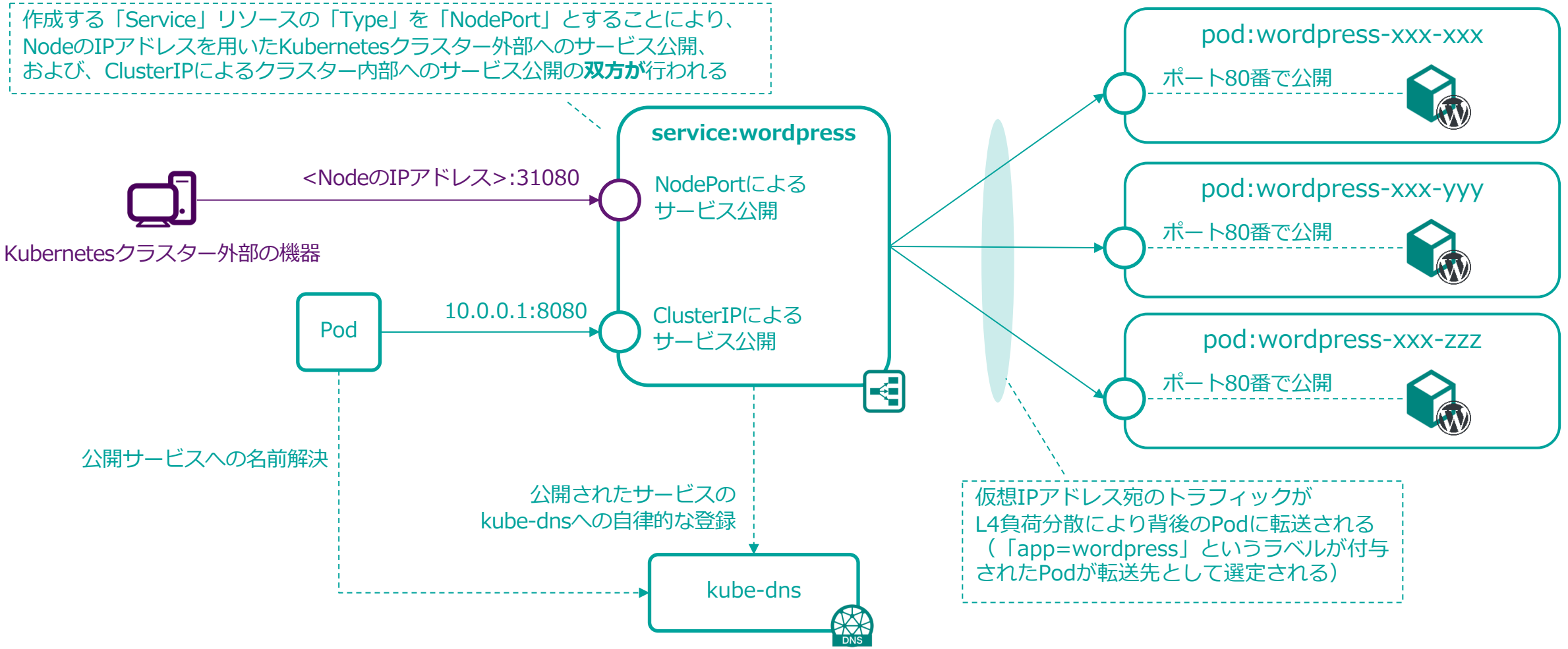
- 「NodePort」構成を採る「Service」リソースのマニフェスト例を以下に示す
  - 参考情報: Service - Kubernetes (Type NodePort)

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
spec:
  type: NodePort ← Serviceの「Type」として「NodePort」を選択することにより、「NodePort」構成を実現する
  selector:
    app: wordpress ← Podに付与されたLabel（key-valueの組み合わせ）に基づき、負荷分散の対象とするPodを識別する
  ports:
  - protocol: TCP      ネットワークアクセスに際して用いるネットワーク・ポート番号を指定する。
    port: 8080         「port」として、クラスター内部からのサービスのアクセス先としてのポート番号を指定する。
    targetPort: 80     「targetPort」として、ネットワーク負荷分散先で公開されるサービスのポート番号を指定する。
    nodePort: 31080    「nodePort」として、クラスター外部に対してノード上でサービスを公開する際に用いるポート番号を指定する。
```

「NodePort」構成の「Service」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す

- 参考情報: Service - Kubernetes (Type NodePort)



「Headless」構成の「Service」リソースのマニフェスト例（1/2）

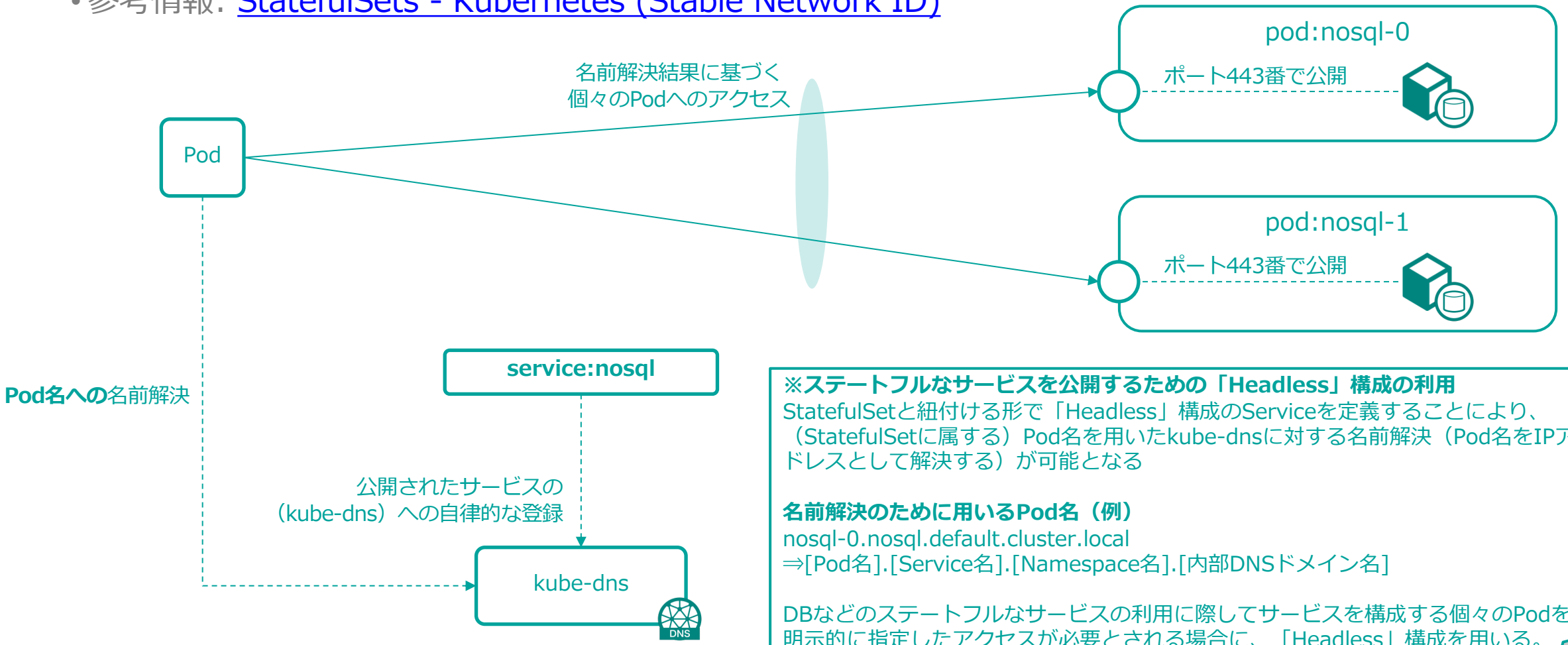
- ステートフルなサービスの公開を主眼として「Headless」構成を採る場合における「Service」リソースのマニフェスト例を以下に示す
  - 参考情報: Service - Kubernetes (Headless Services)
  - 参考情報: StatefulSets - Kubernetes (Stable Network ID)

