



IBM **API** Connect

IBM API Connect v2018 Kubernetes版 クラスター構築ガイド

Disclaimer

❑ 当資料の位置づけ

- 当資料は、IBM API Connect v2018 kubernetesのクラスター構築手順をまとめたものである。

❑ 注意事項

- 当資料に含まれる情報は可能な限り正確を期しているが、当資料に記載された内容に関して何ら保証するものではない。ここでの記載内容はあくまでも支援情報であり、使用者の責任において取扱われるものとし、資料の内容によって受けたいかなる損害に関して一切の保証をしない。
- 製品の新しいリリース、修正などによって動作／仕様が変わる可能性があるので、必ずマニュアル等で最新の情報を確認する必要がある。

❑ 本ガイドの作成で利用したAPI Connect v2018 kubernetesの導入イメージ

コンポーネント	導入したイメージ
Management	management-images-kubernetes_lts_v2018.4.1.1.tgz
Developer Portal	portal-images-kubernetes_lts_v2018.4.1.1.tgz
Analytics	analytics-images-kubernetes_lts_v2018.4.1.1.tgz
DataPower Gateway	(Docker hubからv2018.4.1.1をインストール)

❑ 前提条件

以下のリンクを一読し、Firewall要件等を満たしていることを確認してください。

https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.install.doc/requirements_overview.html

Disclaimer

□ 本ガイドの作成で利用したOS、Docker、kubernetes等のバージョン

項目	バージョン
Ubuntu (Server版)	18.04.1
Docker	18.06.1-ce
kubernetes	1.12.3
Ceph	12.2.7 (luminous)
Ceph-deploy	2.0.1
helm	2.12.0

□ 周辺サーバー

- DNSサーバー、NTPサーバー、SMTPサーバーなど周辺サーバーを構築する必要がある。

□ ネットワーク要件

- kubernetes環境の構築の際に、直接インターネットを経由してモジュールをインストールすることが多いため、インターネットに接続できることを前提としている。
- ネットワーク構成は本ガイド「IBM API Connect v2018 kubernetes版 クラスター基本構成」の「ネットワーク構成」に記載する。

Disclaimer

❑ kubernetes環境に導入するStorageについて

- APIC基盤環境を構築するために、BlockStorageを用意する必要がある。本ガイドではOSSの分散ストレージソフトウェアであるCephをBlockStorageとして構築する手順を記載する。
- 本ガイドは以下のURLの記載に従いBlockStorageを構築する方法を記述しているが、今後記載が更新される可能性があり、自身で構築する際は最新情報を確認すること。

https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.install.doc/tapic_install_reqs_Kubernetes.html

❑ Cephのノードについて

- 本ガイドでは、APIC基盤を構成しているMaster Nodeおよび Worker NodeでCephのクラスターを構築する手順を記載しているが、本番稼働環境ではAPIC基盤を構成しているNodeとは別にCeph用にMaster Node, Worker Nodeを用意し構築することを推奨としている。
- 本ガイドはCeph管理ノードの高可用性は考慮していない。
- Cephについての詳細は以下のURLを参照。

<http://docs.ceph.com/docs/master/>

❑ DNSについて

- IBM API Connect v2018 Kubernetes版では、API Connectのサブシステムのホスト名をDNSで解決させる必要がある。また、逆引き機能(IPアドレスからホスト名を検索)は不要である。

Disclaimer

□ 本ガイドの環境について

- 本ガイドのAPIC基盤構築環境をいかに示す。

	CPU	Memory	Storage	台数	稼働内容
Master Node	8 cores	32 GB	1 st 100GB 2 nd 500GB 3 rd 500GB	1台	Syslog, DNS, SMTP, NTP
Worker Node	16 cores	64 GB	1 st 100GB 2 nd 500GB 3 rd 500GB	3台	API Management, Developer Portal, DataPower Gateway, Analytics

□ Storageについて

本ガイドでIBM API ConnectおよびIBM DataPower Gateway用に利用しているストレージの内訳を以下に示す。

Worker Node Storage 2nd

- ・・・ API Management, Developer Portal, DataPower Gateway, Analyticsの稼働インスタンスとして500GB。

Worker Node Storage 3rd

- ・・・ 各コンポーネントのPersistent Volume用にCephを用意し、そのDiskとして500GB。
(クラスター環境として1500GB)

※実際にサイジングする場合は、以下のSystem Requirementsの内容を確認しておくこと。

https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.install.doc/overview_apimgmt_requirements.html



目次

□ APIC v2018 Kubernetes版 クラスター基本構成

□ 構築の流れ

□ 事前準備

1. ユーザーの作成（sudo、証明書でのsshログイン）
2. dockerコマンドのインストール
3. kubernetesコマンドのインストール
4. helmコマンドのインストール
5. ceph-deployコマンドのインストール
6. apicupコマンドのインストール

□ kubernetesランタイム環境の構築

1. Cephのセットアップ
2. kubernetesのクラスター・セットアップ
3. ネットワークプラグインのインストール
4. Worker Nodeのセットアップ
5. corednsの修正
6. ストレージのセットアップ
7. Docker Repositoryのセットアップ
8. helmの初期設定、Ingressのセットアップ

□ API Connectのセットアップ

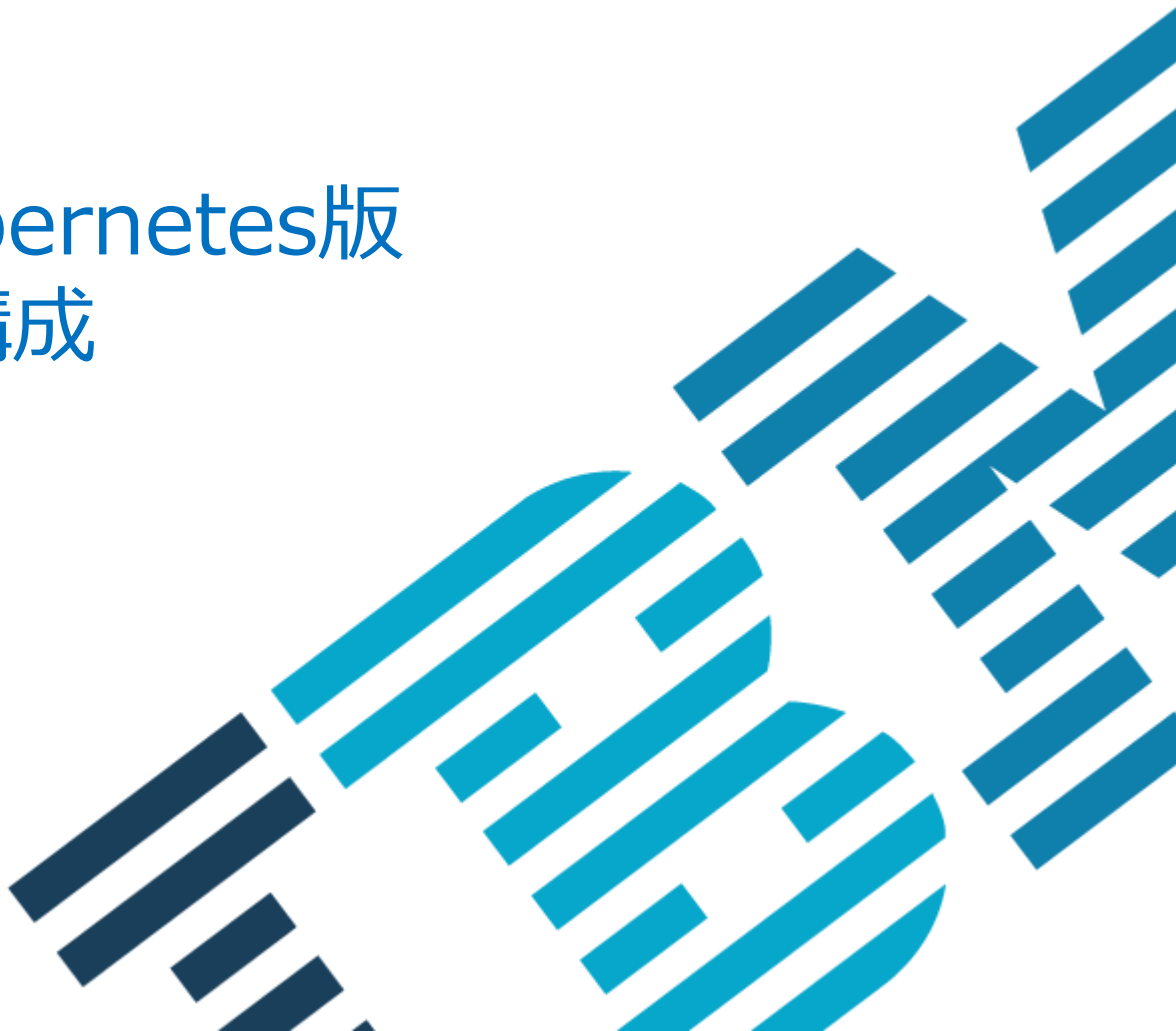
□ API Connectの初期設定

□ 補足資料



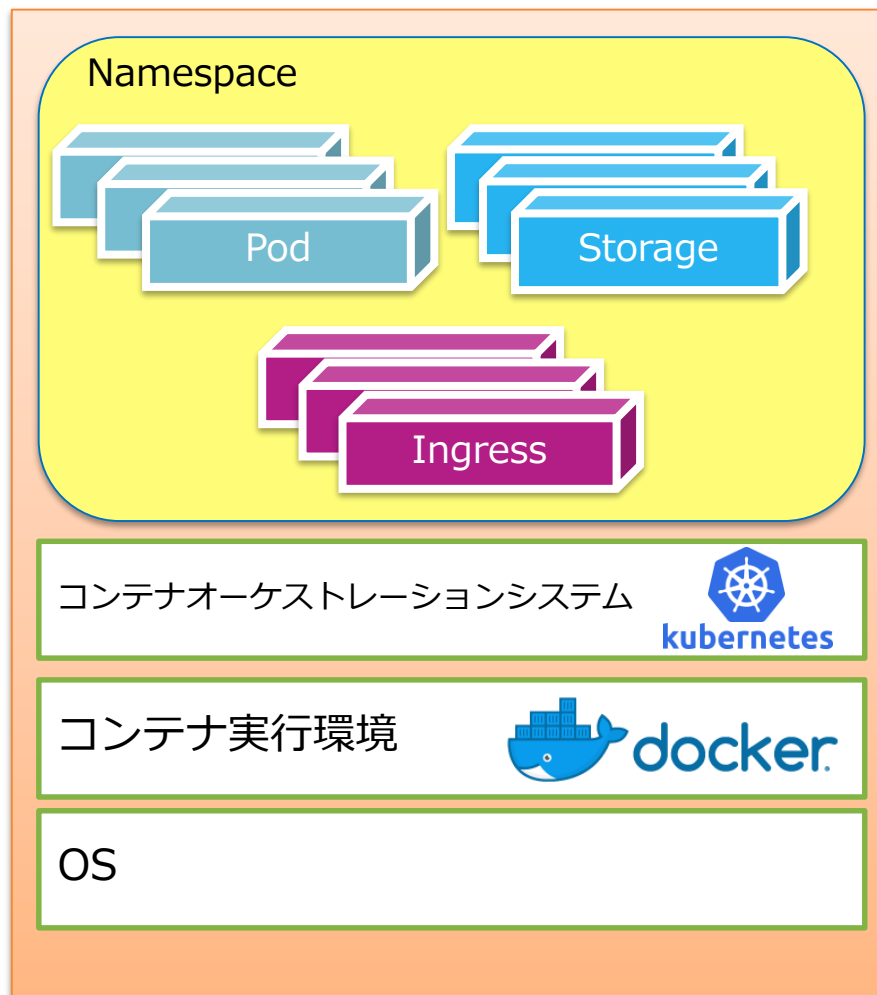
IBM **API** Connect

APIC v2018 Kubernetes版 クラスター基本構成



kubernetes環境の基本概念図

このページではkubernetes環境の一般的な構成と基本概念について解説する。



- アプリケーションはWorker Nodeにデプロイされ、それぞれNodeはMaster Nodeに管理される。
- Nodeは主にOS、コンテナ実行環境、コンテナオーケストレーションシステム（コンテナ管理ソフト）、および各種リソースから構成される。
- 以下は主要なリソースを紹介する。
 - Namespace** : 仮想的なkubernetesクラスターの分離機能。
 - Pod** : コンテナを起動させるのに利用するオブジェクト。APICの各コンポーネントはPod内のコンテナにデプロイされる。
 - Storage** : コンテナ（Pod）のデータを永続化したり、コンテナ間でファイルを共有するために使用される（※）。
 - Ingress** : L7ロードバランサーであり、URLやHTTPヘッダーで負荷分散できるため、外部の受け口として利用する

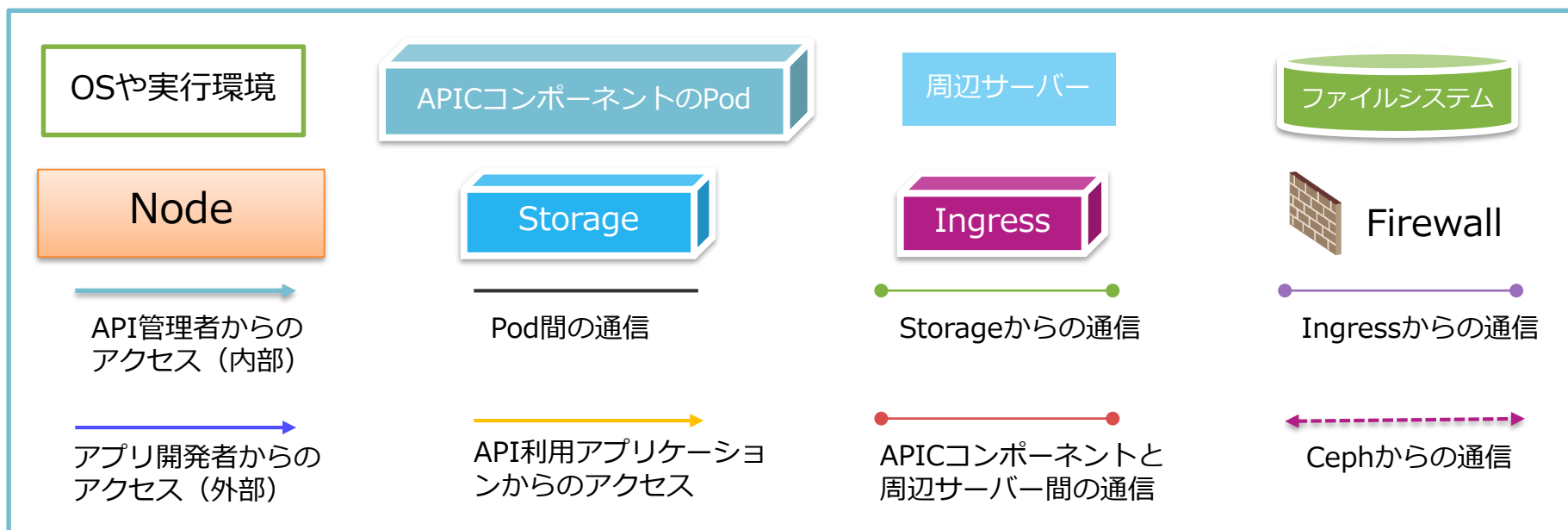
※Storageを使用しない場合は、Podのクラッシュや再起動でデータや状態が消失する。

図 : kubernetes基本構成

IBM API Connect Kubernetes版の構成要素

□ 凡例

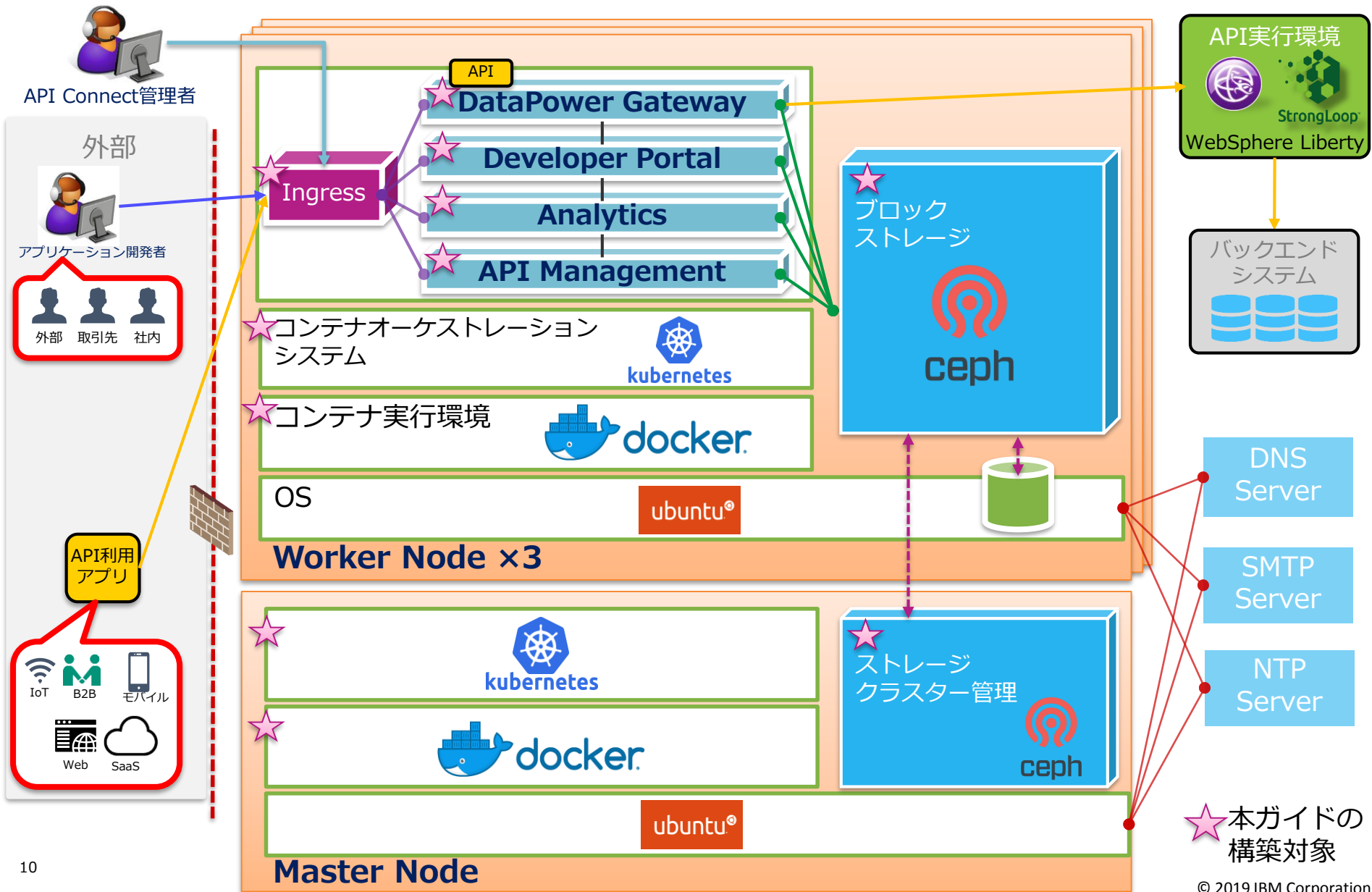
- 次のページでAPIC Kubernetes版（以降、k8s）クラスタリングの基本構成を図示するが、構成図で使用されている凡例、構成図に関する説明をここで示す。



□ 説明

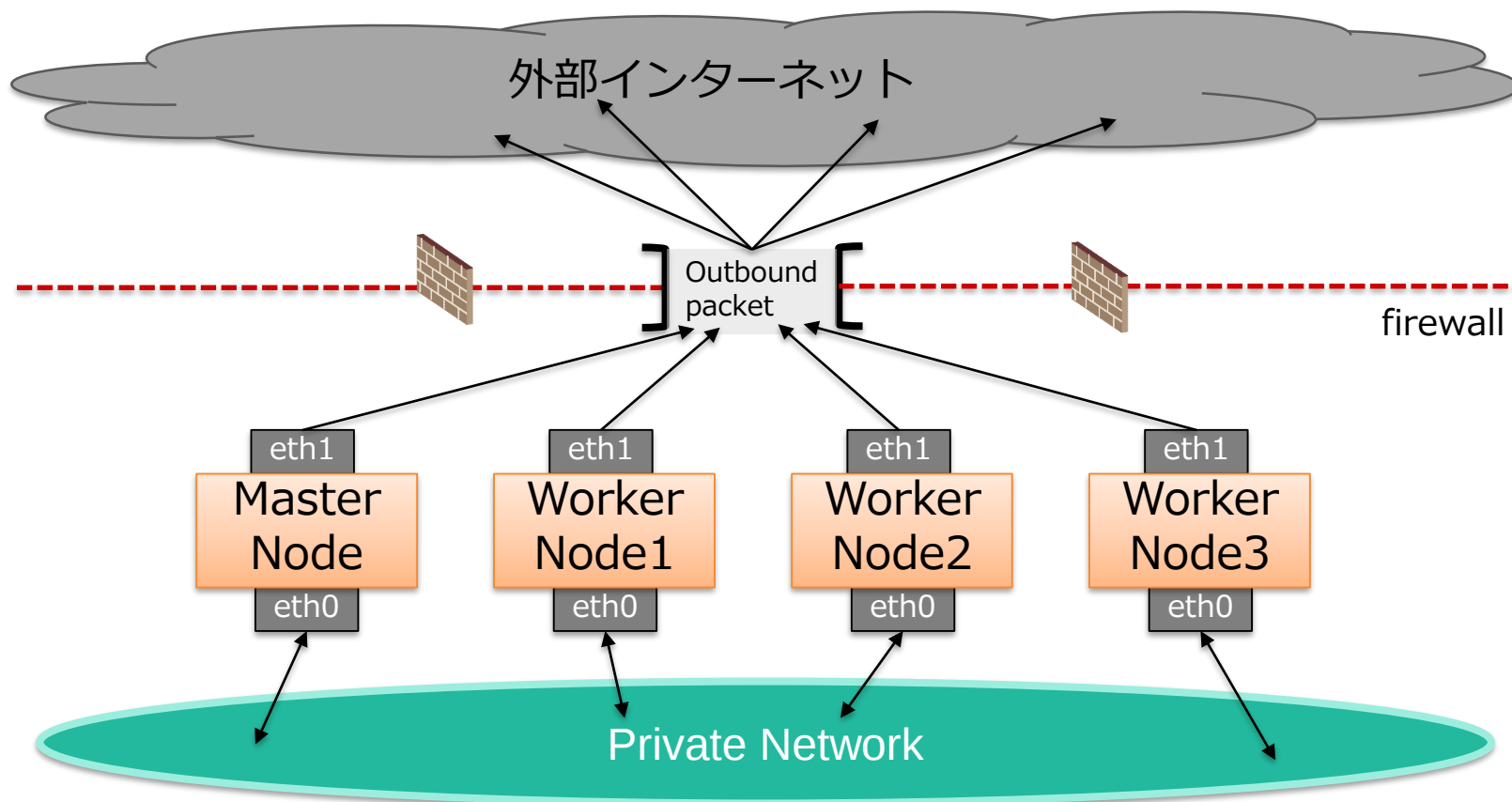
- 本ガイドでは、Master Node1台とWorker Node3台の計4台でAPIC基盤環境を構築する。
- 本ガイドの範囲外になるが、SMTPサーバー（アカウント作成、組織の作成の際にアクティベーションリンクの通知メールを送信するため）、NTPサーバー（各APICコンポーネント間で時間の表示を整合させるため）、DNSサーバー（サブシステムのホスト名を解決させるため）の設定が必要である。
- ServiceはPod間やクラスタ間の通信用ロードバランサーであるため、基本構成図での表示は省略している。

IBM API Connect Kubernetes版 クラスタ基本構成



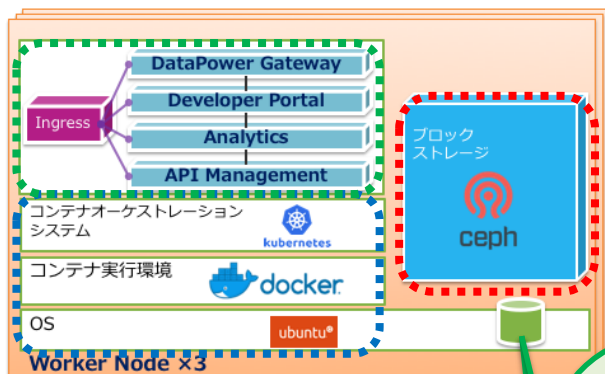
ネットワーク構成について

本ガイドでは本番稼働環境を想定して、各NodeにはNIC(Network Interface Card)を2つ用意し内部通信と外部通信を分離した構成を前提としている。NICが2つある場合に必要な設定も含めた設定を本ガイドで記載している。



(参考)物理構成について

本ガイド環境でのファイルシステムの割り当てについて以下に示す。
(Master NodeおよびWorker Nodeで同様の割り当て)



デバイス名	サイズ	マウント先	割り当て場所
xvda	100 GB		OS
└─ xvda1	256 MB	/boot	OS
└─ xvda2	99.8 GB	/	OS
xvdb	2GB		OS
└─ xvdb1	2GB		OS
xvdc	500 GB		APIc稼働インスタンス
└─ xvdc1	500 GB	/var	APIc稼働インスタンス
xvde	500 GB		Ceph
└─ Ceph--xxxxx	500 GB		Ceph



構築の流れ

※この章では、図を用いてIBM API Connect v2018 kubernetesの構築の流れを解説する。具体的な構築方法、利用するコマンド、実行例などは、後述の章に記載されている。

構築の流れ

- 本ガイドは下記手順を参考に行っている

API Connect

https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.install.doc/tapic_install_Kubernetes_overview.html

Ceph(luminous)

<http://docs.ceph.com/docs/luminous/start/quick-start-preflight/#debian-ubuntu>

<http://docs.ceph.com/docs/luminous/start/quick-ceph-deploy/>

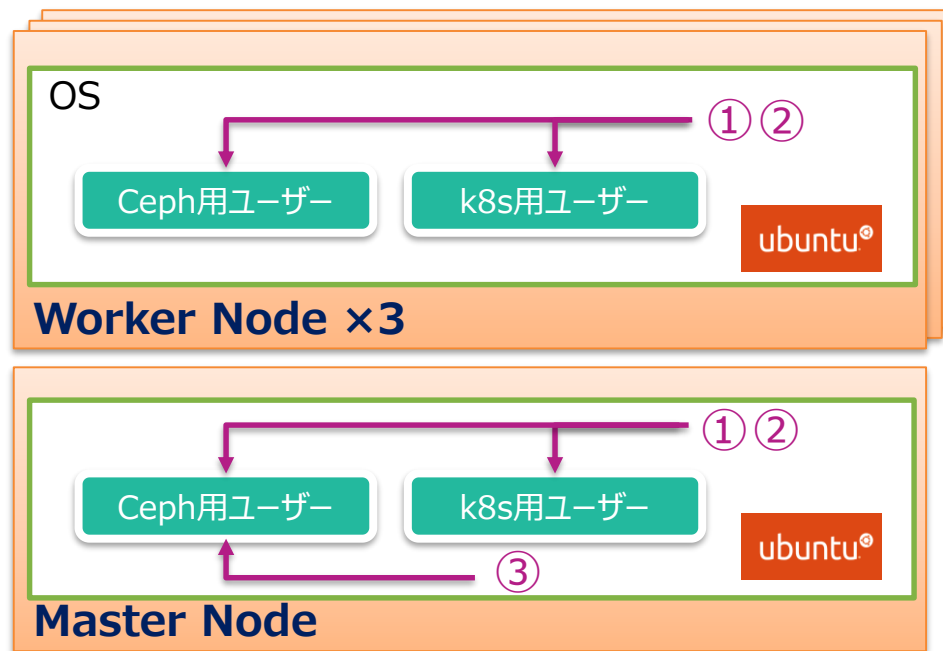
kubernetes

<https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm>

構築の流れ (1) -事前準備1

□ ユーザーの作成 (sudo、証明書でのsshログイン)

