

# Rancher と Trident による 本番運用を実現する Kubernetes 環境の実証

ネットアップ合同会社/株式会社スタイルズ



公開日 : 2019 年 1 月 8 日

## はじめに・概要

本ホワイトペーパーでは Kubernetes を使用したコンテナプラットフォームを実現し本番運用上課題として挙げられる複雑性、データ永続化についての解決方法を提示します。

複雑性を低減する方法として、「Rancher」を活用し Kubernetes に対し直接 CLI からコマンドを実行せずにコンテナのデプロイやデータ永続化領域の作成等を行う検証をしました。また、データ永続化の課題を解決する方法としてエンタープライズストレージをコンテナプラットフォームに提供する「Trident」を採用しました。

本ホワイトペーパー内では、これらコンポーネントを実際にデプロイする方法、利用方法を詳細に記載しています。

多くの皆様にとって、コンテナ活用の課題を解決するための手助けになれば幸いです。

ネットアップ合同会社/株式会社スタイルズ

## 目次

はじめに・概要.....	1
1. Rancher/Kubernetes の構築 .....	5
1.1. Kubernetes 環境の構成説明 .....	5
1.2. Kubernetes のインストール .....	5
1.3. Rancher の特徴.....	5
1.4. Rancher のインストール .....	7
1.4.1. Rancher インストールに必要な構成 .....	7
1.4.2. Docker のインストール .....	7
1.4.3. Rancher のインストール.....	8
1.5. Rancher から Kubernetes のインポート .....	8
1.5.1. Rancher へのログイン .....	8
1.5.2. Kubernetes クラスタ追加方法 .....	8
2. Kubernetes のストレージについて.....	13
2.1. PersistentStorage の基本.....	13
2.2. DynamicProvisioning とは .....	14
2.3. StorageClass の基本 .....	15
3. Trident のコンポーネント・構築/構成 .....	16
3.1. Trident の概要 .....	16
3.2. ホスト OS 設定(iscsi、nfs クライアントのインストール).....	17
3.2.1. NFS .....	17

3.2.2.	iSCSI.....	17
3.3.	バックエンドストレージ設定.....	18
3.3.1.	ONTAP の設定 .....	18
3.3.2.	SolidFire の設定 .....	18
3.4.	Trident のインストール.....	19
3.4.1.	Trident の構成 .....	19
3.4.2.	NFS バックエンドの場合(ONTAP バックエンド) .....	20
3.4.3.	iSCSI バックエンドの場合(SolidFire バックエンド).....	20
3.5.	StorageClass の定義例 .....	21
4.	コンテナでの利用(Rancher での利用方法) .....	24
4.1.	Dynamic Provisioning でのコンテナへのストレージの割り当て .....	24
4.1.1.	Rancher のメニュー構成について.....	24
4.1.2.	クラスタのストレージ .....	24
4.1.3.	プロジェクトのメニュー構成について.....	25
4.1.4.	コンテナをデプロイする .....	27
4.1.5.	ボリュームの割り当て .....	29
4.1.6.	ボリュームのマウント先を指定 .....	31
4.1.7.	マウントを確認.....	31
4.2.	StatefulSet を使ったダイナミックプロビジョニングの例.....	33
4.2.1.	MongoDB レプリカセット.....	33
4.2.2.	Helm 用設定ファイルの準備.....	34
4.2.3.	MongoDB デプロイ .....	36

4.2.4.	ノード障害発生時の動作確認 .....	45
4.3.	Trident のバックエンドストレージに設定した ONTAP での使い方.....	49
4.3.1.	Heptio Ark .....	49
4.3.2.	アプリケーションデータのレプリケーション .....	50
4.3.3.	実際の構成 .....	51
4.3.4.	クラスターフェイルオーバーを実施.....	52
4.3.5.	考慮点 .....	53
4.4.	Trident Fast PVC Cloning の使い方とユースケース .....	54
4.4.1.	PVC Fast Cloning とは.....	54
4.4.2.	実際の設定方法.....	54
5.	総括.....	57
6.	会社概要・問い合わせ先 .....	58

## ■注意

バージョン、利用環境により操作方法、画面イメージが異なることがあります。掲載されているサンプルスクリプト、および実行結果を記した画面イメージなどは、特定の設定に基づいた環境にて再現される一例です。実行結果を保証するものではありません。