```
apiVersion: v1
kind: Service
metadata:
  name: nosql ← Serviceの名称を、そのServiceに紐付けるStatefulSetでの「spec.serviceName」名称に揃える
spec:
  type: ClusterIP ← Serviceの「Type」として「ClsuterIP」を選択する（「ClusterIP」が既定の「Type」であることにより、省略可能）
  clusterIP: None ← 「clusterIP」を「None」とすることにより、「ClusterIP（Headless）」構成が実現する
  selector:
    app: nosql ← Podに付与されたLabel（key-valueの組み合わせ）に基づき、負荷分散の対象とするPodを識別する
  ports:
  - port: 443
```

「Headless」構成の「Service」リソースのマニフェスト例（2/2）

■ 前ページのマニフェストに基づき生成されるリソースの概念図を以下に示す

- 参考情報: Service - Kubernetes (Headless Services)
- 参考情報: StatefulSets - Kubernetes (Stable Network ID)



※ステートフルなサービスを公開するための「Headless」構成の利用
StatefulSetと紐付ける形で「Headless」構成のServiceを定義することにより、（StatefulSetに属する）Pod名を用いたkube-dnsに対する名前解決（Pod名をIPアドレスとして解決する）が可能となる

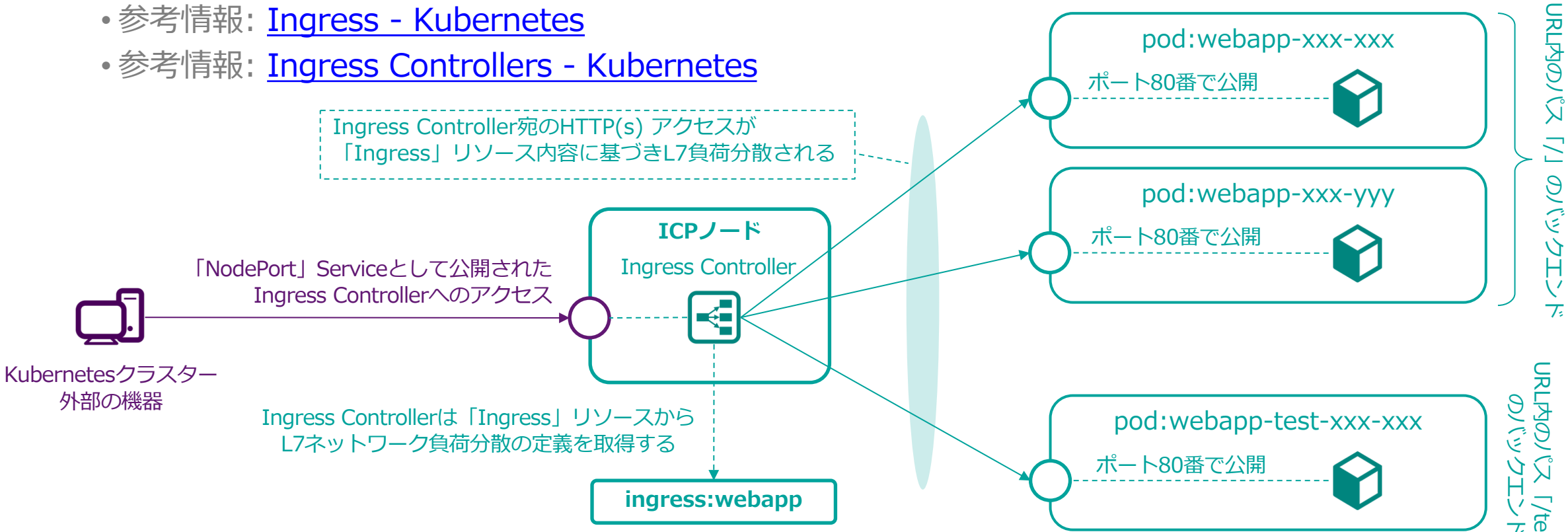
名前解決のために用いるPod名（例）
nosql-0.nosql.default.cluster.local
⇒[Pod名].[Service名].[Namespace名].[内部DNSドメイン名]

DBなどのステートフルなサービスの利用に際してサービスを構成する個々のPodを明示的に指定したアクセスが必要とされる場合に、「Headless」構成を用いる。

「Ingress Controller」を用いたL7負荷分散（※1）

■ L7負荷分散機能を備えた「Ingress Controller」をKubernetes環境上で運用することにより、公開するサービスに対するL7負荷分散、およびクラスター外部からのサービスへのHTTP(s)アクセス経路を実装することが出来る（※2）

- 参考情報: Ingress - Kubernetes
- 参考情報: Ingress Controllers - Kubernetes



※1: ICP Kubernetes環境における「Ingress Controller」実装に基づく解説を行う（Ingress Controllerの構成や動作は実装により若干のバリエーションが生じる）
※2: 「Ingress Controller」の利用（例:「NodePort」構成のServiceリソースとの使い分け）について、本ガイドの3.2節でより具体的に解説する

「Ingress」リソースのマニフェスト例

- 「Ingress」リソースによるL7負荷分散定義のマニフェスト例を以下に示す
  - 参考情報: Ingress - Kubernetes (Simple fanout)

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: webapp
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: webapp
          servicePort: 80
      - path: /test
        backend:
          serviceName: webapp-test
          servicePort: 80
```