今回の構成では各ノード間でkubernetes, Cephでの通信を行うため、以下のような操作を実施する必要がある。



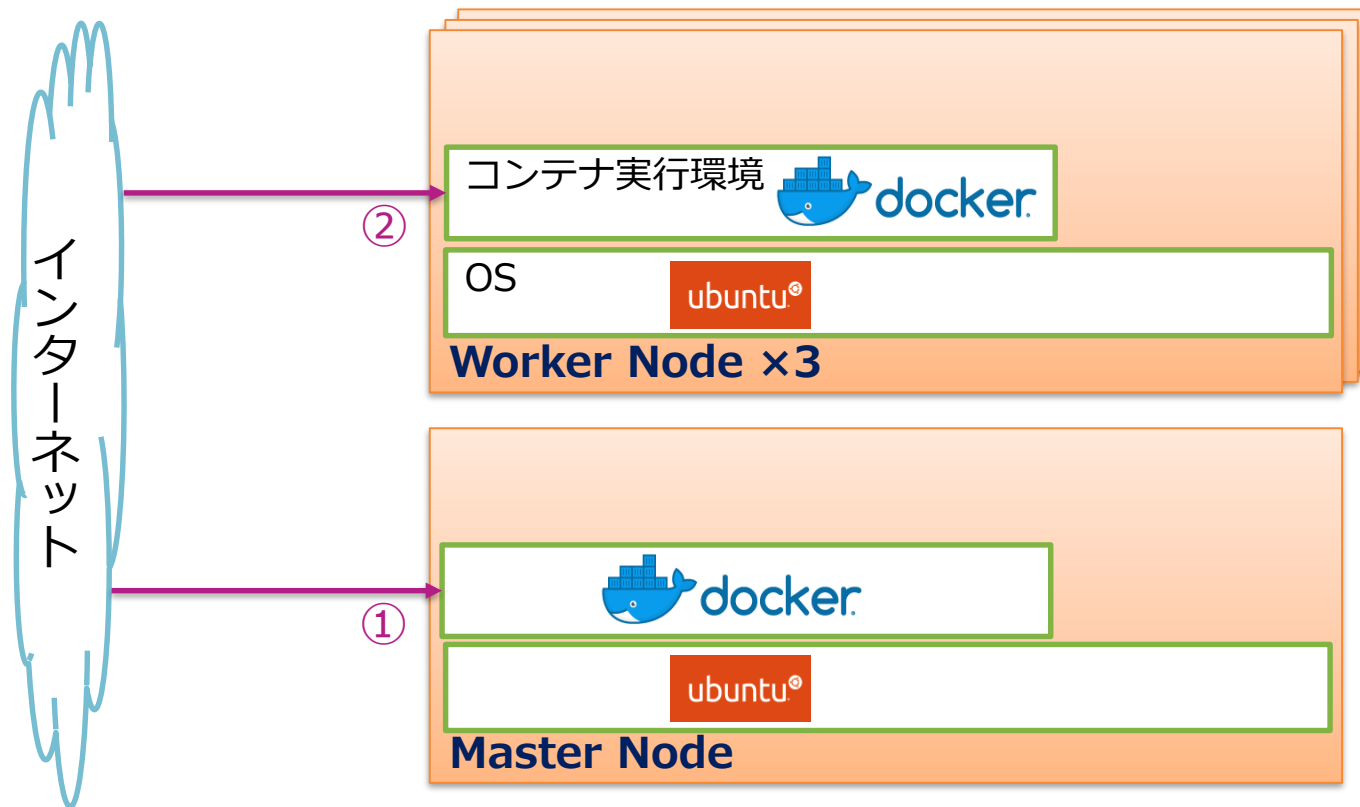
① kubernetes(=k8s)およびCeph用ユーザーを作成する。

② sudoを使用できるように設定する。

③ Master Nodeから各NodeへCeph用ユーザーで証明書ログインができるように設定する。

構築の流れ (1) 事前準備2

❑ dockerコマンドのインストール

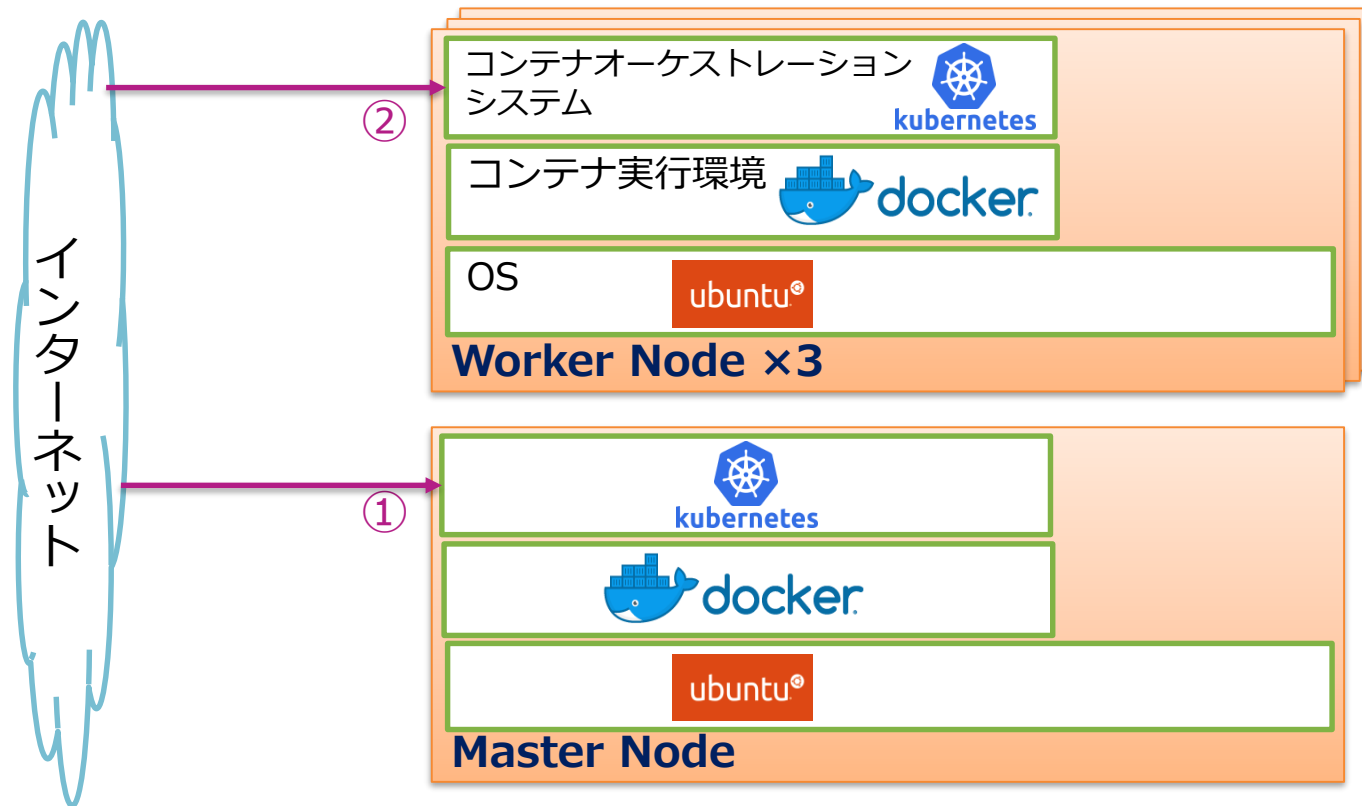


① Master Nodeにおいてコマンドでインターネットからdockerをインストールする。

② Worker Nodeにおいて上記の操作を行う。

構築の流れ (1) 事前準備3

□ kubernetesコマンドのインストール

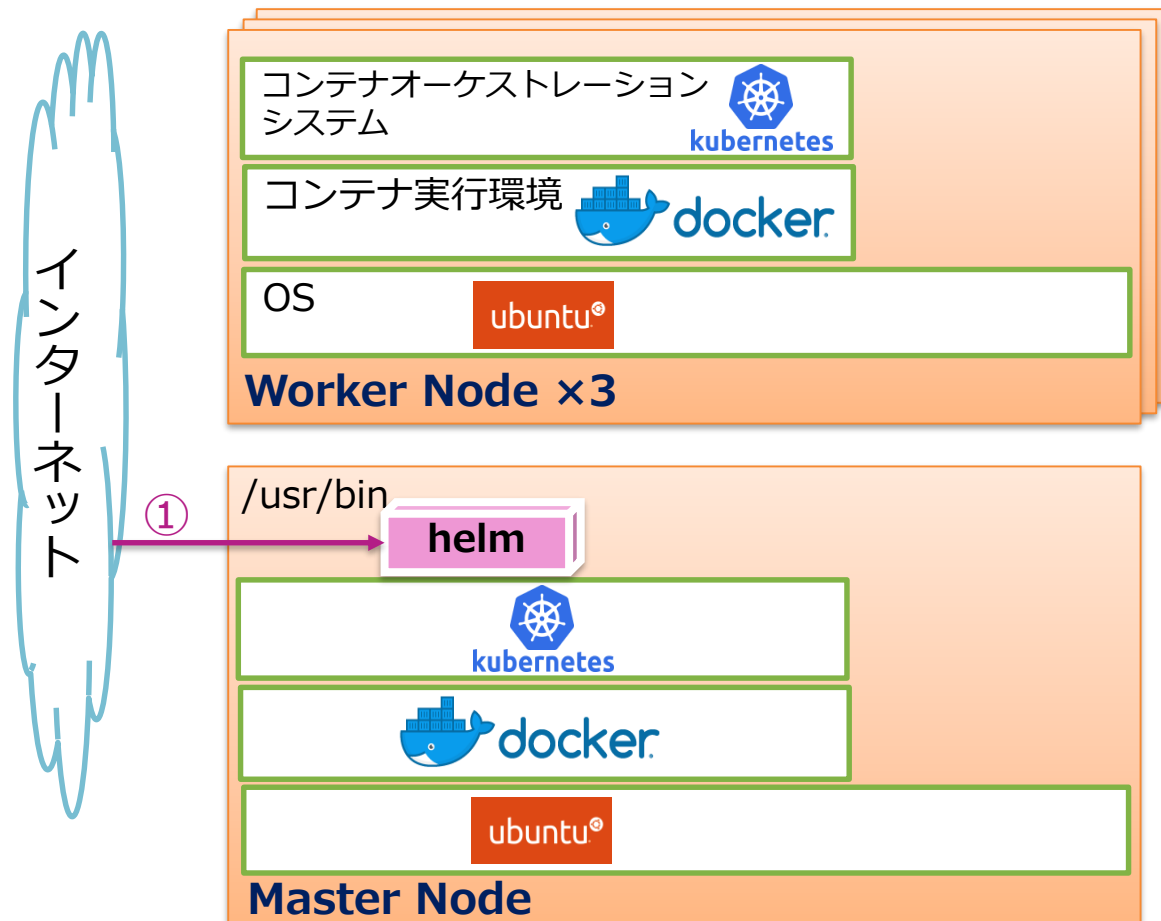


① Master Nodeにおいてコマンドでインターネットからkubernetesをインストールする。

② Worker Nodeにおいて上記の操作を行う。

構築の流れ（1） -事前準備4

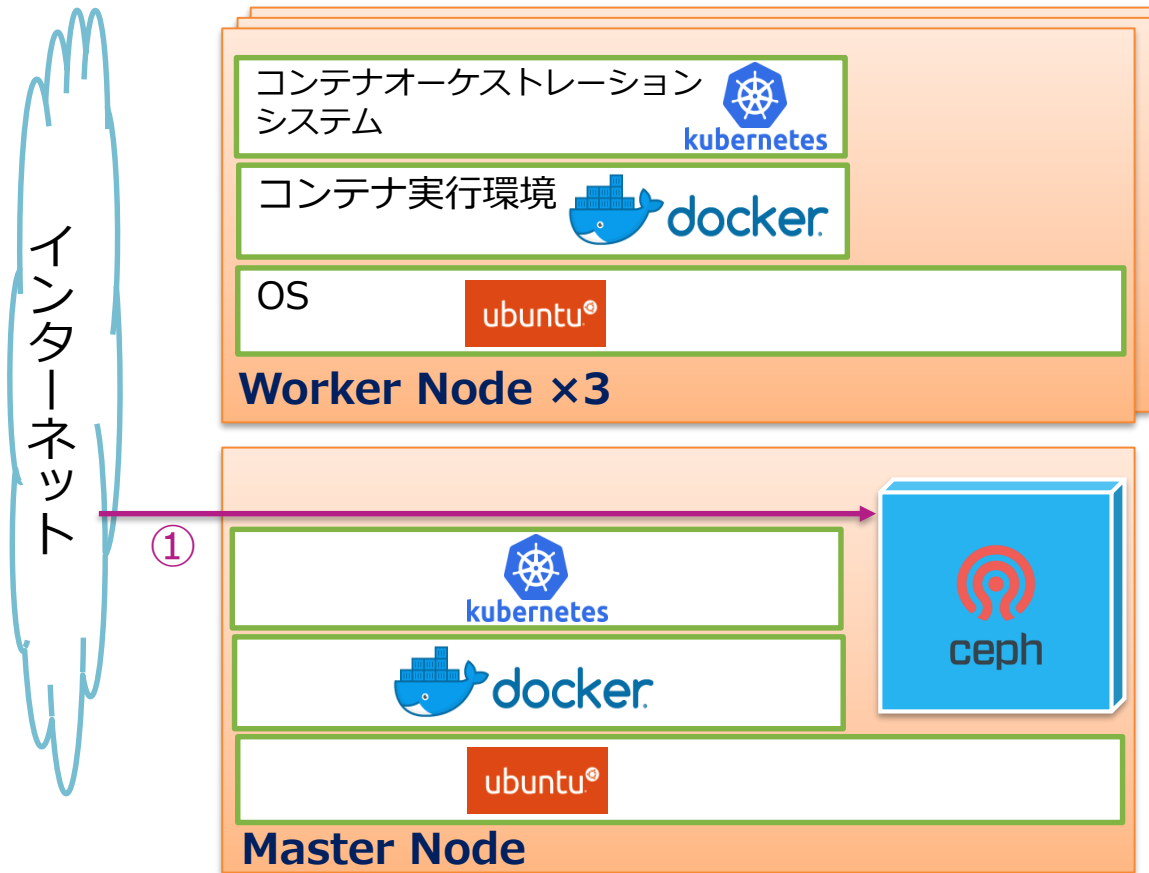
□ helmコマンドのインストール



① helmをインターネットからダウンロードし、解凍した状態で /usr/bin のディレクトリ配下に格納する（Master Nodeのみ）。

構築の流れ（1） -事前準備5

❑ ceph-deployコマンドのインストール

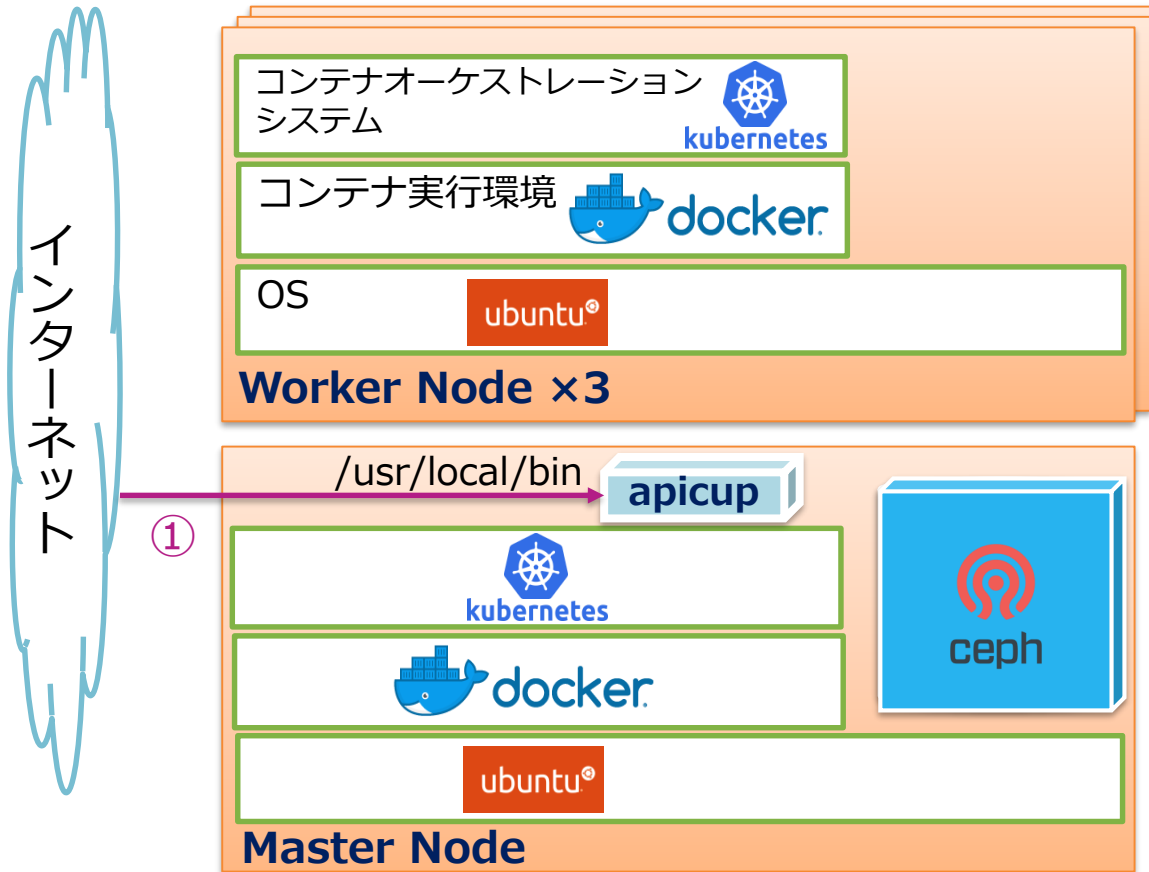


① コマンドでインターネットから ceph-deploy コマンドをインストールする（Master Nodeのみ）。

※ceph-deploy コマンドはCephの設定、クラスター作成に使用するコマンド

構築の流れ（1） -事前準備6

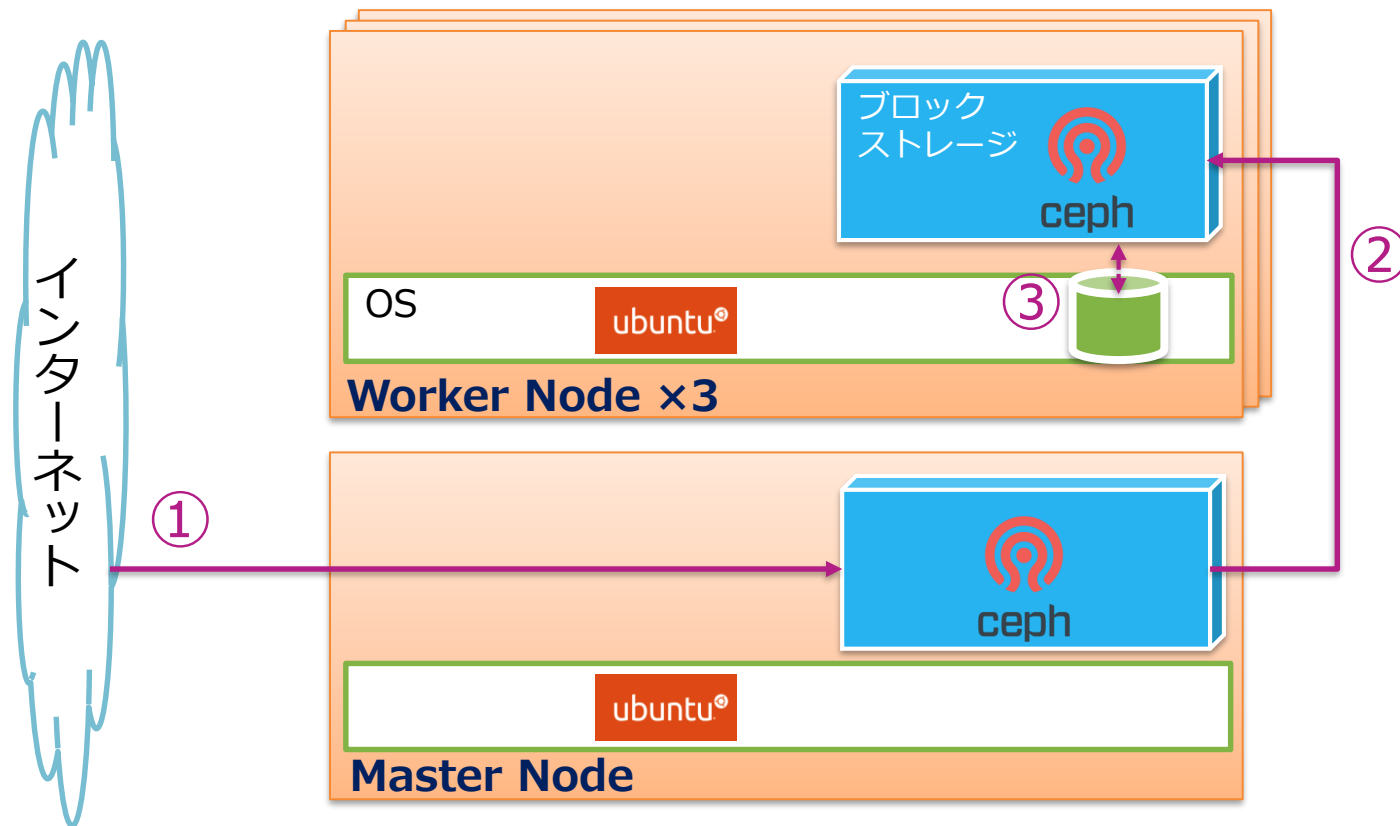
□ apicupコマンドのインストール



① apicup-linuxをインターネットからダウンロードし、
/usr/local/binディレクトリー配下に格納する（Master Nodeのみ）。

構築の流れ (2) kubernetesランタイム環境の構築1

□ Cephのセットアップ



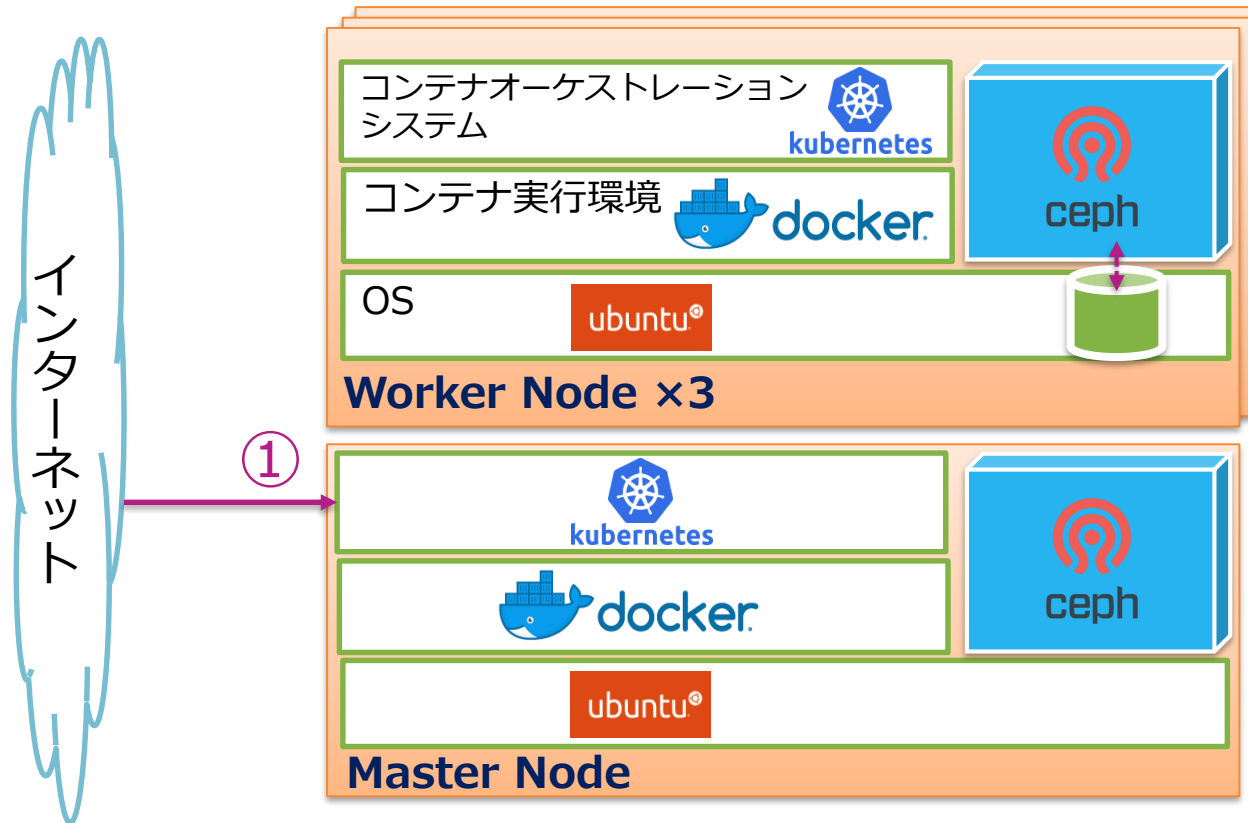
① Master NodeにコマンドでインターネットからCephをインストールする。