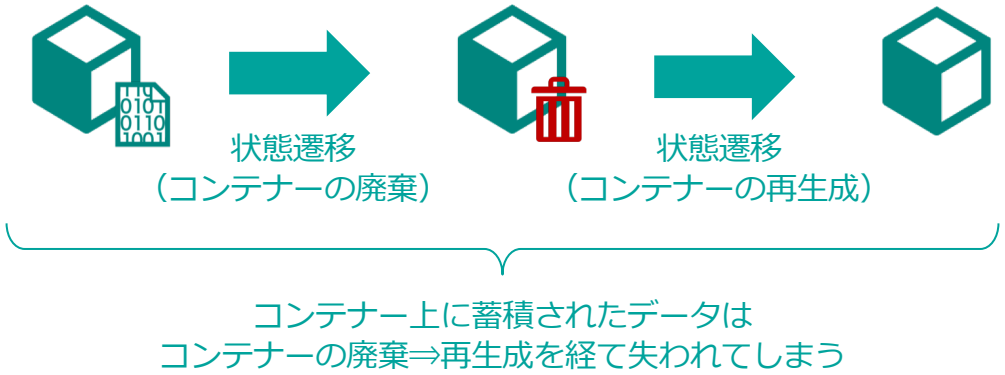
HTTP(s)アクセスに含まれるパスに基づくL7負荷分散定義を記述する
この例では、
・パス「/」へのHTTPアクセスを「webapp」Serviceに属するPodに転送する
・パス「/test」へのHTTPアクセスを「webapp-test」Serviceに属するPodに転送する

第1章 第4節:
永続ストレージ、および環境変数の提供機能

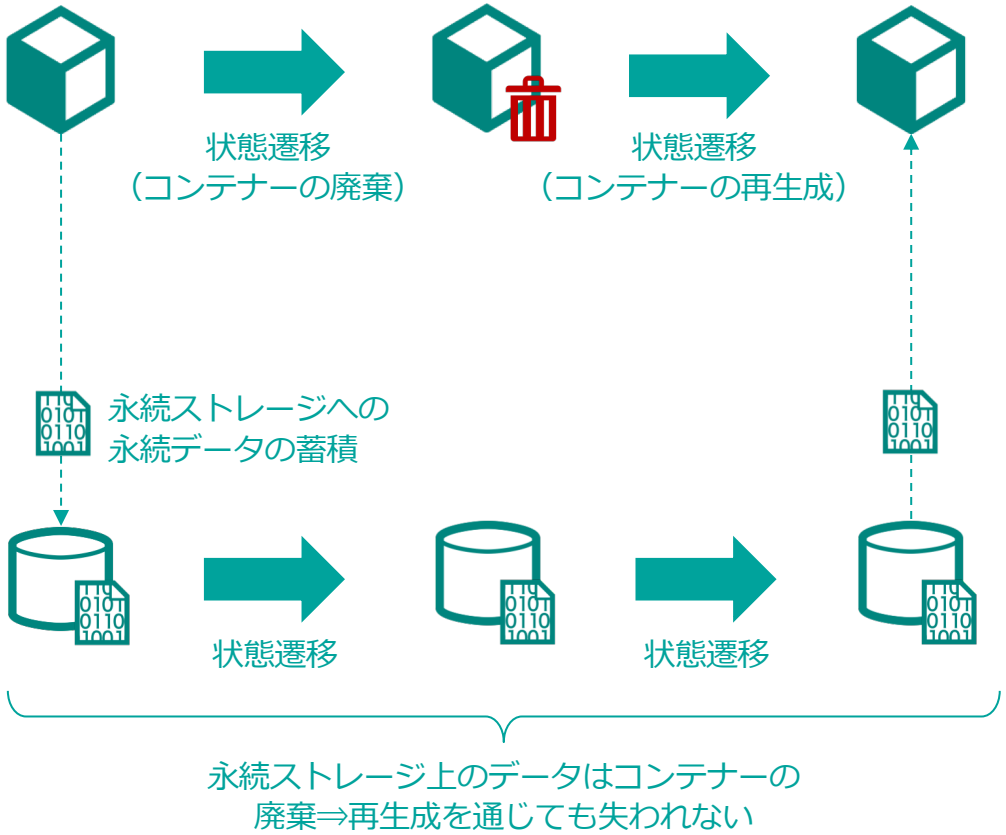
コンテナーへの永続ストレージ機能の提供が求められる背景

■ 廃棄を前提として運用するコンテナーによる永続データ利用を実現するために、Kubernetes環境により永続ストレージ機能が提供される

永続ストレージが利用出来ない場合



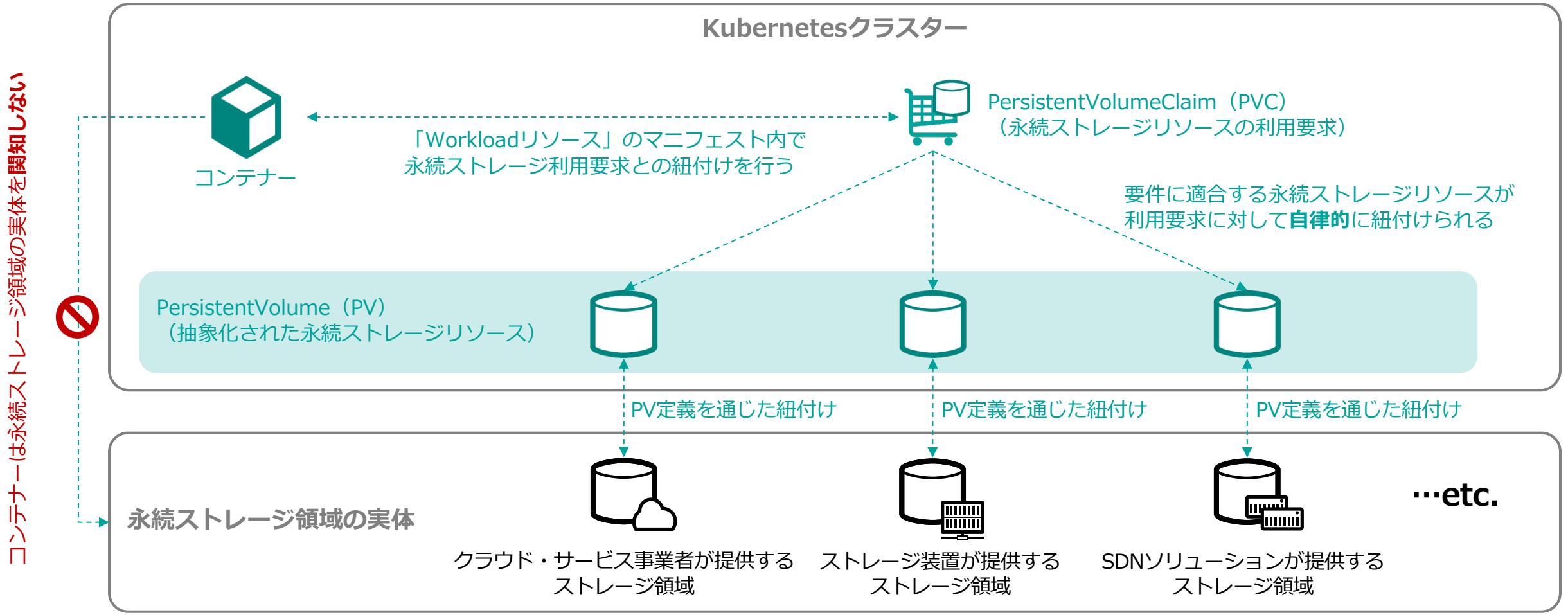
永続ストレージが利用出来る場合



Kubernetes環境における永続ストレージの提供

■ 永続データを保管するためのストレージ領域をリソースとして抽象化した上で、抽象化されたストレージ領域をコンテナーに提供する機能が提供される

- 参考情報: Persistent Volumes - Kubernetes



永続ストレージ利用構成のマニフェスト例（1/4）

■ 永続ストレージを利用するためのマニフェスト例を以下に示す

- 参考情報: Persistent Volumes - Kubernetes

```
apiVersion: v1
kind: PersistentVolume ← 永続ストレージを「PersistentVolume」リソースとして抽象化する
metadata:
  name: mypv
  labels:
    speed: slow
    environment: test
                        } ← 抽象化した永続ストレージの特性をリソースに付与するラベルとして表現する
spec:
  capacity:
    storage: 8Gi ← 抽象化した永続ストレージの容量を定義する
  accessModes:
    - ReadWriteOnce ← 抽象化した永続ストレージへの同時アクセスの可否を定義する
  persistentVolumeReclaimPolicy: Retain