② 各Worker NodeにCephクラスターを作成する。

③ OSのファイルシステムからOSD(Object Storage Daemon)を割り当てる。

構築の流れ (2) kubernetesランタイム環境の構築2

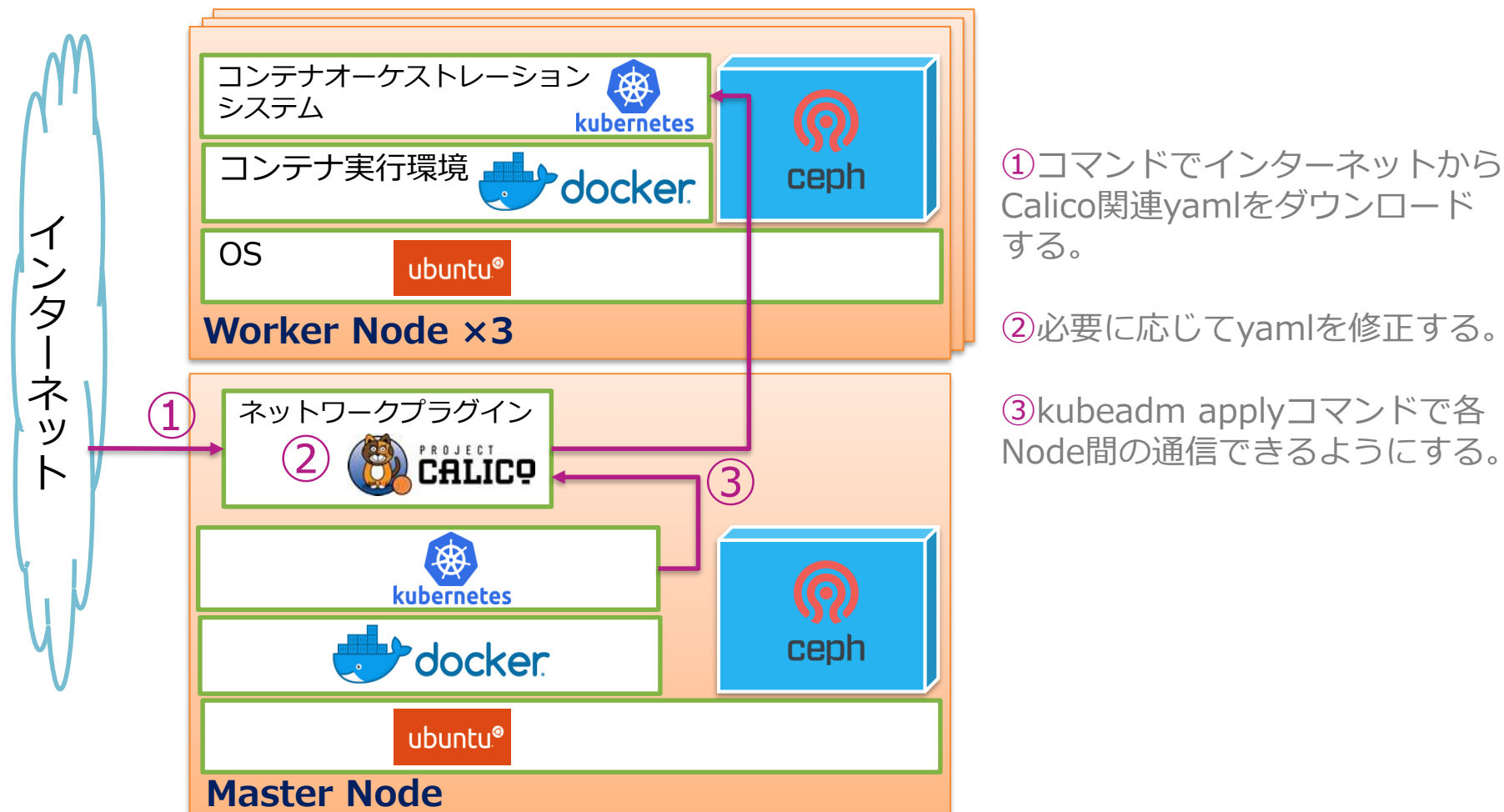
□ kubernetesのクラスター・セットアップ



① Master Nodeでkubeadm initコマンドを実行し、クラスターを作成する。

構築の流れ（2） kubernetesランタイム環境の構築3

□ ネットワークプラグインのインストール

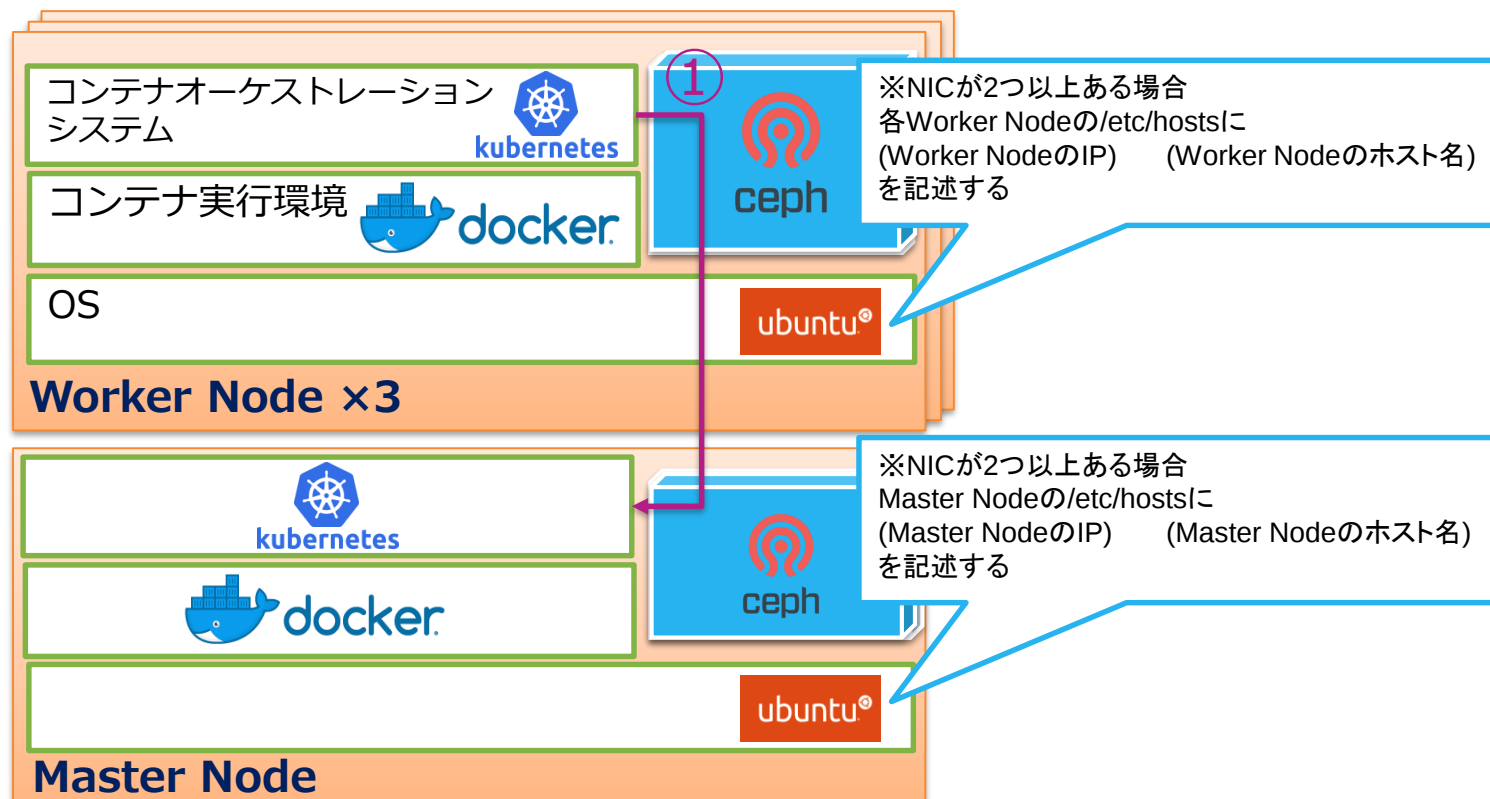


※CalicoとはOSSのネットワークプラグインであり、
本ガイドではCalicoを使用して、kubernetesのネットワークを構成する。

構築の流れ (2) kubernetesランタイム環境の構築4

□ Worker Nodeのセットアップ

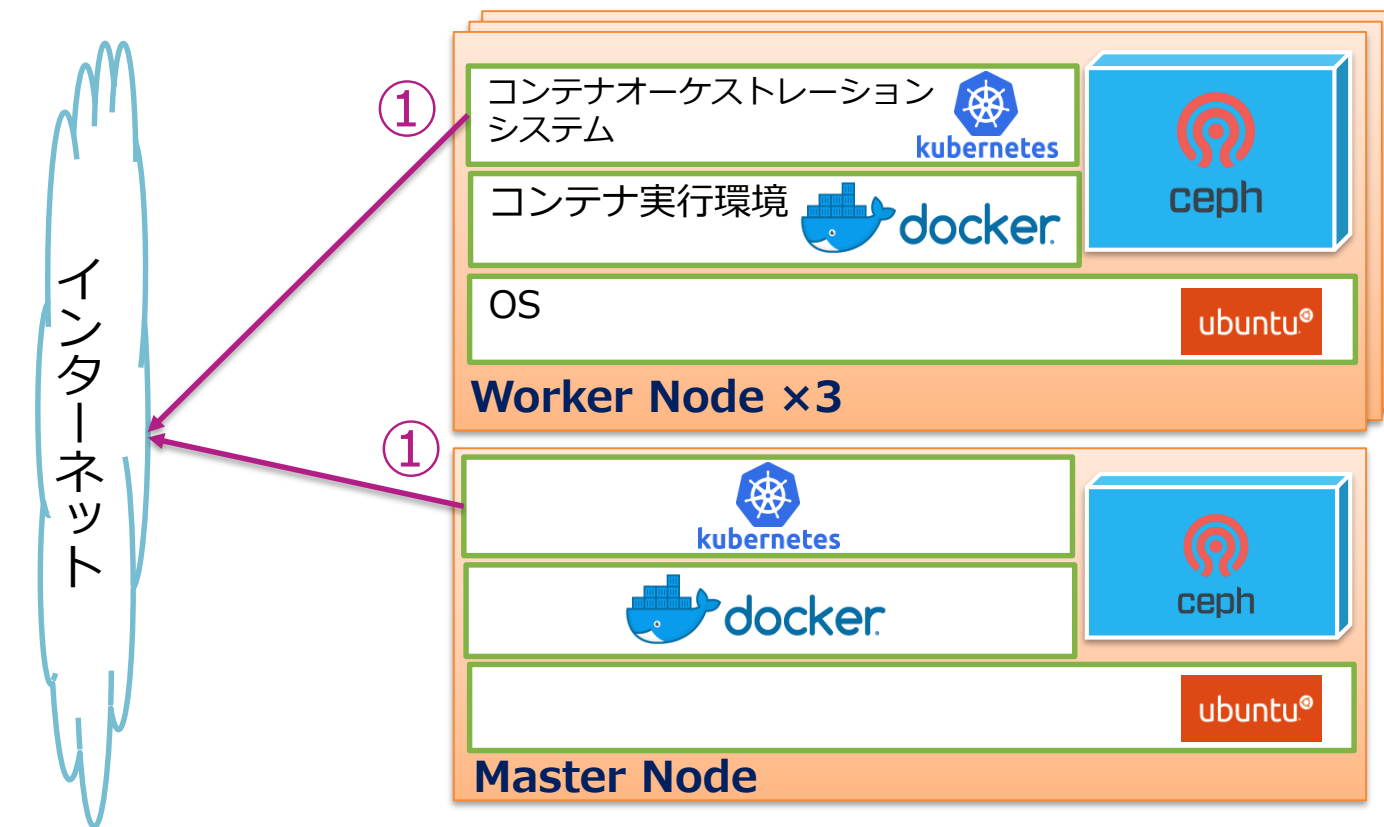
インターネット



① コマンドで各Worker Nodeをクラスターに参加させる。

構築の流れ (2) kubernetesランタイム環境の構築5

□ corednsの修正

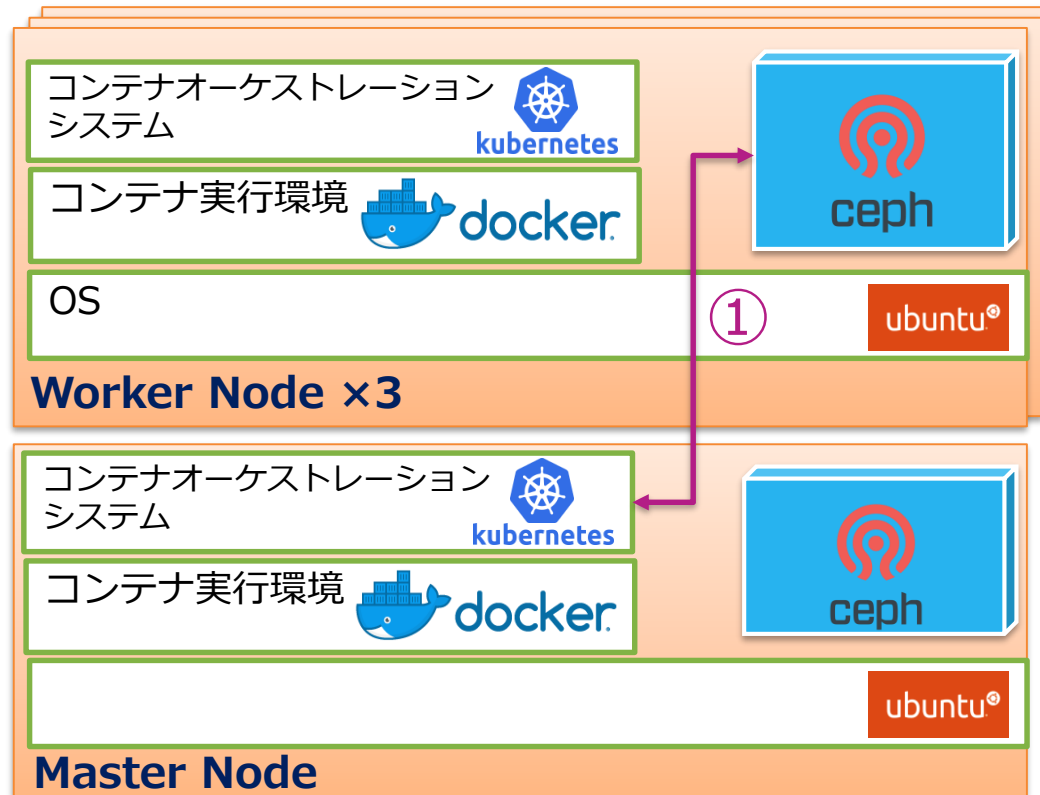


① corednsのpodを修正し、各ノードが外部のDNSを参照可能にする。

構築の流れ (2) kubernetesランタイム環境の構築6

□ ストレージのセットアップ

インターネット

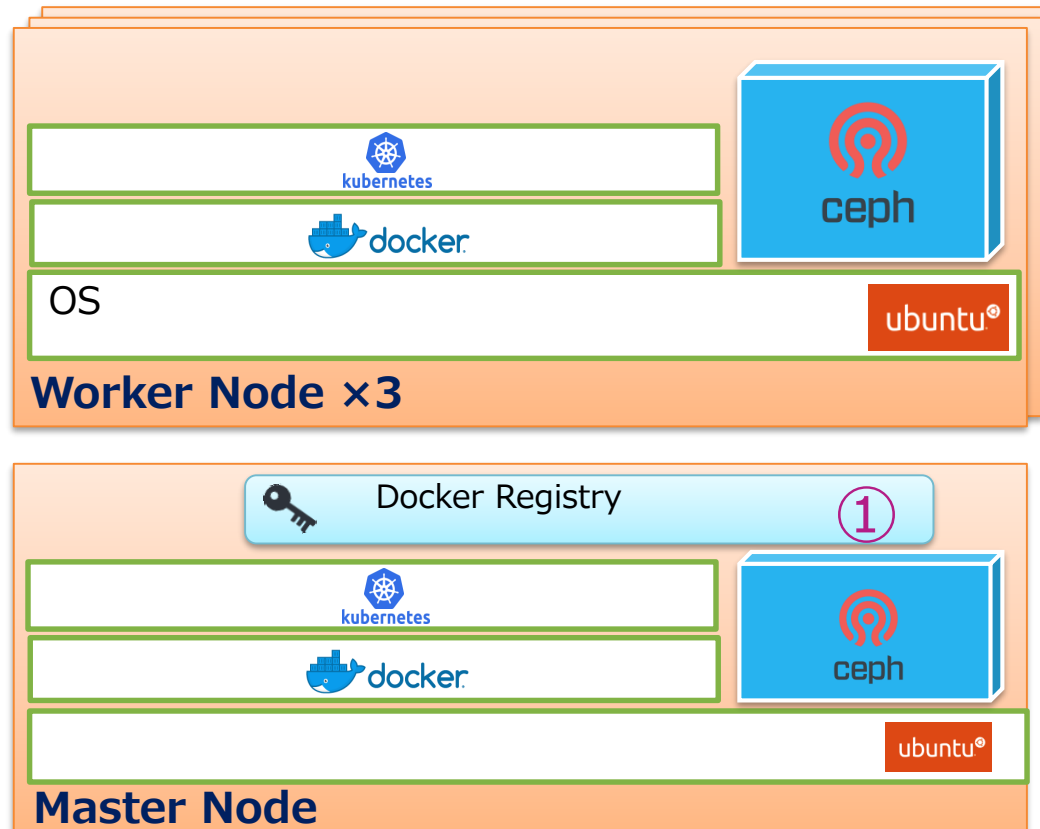


① コマンドでCephをPersistent Volumeとして作成する。

構築の流れ（2） kubernetesランタイム環境の構築7

□ Docker Repositoryのセットアップ

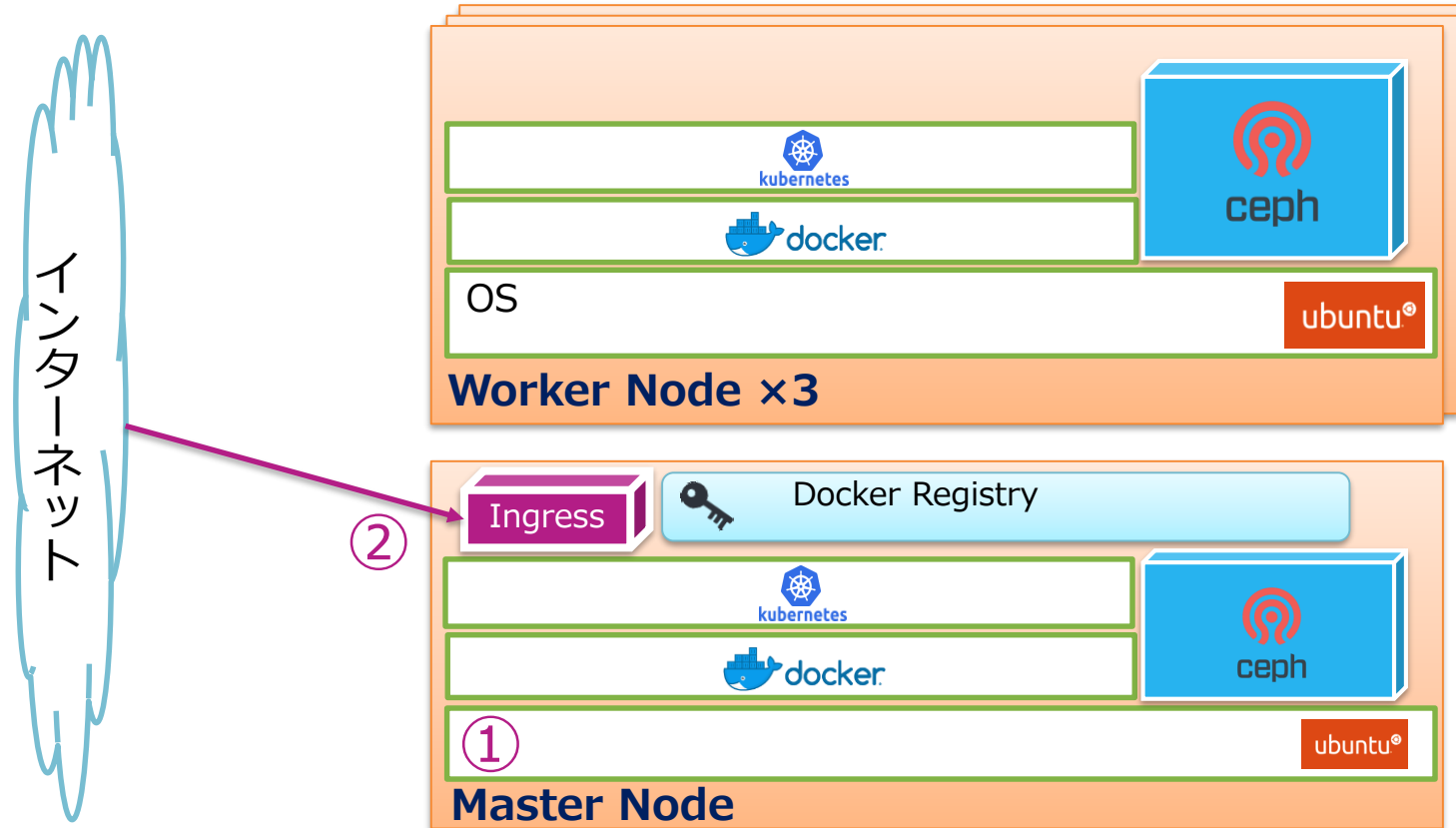
インターネット



① Docker Registryを作成し、API Connectのベースイメージを保管可能にする（Master Nodeのみ）。

構築の流れ（2） kubernetesランタイム環境の構築8

□ helmの初期設定、Ingressのセットアップ



① Master Nodeでhelmコマンドを利用可能にする。

② Ingressを導入する（Master Nodeのみ）。



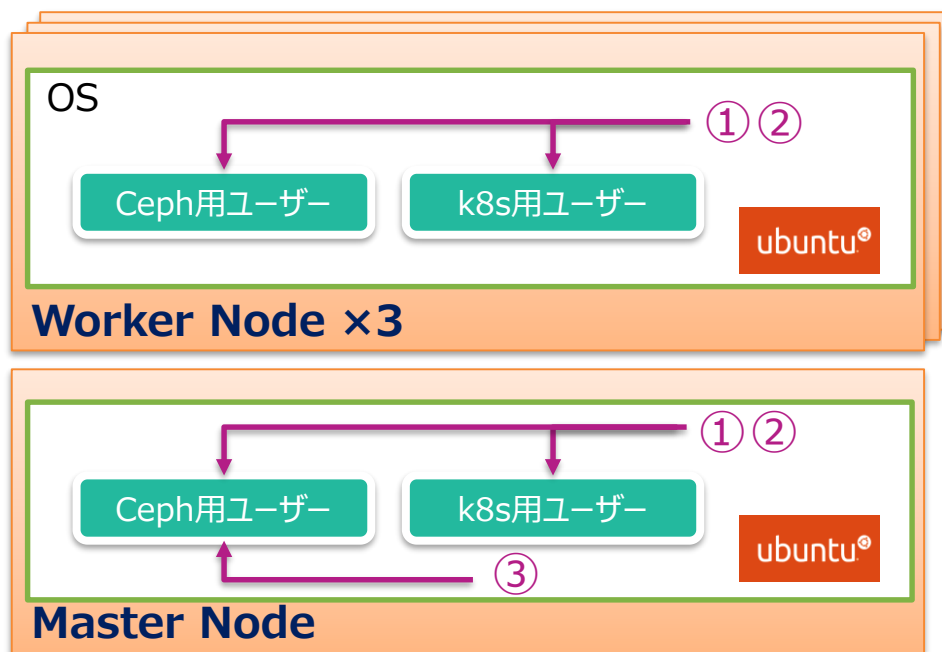
事前準備1

ユーザーの作成（sudo、証明書でのsshログイン）

事前準備1

□ ユーザーの作成 (sudo、証明書でのsshログイン)

今回の構成ではkubernetes, Cephを構築するユーザーを作成する。



① kubernetes(=k8s)およびCeph用ユーザーを作成する。

② sudoを使用できるように設定する。

③ Master Nodeから各NodeへCeph用ユーザーで証明書ログインができるように設定する。

Ceph用ユーザーの作成(全サーバー)

1. Ceph用ユーザーを作成する (本ガイドではユーザー名をceph_adminとする)

```
# adduser ceph_admin
```

2. 作成したユーザーでsudoが使えるように設定

```
# echo "ceph_admin ALL = (root) NOPASSWD:ALL" | tee /etc/sudoers.d/ceph_admin
```

3. sudoer.d/ceph_adminのパーマッションを変更する

```
# chmod 0440 /etc/sudoers.d/ceph_admin
```

Ceph用ユーザーの作成(全サーバー)

4. sudoersに登録されていることを確認する

```
# su - ceph_admin  
$ sudo -l
```

コマンドの絶対パスが表示される場合、コマンドが正常に終了したと判断する。

実行例(Master Node)

```
# su - ceph_admin  
$ sudo -l  
Matching Defaults entries for ceph_admin on master:  
    env_reset, mail_badpass,  
secure_path=/usr/local/sbin¥:/usr/local/bin¥:/usr/sbin¥:/usr/bin¥:/sbin¥:/bin¥:  
/snap/bin  
  
User ceph_admin may run the following commands on master:  
    (root) NOPASSWD: ALL
```

パスワードなしでSSH接続をできるようにする(Master Node)

1. ceph_adminに切り替える

```
# su - ceph_admin
```

2. .sshディレクトリを作成

```
$ mkdir .ssh
```

3. .sshのパーミッションを変更する

```
$ chmod 700 .ssh
```

4. SSHで利用する認証の鍵を生成する

```
$ ssh-keygen
```

5. ssh-keygenで生成されたid_rsa.pubの値を全Nodeの /home/ceph_admin/.ssh/authorized_keysにコピーする



パスワードなしでSSH接続をできるようにする(Master Node)

6. ceph_adminで(worker1)にssh接続をする

```
#ssh (worker1)
```

パスワードを聞かれることなく、worker1にceph_adminでログインが成功した場合、コマンドが正常に終了したと判断する。

k8s用ユーザーの作成(全サーバー)

1. k8s用ユーザーを作成する (本ガイドではユーザー名をk8s_adminとする)

```
#useradd -d /home/k8s_admin -m k8s_admin
```

2. 作成したユーザーでsudoが使えるように設定

```
#echo "k8s_admin ALL = (root) NOPASSWD:ALL" | tee /etc/sudoers.d/k8s_admin
```

3. sudoer.d/ceph_adminのパーミッションを変更する

```
#chmod 0440 /etc/sudoers.d/k8s_admin
```

k8s用ユーザーの作成(全サーバー)

4. sudoersに登録されていることを確認する

```
# su - k8s_admin
$ sudo -l
```

コマンドの絶対パスが表示される場合、コマンドが正常に終了したと判断する。

実行例(Master Node)

```
# su - k8s_admin
$ sudo -l
Matching Defaults entries for k8s_admin on master:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin¥:/usr/local/bin¥:/usr/sbin¥:/usr/bin¥:/sbin¥:/bin¥:/snap/bin

User k8s_admin may run the following commands on master:
    (root) NOPASSWD: ALL
```

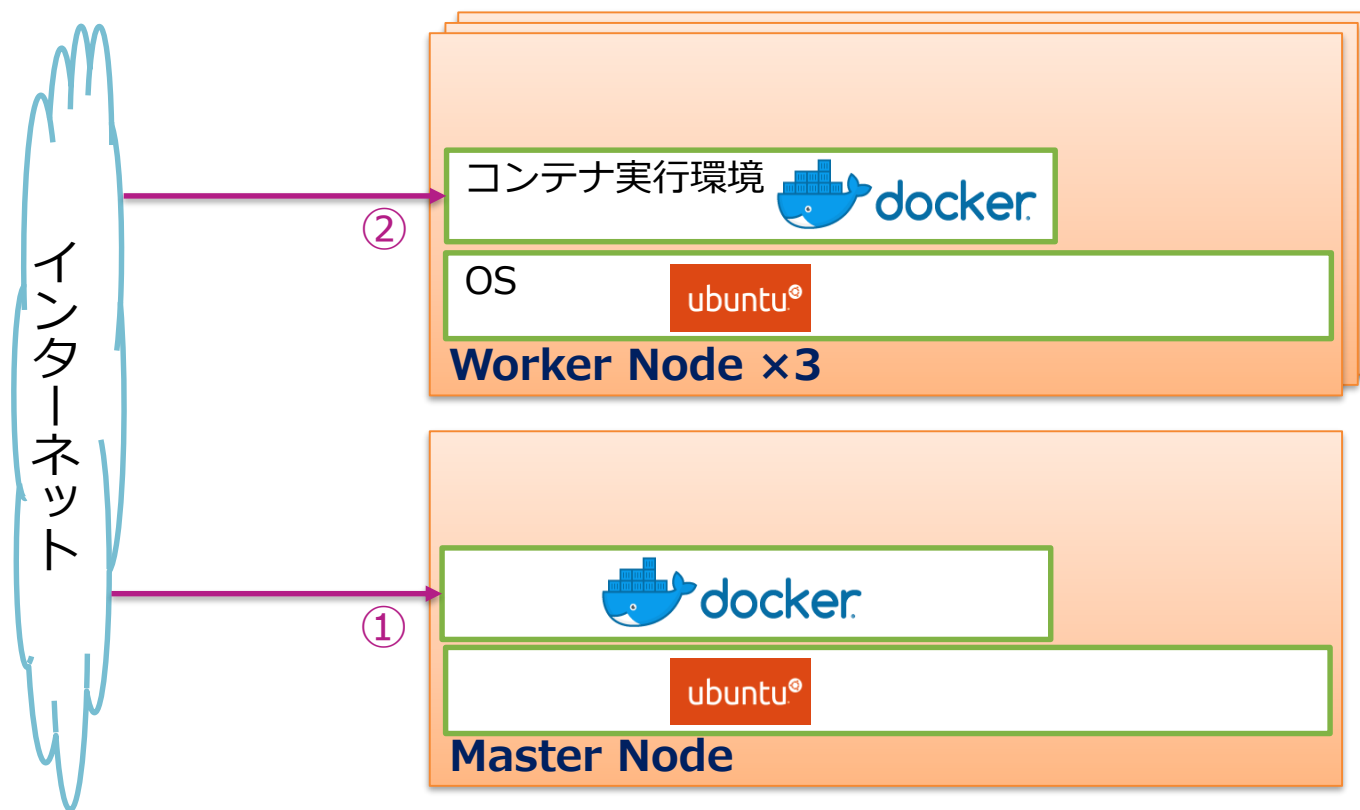


事前準備2

dockerコマンドのインストール

事前準備2

❑ dockerコマンドのインストール



① Master Nodeにおいてコマンドでインターネットからdockerをインストールする。

② Worker Nodeにおいて上記の操作を行う。



dockerコマンドのインストール(Master Node)

1. k8s用ユーザーに切り替える

```
# su - k8s_admin
```

2. docker.ioをインストールする

```
$ sudo apt install docker.io
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する。

3. dockerコマンドが使用できることを確認する