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

永続ストレージの実体の利用設定を行う
（この例の場合、NFSを永続ストレージの実体として利用している）

永続ストレージ利用構成のマニフェスト例（2/4）

■ 永続ストレージを利用するためのマニフェスト例を以下に示す

- 参考情報: Persistent Volumes - Kubernetes

```
apiVersion: v1
kind: PersistentVolumeClaim ← 永続ストレージの利用要求を「PersistentVolumeClaim」リソースとして抽象化する
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  selector:
    matchLabels:
      environment: test
      speed: slow
```

要求する永続ストレージに求める仕様を記述する
（仕様に合致する永続ストレージが、この永続ストレージ要求に対して自律的に紐付けられる）

永続ストレージ利用構成のマニフェスト例（3/4）

■ 永続ストレージを利用するためのマニフェスト例を以下に示す

- 参考情報: Persistent Volumes - Kubernetes

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mycontainer
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: myvolume
  volumes:
    - name: myvolume
      persistentVolumeClaim:
        claimName: myclaim
```

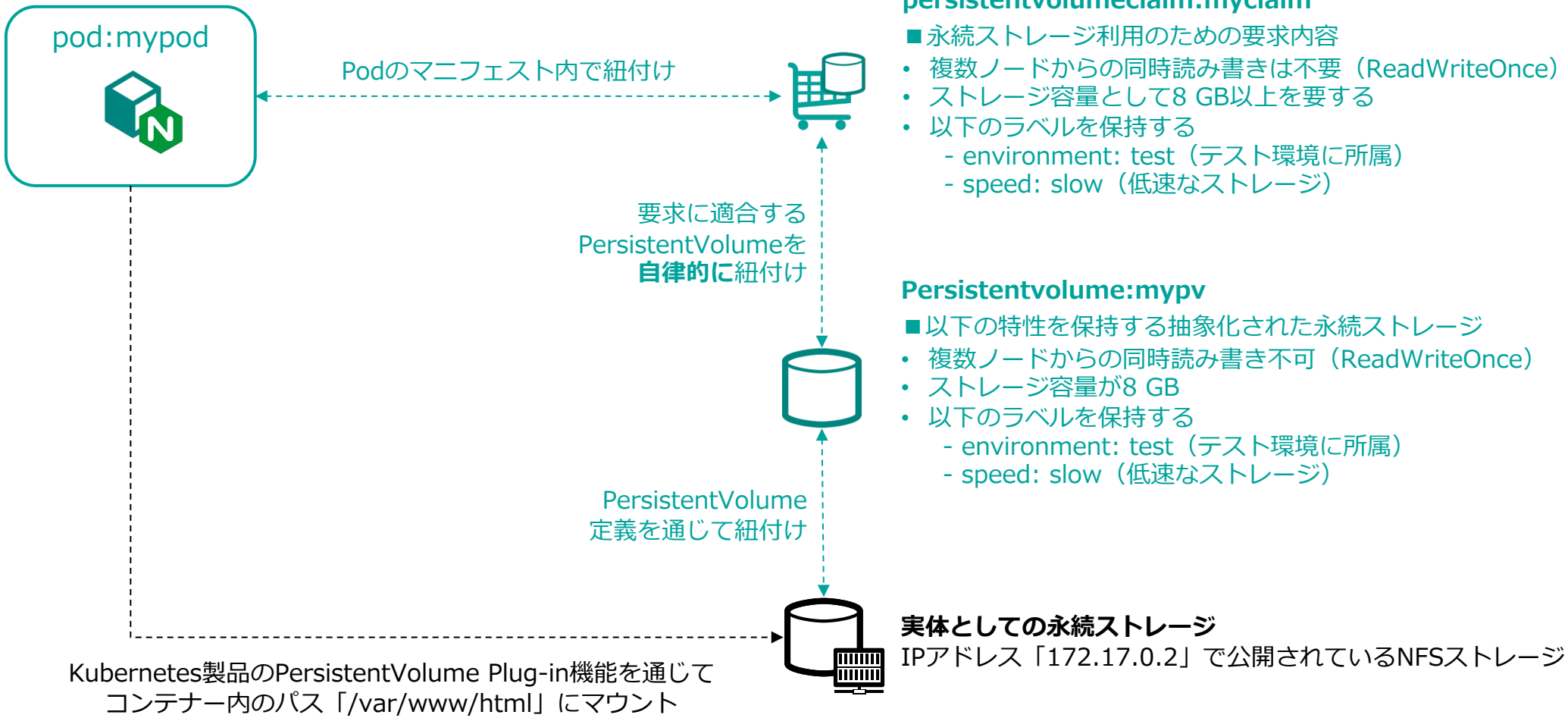
要求した永続ストレージのボリュームをコンテナーから利用出来るファイルシステム上のパスとしてマウントする