```
$ sudo docker version
```

「client」と「server」の後にバージョン情報が表示される場合、コマンドが正常に終了したと判断する。

※dockerのパッケージはdocker.io, docker.ce, docker.engineがあるが、本ガイドではUbuntu公式のリポジトリのdockerであるdocker.ioを使用する。

IBM API ConnectがサポートするDockerのVersion情報は[Software Product Compatibility Reports]で確認しておくこと。docker.ioはdocker.ceのVersion情報を読み替えること。

<https://www.ibm.com/software/reports/compatibility/clarity/index.html>



dockerコマンドのインストール(Worker Node)

「dockerコマンドのインストール(Master Node)」で実施した同様の操作を各 Worker Nodeにおいて実施する。

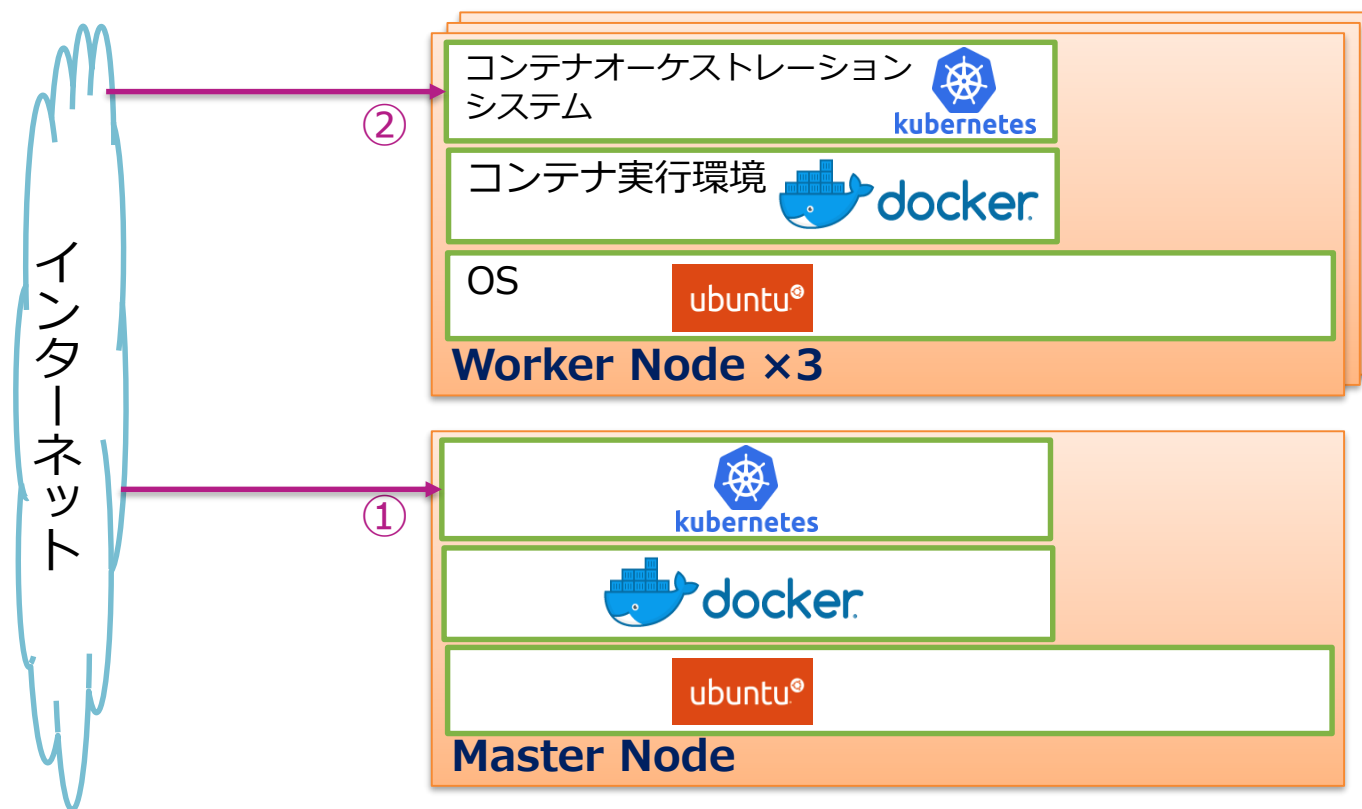


事前準備3

kubernetesコマンドのインストール

事前準備3

□ kubernetesコマンドのインストール



① Master Nodeにおいてコマンドでインターネットからkubernetesをインストールする。

② Worker Nodeにおいて上記の操作を行う。

kubernetesコマンドのインストール(Master Node)

1. kubernetes.listを作成し、以下の内容を記述する

```
$ sudo vim /etc/apt/sources.list.d/kubernetes.list
```

作成したkubernetes.listの例

```
/etc/apt/sources.list.d/kubernetes.list
```

```
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

2. Google Cloud Platformの公開鍵をインポートする

```
$ curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

コマンド実行後「OK」が確認される場合、コマンドが正常に終了したと判断する。

※上記の操作はapt updateを正常に実行させるために必要

3. リポジトリを更新する

```
$ sudo apt update
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する。

kubernetesコマンドのインストール(Master Node)

4. kubeletをコマンドを実行可能にさせるためにスワップをオフにする

```
$sudo swapoff -a
```

(補足)上記操作は再起動すると元に戻ってしまうため、永続的に変更する場合は「/etc/fstab」のswap領域の行をコメントアウトする必要がある

修正したfstabの例

/etc/fstab

```
# CLOUD_IMG: This file was created/modified by the Cloud Image build process
LABEL=cloudimg-bootfs /boot ext3 defaults,relatime 0 0
# LABEL=SWAP-xvdb1 none swap sw,comment=cloudconfig 0 2
/dev/xvdc1 /var ext4 defaults,noatime 0 2
```

5. kubernetesコマンドが使用できることを確認する

```
$ kubectl version
```

「client」と「server」の後にバージョン情報が表示される場合、コマンドが正常に終了したと判断する。

```
$ kubeadm version
```

「kubeadm version」の後にバージョン情報が表示される場合、コマンドが正常に終了したと判断する。



dockerコマンド, kubernetesコマンドのインストール(Worker Node)

「dockerコマンドのインストール(Master Node)」と「kubernetesコマンドのインストール(Master Node)」で実施した同様の操作を各Worker Nodeにおいて実施する。

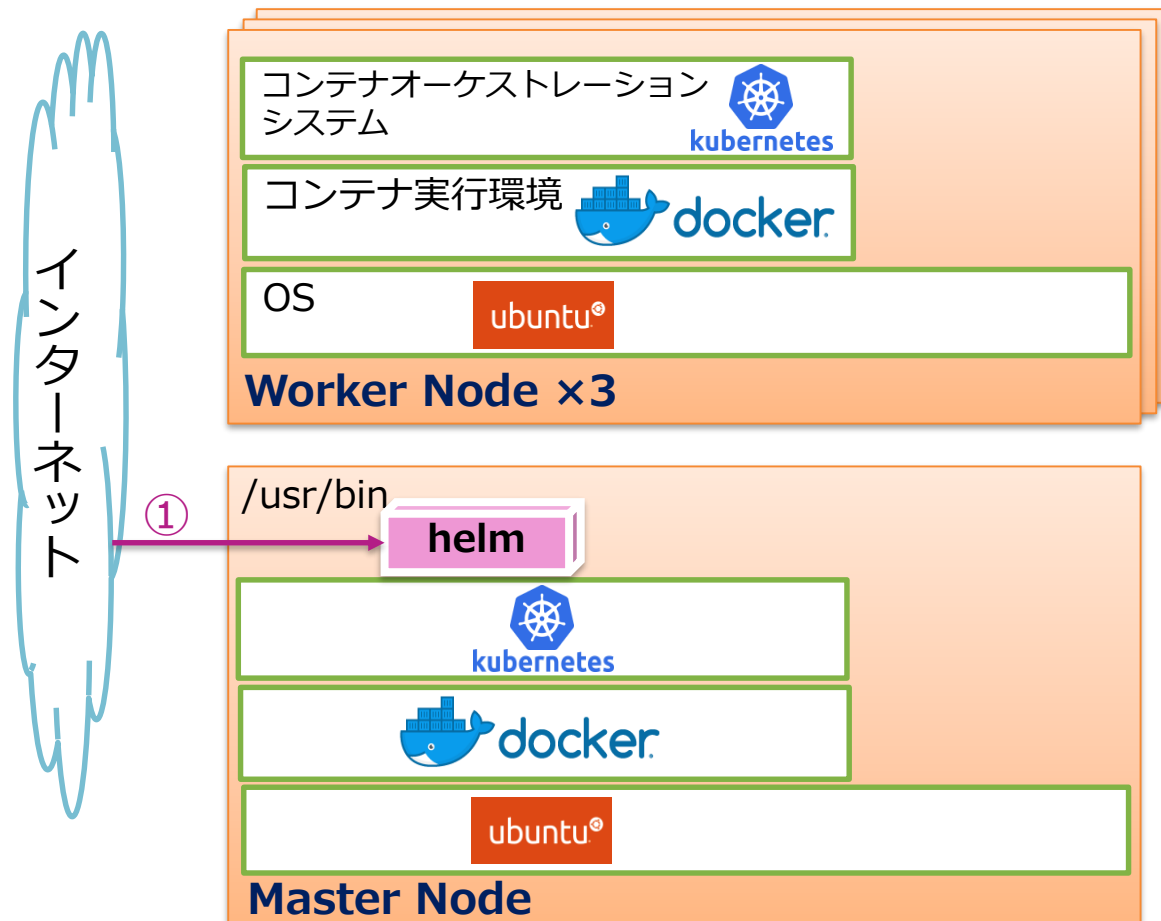


事前準備4

helmコマンドのインストール

事前準備4

□ helmコマンドのインストール



① helmをインターネットからダウンロードし、解凍した状態で /usr/bin のディレクトリー配下に格納する



helmコマンドのインストール(Master Node)

下記リンク先からHelmのバイナリーファイルをダウンロードする。

<https://github.com/helm/helm/releases>

1. 構築環境にアップロードして、解凍する

実行例(Master Node)

```
$ tar -xvf Helm-vx.xx.x-linux-amd64.tar.gz
linux-amd64/
linux-amd64/LICENSE
linux-amd64/helm
linux-amd64/README.md
```

2. linux-amd64/helmをusr/bin配下に移動させる

```
$ mv linux-amd64/helm /usr/bin/helm
```

3. helmコマンドが使用できることを確認する

```
$ helm version
```

「Client:」以降にバージョン情報が表示される場合、コマンドが正常に終了したと判断する。

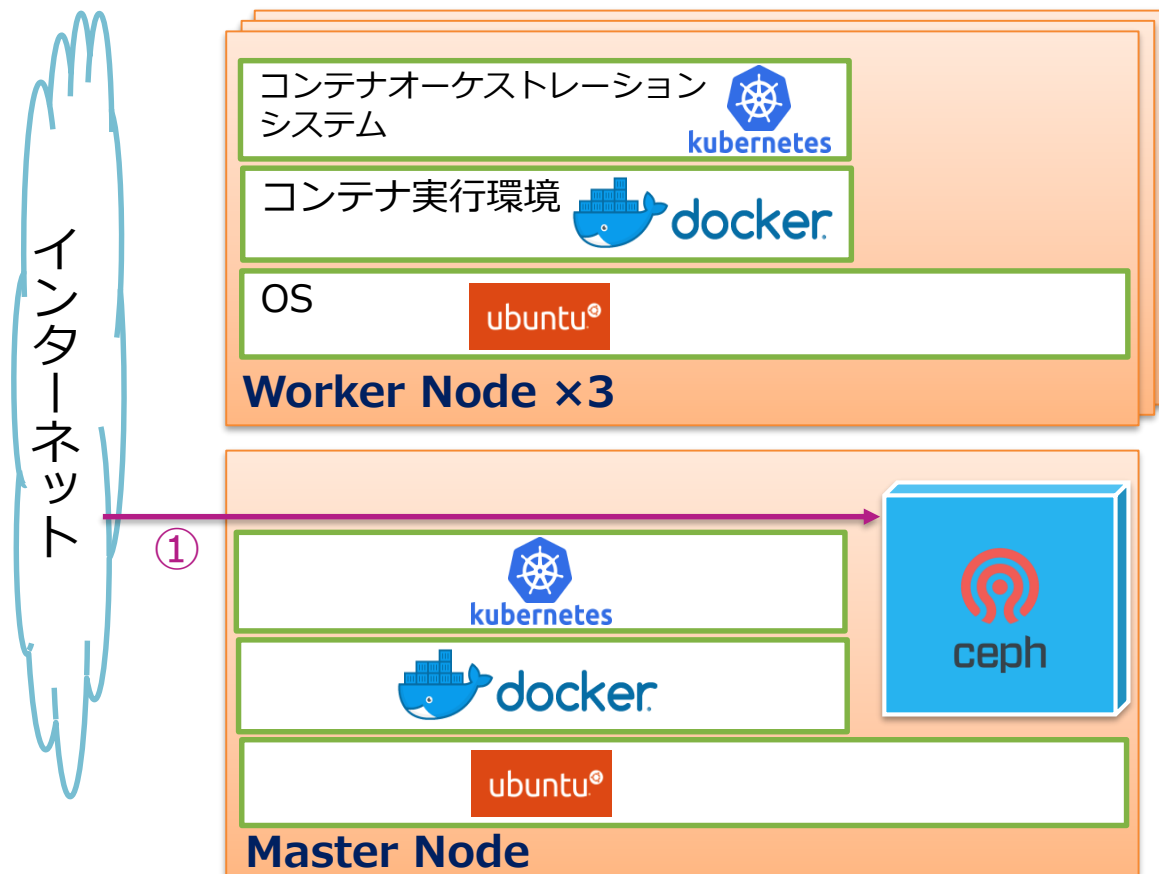


事前準備5

ceph-deployコマンドのインストール

事前準備5

□ ceph-deployコマンドのインストール



① コマンドでインターネットから ceph-deploy コマンドをインストールする（Master Nodeのみ）。

※ceph-deploy コマンドはcephの設定、クラスター作成に使用するコマンド

ceph-deployのインストール(Master Node)

1. リリースキーを取得する

```
$ wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-key add -
```

2. Cephのパッケージをリポジトリに追加する

```
$ echo deb https://download.ceph.com/debian-luminous/ $(lsb_release -sc) main | sudo tee  
/etc/apt/sources.list.d/ceph.list
```

3. リポジトリを更新する

```
$ sudo apt update
```

4. ceph-deployをインストールする

```
$ sudo apt install ceph-deploy
```



ceph-deployコマンドのインストール(Master Node)

1. ceph-deployコマンドが使用できることを確認する

```
$ ceph-deploy --version
```

バージョン情報が表示される場合、コマンドが正常に終了したと判断する。

※ceph-deployはCephと異なる独自のバージョンを持つ。

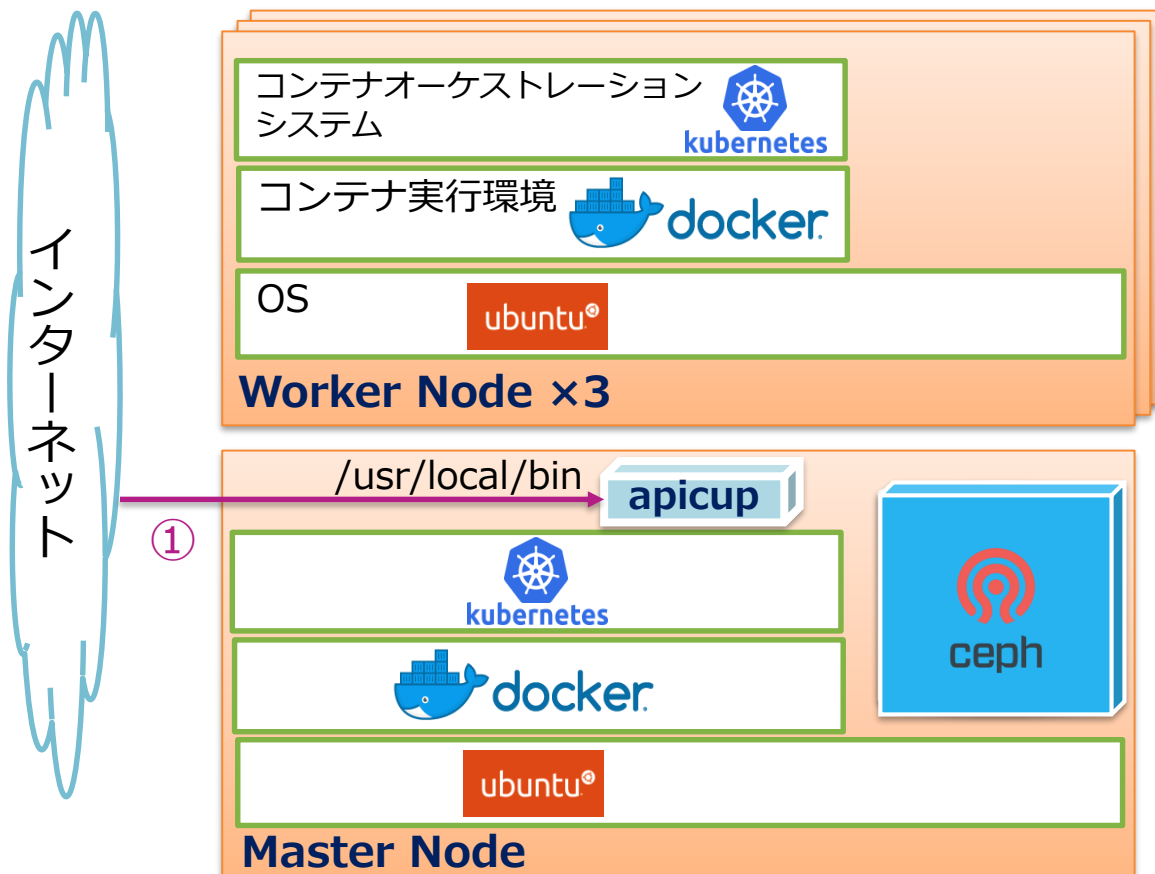


事前準備6

apicupコマンドのインストール

事前準備6

□ apicupコマンドのインストール



① apicup-linuxをインターネットからダウンロードし、
/usr/local/binディレクトリー配下に格納する。

apicupコマンドのインストール(Master Node)

1. FixCentra等から以下のコンポーネントをダウンロードする。

apicup-linux_lts_v20xx.x.x.x （本ガイドではv2018.4.1.1を使用）

2. ダウンロードしたファイルを配置する。

ダウンロードしたファイルに実行権限を付与し、パスの通ったディレクトリに配置する。
この際入力を簡易化するために「apicup」などにファイル名を変更しても問題ない。

```
$ chmod +x ./apicup-linux_lts_v2018.4.1.1  
$ mv ./apicup-linux_lts_v2018.4.1.1 /usr/local/bin/apicup
```

3. apicupコマンドが使用できることを確認する

```
$ apicup version
```

APIC v2018 Kubernetes版 導入前の環境設定(全サーバー)

API ManagementおよびAnalyticsではOSデフォルトのmmap(メモリマップ)の値より高い値(1048575以上)にする必要がある。以下に値の確認と設定変更の手順を記す。

1. 一つのプロセスで使えるメモリマップ数上限を確認

```
$ cat /proc/sys/vm/max_map_count
```

2. 1の値が1048575より小さい場合、mmapの設定値を変更する

```
$ sudo sysctl -w vm.max_map_count=1048575
```

(補足)上記操作は再起動すると元に戻ってしまうため、永続的に変更する場合は「/etc/sysctl.conf」の最終行にvm.max_map_countの値を追記する。

etc/sysctl.conf

~中略~

```
#fs.protected_hardlinks=0
```

```
#fs.protected_symlinks=0
```

```
vm.max_map_count = 1048575
```



IBM **API** Connect

kubernetesランタイム環境の構築1

Cephのセットアップ



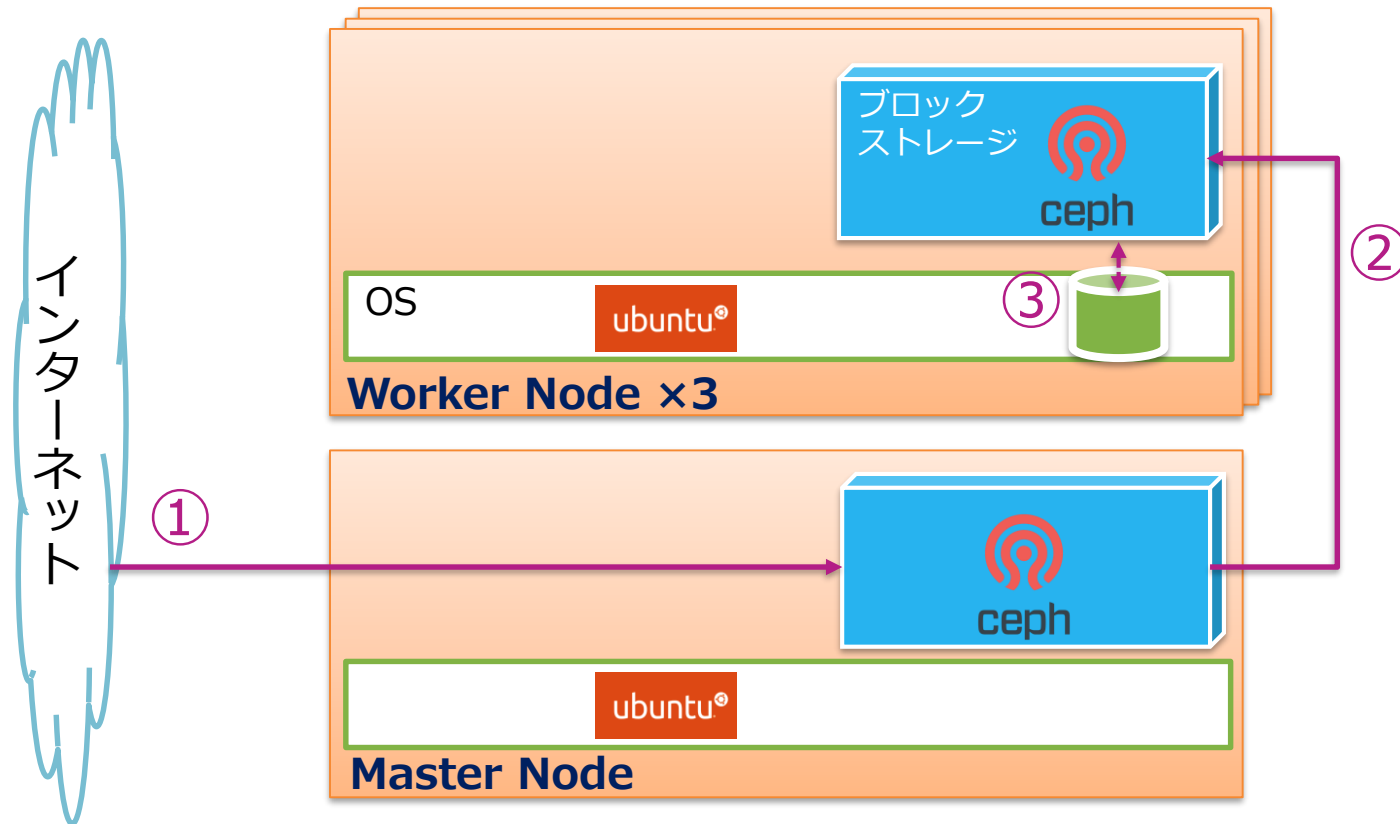
Cephのセットアップ

□ 前提

- ceph_deployを実行するサーバー、osdを作成するサーバーにcephコマンド実行ユーザーが作成されていること
 - ※osd:ディスクにデータを格納する際に利用されるDaemonプロセス
- cephコマンド実行ユーザーにsudoの権限があること
- ceph_deployを実行するサーバーからosdを作成するサーバーにcephコマンド実行ユーザーでパスワードなしSSH接続が可能であること
- ceph_deployを実行するサーバーに「ceph_deploy」コマンドがインストールされていること
- Cephに利用するディスクは、ファイルシステムが作成されていない状態でサーバーに接続すること
 - ※サーバーからは「/dev/xvdc」等で認識されているが、mkfsコマンドでファイルシステムを作成していない状態にする必要がある
- 本資料ではCeph 12.2.7 (luminous)を利用している
- 本資料ではceph実行ユーザーとしてceph_adminユーザーを利用している
 - 特に記載がない限り、コマンドはceph_adminで実行している
- 本資料ではceph_deployを実行するサーバーをMaster Node、Cephの管理サーバーをWorker1 Node、osdを作成するサーバーをWorker Node3台としている

kubernetesランタイム環境の構築2

□ cephのセットアップ



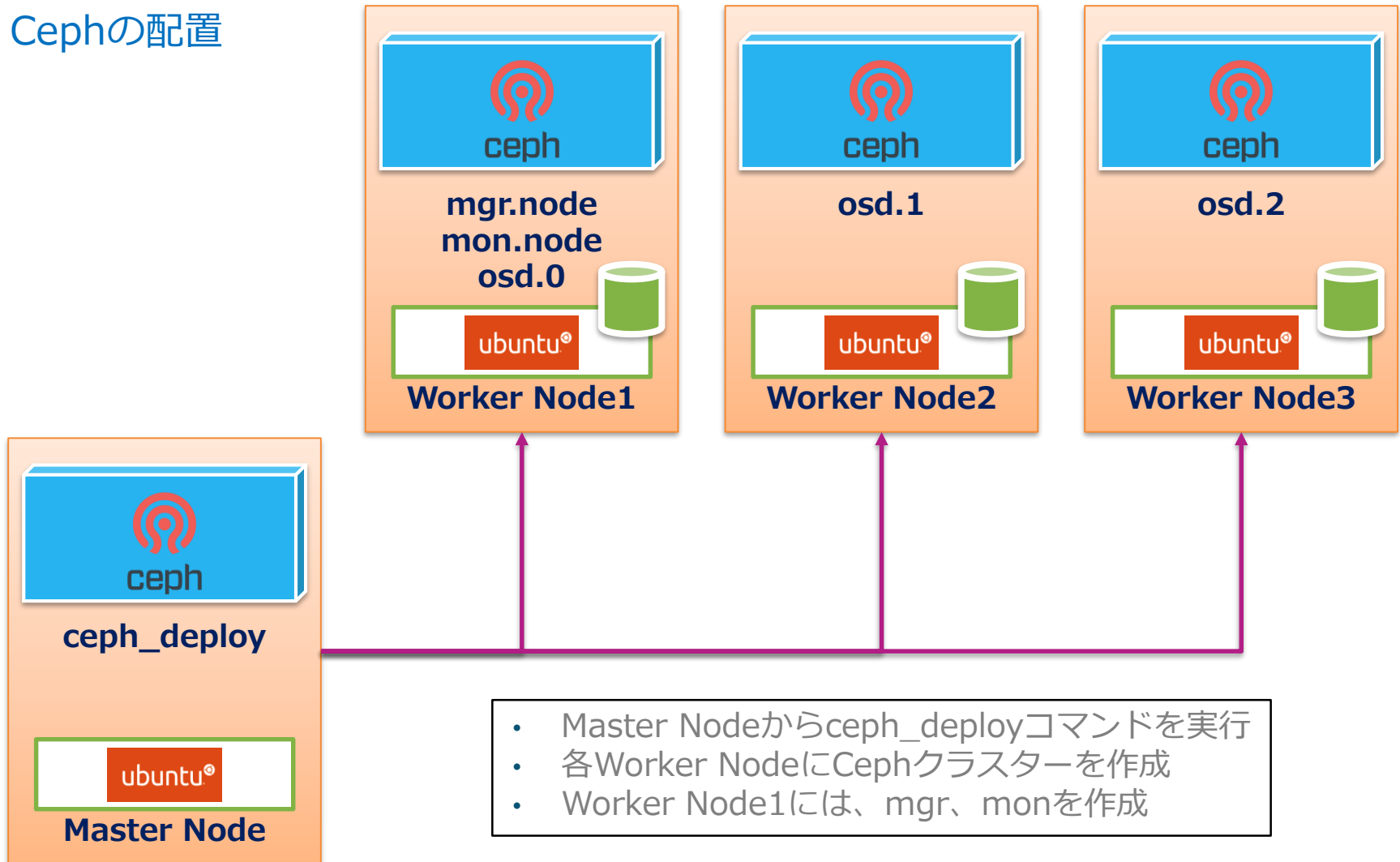
① Master NodeにコマンドでインターネットからCephをインストールする。

② 各Worker NodeにCephクラスターを作成する。

③ OSのファイルシステムからOSD(Object Storage Daemon)を割り当てる。

Cephのセットアップ

□ Cephの配置





Cephクラスタの作成(Master Node)

1. cephクラスタ用にディレクトリを作成(当ガイドではceph-clusterとする)

```
$ mkdir ceph-cluster  
$ cd ceph-cluster
```

2. クラスタを作成

```
$ ceph-deploy new (Cephの管理サーバー)
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する

3. ceph.confを編集する

```
$ vim ./ceph.conf
```

ceph.confは使用環境で2つ以上ネットワーク・インターフェースを持つ場合に、
globalセクションに内部通信に利用したいIPアドレスのレンジを追記する

public network = (内部通信に利用したいIPアドレス)/(bits)

Cephクラスタの作成(Master Node)

実行例(Master Node)

```
$ mkdir my-cluster
$ cd my-cluster
$ pwd
/home/ceph_admin/my-cluster
$ ceph-deploy new worker1
[ceph_deploy.conf][DEBUG ] found configuration file at:
/home/ceph_admin/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked (2.0.1): /usr/bin/ceph-deploy new worker1
[ceph_deploy.cli][INFO  ] ceph-deploy options:
--- 中略 ---
[ceph_deploy.new][DEBUG ] Writing monitor keyring to ceph.mon.keyring...
[ceph_deploy.new][DEBUG ] Writing initial config to ceph.conf...