← 永続ストレージを要求することを宣言する

永続ストレージ利用構成のマニフェスト例（4/4）

■ ここまでのマニフェストに基づく永続ストレージ利用の概念図を以下に示す

- 参考情報: Persistent Volumes - Kubernetes

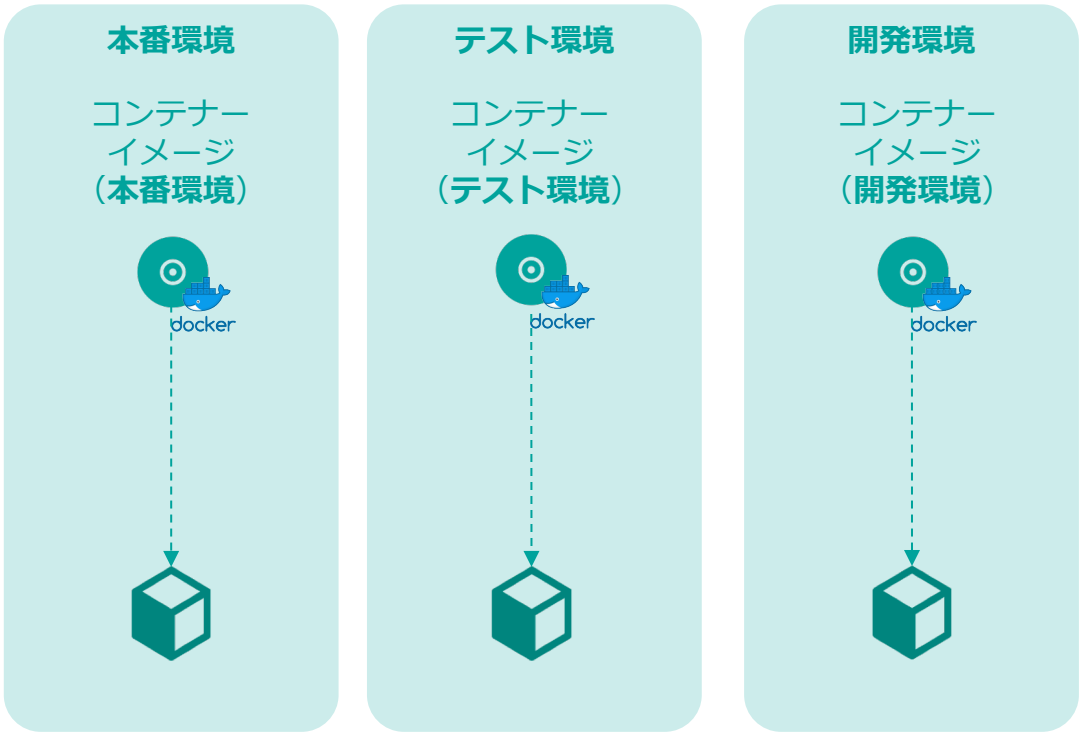


コンテナーへの環境変数の注入が求められる背景

■ 環境により変化する各種の設定値（環境変数）をコンテナー外部から注入する運用を通じて、管理すべきコンテナー・イメージ数を抑制することが出来る

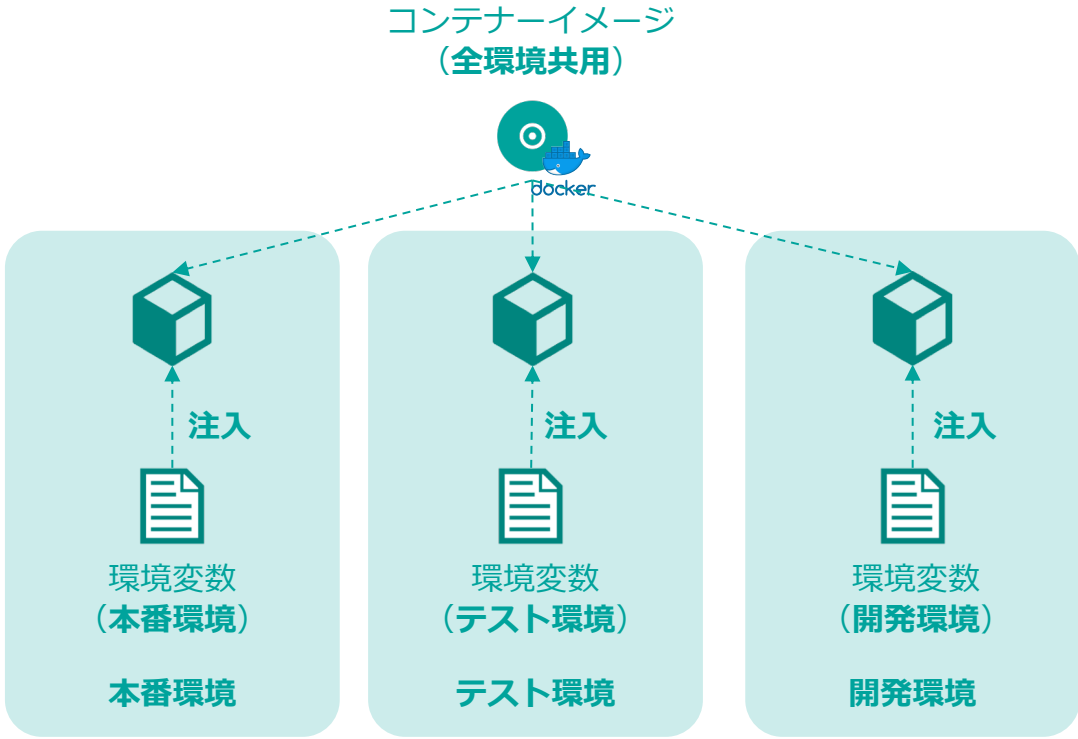
環境変数の注入が利用出来ない場合

環境毎の設定値をコンテナー・イメージに含める必要が生じるため、
環境毎にコンテナー・イメージを準備しなければならない



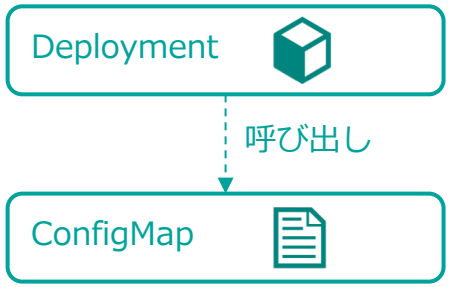
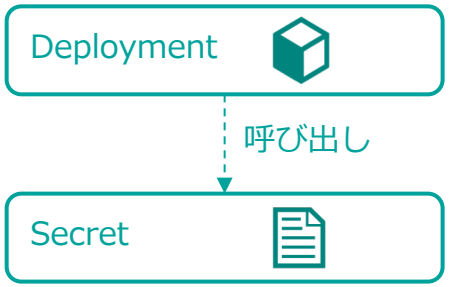
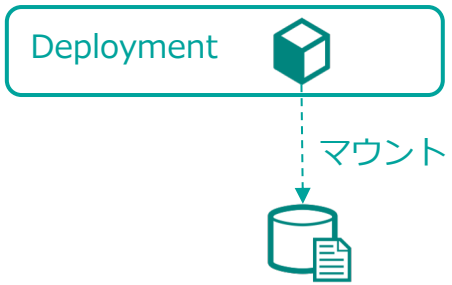
環境変数の注入が利用出来る場合

環境毎の設定値を展開されたコンテナーに注入することにより、
単一のコンテナー・イメージを環境間で共用することが可能となる



Kubernetes環境におけるコンテナーへの環境変数の注入手段

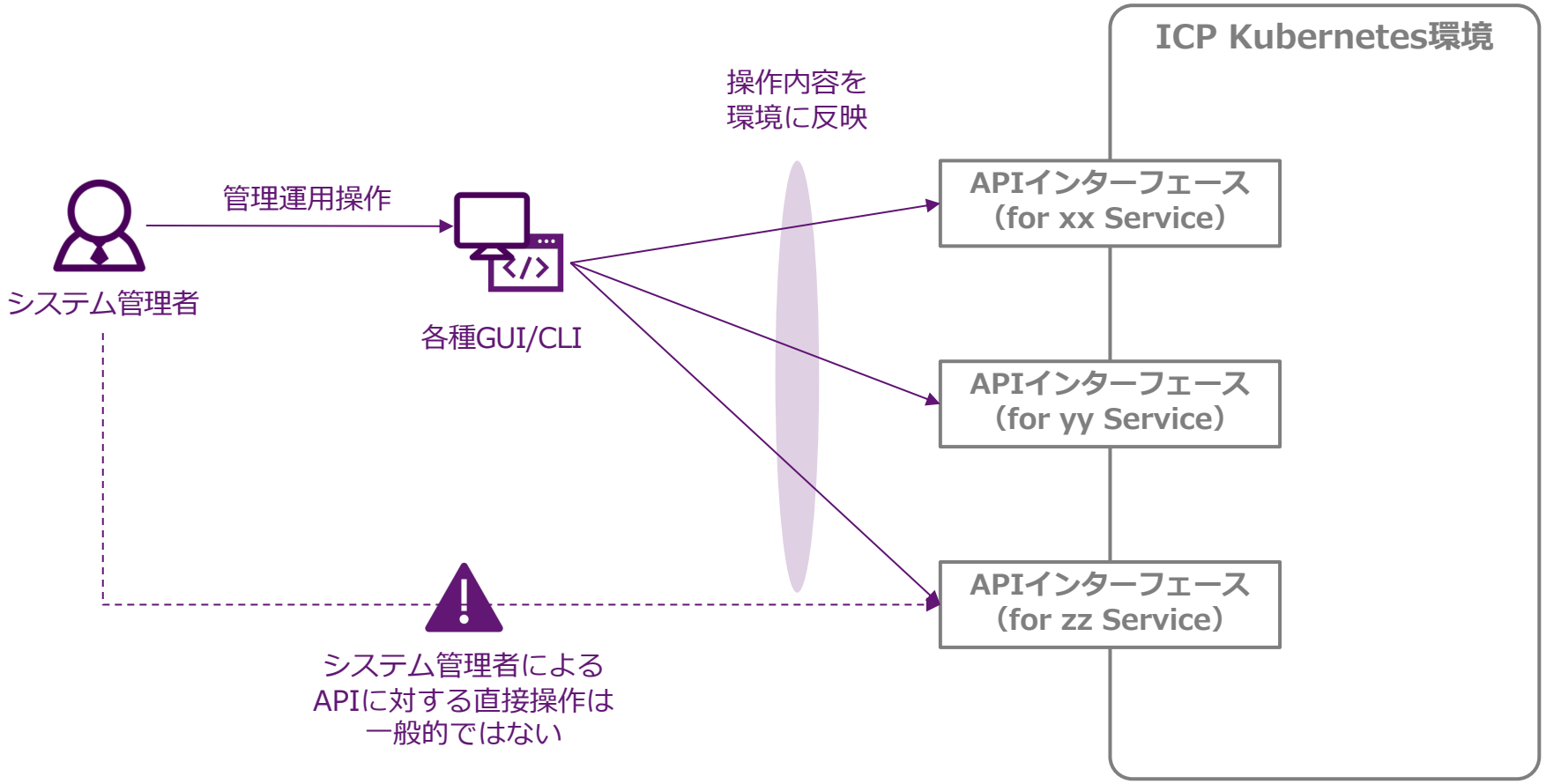
■ Kubernetes環境においては、環境変数を定義・保持・注入するために複数の手段を使い分けられる（下表）