$ vim ceph.conf
$ cat ceph.conf
[global]
--- 中略 ---
public network = 10.193.32.0/24
--- 以下略 ---
```

Cephを各Worker Nodeにインストール(Master Node)

1. Cephを各Worker Nodeにインストールする

```
$ ceph-deploy install --release luminous (worker1) (worker2) (worker3)
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する

2. monitorをインストールする

```
$ ceph-deploy mon create-initial
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する
monitorはCephの管理サーバーとして指定されたノードにインストールされ、osdの構成や状態監視を行う

3. adminをインストールする

```
$ ceph-deploy admin (worker1) (worker2) (worker3)
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する
この時点で、各osd作成用サーバーでcephコマンドが利用可能となる

4. managerをインストールする

```
$ ceph-deploy mgr create (worker1)
```

エラーなどの文字列がない場合、コマンドが正常に終了したと判断する
今後、Ceph mgr Nodeで実施と記述があった場合、ここで指定したNodeで実行する

Cephを各Worker Nodeにインストール(Master Node)

実行例(Master Node)

```
$ ceph-deploy install --release luminous worker1 worker2 worker3
--- 中略 ---
[worker3][INFO ] Running command: sudo ceph --version
[worker3][DEBUG ] ceph version 12.2.7 (3ec878d1e53e1aeb47a9f619c49d9e7c0aa384d5)
luminous (stable)

$ ceph-deploy mon create-initial
--- 中略 ---
[ceph_deploy.gatherkeys][INFO ] Storing ceph.bootstrap-rgw.keyring
[ceph_deploy.gatherkeys][INFO ] Destroy temp directory /tmp/tmp3u4k6H

$ ceph-deploy admin worker1 worker2 worker3
--- 中略 ---
[worker3][DEBUG ] detect machine type
[worker3][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.conf

$ ceph-deploy mgr create worker1
--- 中略 ---
[worker1][INFO ] Running command: sudo systemctl start ceph-mgr@worker1
[worker1][INFO ] Running command: sudo systemctl enable ceph.target
```

各Worker NodeにOSDを作成(Master Node)

1. 各Worker NodeにOSDを作成する

```
$ ceph-deploy osd create --data (worker1に接続されているディスクのパス) (worker1)
$ ceph-deploy osd create --data (worker2に接続されているディスクのパス) (worker2)
$ ceph-deploy osd create --data (worker3に接続されているディスクのパス) (worker3)
```

OSD(Object Storage Daemon)は、ディスクにデータを格納するためのDaemon
1つのディスクにつき、1つのプロセスが必要となる
本資料では3つのNodeに3つのディスクを配置し、それぞれをクラスタリングして利用する
「Host (Worker Node) is now ready for osd use.」
と表示された場合、コマンドが正常に終了したと判断する

各Worker NodeにOSDを作成(Master Node)

実行例(Master Node)

```
$ ceph-deploy osd create --data /dev/xvde worker1
```

```
--- 中略 ---
```

```
[worker1][INFO ] Running command: sudo /usr/bin/ceph --cluster=ceph osd stat --format=json
```

```
[ceph_deploy.osd][DEBUG ] Host worker1 is now ready for osd use.
```

```
$ ceph-deploy osd create --data /dev/xvde worker2
```

```
--- 中略 ---
```

```
[worker2][INFO ] Running command: sudo /usr/bin/ceph --cluster=ceph osd stat --format=json
```

```
[ceph_deploy.osd][DEBUG ] Host worker2 is now ready for osd use.
```

```
$ ceph-deploy osd create --data /dev/xvde worker3
```

```
--- 中略 ---
```

```
[worker3][INFO ] Running command: sudo /usr/bin/ceph --cluster=ceph osd stat --format=json
```

```
[ceph_deploy.osd][DEBUG ] Host worker3 is now ready for osd use.
```



Cephの稼動確認(Master Node)

1. ヘルスチェック

```
$ ssh (worker1) sudo ceph health
```

「HEALTH_OK」

と表示された場合、Cephのセットアップが正常に終了したと判断する

Cephの稼動確認(Master Node)

実行例(Master Node)

```
$ ssh worker1 sudo ceph health  
HEALTH_OK
```



kubernetesランタイム環境の構築2

kubernetesのクラスター・セットアップ



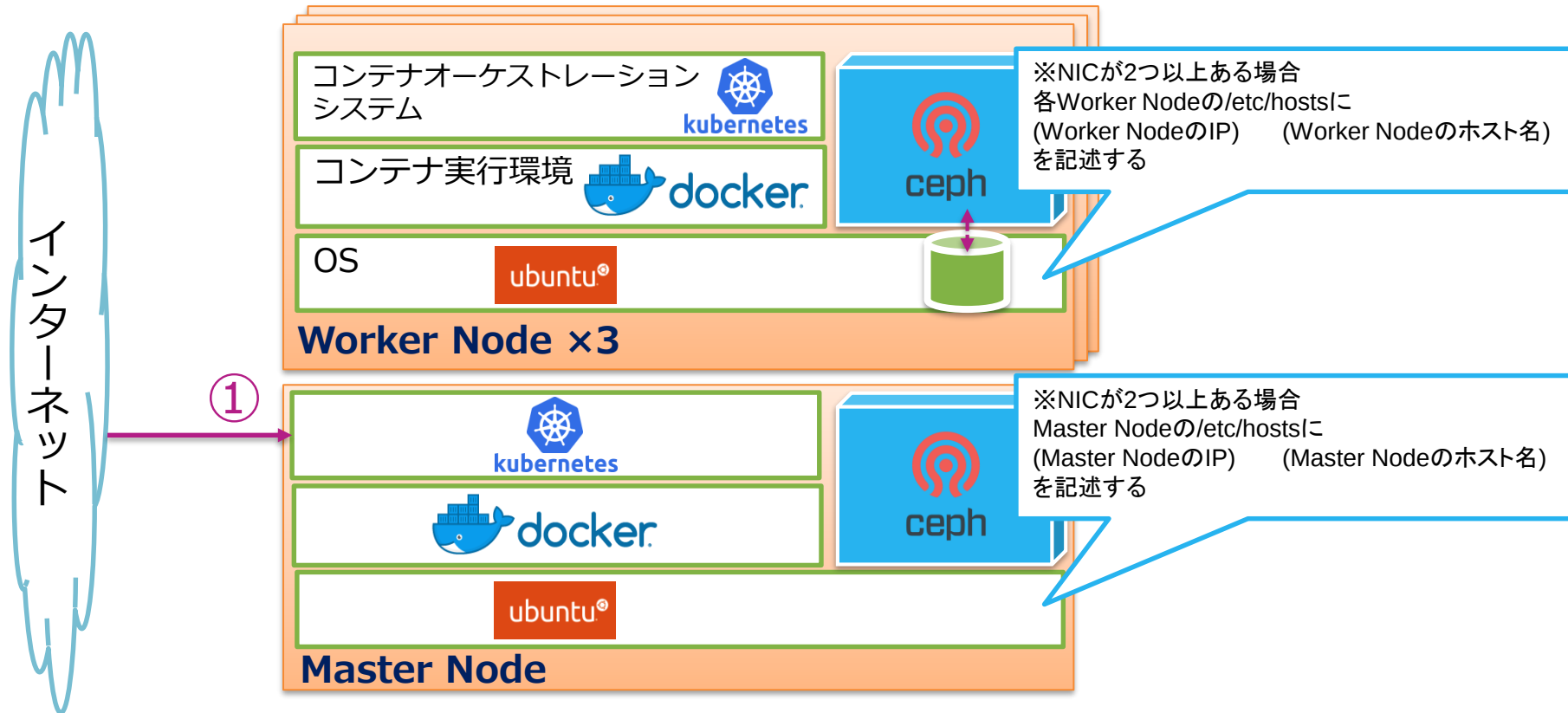
kubernetesのクラスター・セットアップ

□ 前提

- Master Node、Worker Nodeそれぞれにkubectlコマンド実行ユーザーが作成されていること
- kubectlコマンド実行ユーザーにsudoの権限があること
- kubeadmコマンドを実行するMaster NodeからWorker Nodeにkubectlコマンド実行ユーザーでパスワードなしSSH接続が可能であること
- kubeadmコマンドを実行するMaster Nodeに「kubeadm」コマンドがインストールされていること
- kubeadmコマンドを実行するMaster Nodeに「helm」コマンドがインストールされていること
- (NICが2つ以上ある場合)Master Node、Worker Nodeそれぞれの「/etc/hosts」に、自身のkubernetesのノード間通信で利用するIPとホスト名のペアが記載されていること
- 本資料ではkubernetes 1.12.3を利用している
- 本資料ではhelm 2.12.0を利用している
- 本資料ではkubernetesのネットワークプラグインとしてCalico 3.3を利用している
- 本資料ではkubectl実行ユーザーとしてk8s_adminユーザーを利用している
特に記載がない限り、コマンドはk8s_adminで実行している

kubernetesランタイム環境の構築2

□ kubernetesのクラスター・セットアップ



① Master Nodeでkubeadm initコマンドを実行し、クラスターを作成する。



Master Nodeのセットアップ(Master Node)

1. k8sをインストールする

```
$ sudo kubeadm init --apiserver-advertise-address=(Master NodeのIPアドレス) --pod-network-cidr=192.168.0.0/16  
$ mkdir -p $HOME/.kube  
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

ネットワークプラグインにCalicoを利用する場合、
--pod-network-cidrは「192.168.0.0/16」を指定する

「Your kubernetes master has initialized successfully!」
と表示された場合、コマンドが正常に終了したと判断する

「kubeadm init」が正常終了後表示される「kubeadm join ~」から始まる文字列は、Worker Nodeのセットアップで利用するため、保管しておくこと

サーバーにNICが2つ以上ある場合、kubeadmコマンドはデフォルトでデフォルト・ゲートウェイが付与されているNICをkubernetesのノード間通信に利用する
デフォルト・ゲートウェイが付与されていないNICをノード間通信に利用したい場合は、各サーバーの「/etc/hosts」に利用したいNICに付与されているIPアドレスと自身のホスト名を記載する必要がある **(DNSを利用している場合でも必須)**

Master Nodeのセットアップ(Master Node)

実行例(Master Node)

```
$ cat /etc/hosts
--- 中略 ---
10.193.32.164    master.apitechsalesjp.com    master

$ sudo kubeadm init --apiserver-advertise-address=10.193.32.164 --pod-network-
cidr=192.168.0.0/16
I1204 18:05:12.760065    23067 version.go:236] remote version is much newer:
v1.13.0;falling back to: stable-1.12
[init] using kubernetes version: v1.12.3
--- 中略 ---
Your kubernetes master has initialized successfully!

--- 中略 ---
You can now join any number of machines by running the following on each node
as root:

    kubeadm join 10.193.32.164:6443 --token xatvg9.n0ffnlkcw4aebpvl --discovery-
token-ca-cert-hash
sha256:b954d006a0276cd3656543db5f981881d6357f419a8379bb6697c70bc8bc782c
```

赤字部分を保管しておくこと



Master Nodeのセットアップ(Master Node)

実行例(Master Node)

```
$ mkdir -p $HOME/.kube  
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```



Master Nodeの稼働確認(Master Node)

1. Nodeの確認

```
$ kubectl get nodes
```

Master Nodeが作成されていることを確認する

この時点ではネットワークプラグインが作成されていないため、STATUSは「NotReady」となっている

Master Nodeの稼働確認(Master Node)

実行例(Master Node)

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	NotReady	master	2m39s	v1.12.3

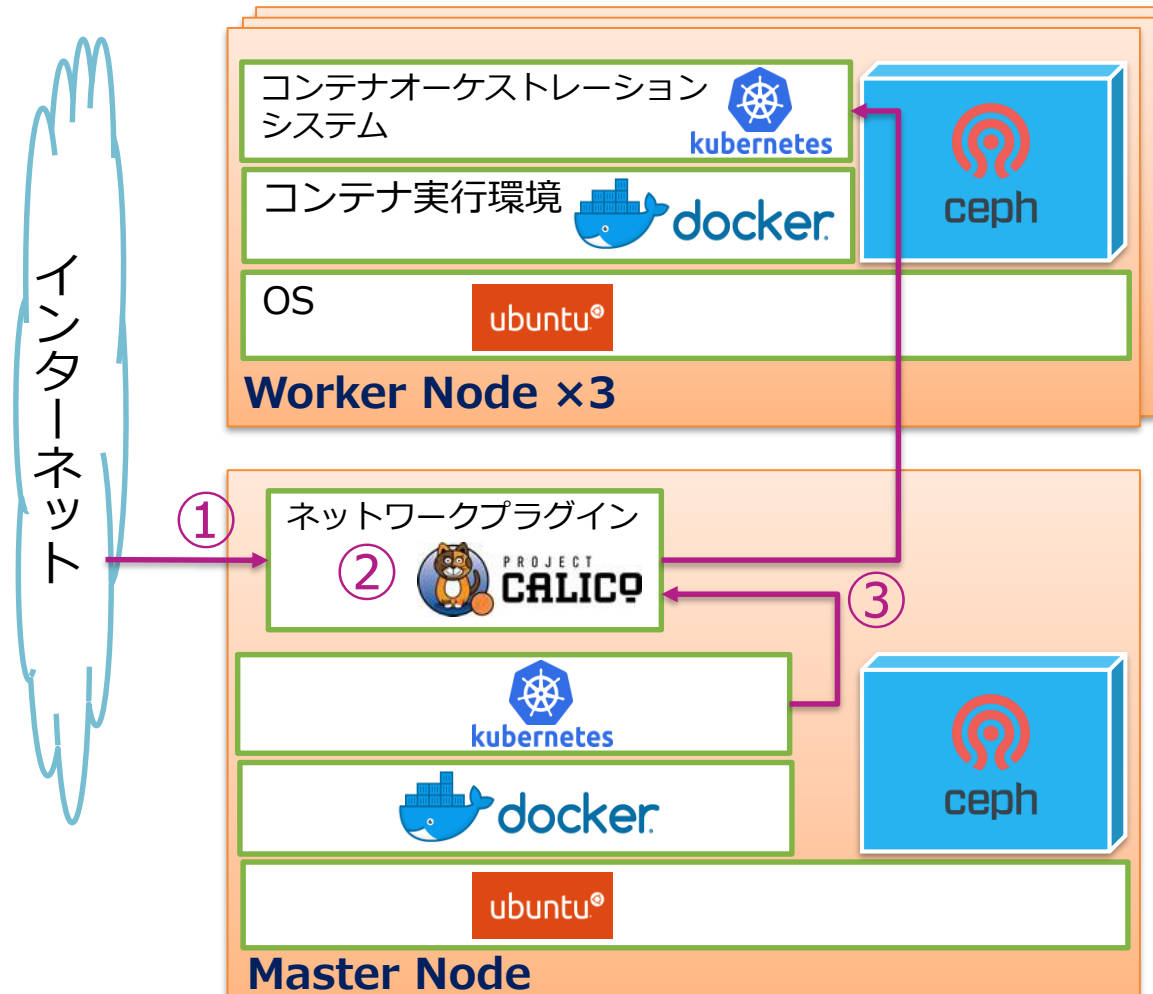


kubernetesランタイム環境の構築3

ネットワークプラグインのインストール

kubernetesランタイム環境の構築3

□ ネットワークプラグインのインストール



① コマンドでインターネットからcalico関連yamlをダウンロードする。

② 必要に応じてyamlを修正する。

③ kubectl applyコマンドで各Node間の通信できるようにする。

ネットワークプラグインのセットアップ(Master Node)

1. Calicoをインストールする

```
$ curl https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/etcd.yaml -O
$ curl https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/rbac.yaml -O
$ curl https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/calico.yaml -O
$ vim calico.yaml
$ kubectl apply -f etcd.yaml
$ kubectl apply -f rbac.yaml
$ kubectl apply -f calico.yaml
```

「〜〜 created」

と表示された場合、コマンドが正常に終了したと判断する

サーバーにNICが2つ以上ある場合、Calicoはデフォルトでデフォルト・ゲートウェイが付与されているNICをCalicoのノード間通信に利用する
デフォルト・ゲートウェイが付与されていないNICをノード間通信に利用したい場合は、「kubectl apply」コマンドを実施する前にcalico.yamlを修正する必要がある
修正箇所は次ページ以降に記載する

ネットワークプラグインのセットアップ(Master Node)

実行例(Master Node)

```
--- curlコマンドは省略 ---  
$ vim calico.yaml  
$ kubectl apply -f etcd.yaml  
daemonset.extensions/calico-etcd created  
service/calico-etcd created  
$ kubectl apply -f rbac.yaml  
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created  
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created  
clusterrole.rbac.authorization.k8s.io/calico-node created  
clusterrolebinding.rbac.authorization.k8s.io/calico-node created  
$ kubectl apply -f calico.yaml  
configmap/calico-config created  
secret/calico-etcd-secrets created  
daemonset.extensions/calico-node created  
serviceaccount/calico-node created  
deployment.extensions/calico-kube-controllers created  
serviceaccount/calico-kube-controllers created
```

ネットワークプラグインのセットアップ(Master Node)

修正したcalico.yaml(例)

yamlファイル

```
# Calico Version v3.3.2
# https://docs.projectcalico.org/v3.3/releases#v3.3.2
# This manifest includes the following component versions:

--- 中略 ---

  # Auto-detect the BGP IP address.
  - name: IP
    value: "autodetect"
  # Valid IP address on interface eth0, eth1, eth2 etc.
  - name: IP_AUTODETECTION_METHOD
    value: "interface=eth0"
  # Enable IPIP
  - name: CALICO_IPV4POOL_IPIP
    value: "Always"

--- 以下略 ---
```

赤字部分を追記

value部分にノード間通信で利用したいNICインターフェース名を記載する

ネットワークプラグインの稼働確認(Master Node)

1. Podの確認

```
$ kubectl get pods --all-namespaces
```

すべてのPodが「Running」となっている場合、ネットワークプラグインのセットアップが正常に終了したと判断する

2. Nodeの確認

```
$ kubectl get nodes
```

Master NodeのSTATUSが「Ready」となっている場合、ネットワークプラグインのセットアップが正常に終了したと判断する

ネットワークプラグインの稼働確認(Master Node)

実行例(Master Node)

```
$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS
AGE				
kube-system	calico-etcd-58n8c	1/1	Running	0 63s
kube-system	calico-kube-controllers-85cf9c8b79-5k4h2	1/1	Running	0 78s
kube-system	calico-node-n7466	2/2	Running	2 78s
kube-system	coredns-576cbf47c7-5swcf	1/1	Running	0 12m
kube-system	coredns-576cbf47c7-nvcdj	1/1	Running	0 12m
kube-system	etcd-master	1/1	Running	0 63s
kube-system	kube-apiserver-master	1/1	Running	0 64s
kube-system	kube-controller-manager-master	1/1	Running	0 64s
kube-system	kube-proxy-4ftd4	1/1	Running	0 12m
kube-system	kube-scheduler-master	1/1	Running	0 59s

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	15m	v1.12.3



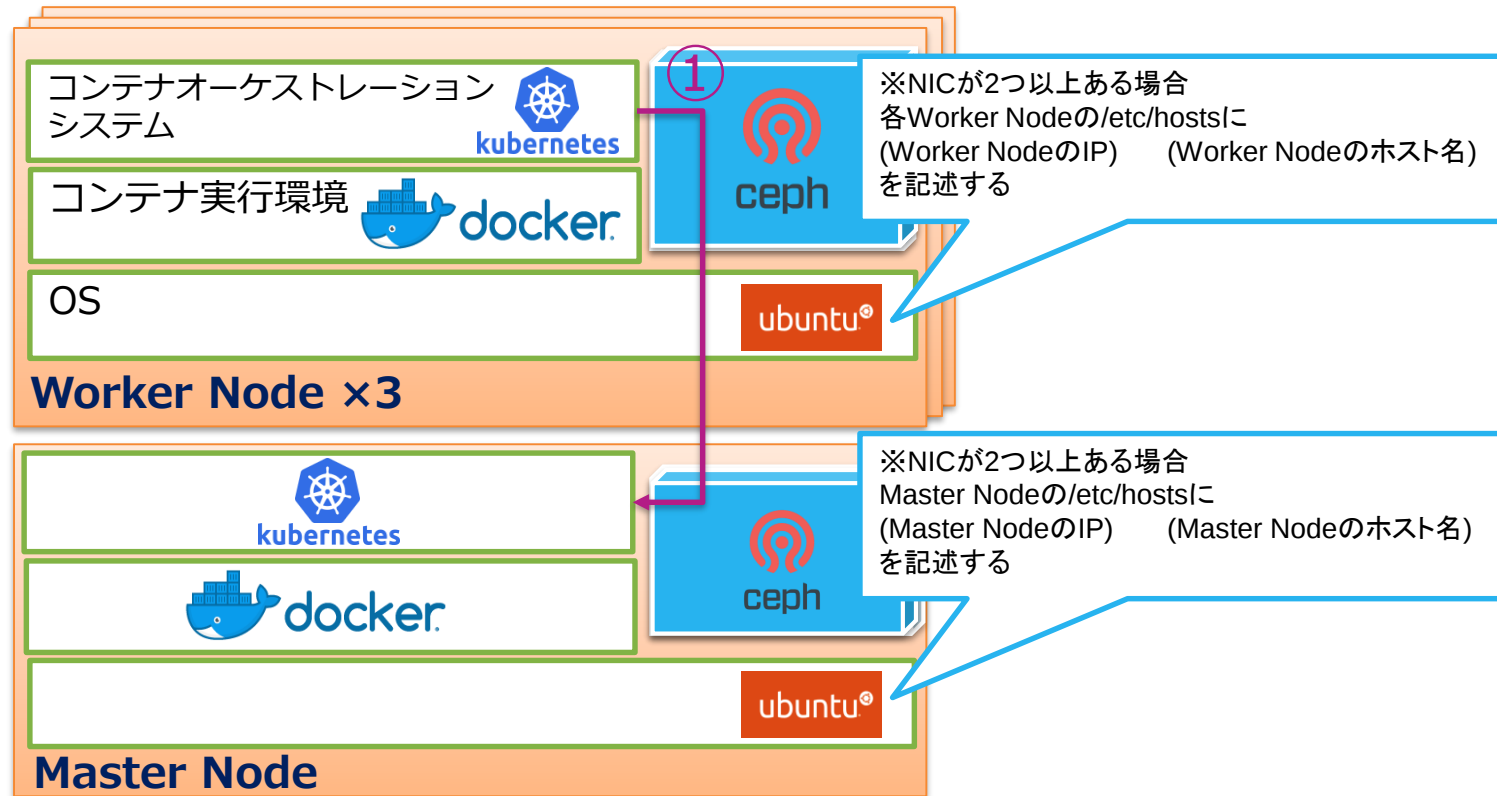
kubernetesランタイム環境の構築4

Worker Nodeのセットアップ

kubernetesランタイム環境の構築4

Worker Nodeのセットアップ

インターネット



① コマンドで各Worker Nodeをクラスターに参加させる。



Worker Nodeのセットアップ(Worker Node)

1. クラスターに参加する

\$ sudo (事前にMaster Nodeで「kubeadm init」コマンドを実行した際に表示されたコマンド)

利用するすべてのWorker Nodeで実行する

実際のコマンドは

```
sudo kubeadm join (Master NodeのIPアドレス):6443 --token (ランダムな文字列)
--discovery-token-ca-cert-hash sha256(ランダムな文字列)
```

のような形式となる

「This node has joined the cluster:」

と表示された場合、コマンドが正常に終了したと判断する

サーバーにNICが2つ以上ある場合、kubeadmコマンドはデフォルトでデフォルト・ゲートウェイが付与されているNICをkubernetesのノード間通信に利用する

デフォルト・ゲートウェイが付与されていないNICをノード間通信に利用したい場合は、各サーバーの「/etc/hosts」に利用したいNICに付与されているIPアドレスと自身のホスト名を記載する必要がある **(DNSを利用している場合でも必須)**

Worker Nodeのセットアップ(Worker Node)

実行例(Worker Node)

```
$ cat /etc/hosts
```

```
--- 中略 ---
```

```
10.193.32.169    worker1.apitechsalesjp.com    worker1
```

```
$ sudo kubeadm join 10.193.32.164:6443 --token xatvg9.n0ffnlkcw4aebpvl --discovery-  
token-ca-cert-hash
```

```
sha256:b954d006a0276cd3656543db5f981881d6357f419a8379bb6697c70bc8bc782c
```

```
[preflight] running pre-flight checks
```

```
--- 中略 ---
```

This node has joined the cluster:

- * Certificate signing request was sent to apiserver and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.



Worker Nodeの稼働確認(Master Node)

1. Nodeの確認

```
$ kubectl get nodes
```

Worker Nodeが追加されていることを確認する

Worker Nodeの稼働確認(Master Node)

実行例(Master Node)

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	20m	v1.12.3
worker1	Ready	<none>	14m	v1.12.3
worker2	Ready	<none>	13m	v1.12.3
worker3	Ready	<none>	12m	v1.12.3

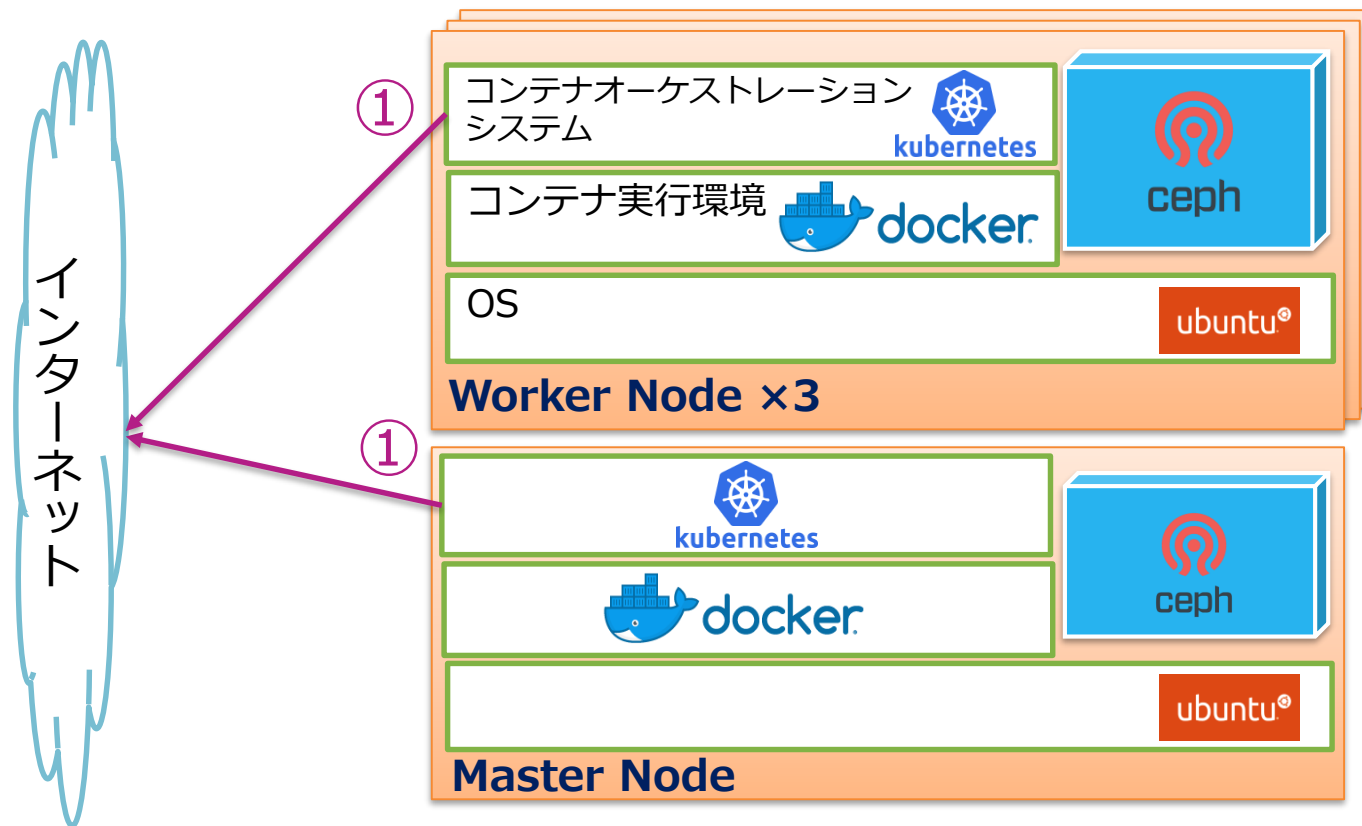


kubernetesランタイム環境の構築5

corednsの修正

kubernetesランタイム環境の構築5

□ corednsの修正



① corednsのpodを修正し、各ノードが外部のDNSを参照可能にする。

corednsの修正(Master Node)

1. corednsの設定確認