	ConfigMapリソースの利用	Secretリソースの利用	静的な定義	ファイル渡し
注入手段				
構成の概要	「ConfigMap」リソースとして注入する環境変数を定義した上で、「Workloadリソース」のマニフェストから「ConfigMap」リソースを呼び出す。	「Secret」リソースとして注入する環境変数を定義した上で、「Workloadリソース」のマニフェストから「Secret」リソースを呼び出す。	注入する環境変数を「Workloadリソース」のマニフェストに直接記述する。	注入する環境変数をファイル内に保管した上で、当該のファイルを格納するボリュームをコンテナーにマウントする。
セキュリティ水準	ConfigMapリソースに対するRBAC運用が可能。ただし、環境変数が平文で保管される。	Secretリソースに対するRBAC運用が可能。また、環境変数の難読化（Base64）が行われる。	「Workloadリソース」に対するRBAC運用が可能。ただし、環境変数が平文で保管される。	作りこみの内容に依存する。
主な利用用途	大量の環境変数のコンテナーへの注入。例えば、ミドルウェア構成値のコンテナーへの注入。	機密性が高い環境変数のコンテナーへの注入。例えば、認証情報のコンテナーへの注入。	少数の（数えられる程度の）環境変数のコンテナーへの注入。	現行での環境変数運用との整合確保。例えば、ミドルウェア環境変数を設定ファイルとして管理している現行運用を引き継ぐ。
参考情報	Configure a Pod to Use a ConfigMap - Kubernetes	Secrets - Kubernetes		

第1章 第5節:
管理インターフェース

ICP Kubernetes環境でのユーザーインターフェースの概略

- ICP Kubernetes環境では（通常は）GUI/CLIを用いた運用操作を行う
  - GUI/CLIを代替・補完するAPIも提供されているものの、広く用いられるものではない
  - ICP製品が提供するCLIについての解説は4.10節で行う



ICP製品が提供するGUIの実例（1/5）

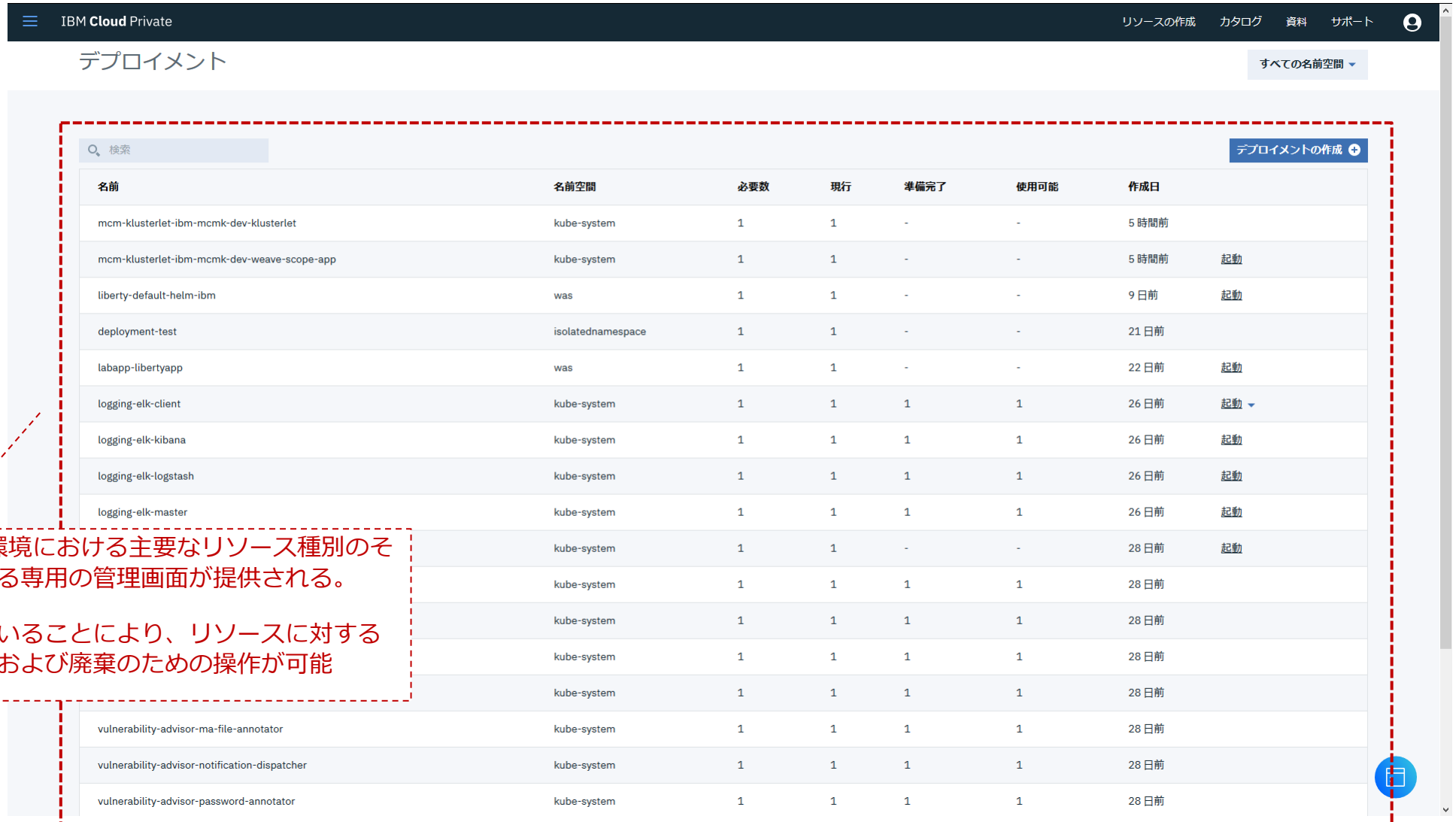
ユーザー認証機能、および、認証結果に基づく
RBACベースでの認可機能が提供される