```
$ kubectl get configmap coredns -n kube-system -o yaml > ./coredns.yaml
```

kubernetesではNode間の通信の名前解決をするためにcorednsという機能を利用している
corednsではデフォルトでkubernetesが稼働しているサーバーの「/etc/resolv.conf」を参照し、
利用しているが今回は直接外部DNSを参照するように修正する

2. coredns.yamlファイルの編集

```
$ vim ./coredns.yaml
```

corednsの設定ファイルを修正し、本資料でAPI Connectサーバーに利用しているドメインの場合
に外部DNSを参照する設定を追加する

3. coredns.yamlファイルの適用、確認

```
$ kubectl apply -f ./coredns.yaml  
$ kubectl describe configmap -n kube-system coredns
```

適用後再度corednsの設定を確認することで、適用されていることを確認する

corednsの修正(Master Node)

実行例(Master Node)

```
$ kubectl get configmap coredns -n kube-system -o yaml > ./coredns.yaml
$ vim coredns.yaml
$ kubectl apply -f ./coredns.yaml
configmap/coredns configured

$ kubectl describe configmap -n kube-system coredns
--- 中略 ---
apitechsalesjp.com:53 {
    errors
    proxy . 10.193.32.164
    cache 30
    loop
    reload
}
--- 以下略 ---
```

corednsの修正(Master Node)

修正したcoredns.yaml(例)

yamlファイル

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    (API Connectで利用するドメイン):53 {
      errors
      proxy . (DNSサーバーのIPアドレス)
      cache 30
      loop
      reload
    }

    .:53 {
    --- 以下略 ---
```

赤字部分を追記

API Connectで利用するドメインからのDNSリクエストが来た場合に、(DNSサーバーのIPアドレス)で記載したIPのDNSにリダイレクトする、という設定になっている



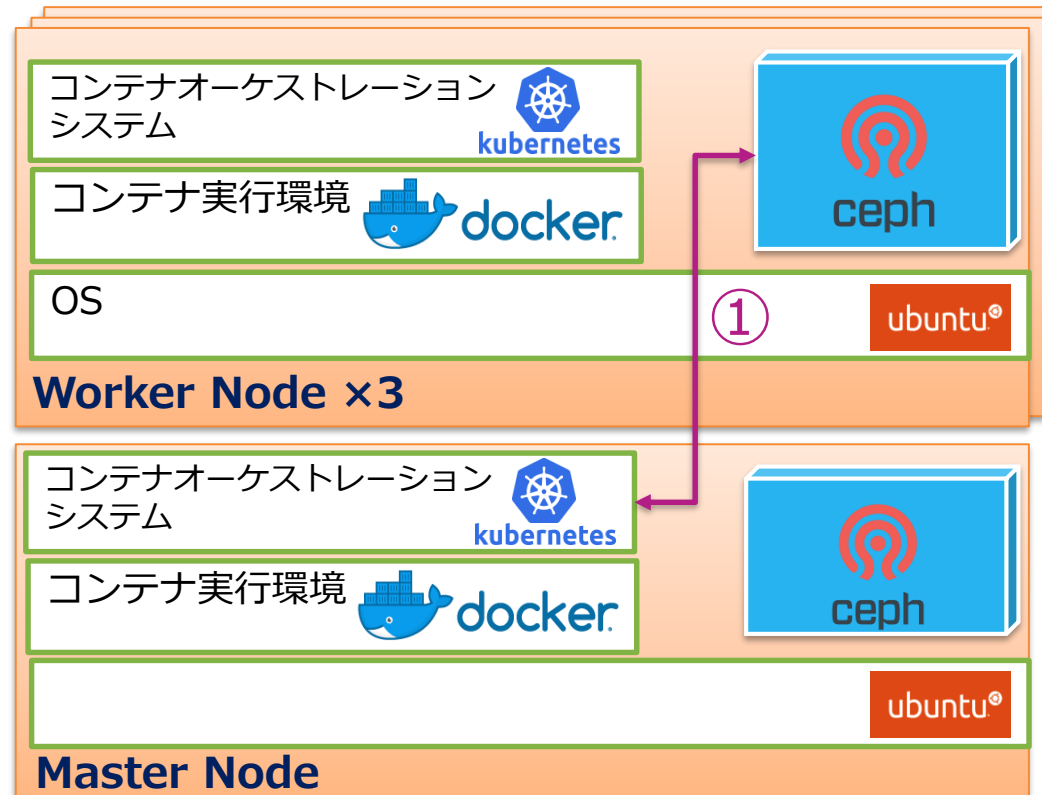
kubernetesランタイム環境の構築6

ストレージのセットアップ

kubernetesランタイム環境の構築6

□ ストレージのセットアップ

インターネット



① コマンドでCephをPersistent Volumeとして作成する。



ストレージのセットアップ(Master Node)

1. API Connect用のnamespaceを作成する

```
$ kubectl create namespace apiconnect
```

2. rbd-provisioner用の権限を作成する

```
$ kubectl create clusterrolebinding add-on-cluster-admin --clusterrole=cluster-admin --  
serviceaccount=apiconnect:default
```

3. rbd-provisionerをインストールする

```
$ kubectl apply -n apiconnect -f ./StorageRBAC.yaml  
$ kubectl get pods -l app=rbd-provisioner -n apiconnect
```

kubernetes環境とCephストレージを紐付けるために、rbd-provisionerが必要になる
rbd-provisionerのpodが作成されていることを確認する

ストレージのセットアップ(Master Node)

実行例(Master Node)

```
$ kubectl create namespace apiconnect
namespace/apiconnect created
$ kubectl create clusterrolebinding add-on-cluster-admin --clusterrole=cluster-admin --serviceaccount=apiconnect:default
clusterrolebinding.rbac.authorization.k8s.io/add-on-cluster-admin created
$ kubectl apply -n apiconnect -f ./StorageRBAC.yaml
clusterrole.rbac.authorization.k8s.io/rbd-provisioner created
clusterrolebinding.rbac.authorization.k8s.io/rbd-provisioner created
role.rbac.authorization.k8s.io/rbd-provisioner created
rolebinding.rbac.authorization.k8s.io/rbd-provisioner created
serviceaccount/rbd-provisioner created
deployment.extensions/rbd-provisioner created
$ kubectl get pods -l app=rbd-provisioner -n apiconnect
```

NAME	READY	STATUS	RESTARTS	AGE
rbd-provisioner-5d6b9c6b89-4c655	1/1	Running	0	66s

ストレージのセットアップ(Master Node)

作成したStorageRBAC.yaml(例)

yamlファイル

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rbd-provisioner
  namespace: apiconnect
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: [""]
    resources: ["services"]
    resourceNames: ["kube-dns"]
    verbs: ["list", "get"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rbd-provisioner
  namespace: apiconnect
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rbd-provisioner
subjects:
  - kind: ServiceAccount
    name: default
    namespace: apiconnect
---
kind: ClusterRole
name: rbd-provisioner
subjects:
  - kind: ServiceAccount
    name: default
    namespace: apiconnect
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: rbd-provisioner
  namespace: apiconnect
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rbd-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rbd-provisioner
subjects:
  - kind: ServiceAccount
    name: default
    namespace: apiconnect
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: rbd-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: rbd-provisioner
    spec:
      containers:
        - name: rbd-provisioner
          image: "quay.io/external_storage/rbd-provisioner:latest"
          env:
            - name: PROVISIONER_NAME
              value: ceph.com/rbd
```

kubernetes環境とCephストレージを紐付けるためのrbd-provisionerのアクセス権限を定義している namespaceにAPI Connectで利用するNameSpaceを指定すること
rbd-provisionerのバージョンは、赤字で示したimageで指定する
本資料では、latest(v2.1.1-k8s1.11)で動作確認している

ストレージのセットアップ(Ceph mgr Node)

4. API Connect用のブロックストレージを作成する

```
$ sudo ceph --cluster ceph osd pool create apic-pool (pgサイズ) (pgpサイズ)  
$ sudo ceph osd pool application enable apic-pool rbd
```

apic-poolのpgサイズ、pgpサイズは、環境の大きさによって変更する

【参考資料:A preselection of pg_num】

<http://docs.ceph.com/docs/luminous/rados/operations/placement-groups/>

5. ceph-secret-kube用の鍵を作成する

```
$ sudo ceph --cluster ceph auth get-or-create client.kube mon 'allow r' osd 'allow rwx  
pool=apic-pool'  
$ sudo ceph --cluster ceph auth get-key client.kube
```

auth get-key client.kubeで表示された鍵は利用するため、保管しておくこと
鍵はCephとkubernetes間の通信に利用される

6. ceph-secret用の鍵を作成する

```
$ sudo ceph --cluster ceph auth get-key client.admin
```

表示された鍵は利用するため、保管しておくこと
鍵はCephとkubernetes間の通信（管理）に利用される

ストレージのセットアップ(Ceph mgr Node)

実行例(Ceph mgr Node)

```
$ sudo ceph --cluster ceph osd pool create apic-pool 128 128
pool 'apic-pool' created
$ sudo ceph osd pool application enable apic-pool rbd
enabled application 'rbd' on pool 'apic-pool'

$ sudo ceph --cluster ceph auth get-or-create client.kube mon 'allow r' osd 'allow
rwx pool=apic-pool'
[client.kube]
    key = AQBubA9cf10jHhAAIZLXkaz8KjypsTbDRRRP3w==
$ sudo ceph --cluster ceph auth get-key client.kube
AQBubA9cf10jHhAAIZLXkaz8KjypsTbDRRRP3w==

$ sudo ceph --cluster ceph auth get-key client.admin
AQAfKAZcZXZMFhAAGafeefTT76NTirernqQXwQ==
```

赤字部分を保管しておくこと

ストレージのセットアップ(Master Node)

7. ストレージ用の鍵を登録する

```
$ kubectl create secret generic ceph-secret-kube ¥  
  --type="ceph.com/rbd" ¥  
  --from-literal=key='(ceph-secret-kube用鍵)' ¥  
  --namespace=apiconnect  
$ kubectl create secret generic ceph-secret ¥  
  --type="ceph.com/rbd" ¥  
  --from-literal=key=(ceph-secret用の鍵)' ¥  
  --namespace=apiconnect
```

「〜〜 created」
と表示された場合、コマンドが正常に終了したと判断する

ストレージのセットアップ(Master Node)

実行例(Master Node)

```
$ kubectl create secret generic ceph-secret-kube ¥
  --type="ceph.com/rbd" ¥
  --from-literal=key=' AQBubA9cf10jHhAAIZLXkaz8KjypsTbDRRRP3w==' ¥
  --namespace=apiconnect
secret/ceph-secret-kube created
$ kubectl create secret generic ceph-secret ¥
  --type="ceph.com/rbd" ¥
  --from-literal=key=' AQAFKAZcZXZMFhAAGafeefTT76NTirernqQXwQ==' ¥
  --namespace=apiconnect
secret/ceph-secret created
```

赤字部分に保管した鍵を使用する

ストレージのセットアップ(Master Node)

8. kubernetesとCephを紐付ける

```
$ kubectl create -f ./StorageClass.yaml
```

「〜〜 created」

と表示された場合、コマンドが正常に終了したと判断する



ストレージのセットアップ(Master Node)

実行例(Master Node)

```
$ kubectl create -f ./StorageClass.yaml  
storageclass.storage.k8s.io/apic-rbd created
```

ストレージのセットアップ(Master Node)

作成したStorageClass.yaml(例)

yamlファイル

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: apic-rbd
provisioner: ceph.com/rbd
parameters:
  monitors: (Ceph mgr NodeのIPアドレス):6789
  adminId: admin
  adminSecretName: ceph-secret
  adminSecretNamespace: apiconnect
  pool: apic-pool
  userId: kube
  userSecretName: ceph-secret-kube
  userSecretNamespace: apiconnect
  imageFormat: "2"
  imageFeatures: layering
```

赤字で示したnameがAPI Connectで利用するStorageClassとなる
adminSecretName、userSecretNameにはそれぞれceph-secret用の鍵名とceph-secret-kube用の鍵名
を記載する



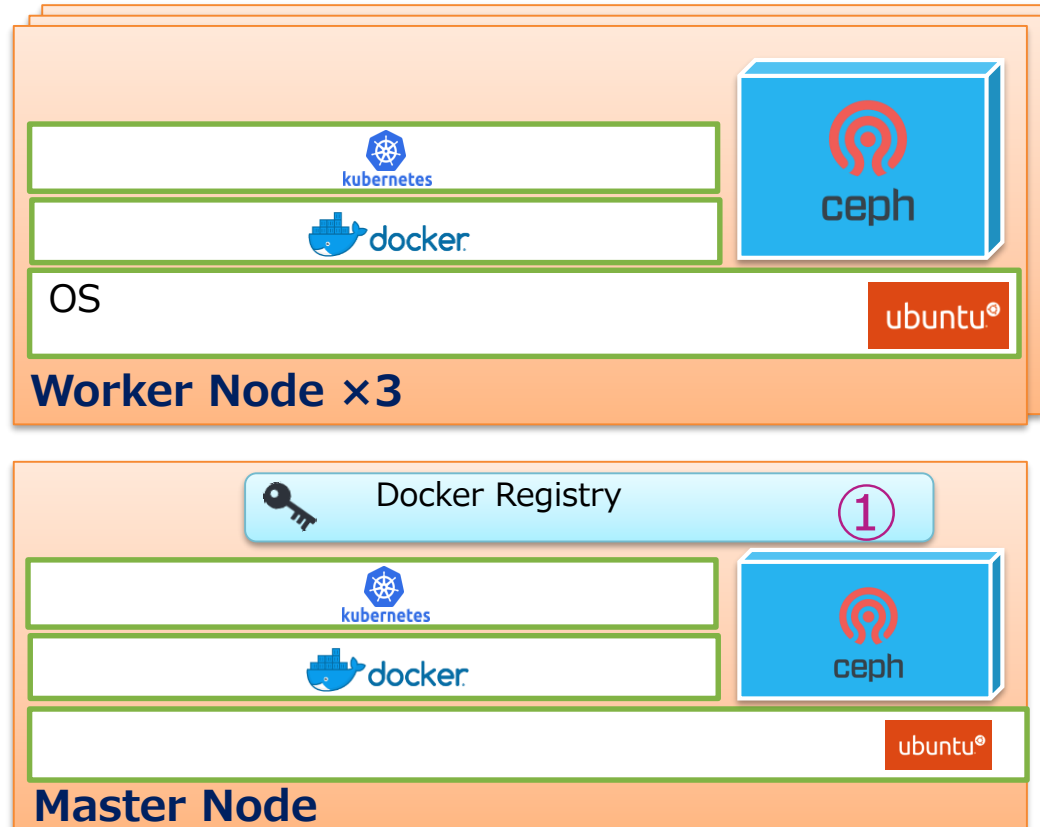
kubernetesランタイム環境の構築7

Docker Repositoryのセットアップ

kubernetesランタイム環境の構築7

❑ Docker Repositoryのセットアップ

インターネット



① Docker Registryを作成し、API Connectのベースイメージを保管可能にする。

Docker Repositoryのセットアップ(Master Node)

1. Docker Repositoryで利用するディレクトリーの作成

```
$ sudo mkdir -p /var/opt/docker-registry/auth  
$ sudo mkdir -p /var/opt/docker-registry/data
```

API Connect用のベースイメージが配置されるため10GB程度の空き容量のあるファイルシステムに作成する

2. Docker Repositoryにアクセスするためのパスワードの設定

```
$ sudo docker run ¥  
  --entrypoint htpasswd ¥  
  registry:2 -Bbn admin (パスワード) > /var/opt/docker-registry/auth/htpasswd
```

3. Docker Repositoryにアクセスするための証明書の設定

```
$ openssl req ¥  
  -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key ¥  
  -x509 -days 365 -out certs/domain.crt
```

Worker NodeからDocker Repositoryにアクセスする場合、パスワードの他にSSLアクセスのための証明書が必要になる

Docker Repositoryのセットアップ(Master Node)

実行例(Master Node)

```
$ sudo mkdir -p /var/opt/docker-registry/auth
$ sudo mkdir -p /var/opt/docker-registry/data
$ sudo docker run ¥
  --entrypoint htpasswd ¥
  registry:2 -Bbn admin APIC1234! > /var/opt/docker-registry/auth/htpasswd
--- 中略 ---
Status: Downloaded newer image for registry:2
$ openssl req ¥
  -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key ¥
  -x509 -days 365 -out certs/domain.crt
Generating a 4096 bit RSA private key
..... ++
--- 中略 ---
Country Name (2 letter code) [AU]:JP
State or Province Name (full name) [Some-State]:Tokyo
Locality Name (eg, city) []:Hakozaki
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ISE
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:(Docker RepositoryのIPアドレス)
Email Address []:root@master.apitechsalesjp.com
```

Docker Repositoryのセットアップ(Master Node)

4. 証明書の配置

```
$ ls certs
$ cat ./certs/domain.crt
$ mkdir -p /etc/docker/certs.d/master.apitechsalesjp.com:5000
$ sudo cp -p ./certs/domain.crt /etc/docker/certs.d/master.apitechsalesjp.com:5000/ca.crt
```

domain.crtファイルを各Worker Nodeの
「/etc/docker/certs.d/master.apitechsalesjp.com:5000」
ディレクトリーに「ca.crt」ファイルとして配置する

5. Docker Repositoryにパスワードなしでログイン可能にする設定

```
$ kubectl create secret docker-registry (Registryシークレット名) ¥
--docker-server=(Master Nodeのホスト名):5000 --docker-username=admin ¥
--docker-password=(パスワード) --docker-email=(Eメールアドレス) ¥
-n apiconnect
```

**API Connectのインストールをする際に必要になる
Registryシークレット名はapicupコマンドを利用する際に指定するので保管しておくこと**

Docker Repositoryのセットアップ(Master Node)

実行例(Master Node)

```
$ ls certs
domain.crt  domain.key
$ cat ./certs/domain.crt
-----BEGIN CERTIFICATE-----
MIIF+jCCA+KgAwIBAgIJA0vWWfI2Wso2MAOGCSqGSIb3DQEBCwUAMIGRMQswCQYD
VQQGEwJKUDEOMAwGA1UECAwFVG9reW8xETAPBgNVBACMEhha296YWtpMQwwCgYD
--- 中略 ---
dpJ1GAeuy5xmks03sEgtL+HK1e+TaRsP2AhETFBMJCBU+fg0r02cxPk5xhdhDmNV
dIt6S04uQxpI Z5563whWBppU6zn8OG4AK6p2uZzG1TaBdcbzC9cR3mW74ST3Tw==
-----END CERTIFICATE-----
$ mkdir -p /etc/docker/certs.d/master.apitechsalesjp.com:5000
$ sudo cp -p ./certs/domain.crt ¥
/etc/docker/certs.d/master.apitechsalesjp.com:5000/ca.crt
$ kubectl create secret docker-registry my-localreg-secret ¥
--docker-server=master.apitechsalesjp.com:5000 --docker-username=admin ¥
--docker-password=APIC1234! --docker-email=root@master.apitechsalesjp.com ¥
-n apiconnect
secret/my-localreg-secret created
```

赤字部分をRegistryシークレット名として保管しておくこと

Docker Repositoryのセットアップ(Worker Node)

実行例(Worker Node)

```
$ mkdir -p /etc/docker/certs.d/master.apitechsalesjp.com:5000
--- 中略 ---
$ ls /etc/docker/certs.d/master.apitechsalesjp.com:5000
ca.crt
$ cat /etc/docker/certs.d/master.apitechsalesjp.com:5000/ca.crt
-----BEGIN CERTIFICATE-----
MIIF+jCCA+KgAwIBAgIJA0vWWfI2Wso2MAOGCSqGSIb3DQEBCwUAMI GRMQswCQYD
VQQGEwJKUDEOMAwGA1UECAwFVG9reW8xETAPBgNVBACMEhha296YWtpMQwwCgYD
--- 中略 ---
dpJ1GAeuy5xmks03sEgtL+HK1e+TaRsP2AhETFBMJCBU+fg0r02cxPk5xhdhDmNV
dIt6S04uQxpI Z5563whWBppU6zn80G4AK6p2uZzG1TaBdcbzC9cR3mW74ST3Tw==
-----END CERTIFICATE-----
```

赤字部分でscp、sftpなどでMaster Nodeのdomain.crtを各Worker Nodeに
 /etc/docker/certs.d/master.apitechsalesjp.com:5000/ca.crt
 としてコピーすること



Docker Repositoryのセットアップ(Master Node)

6. Docker Repositoryコンテナの作成

```
$ sudo docker run -d ¥  
-p 5000:5000 ¥  
--restart=always ¥  
--name registry ¥  
-v /var/opt/docker-registry/auth:/auth ¥  
-v /var/opt/docker-registry/data:/var/lib/registry ¥  
-v `pwd`/certs:/certs ¥  
-e "REGISTRY_AUTH=htpasswd" ¥  
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" ¥  
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd ¥  
-e REGISTRY_HTTP_ADDR=0.0.0.0:5000 ¥  
-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt ¥  
-e REGISTRY_HTTP_TLS_KEY=/certs/domain.key ¥  
registry:2
```

必ずopensslコマンドを実行したディレクトリーで実行すること

Docker Repositoryのセットアップ(Master Node)

実行例(Master Node)

```
$ sudo docker run -d ¥
-p 5000:5000 ¥
--restart=always ¥
--name registry ¥
-v /var/opt/docker-registry/auth:/auth ¥
-v /var/opt/docker-registry/data:/var/lib/registry ¥
-v `pwd`/certs:/certs ¥
-e "REGISTRY_AUTH=htpasswd" ¥
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" ¥
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd ¥
-e REGISTRY_HTTP_ADDR=0.0.0.0:5000 ¥
-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt ¥
-e REGISTRY_HTTP_TLS_KEY=/certs/domain.key ¥
registry:2
e103aaa3e919e923369e5eed5f2e4bbe0e96844c7f9447b9aa22f49f549379e0
```

Docker Repositoryの稼働確認(全Node)

1. Docker Repositoryコンテナへのログイン確認

```
$ sudo docker login (Docker Repositoryのホスト名):5000 -u admin
```

Master Node、Worker Nodeすべてで実行し、「Login Succeeded」と表示されることを確認する

Docker Repositoryの稼働確認(全Node)

実行例(全Node)

```
$ sudo docker login master.apitechsales.jp.com:5000 -u admin
Password:
--- 中略 ---

Login Succeeded
```

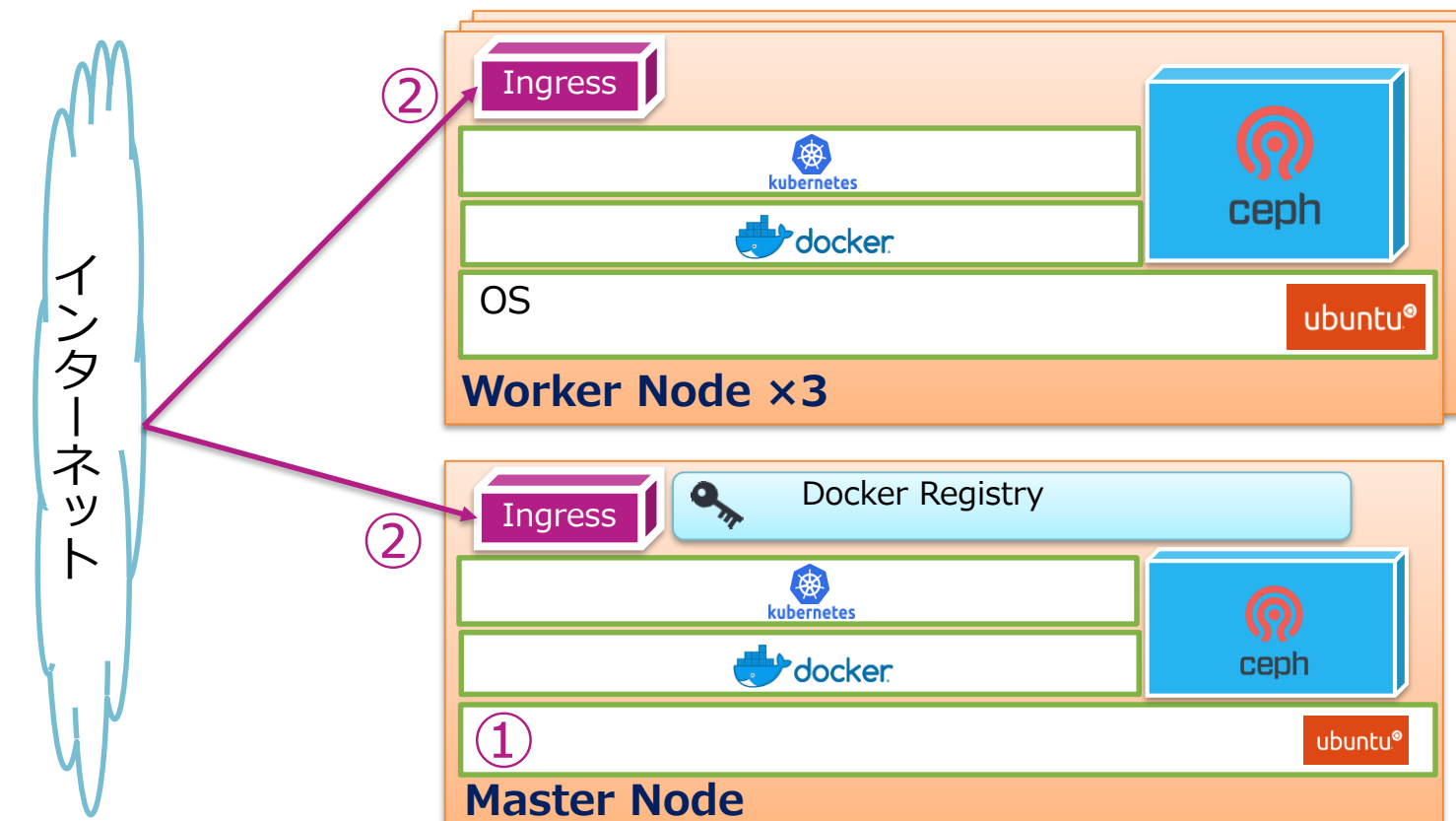


kubernetesランタイム環境の構築8

helmの初期設定、Ingressのセットアップ

kubernetesランタイム環境の構築8

□ helmの初期設定、Ingressのセットアップ



① Master Nodeでhelmコマンドを利用可能にする。

② Ingressを導入する。

helmの初期設定(Master Node)

1. helmの初期化

```
$ helm init --tiller-namespace apiconnect
```

IngressおよびAPI Connectコンポーネントの導入には、helmコマンドが必要になる

「Happy Helming!」

と表示された場合、コマンドが正常に終了したと判断する

helmの初期設定(Master Node)

実行例(Master Node)

```
$ helm init --tiller-namespace apiconnect
Creating /home/k8s_admin/.helm
Creating /home/k8s_admin/.helm/repository
Creating /home/k8s_admin/.helm/repository/cache
Creating /home/k8s_admin/.helm/repository/local
--- 中略 ---
```

Tiller (the Helm server-side component) has been installed into your kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.

To prevent this, run `helm init` with the `--tiller-tls-verify` flag.

For more information on securing your installation see:

https://docs.helm.sh/using_helm/#securing-your-helm-installation

Happy Helming!

Ingressのセットアップ(Master Node)

1. Ingressのインストール

```
$ helm install --name ingress -f ./nginx-ingress-values.yaml stable/nginx-ingress ¥  
--namespace apiconnect --tiller-namespace apiconnect
```

API Connectコンポーネントに外部から接続可能とするために、Ingressを導入する

「The nginx-ingress controller has been installed.」
と表示された場合、コマンドが正常に終了したと判断する

Ingressのセットアップ(Master Node)

実行例(Master Node)

```
$ helm install --name ingress -f ./nginx-ingress-values.yaml stable/nginx-ingress ¥  
--namespace apiconnect --tiller-namespace apiconnect
```

```
NAME:    ingress
```

```
LAST DEPLOYED: Wed Dec 12 19:41:34 2018
```

```
NAMESPACE: apiconnect
```

```
STATUS: DEPLOYED
```

```
--- 中略 ---
```

```
NOTES:
```

```
The nginx-ingress controller has been installed.
```

```
It may take a few minutes for the LoadBalancer IP to be available.
```

```
--- 以下略 ---
```

Ingressのセットアップ(Master Node)

作成したnginx-ingress-values.yaml(例)

yamlファイル

```
rbac:
  create: "true"
controller:
  image:
    repository: "quay.io/kubernetes-ingress-controller/nginx-ingress-controller"
    tag: "0.11.0"
  daemonset:
    useHostPort: false
  name: controller
  kind: DaemonSet
  hostNetwork: true
  extraArgs:
    enable-ssl-passthrough: true
    annotations-prefix: "ingress.kubernetes.io"
  config:
    proxy-body-size: "0"
    server-name-hash-bucket-size: "64"
    server-name-hash-max-size: "1024"
    use-http2: "true"
    proxy-buffering: "off"
    log-format: '{ "@timestamp": "$time_iso8601", "@version": "1", "clientip": "$remote_addr", "tag": "ingress", "remote_user": "$remote_user", "bytes": $bytes_sent,
    "duration": $request_time, "status": $status, "request": "$request_uri", "urlpath": "$uri", "urlquery": "$args", "method": "$request_method", "referer":
    "$http_referer", "useragent": "$http_user_agent", "software": "nginx", "version": "$nginx_version", "host": "$host", "upstream": "$upstream_addr", "upstream-status":
    "$upstream_status" }'
    hsts-max-age: "31536000"
    ssl-protocols: "TLSv1.2"
    ssl-prefer-server-ciphers: "True"
    ssl-ciphers: "HIGH:!aNULL:!MD5"
    server-tokens: "False"
    main-snippets: 'load_module "modules/ngx_stream_module.so"'
    worker-processes: "1"
    worker-rlimit-nofile: "65536"
    worker-connections: "10240"
    worker-cpu-affinity: "auto"
    worker-shutdown-timeout: "5m"
    keepalive: "32"
```

nginx-ingress-values.yamlに関しては以下のURLのvalues.yamlを参考に作成する

【参考資料:nginx-ingress】

124 <https://github.com/helm/charts/tree/master/stable/nginx-ingress>

Ingressの稼働確認(Master Node)

1. Ingressのhelmの確認

```
$ helm list --tiller-namespace apiconnect
```

Ingressのhelmパッケージが稼働していることを確認する

2. IngressのPod確認

```
$ kubectl get pods -n apiconnect -o wide
```

Ingressに関するPodが各Worker Nodeで「Running」となっていることを確認する

Ingressの稼働確認(Master Node)

実行例(Master Node)

```
$ helm list --tiller-namespace apiconnect
```

NAME	REVISION	UPDATED	STATUS
ingress	1	Wed Dec 12 19:41:34 2018	DEPLOYED
nginx-ingress-1.0.1		0.21.0	apiconnect

```
$ kubectl get pods -n apiconnect -o wide
```

NAME	READY	STATUS
ingress-nginx-ingress-controller-sfgjl	1/1	Running
14d 10.193.32.160 worker3 <none>		
ingress-nginx-ingress-controller-t2vr8	1/1	Running
14d 10.193.32.169 worker1 <none>		
ingress-nginx-ingress-controller-zhmqm	1/1	Running
14d 10.193.32.141 worker2 <none>		
ingress-nginx-ingress-default-backend-76cb88865d-7h4w9	1/1	Running
14d 192.168.190.195 worker2 <none>		

--- 以下略 ---



API Connectのセットアップ

API Connectのセットアップ

□ 前提

- apicupコマンドを実行するMaster Nodeに「apicup」コマンドがインストールされていること
- 以下のファイルをfixcentral等から事前にダウンロードし、Master Node上に配置していること
 - management-images-kubernetes_lts_v2018.4.1.1.tgz
 - analytics-images-kubernetes_lts_v2018.4.1.1.tgz
 - portal-images-kubernetes_lts_v2018.4.1.1.tgz
- Management、Analytics、Portal、DataPowerGatewayで指定する各URLについて、DNSで解決できること
 - Ingressの配置されたNodeを指定する
 - 本資料ではIngressは各Worker Nodeに作成しているが、DNSではWorker Node1を指定している
 - 実際の本番環境で利用する場合は、別途ロードバランサーを用意して各Worker Nodeに配分するよう設定すること
- 本資料ではkubernetes 1.12.3を利用している
- 本資料ではhelm 2.12.0を利用している
- apicupコマンドは利用するAPI Connectのバージョンにあわせる(本資料では2018.4.1.1)
- 本資料ではkubectl実行ユーザーとしてk8s_adminユーザーを利用している
特に記載がない限り、コマンドはk8s_adminで実行している

API Connectのベースイメージのアップロード(Master Node)

1. API ConnectのベースイメージをDocker Repositoryにアップロードする

```
$ cd (イメージを保存したディレクトリー)
$ sudo apicup registry-upload management ¥
management-images-kubernetes_lts_v2018.4.1.1.tgz (Docker Repositoryのホスト名):5000
$ sudo apicup registry-upload analytics ¥
analytics-images-kubernetes_lts_v2018.4.1.1.tgz (Docker Repositoryのホスト名):5000
$ sudo apicup registry-upload portal ¥
portal-images-kubernetes_lts_v2018.4.1.1.tgz (Docker Repositoryのホスト名):5000
```

2. Docker Repositoryの確認

```
$ curl -X GET https://(Docker Repositoryのホスト名):5000/v2/_catalog -u admin:(パスワード)
```

実行例と同様の出力となることを確認する

API Connectのベースイメージのアップロード(Master Node)

実行例(Master Node)

```
$ cd /var/work/apiconnect/4.1.1
$ sudo apicup registry-upload management ¥
  management-images-kubernetes_lts_v2018.4.1.1.tgz master.apitechsalesjp.com:5000
INFO[0000] Loading images into local docker (will take a while)
--- 中略 ---
$ sudo apicup registry-upload analytics ¥
  analytics-images-kubernetes_lts_v2018.4.1.1.tgz master.apitechsalesjp.com:5000
INFO[0000] Loading images into local docker (will take a while)
--- 中略 ---
$ sudo apicup registry-upload portal ¥
  portal-images-kubernetes_lts_v2018.4.1.1.tgz master.apitechsalesjp.com:5000
INFO[0000] Loading images into local docker (will take a while)
--- 中略 ---
$ curl -X GET https://master.apitechsalesjp.com:5000/v2/_catalog -u admin:APIc1234!
{"repositories":["analytics-client","analytics-cronjobs","analytics-
ingestion","analytics-mq-kafka","analytics-mq-zookeeper","analytics-
proxy","analytics-storage","apim","busybox","cassandra","cassandra-health-
check","cassandra-operator","client-downloads-
server","juhu","ldap","lur","migration","openresty","portal-admin","portal-
db","portal-dbproxy","portal-job-alpine","portal-web","ui"]}
```

インストール・プランの作成

□ 事前準備(Management)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

M-01. 外部yaml設定ファイル	例: 本資料では指定しない
説明: オプション設定を変更する際に使用するファイル名を指定する。任意。	
M-02. Ingressタイプ	例: 本資料では指定しない
説明: Ingressのタイプ。OpenShiftを利用する際はrouteを指定する。	
M-03. モード	例: standard
説明: Managementの導入モード。本番環境ではstandardを選択する。	
M-04. NameSpace	例: apiconnect
説明: ManagementをインストールするNameSpaceを指定する。	
M-05. Registry	例: master.apitechsalesjp.com:5000
説明: API ConnectのイメージをアップロードしたDocker Repositoryを指定する。	
M-06. Registry-Secret	例: my-localreg-secret
説明: Docker Repositoryのパスワードを格納したRegistryシークレット名を指定する。	
M-07. Storage-Class	例: apic-rbd
説明: ストレージのセットアップで指定したStorageClassを指定する。	

インストール・プランの作成

□ 事前準備(Management)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

M-08. platform-api	例: platform-api.apic2018mgmt.apitechsalesjp.com
説明: 管理とプロバイダープラットフォーム向けのREST APIを提供するURLを指定する。	
M-09. api-manager-ui	例: api-manager-ui.apic2018mgmt.apitechsalesjp.com
説明: API Management GUI用のURLを指定する。	
M-10. cloud-admin-ui	例: cloud-admin-ui.apic2018mgmt.apitechsalesjp.com
説明: Cloud Management GUI用のURLを指定する。	
M-11. consumer-api	例: consumer-api.apic2018mgmt.apitechsalesjp.com
説明: コンシューマー向けのREST APIを提供するURLを指定する。	
M-12. Cassandraバックアップ用ユーザー	例: bkup_usr
説明: Apache Cassandraのバックアップを保管するサーバーにアクセスするためのユーザー名。任意。	
M-13. Cassandraバックアップ用パスワード	例: APIC1234!
説明: Apache Cassandraのバックアップを保管するサーバーにアクセスするためのパスワード。任意。	
M-14. Cassandraバックアップ先ホスト名	例: master.apitechsalesjp.com
説明: Apache Cassandraのバックアップを保管するサーバーのホスト名。任意。	

インストール・プランの作成

□ 事前準備(Management)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

M-15. Cassandraバックアップ先ポート	例: 22
説明: Apache Cassandraのバックアップを保管するサーバーのSFTPのポート番号。任意。	
M-16. Cassandraバックアッププロトコル	例: sftp
説明: Apache Cassandraのバックアップのためのプロトコル。sftpまたはobjstore。任意。	
M-17. Cassandraバックアップパス	例: /var/work/backups/cassandra-backup
説明: Apache Cassandraのバックアップを保管するサーバーのパス。任意。	
M-18. Cassandraバックアップスケジュール	例: "0 0 * * 1"
説明: Apache Cassandraのバックアップを行うスケジュール(cron形式の記述)。任意。	
M-19. Cassandra最大メモリー	例: 8
説明: Apache Cassandraに利用されるメモリーの最大値(GB)。	
M-20. Cassandraクラスター数	例: 3
説明: Apache Cassandraのクラスターの数。	
M-21. Cassandraボリュームサイズ	例: 64
説明: Apache Cassandraに利用するボリュームのサイズ(GB)。デプロイ後変更不可。	

インストール・プランの作成

□ 事前準備(Management)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

M-22. CRDの作成	例: true
説明: CustomResourceDefinitionsの自動作成。falseにすると手動作成の必要がある。	
M-23. 外部Cassandraホスト	例: 本資料では指定しない
説明: 外部Apache Cassandraを利用する場合に指定する。任意。	
M-24. 証明書	例: 本資料では指定しない
説明: Managementに利用する証明書を手動で作成した場合に指定する。任意。	

インストール・プランの作成

□ 事前準備(Analytics)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

A-01. 外部yaml設定ファイル	例: 本資料では指定しない
説明: オプション設定を変更する際に利用するファイル名を指定する。任意。	
A-02. analytics-ingestion エンドポイントの名前	例: ingestion.apic2018analyt.apitechsalesjp.com
説明: Analyticsにおける、データ収集のためのURLを指定する。	
A-03. analytics-client エンドポイントの名前	例: client.apic2018analyt.apitechsalesjp.com
説明: Analyticsにおける、分析データ表示のためのURLを指定する。	
A-04. NameSpace	例: apiconnect
説明: AnalyticsをインストールするNameSpaceを指定する。	
A-05. Registry	例: master.apitechsalesjp.com:5000
説明: API ConnectのイメージをアップロードしたDocker Repositoryを指定する。	
A-06. Registry-Secret	例: my-localreg-secret
説明: Docker Repositoryのパスワードを格納したRegistryシークレット名を指定する。	
A-07. Coordinating最大メモリー	例: 12
説明: Coodinatingに利用されるメモリーの最大値(GB)。	

インストール・プランの作成

□ 事前準備(Analytics)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

A-08. 分析データ用最大メモリー	例: 12
説明: Analyticsのデータに利用されるメモリーの最大値(GB)。	
A-09. 分析データ用ボリュームサイズ	例: 200
説明: Analyticsのデータに利用するボリュームのサイズ(GB)。	
A-10. 分析マスター用最大メモリー	例: 12
説明: Analyticsのマスターに利用されるメモリーの最大値(GB)。	
A-11. 分析マスター用ボリュームサイズ	例: 5
説明: Analyticsのマスターに利用するボリュームのサイズ(GB)。	
A-12. MessageQueueの利用	例: false
説明: AnalyticsのパイプラインにMessageQueueを利用する場合はtrueを指定する。	
A-13. Storage-Class	例: apic-rbd
説明: ストレージのセットアップで指定したStorageClassを指定する。	
A-14. Elastic Search用Storage-Class	例: apic-rbd
説明: Elastic Searchに利用するストレージのStorageClassを指定する。	

インストール・プランの作成

□ 事前準備(Analytics)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

A-15. Message Queue用Storage-Class

例: 本資料では指定しない

説明: Message Queueに利用するストレージのStorageClassを指定する。

A-16. モード

例: standard

説明: Analyticsの導入モード。本番環境ではstandardを選択する。

A-17. Ingressタイプ

例: 本資料では指定しない

説明: Ingressのタイプ。OpenShiftを利用する際はrouteを指定する。

A-18. 証明書

例: 本資料では指定しない

説明: Managementに利用する証明書を手動で作成した場合に指定する。任意。

インストール・プランの作成

□ 事前準備(Portal)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

P-01. portal-admin エンドポイントの名前

例: admin.apic2018ptl.apitechsalesjp.com

説明: Portalにおける、ポータル管理のGUIにアクセスするためのURLを指定する。

P-02. portal-www エンドポイントの名前

例: www.apic2018ptl.apitechsalesjp.com

説明: Portalにおける、ポータル画面のGUIにアクセスするためのURLを指定する。

P-03. NameSpace

例: apiconnect

説明: AnalyticsをインストールするNameSpaceを指定する。

P-04. Storage-Class

例: apic-rbd

説明: ストレージのセットアップで指定したStorageClassを指定する。

P-05. Registry

例: master.apitechsalesjp.com:5000

説明: API ConnectのイメージをアップロードしたDocker Repositoryを指定する。

P-06. Registry-Secret

例: my-localreg-secret

説明: Docker Repositoryのパスワードを格納したRegistryシークレット名を指定する。

P-07. Portal用ボリュームサイズ

例: 5

説明: Portalに利用するボリュームのサイズ(GB)。

インストール・プランの作成

□ 事前準備(Portal)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

P-08. Portalバックアップ用ボリュームサイズ	例: 5
説明: Portalのバックアップに利用するボリュームのサイズ(GB)。	
P-09. Portalデータベース用ボリュームサイズ	例: 12
説明: Portalのデータベースに利用するボリュームのサイズ(GB)。	
P-10. Portalデータベースログ用ボリュームサイズ	例: 2
説明: Portalのデータベースログに利用するボリュームのサイズ(GB)。	
P-11. Portal管理コンテナ用ボリュームサイズ	例: 1
説明: Portalの管理コンテナに利用するボリュームのサイズ(GB)。	
P-12. モード	例: standard
説明: Portalの導入モード。本番環境ではstandardを選択する。	
P-13. Ingressタイプ	例: 本資料では指定しない
説明: Ingressのタイプ。OpenShiftを利用する際はrouteを指定する。	
P-14. Portalバックアップ先ホスト名	例: master.apitechsalesjp.com
説明: Portalのサイトバックアップを保管するサーバーのホスト名。Object Storageの場合はEndpoint/Region。	

インストール・プランの作成

□ 事前準備(Portal)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

P-15. Portalバックアップ先ポート	例: 22
説明: Portalのサイトバックアップを保管するサーバーのSFTPのポート番号。Object Storageの場合は不要。	
P-16. Portalバックアップ用ユーザー	例: bkup_usr
説明: Portalのサイトバックアップを保管するサーバーにアクセスするためのユーザー名。	
P-17. Portalバックアップ用パスワード	例: APIC1234!
説明: Portalのサイトバックアップを保管するサーバーにアクセスするためのパスワード。	
P-18. Portalバックアップパス	例: /var/work/backups/site-backup
説明: Portalのサイトバックアップを保管するサーバーのパス。	
P-19. Portalバックアッププロトコル	例: sftp
説明: Portalのサイトバックアップのためのプロトコル。sftpまたはobjstore。	
P-20. Portalバックアップスケジュール	例: "0 0 * * 1"
説明: Portalのサイトバックアップを行うスケジュール(cron形式の記述)。	
P-21. 証明書	例: 本資料では指定しない
説明: Portalに利用する証明書を手動で作成した場合に指定する。任意。	

インストール・プランの作成

□ 事前準備(DataPowerGateway)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

G-01. 外部yaml設定ファイル	例: 本資料では指定しない
説明: オプション設定を変更する際に利用するファイル名を指定する。任意。	
G-02. DataPower GatewayサーバーのURL	例: gw.apic2018gwy.apitechsalesjp.com
説明: DataPower GatewayサーバーにアクセスするためのURLを指定する。	
G-03. DataPower GatewayサーバーDirectorのURL	例: gwd.apic2018gwy.apitechsalesjp.com
説明: DataPower Gatewayサーバーの内部通信のためのURLを指定する。	
G-04. NameSpace	例: apiconnect
説明: ManagementをインストールするNameSpaceを指定する。	
G-05. Registry	例: 本資料では指定しない
説明: API ConnectのイメージをアップロードしたDocker Repositoryを指定する。	
G-06. Registry-Secret	例: 本資料では指定しない
説明: Docker Repositoryのパスワードを格納したRegistryシークレット名を指定する。	
G-07. Image Repository	例: ibmcom/datapower
説明: DataPowerのイメージを配置したDocker Repositoryを指定する。	

インストール・プランの作成

□ 事前準備(DataPowerGateway)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

G-08. Image Tag	例: 2018.4.1
説明: pullするDataPowerイメージのバージョンを指定する。	
G-09. Image Pull Policy	例: IfNotPresent
説明: イメージをpullするタイミングを指定する。	
G-10. Gatewayのレプリカ数	例: 3
説明: DataPowerGatewayのレプリカ数を指定する。	
G-11. Gateway用最大CPU	例: 4
説明: DataPowerGatewayに利用されるCPUの最大値(GB)。	
G-12. Gateway用最大メモリー	例: 6
説明: DataPowerGatewayに利用されるメモリーの最大値(GB)。	
G-13. Storage-Class	例: apic-rbd
説明: ストレージのセットアップで指定したStorageClassを指定する。	
G-14. v5互換モード	例: false
説明: DataPowerGatewayをAPI Connect v5互換モードで稼働する場合はtrueを指定する。	

インストール・プランの作成

□ 事前準備(DataPowerGateway)

インストール・プランを作成する前に、以下の情報について事前に決定する必要がある

G-15. tmsの有効化	例: true
説明: OAuth Token Management Systemの有効化。v5互換モードでない場合はtrueを指定する。	
G-16. tmsのピアリングストレージサイズ	例: 10
説明: OAuth Token Management Systemに利用するボリュームのサイズ(GB)。	
G-17. モード	例: standard
説明: DataPowerGatewayの導入モード。本番環境ではstandardを選択する。	
G-18. Ingressタイプ	例: 本資料では指定しない
説明: Ingressのタイプ。OpenShiftを利用する際はrouteを指定する。	
G-19. 証明書	例: 本資料では指定しない
説明: DataPowerGatewayに利用する証明書を手動で作成した場合に指定する。任意。	

インストール・プロジェクトの作成(Master Node)

1. API Connectのプロジェクトを作成する

```
$ mkdir (プロジェクト名)  
$ cd (プロジェクト名)  
$ apicup init
```

プロジェクト名は作成するAPI Connect環境には特に影響しない
apiconnect-up.ymlが作成されていることを確認する



インストール・プロジェクトの作成(Master Node)

実行例(Master Node)

```
$ mkdir apic_yaml  
$ cd apic_yaml  
$ apicup init  
$ ls  
apiconnect-up.yml
```

インストール・yamlの作成(Master Node)

1. Managerの情報を入力する

```
$ apicup subsys create mgmt management --k8s
$ apicup subsys set mgmt mode=(M-03の値)
$ apicup subsys set mgmt namespace=(M-04の値)
$ apicup subsys set mgmt registry=(M-05の値)
$ apicup subsys set mgmt registry-secret=(M-06の値)
$ apicup subsys set mgmt storage-class=(M-07の値)
$ apicup subsys set mgmt platform-api=(M-08の値)
$ apicup subsys set mgmt api-manager-ui=(M-09の値)
$ apicup subsys set mgmt cloud-admin-ui=(M-10の値)
$ apicup subsys set mgmt consumer-api=(M-11の値)
$ apicup subsys set mgmt cassandra-backup-auth-user=(M-12の値)
$ apicup subsys set mgmt cassandra-backup-auth-pass=(M-13の値)
$ apicup subsys set mgmt cassandra-backup-host=(M-14の値)
$ apicup subsys set mgmt cassandra-backup-port=(M-15の値)
$ apicup subsys set mgmt cassandra-backup-protocol=(M-16の値)
$ apicup subsys set mgmt cassandra-backup-path=(M-17の値)
$ apicup subsys set mgmt cassandra-backup-schedule=(M-18の値)
$ apicup subsys set mgmt cassandra-max-memory-gb=(M-19の値)
```

インストール・yamlの作成(Master Node)

1. Managerの情報を入力する

```
$ apicup subsys set mgmt cassandra-cluster-size=(M-20の値)  
$ apicup subsys set mgmt cassandra-volume-size-gb=(M-21の値)  
$ apicup subsys set mgmt create-crd=(M-22の値)
```

コマンドが正常終了した場合、特に出力はない

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys create mgmt management --k8s
$ apicup subsys set mgmt mode=standard
$ apicup subsys set mgmt namespace=apiconnect
$ apicup subsys set mgmt registry=master.apitechsalesjp.com:5000
$ apicup subsys set mgmt registry-secret=my-localreg-secret
$ apicup subsys set mgmt storage-class=apic-rbd
$ apicup subsys set mgmt platform-api=platform-api.apic2018mgmt.apitechsalesjp.com
$ apicup subsys set mgmt api-manager-ui=api-manager-
ui.apic2018mgmt.apitechsalesjp.com
$ apicup subsys set mgmt cloud-admin-ui=cloud-admin-
ui.apic2018mgmt.apitechsalesjp.com
$ apicup subsys set mgmt consumer-api=consumer-api.apic2018mgmt.apitechsalesjp.com
$ apicup subsys set mgmt cassandra-backup-auth-user=bkup_usr
$ apicup subsys set mgmt cassandra-backup-auth-pass=APIC1234!
$ apicup subsys set mgmt cassandra-backup-host=master.apitechsalesjp.com
$ apicup subsys set mgmt cassandra-backup-port=22
$ apicup subsys set mgmt cassandra-backup-protocol=sftp
$ apicup subsys set mgmt cassandra-backup-path=/var/work/backups/cassandra-backup
$ apicup subsys set mgmt cassandra-backup-schedule="0 0 * * 1"
$ apicup subsys set mgmt cassandra-max-memory-gb=8
```



インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys set mgmt cassandra-cluster-size=3  
$ apicup subsys set mgmt cassandra-volume-size-gb=64  
$ apicup subsys set mgmt create-crd=true
```



インストール・yamlの作成(Master Node)

2. Managerの情報を確認する

```
$ apicup subsys get mgmt --validate
```

入力した値が適正であることを確認することができる

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys get mgmt --validate
```

```
kubernetes settings
```

```
=====
```

Name	Value	
-----	-----	
extra-values-file		✓
ingress-type	ingress	✓
mode	standard	✓
namespace	apiconnect	✓

```
--- 中略 ---
```

```
Endpoints
```

```
=====
```

Name	Value	
-----	-----	
api-manager-ui	api-manager-ui.apic2018mgmt.apitechsalesjp.com	✓
cloud-admin-ui	cloud-admin-ui.apic2018mgmt.apitechsalesjp.com	✓
consumer-api	consumer-api.apic2018mgmt.apitechsalesjp.com	✓
platform-api	platform-api.apic2018mgmt.apitechsalesjp.com	✓

インストール・yamlの作成(Master Node)

3. Analyticsの情報を入力する

```
$ apicup subsys create analyt analytics --k8s
$ apicup subsys set analyt analytics-ingestion=(A-02の値)
$ apicup subsys set analyt analytics-client=(A-03の値)
$ apicup subsys set analyt namespace=(A-04の値)
$ apicup subsys set analyt registry=(A-05の値)
$ apicup subsys set analyt registry-secret=(A-06の値)
$ apicup subsys set analyt coordinating-max-memory-gb=(A-07の値)
$ apicup subsys set analyt data-max-memory-gb=(A-08の値)
$ apicup subsys set analyt data-storage-size-gb=(A-09の値)
$ apicup subsys set analyt master-max-memory-gb=(A-10の値)
$ apicup subsys set analyt master-storage-size-gb=(A-11の値)
$ apicup subsys set analyt enable-message-queue=(A-12の値)
$ apicup subsys set analyt storage-class=(A-13の値)
$ apicup subsys set analyt es-storage-class=(A-14の値)
$ apicup subsys set analyt mode=(A-16の値)
```

コマンドが正常終了した場合、特に出力はない

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys create analyt analytics --k8s
$ apicup subsys set analyt analytics-
ingestion=ingestion.apic2018analyt.apitechsalesjp.com
$ apicup subsys set analyt analytics-
client=client.apic2018analyt.apitechsalesjp.com
$ apicup subsys set analyt namespace=apiconnect
$ apicup subsys set analyt registry=localhost:5000
$ apicup subsys set analyt registry-secret=my-localreg-secret
$ apicup subsys set analyt coordinating-max-memory-gb=12
$ apicup subsys set analyt data-max-memory-gb=12
$ apicup subsys set analyt data-storage-size-gb=200
$ apicup subsys set analyt master-max-memory-gb=12
$ apicup subsys set analyt master-storage-size-gb=5
$ apicup subsys set analyt enable-message-queue=false
$ apicup subsys set analyt storage-class=apic-rbd
$ apicup subsys set analyt es-storage-class=apic-rbd
$ apicup subsys set analyt mode=standard
```



インストール・yamlの作成(Master Node)

4. Analyticsの情報を確認する

```
$ apicup subsys get analyt --validate
```

入力した値が適正であることを確認することができる

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys get analyt --validate
```

```
kubernetes settings
```

```
=====
```

Name	Value	
-----	-----	
extra-values-file		✓
ingress-type	ingress	✓
mode	standard	✓
namespace	apiconnect	✓
registry	master.apitechsalesjp.com:5000	✓
registry-secret	my-localreg-secret	✓
storage-class	apic-rbd	✓

--- 中略 ---

```
Endpoints
```

```
=====
```

Name	Value	
-----	-----	
analytics-client	client.apic2018analyt.apitechsalesjp.com	✓
analytics-ingestion	ingestion.apic2018analyt.apitechsalesjp.com	✓



インストール・yamlの作成(Master Node)

5. Portalの情報を入力する

```
$ apicup subsys create ptl portal --k8s
$ apicup subsys set ptl portal-admin=(P-01の値)
$ apicup subsys set ptl portal-www=(P-02の値)
$ apicup subsys set ptl namespace=(P-03の値)
$ apicup subsys set ptl storage-class=(P-04の値)
$ apicup subsys set ptl registry=(P-05の値)
$ apicup subsys set ptl registry-secret=(P-06の値)
$ apicup subsys set ptl www-storage-size-gb=(P-07の値)
$ apicup subsys set ptl backup-storage-size-gb=(P-08の値)
$ apicup subsys set ptl db-storage-size-gb=(P-09の値)
$ apicup subsys set ptl db-logs-storage-size-gb=(P-10の値)
$ apicup subsys set ptl admin-storage-size-gb=(P-11の値)
$ apicup subsys set ptl mode=(P-12の値)
$ apicup subsys set ptl site-backup-host=(P-14の値)
$ apicup subsys set ptl site-backup-port=(P-15の値)
$ apicup subsys set ptl site-backup-auth-user=(P-16の値)
$ apicup subsys set ptl site-backup-auth-pass=(P-17の値)
$ apicup subsys set ptl site-backup-path=(P-18の値)
```



インストール・yamlの作成(Master Node)

5. Portalの情報を入力する

```
$ apicup subsys set ptl site-backup-protocol=(P-19の値)
```

```
$ apicup subsys set ptl site-backup-schedule=(P-20の値)
```

コマンドが正常終了した場合、特に出力はない

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys create ptl portal --k8s
$ apicup subsys set ptl portal-admin=admin.apic2018ptl.apitechsalesjp.com
$ apicup subsys set ptl portal-www=www.apic2018ptl.apitechsalesjp.com
$ apicup subsys set ptl namespace=apiconnect
$ apicup subsys set ptl storage-class=apic-rbd
$ apicup subsys set ptl registry=localhost:5000
$ apicup subsys set ptl registry-secret=my-localreg-secret
$ apicup subsys set ptl www-storage-size-gb=5
$ apicup subsys set ptl backup-storage-size-gb=5
$ apicup subsys set ptl db-storage-size-gb=12
$ apicup subsys set ptl db-logs-storage-size-gb=2
$ apicup subsys set ptl admin-storage-size-gb=1
$ apicup subsys set ptl mode=standard
$ apicup subsys set ptl site-backup-host=master.apitechsalesjp.com
$ apicup subsys set ptl site-backup-port=22
$ apicup subsys set ptl site-backup-auth-user=bkup_usr
$ apicup subsys set ptl site-backup-auth-pass=APIC1234!
$ apicup subsys set ptl site-backup-path=/var/work/backups/site-backup
$ apicup subsys set ptl site-backup-protocol=sftp
$ apicup subsys set ptl site-backup-schedule="0 0 * * 1"
```