Webブラウザベースでの管理画面（管理画面の実体は
ICP Kubernetes環境上で稼働するWebサービス）

ICP製品が提供するGUIの実例（2/5）

GUI管理画面のメニュー

ICP製品が提供するGUIの実例（3/5）

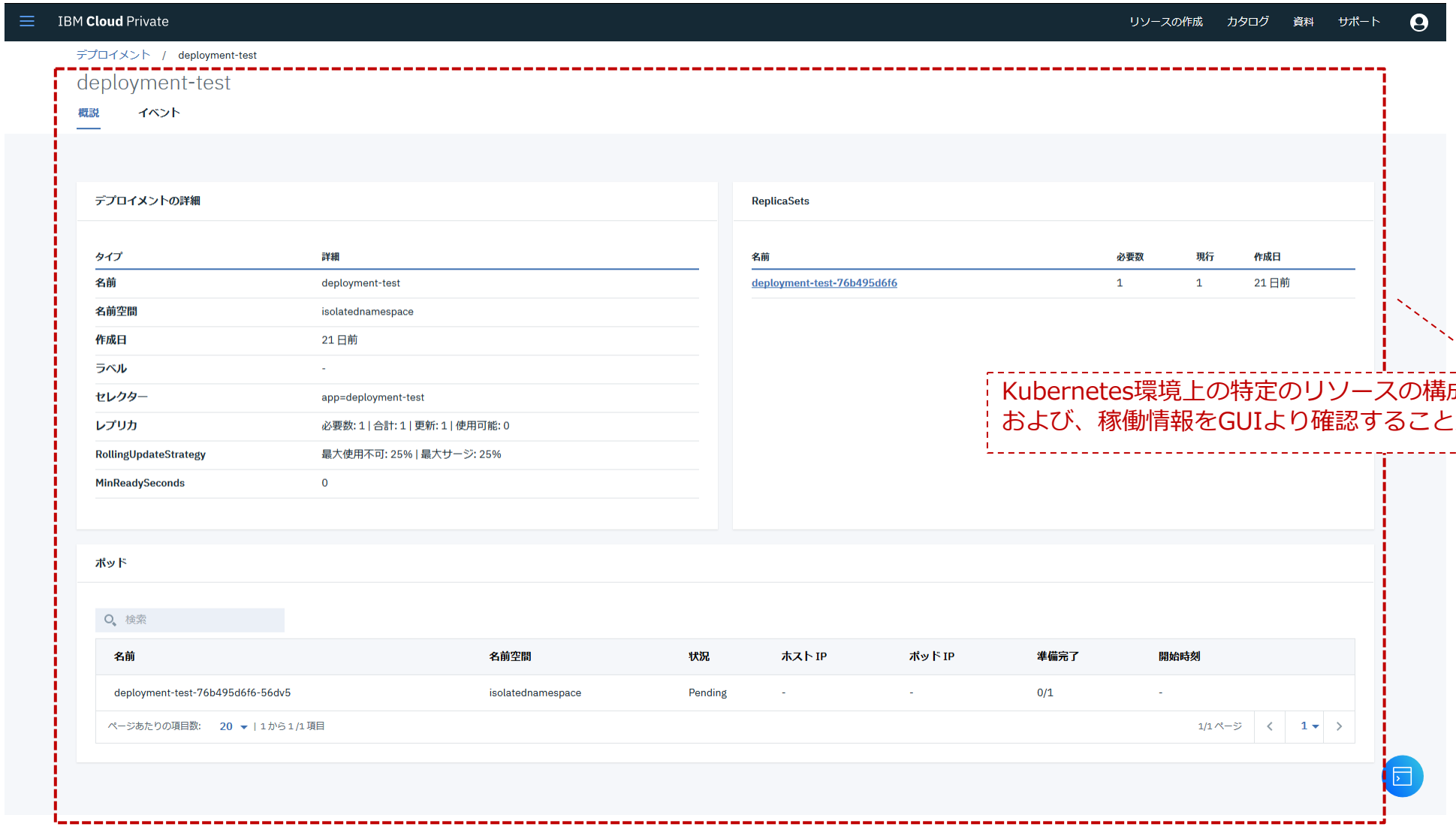


名前	名前空間	必要数	現行	準備完了	使用可能	作成日	
mcm-klusterlet-ibm-mcmk-dev-klusterlet	kube-system	1	1	-	-	5 時間前	
mcm-klusterlet-ibm-mcmk-dev-weave-scope-app	kube-system	1	1	-	-	5 時間前	起動
liberty-default-helm-ibm	was	1	1	-	-	9 日前	起動
deployment-test	isolatednamespace	1	1	-	-	21 日前	
labapp-libertyapp	was	1	1	-	-	22 日前	起動
logging-elk-client	kube-system	1	1	1	1	26 日前	起動
logging-elk-kibana	kube-system	1	1	1	1	26 日前	起動
logging-elk-logstash	kube-system	1	1	1	1	26 日前	起動
logging-elk-master	kube-system	1	1	1	1	26 日前	起動
	kube-system	1	1	-	-	28 日前	起動
	kube-system	1	1	1	1	28 日前	
	kube-system	1	1	1	1	28 日前	
	kube-system	1	1	1	1	28 日前	
	kube-system	1	1	1	1	28 日前	
vulnerability-advisor-ma-file-annotator	kube-system	1	1	1	1	28 日前	
vulnerability-advisor-notification-dispatcher	kube-system	1	1	1	1	28 日前	
vulnerability-advisor-password-annotator	kube-system	1	1	1	1	28 日前	

Kubernetes環境における主要なリソース種別のそれぞれに対する専用の管理画面が提供される。

管理画面を用いることにより、リソースに対する作成、閲覧、および廃棄のための操作が可能

ICP製品が提供するGUIの実例（4/5）



ICP製品が提供するGUIの実例（5/5）

GUIとして提供される各種のダッシュボード類を用いることにより、クラスターの稼働状況を一元的に確認、把握することが可能