インストール・yamlの作成(Master Node)

6. Portalの情報を確認する

```
$ apicup subsys get ptl --validate
```

入力した値が適正であることを確認することができる

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys get ptl --validate
```

```
kubernetes settings
```

```
=====
```

Name	Value	
extra-values-file		✓
ingress-type	ingress	✓
mode	standard	✓
namespace	apiconnect	✓
registry	master.apitechsalesjp.com:5000	✓
registry-secret	my-localreg-secret	✓
storage-class	apic-rbd	✓
--- 中略 ---		
Endpoints		
=====		
Name	Value	
portal-admin	admin.apic2018ptl.apitechsalesjp.com	✓
portal-www	www.apic2018ptl.apitechsalesjp.com	✓



インストール・yamlの作成(Master Node)

7. DataPowerGatewayの情報を入力する

```
$ apicup subsys create gwy gateway --k8s
$ apicup subsys set gwy api-gateway=(G-02の値)
$ apicup subsys set gwy apic-gw-service=(G-03の値)
$ apicup subsys set gwy namespace=(G-04の値)
$ apicup subsys set gwy image-repository=(G-07の値)
$ apicup subsys set gwy image-tag=(G-08の値)
$ apicup subsys set gwy image-pull-policy=(G-09の値)
$ apicup subsys set gwy replica-count=(G-10の値)
$ apicup subsys set gwy max-cpu=(G-11の値)
$ apicup subsys set gwy max-memory-gb=(G-12の値)
$ apicup subsys set gwy storage-class=(G-13の値)
$ apicup subsys set gwy v5-compatibility-mode=(G-14の値)
$ apicup subsys set gwy enable-tms=(G-15の値)
$ apicup subsys set gwy tms-peering-storage-size-gb=(G-16の値)
$ apicup subsys set gwy mode=(G-17の値)
```

コマンドが正常終了した場合、特に出力はない

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys create gwy gateway --k8s
$ apicup subsys set gwy api-gateway=gw.apic2018gwy.apitechsalesjp.com
$ apicup subsys set gwy apic-gw-service=gwd.apic2018gwy.apitechsalesjp.com
$ apicup subsys set gwy namespace=apiconnect
$ apicup subsys set gwy image-repository=ibmcom/datapower
$ apicup subsys set gwy image-tag="2018.4.1-release"
$ apicup subsys set gwy image-pull-policy=IfNotPresent
$ apicup subsys set gwy replica-count=3
$ apicup subsys set gwy max-cpu=4
$ apicup subsys set gwy max-memory-gb=6
$ apicup subsys set gwy storage-class=apic-rbd
$ apicup subsys set gwy v5-compatibility-mode=false
$ apicup subsys set gwy enable-tms=true
$ apicup subsys set gwy tms-peering-storage-size-gb=10
$ apicup subsys set gwy mode=standard
```



インストール・yamlの作成(Master Node)

8. DataPowerGatewayの情報を確認する

```
$ apicup subsys get gwy --validate
```

入力した値が適正であることを確認することができる

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ apicup subsys get gwy --validate
```

```
kubernetes settings
```

```
=====
```

Name	Value	
-----	-----	
extra-values-file		✓
ingress-type	ingress	✓
mode	standard	✓
namespace	apiconnect	✓
registry		✓
registry-secret		✓
storage-class	apic-rbd	✓

```
--- 中略 ---
```

```
Endpoints
```

```
=====
```

Name	Value	
-----	-----	
api-gateway	gw. apic2018gwy. apitechsalesjp. com	✓
apic-gw-service	gwd. apic2018gwy. apitechsalesjp. com	✓



API Connectのインストール(Master Node)

1. 作成したyamlをファイルに変換する

```
$ apicup subsys install mgmt --out mgmt-out  
$ apicup subsys install analyt --out analyt-out  
$ apicup subsys install ptl --out ptl-out  
$ apicup subsys install gwy --out gwy-out
```

ファイルが作成されていることで正常に終了していることを確認する

インストール・yamlの作成(Master Node)

実行例(Master Node)

```
$ ls
apiconnect-up.yml
$ apicup subsys install mgmt --out mgmt-out
$ apicup subsys install analyt --out analyt-out
$ apicup subsys install ptl --out ptl-out
$ apicup subsys install gwy --out gwy-out
$ ls -al
total 256
drwxrwxr-x  6 k8s_admin k8s_admin  4096 Dec 28 02:59 .
drwxr-xr-x 14 k8s_admin k8s_admin  4096 Dec 24 11:59 ..
drwx-----  5 k8s_admin k8s_admin  4096 Dec 28 02:59 analyt-out
-rw-rw----  1 k8s_admin k8s_admin 33185 Dec 28 02:59 apic_yaml-analyt-apiconnect-secrets.yml
-rw-rw----  1 k8s_admin k8s_admin 34710 Dec 28 02:59 apic_yaml-common-secrets.yml
-rw-rw----  1 k8s_admin k8s_admin 25901 Dec 28 02:59 apic_yaml-gwy-apiconnect-secrets.yml
-rw-rw----  1 k8s_admin k8s_admin 73913 Dec 28 02:59 apic_yaml-mgmt-apiconnect-secrets.yml
-rw-rw----  1 k8s_admin k8s_admin 46327 Dec 28 02:59 apic_yaml-ptl-apiconnect-secrets.yml
-rw-rw----  1 k8s_admin k8s_admin  4570 Dec 28 02:52 apiconnect-up.yml
drwx-----  5 k8s_admin k8s_admin  4096 Dec 28 02:59 gwy-out
drwx-----  5 k8s_admin k8s_admin  4096 Dec 28 02:59 mgmt-out
drwx-----  5 k8s_admin k8s_admin  4096 Dec 28 02:59 ptl-out
```

API Connectのインストール(Master Node)

2. Managementのインストール

```
$ export TILLER_NAMESPACE=apiconnect  
$ apicup subsys install mgmt --plan-dir ./mgmt-out
```

「Management service service online」
と表示された場合、コマンドが正常に終了したと判断する

3. Analyticsのインストール

```
$ apicup subsys install analyt --plan-dir ./analyt-out
```

「Analytics client service online」
と表示された場合、コマンドが正常に終了したと判断する

4. Portalのインストール

```
$ apicup subsys install ptl --plan-dir ./ptl-out
```

「Portal manager service online」
と表示された場合、コマンドが正常に終了したと判断する

5. DataPowerGatewayのインストール

```
$ apicup subsys install gwy --plan-dir ./gwy-out
```

「Installed chart dynamic-gateway-service」
と表示された場合、コマンドが正常に終了したと判断する

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ export TILLER_NAMESPACE=apiconnect
$ apicup subsys install mgmt --plan-dir ./mgmt-out

Beginning installation...
--- 中略 ---
Management service service online

$ apicup subsys install analyt --plan-dir ./analyt-out

Beginning installation...
--- 中略 ---
Analytics client service online

$ apicup subsys install ptl --plan-dir ./ptl-out

Beginning installation...
--- 中略 ---
Portal manager service online

$ apicup subsys install gwy --plan-dir ./gwy-out

Beginning installation...
Installing chart dynamic-gateway-service...
Installed chart dynamic-gateway-service
```

API Connectのインストール(Master Node)

6. API Connectの稼働確認

```
$ kubectl get pods -n apiconnect
```

すべてのpodが「Running」または「Completed」となっていることを確認する
※一部、repair等Errorになっていても問題なく稼働している場合がある

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ kubectl get pods -n apiconnect
```

NAME	READY	STATUS	RESTARTS	AGE
ingress-nginx-ingress-controller-sfgjl	1/1	Running	0	15d
ingress-nginx-ingress-controller-t2vr8	1/1	Running	0	15d
ingress-nginx-ingress-controller-zhmqm	1/1	Running	0	15d
ingress-nginx-ingress-default-backend-76cb88865d-7h4w9	1/1	Running	0	15d
r1a68a36b02-analytics-client-59d6d549c9-mg5tx	1/1	Running	0	7d8h
r1a68a36b02-analytics-client-59d6d549c9-q5zsz	1/1	Running	0	7d8h
r1a68a36b02-analytics-cronjobs-retention-1545874200-7tbdn	0/1	Completed	0	16h
r1a68a36b02-analytics-cronjobs-rollover-1545932700-lptd8	0/1	Completed	0	27m
r1a68a36b02-analytics-ingestion-7648db8d48-9b5fd	1/1	Running	0	7d8h
r1a68a36b02-analytics-ingestion-7648db8d48-bn9l9	1/1	Running	0	7d8h
--- 中略 ---				
rbd-provisioner-65699b8bfc-wkhmz	1/1	Running	8	15d
rc4ea5d1d3e-apic-portal-db-0	2/2	Running	0	7d8h
rc4ea5d1d3e-apic-portal-db-1	2/2	Running	0	7d8h
rc4ea5d1d3e-apic-portal-db-2	2/2	Running	0	7d8h
rc4ea5d1d3e-apic-portal-nginx-74c46fd46d-jbxw4	1/1	Running	0	7d8h
rc4ea5d1d3e-apic-portal-nginx-74c46fd46d-kktj4	1/1	Running	0	7d8h
rc4ea5d1d3e-apic-portal-nginx-74c46fd46d-slqs4	1/1	Running	0	7d8h
rc4ea5d1d3e-apic-portal-www-0	2/2	Running	2	7d8h
rc4ea5d1d3e-apic-portal-www-1	2/2	Running	2	7d8h
rc4ea5d1d3e-apic-portal-www-2	2/2	Running	1	7d8h
tiller-deploy-db548578-pxblh	1/1	Running	0	15d

API Connectのインストール(Master Node)

7. (補足)インストールに失敗した場合

```
$ helm list --namespace=apiconnect --tiller-namespace=apiconnect  
$ helm delete --purge xxxxx
```

インストールに失敗した場合、helm listから該当のhelmパッケージを確認し、NAMEに対してdeleteを実行することでインストール前の状態に戻すことができる

Managerのインストールに失敗した場合は、CHART名が「apiconnect-2.0.0」と「cassandra-operator-1.0.0」の2つをdeleteする必要がある

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ helm list --namespace=apiconnect --tiller-namespace=apiconnect
```

NAME	REVISION	UPDATED	STATUS	CHART
APP VERSION	NAMESPACE			
ingress	1	Wed Dec 12 19:41:34 2018	DEPLOYED	nginx-ingress-
1.0.1	0.21.0	apiconnect		
r1a68a36b02	1	Thu Dec 20 19:07:01 2018	DEPLOYED	apic-analytics-
2.0.0		apiconnect		
r31f4a26f5e	1	Thu Dec 20 18:13:32 2018	DEPLOYED	apiconnect-
2.0.0		apiconnect		
r554d996560	1	Thu Dec 20 19:15:22 2018	DEPLOYED	dynamic-
gateway-service-1.0.16	1.0	apiconnect		
r5d88581ca1	1	Thu Dec 20 18:13:26 2018	DEPLOYED	cassandra-
operator-1.0.0	1.0.1	apiconnect		
rc4ea5d1d3e	1	Thu Dec 20 19:09:44 2018	DEPLOYED	apic-portal-
2.0.0		apiconnect		

```
$ helm delete --purge r31f4a26f5e
```

```
release "r31f4a26f5e" deleted
```

```
$ helm delete --purge r5d88581ca1
```

```
release "r5d88581ca1" deleted
```

API Connectのインストール(Master Node)

8. (補足)DataPower GatewayのCLIログイン

```
$ kubectl get pods -n apiconnect | grep gateway  
$ kubectl attach -n apiconnect -it (対象のDataPower Gatewayのpod)
```

DataPower Gatewayの設定変更等、CLIにログインする必要がある場合に実行する
ユーザー名 / パスワードは「admin / admin」

DataPower Gatewayから抜ける際は、「Ctrl-c」ではなく「Ctrl-P、Ctrl-Q」で抜ける必要がある
<https://kubernetes.io/docs/reference/kubectl/docker-cli-to-kubectl/#docker-attach>

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ kubectl get pods -n apiconnect | grep gateway
r554d996560-dynamic-gateway-service-0          1/1      Running    0
2d9h
r554d996560-dynamic-gateway-service-1          1/1      Running    0
2d9h
r554d996560-dynamic-gateway-service-2          1/1      Running    1
2d9h
$ kubectl attach -n apiconnect -it r554d996560-dynamic-gateway-service-2
Defaulting container name to dynamic-gateway-service.
Use 'kubectl describe pod/ -n apiconnect' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.

idg#

Ctrl-P、Ctrl-Qを押下
idg# Session ended, resume using 'kubectl attach r554d996560-dynamic-gateway-service-2 -c
dynamic-gateway-service -i -t' command when the pod is running
```

API Connectのインストール(Master Node)

9. (補足)DataPower GatewayのGUI有効化

```
$ kubectl attach -n apiconnect -it (対象のDataPower Gatewayのpod)
idg# configure
idg(config)# web-mgmt
idg(config web-mgmt)# admin-state enabled
idg(config web-mgmt)# exit
idg(config)# write memory
```

DataPower Gatewayの設定変更等、GUIにログインする必要がある場合に実行する
ユーザー名 / パスワードは「admin / admin」

サポートがないため、実施する場合は自己責任にて行う必要がある

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ kubectl attach -n apiconnect -it r554d996560-dynamic-gateway-service-2
Defaulting container name to dynamic-gateway-service.
Use 'kubectl describe pod/ -n apiconnect' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.

idg# configure
Global configuration mode
idg(config)# web-mgmt
Modify Web Management Service configuration

idg(config web-mgmt)# show web-mgmt

web-mgmt [down]
-----
    admin-state disabled
--- 中略 ---
idg(config web-mgmt)# admin-state enabled
idg(config web-mgmt)# exit
idg(config)# write memory
Overwrite previously saved configuration? Yes/No [y/n]: y
Configuration saved successfully.
idg(config)# show web-mgmt

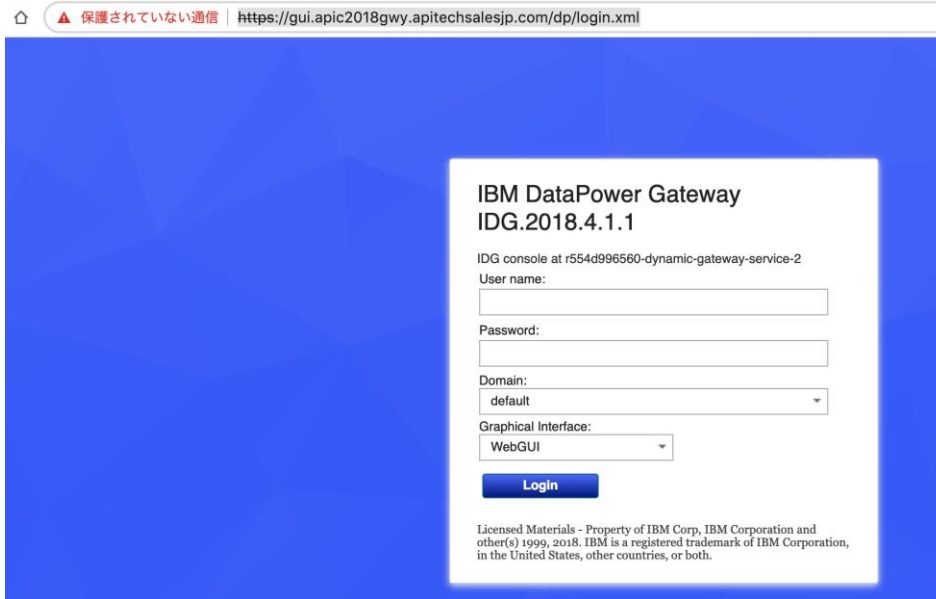
web-mgmt [up]
-----
    admin-state enabled
--- 以下略 ---
```

API Connectのインストール(Master Node)

9. (補足)DataPower GatewayのGUIポート開放

```
$ kubectl apply -f ./gwy_gui.yaml
$ kubectl get svc -n apiconnect
$ kubectl edit svc -n apiconnect (対象のDataPower GatewayのService)
```

DataPower Gatewayの設定変更等、GUIにログインする必要がある場合に実行する初期設定ではポートに接続できないため、Ingressの設定を変更する必要がある
サポートがないため、実施する場合は自己責任にて行う必要がある



IBM DataPower Gateway
IDG.2018.4.1.1

IDG console at r554d996560-dynamic-gateway-service-2

User name:

Password:

Domain:

Graphical Interface:

Login

Licensed Materials - Property of IBM Corp, IBM Corporation and other(s) 1999, 2018. IBM is a registered trademark of IBM Corporation, in the United States, other countries, or both.

API Connectのインストール(Master Node)

実行例(Master Node)

```
$ vim gwy_gui.yaml
$ kubectl apply -f ./gwy_gui.yaml
ingress.extensions/dynamic-gateway-service-gui created
$ kubectl edit svc -n apiconnect r554d996560-dynamic-gateway-service-ingress
service/r554d996560-dynamic-gateway-service-ingress edited
$ kubectl get ingress -n apiconnect
```

NAME	HOSTS	ADDRESS	PORTS
dynamic-gateway-service-gui	gui.apic2018gwy.apitechsalesjp.com		80, 443

```
9s
--- 中略 ---
$ kubectl get svc -n apiconnect
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
r554d996560-dynamic-gateway-service-ingress	ClusterIP	10.101.19.170	<none>	3000/TCP, 9443/TCP, 9090/TCP

```
2d5h
--- 以下略 ---
```

ingressが作成され、svcで9090ポートが開放されていることを確認する

API Connectのインストール(Master Node)

作成したgwy_gui.yaml(例)

yamlファイル

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/ssl-passthrough: "true"
    ingress.kubernetes.io/ssl-redirect: "true"
  generation: 1
  name: dynamic-gateway-service-gui
  namespace: apiconnect
  selfLink: /apis/extensions/v1beta1/namespaces/apiconnect/ingresses/dynamic-gateway-service-gui
spec:
  rules:
    - host: gui.apic2018gwy.apitechsalesjp.com
      http:
        paths:
          - backend:
              serviceName: r554d996560-dynamic-gateway-service-ingress
              servicePort: 9090
            path: /
      tls:
        - hosts:
            - gui.apic2018gwy.apitechsalesjp.com
  status:
    loadBalancer:
      ingress:
        - {}
```

赤字で記載したホスト名で、GUIにアクセス可能となる

例では、「https:// gui.apic2018gwy.apitechsalesjp.com/dp/login」でGUIにログイン可能となる

API Connectのインストール(Master Node)

kubectl edit svcにより修正する箇所

yamlファイル

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2019-02-11T22:37:02Z"
  labels:
    app: dynamic-gateway-service
    name: r554d996560-dynamic-gateway-service-ingress
    namespace: apiconnect
    resourceVersion: "56825"
  selfLink: /api/v1/namespaces/apiconnect/services/r554d996560-dynamic-gateway-service-ingress
  uid: 8ccdf116-2e4d-11e9-8f60-067b83020a5b
spec:
  clusterIP: 10.101.19.170
  ports:
    - name: apic-gw-mgmt
      port: 3000
      protocol: TCP
      targetPort: 3000
    - name: api-gw-svc
      port: 9443
      protocol: TCP
      targetPort: 9443
    - name: api-gw-gui
      port: 9090
      protocol: TCP
      targetPort: 9090
  selector:
    app: dynamic-gateway-service
    release: r554d996560
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

赤字部分を追記する



IBM **API** Connect

API Connectの初期設定

API Connectのセットアップ

□ 初期設定の流れはOVA版とほぼ同様のため「IBM API Connect v2018.2.10 OVA版構築ガイド」を参照してください

□ OVA版との違い

- DataPowerGateway
 - kubernetes版ではインストール時に自動で行われるため、初期設定は不要です
 - DataPowerGatewayへのGUIログインは、デフォルトではできません
- NTP設定は不要です
- サービスの登録時、Gatewayのサービスのタイプが追加されています
 - インストール・プランの設定内容に合わせて選択してください

サービス・タイプの選択

 DataPower API Gateway API の保護および実行のための DataPower API ゲートウェイ・サービスを構成します	 DataPower Gateway (v5 互換) API の保護および実行のための DataPower ゲートウェイ・サービスを構成します	 ポータル API コンシューマー用の開発者ポータル・サービスを構成します
 分析 API 呼び出しデータを収集するための分析サービスを構成します		



補足資料



要件と制約事項

□ リソース要件

【参考資料:Requirements for deploying API Connect into a kubernetes environment】

https://www.ibm.com/support/knowledgecenter/en/SSMNE2_2018/com.ibm.apic.install.doc/topic_install_reqs_kubernetes.html

【参考資料 : Detailed system requirements for a specific product】

<https://www.ibm.com/software/reports/compatibility/clarity/softwareReqsForProduct.html>

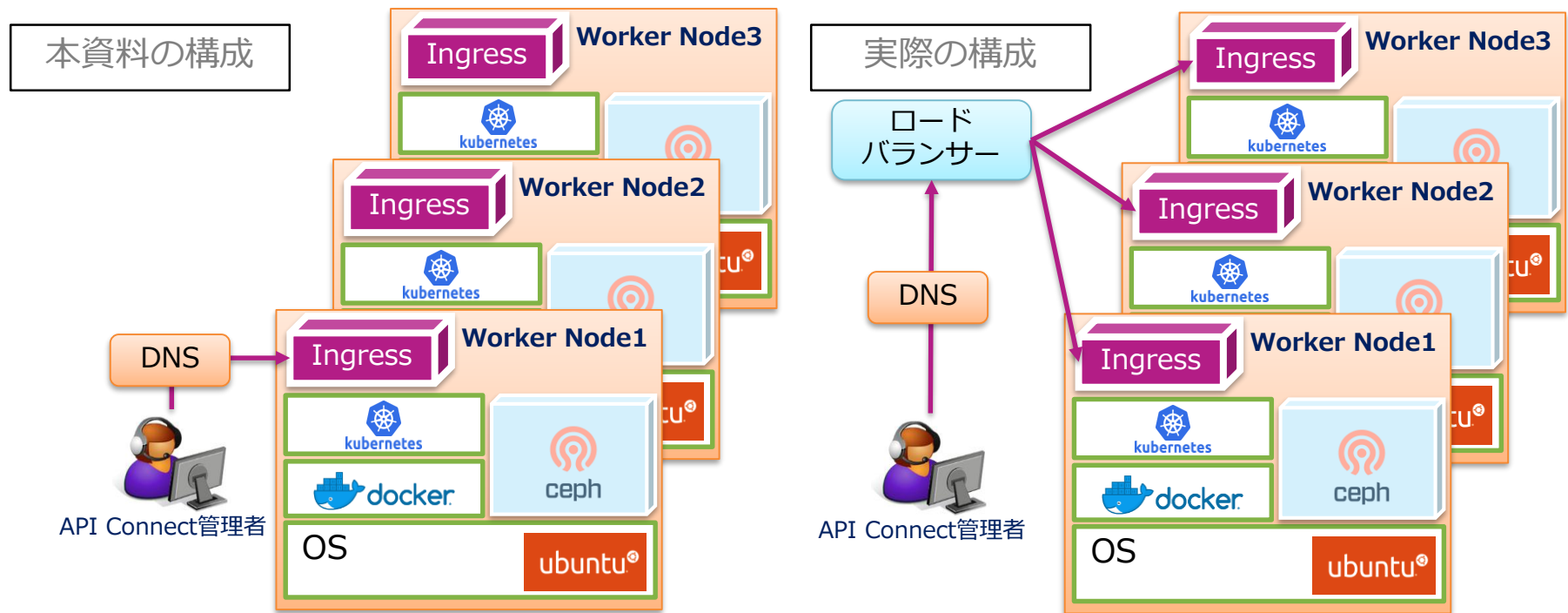
サーバー名	CPU	メモリ (GB)	Disk (GB)
管理サーバー	4core	16	250
開発者ポータル・サーバー	2core	8	50
分析サーバー	2core	16	200
DataPower Gatewayサーバー	4core	8	32

ソフトウェア	バージョン
kubernetes	1.10 - 1.12
Docker	17.03 - 18.03
helm	any version

要件と制約事項

□ DNSについて

- Cloud Managerでは各サーバーがサービスのメンバーとして追加され、管理される
- サービスを登録するには、DNSサーバーを利用する必要がある
- 本資料ではWorker1のIngressを利用しているが、本番環境の場合は別途ロードバランサーなどを利用し各NodeのIngressに配分されるよう設定する必要がある



要件と制約事項

□ DNSについて

本資料でのDNS設定例

```

--- 省略 ---

                IN      NS      master.apitechsalesjp.com.
master          IN      A       10.193.32.164
*.master        IN      A       10.193.32.164
worker1         IN      A       10.193.32.169
*.worker1       IN      A       10.193.32.169
worker2         IN      A       10.193.32.141
*.worker2       IN      A       10.193.32.141
worker3         IN      A       10.193.32.160
*.worker3       IN      A       10.193.32.160
apic2018mgmt    IN      A       10.193.32.169
*.apic2018mgmt  IN      A       10.193.32.169
apic2018ptl     IN      A       10.193.32.169
*.apic2018ptl   IN      A       10.193.32.169
apic2018analyt  IN      A       10.193.32.169
*.apic2018analyt IN      A       10.193.32.169
apic2018gwy     IN      A       10.193.32.169
*.apic2018gwy   IN      A       10.193.32.169
  
```

本資料では、10.193.32.169(Worker Node1のIngressのIP)を指定しているが、
本番環境ではこのIPをロードバランサーのIPとし、そこから各Worker Nodeに配分される構成とすることが望ましい

要件と制約事項

□ 分析サーバー用のゾーン間ポート番号

https://www.ibm.com/support/knowledgecenter/ja/SSMNED_2018/com.ibm.apic.install.doc/overview_apimgmt_portreqs.html

- V2018では分析サーバーを独立したコンポーネントとして構築する必要があるため、製品として新たに以下のポートを使用するように変更されている

番号	通信内容	from	to	プロトコル	ポート番号
1	分析サーバーのオフロード用	分析サーバー	オフロード・システム	HTTP、HTTPS、TCP、UDP、KAFKA	ポート番号はオフロードに利用されるプラグインのタイプとプロトコルによって決める。
2	分析サーバーからNTPへアクセス	分析サーバー	NTPサーバー	TCP、UDP	123 ※k8s版では、APICの各コンポーネントはworker Nodeの時刻を参照しに行くため、worker NodeがNTPを参照するように設定してください。
3	分析サーバーからDNSへアクセス	分析サーバー	DNSサーバー	TCP、UDP	53
4	管理サーバーから分析サーバーへの問い合わせ (Management service queries Analytics service)	管理サーバー	分析サーバー	HTTPS	443
5	ポータルサーバーが分析サーバーからデータを取得するためにのAPI用のポート	ポータルサーバー	分析サーバー	HTTPS	443

要件と制約事項

□ IPアドレス

https://www.ibm.com/support/knowledgecenter/en/SSMNED_2018/com.ibm.apic.install.doc/overview_installing_mgmtvm_apimgmt.html

- 172.16.0.0/16および172.17.0.0/16の範囲のアドレスは、デフォルトではkubernetesのサービスネットワークとして利用される
- 変更する場合は、API Connectデプロイ前のyamlファイル作成時に設定を変更する必要がある

Optional: Modify the IP ranges for the kubernetes pod and the service networks.

You can change the IP ranges of the kubernetes pod and the service networks from the default values of 172.16.0.0/16 and 172.17.0.0/16, respectively. You can modify these ranges during initial installation and configuration only. You cannot modify them once an appliance has been deployed.

□ kubernetesのロードバランサー

- API Connectが外部と通信するためには、API Connectコンポーネント以外にL7ロードバランサーであるIngressをkubernetes環境内で構築する必要がある

□ DataPowerGatewayのGUIアクセス

- kubernetes版のDataPower Gatewayは、ポート番号443と80以外は公開されない実装になっている
- API呼び出しは443ポートを利用し、Ingress経由で9443ポートにアクセスする
- 外部からGUIにアクセスするには9090ポートを有効化し、Ingressの定義にGUIのホスト名を追加する必要がある