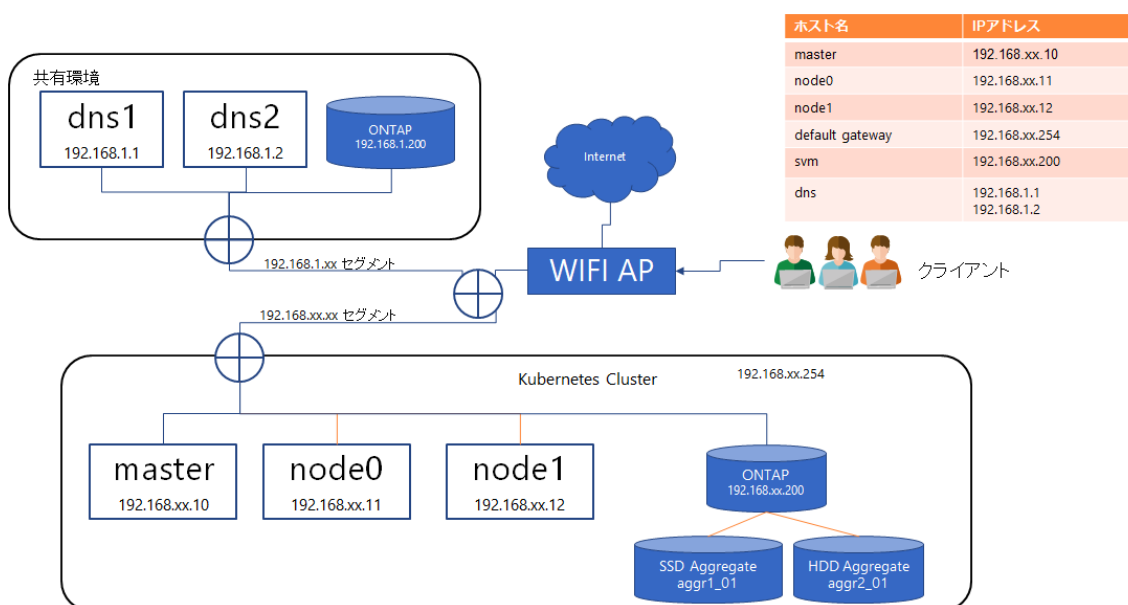
OS、ソフト名、企業名、サービス名は、一般に各メーカー・企業の商標または登録商標です。なお、本文中では TM、および®マークは明記していません。

本ドキュメントの全部、または一部について、許諾を得ずに複製することは禁じられています。

# 1. Rancher/Kubernetes の構築

## 1.1. Kubernetes 環境の構成説明

Kubernetes クラスターの構成を以下のように構成する。共有する環境として DNS を dns1,dns2 を設定し、Kubernetes クラスター毎のセグメントと分離する。Kubernetes クラスターには、Master ノードを 1 つ、Worker ノードを node0,node1 を設置する。同じセグメント内に NetApp ONTAP を配置し SVM を構成、SSD と HDD のアグリゲートを作成し、それぞれ aggr1\_01、aggr2\_01 とした。SolidFire も配置し、複数のストレージバックエンドを取り扱えることも確認した。



## 1.2. Kubernetes のインストール

Kubernetes クラスターのインストールには、kubeadm でインストールを行った。特別な操作はせず通常の方法でセットアップしたため、ここでは操作方法などは省略する。

## 1.3. Rancher の特徴

Rancher は オープンソースで開発されている Kubernetes を管理するアプリケーションです。Enterprise Kubernetes made Easy を目標に Kubernetes の機能を企業ユースで利用しやすい機能が備わっています。

管理するユーザーインターフェースは、Web ブラウザーを使ったグラフィカルなインターフェースとターミナルを使った専用コマンドを通じて操作する方法との 2 種類があります。

通常、Kubernetes では Manifest と呼ばれるコンテナ操作ファイルを作成し、それを kubectl コマンドにより反映させてコンテナを起動・停止します。しかし、Rancher ではそのようなコマンドラインで操作することなく、ブラウザでの操作でコンテナを操作します。また、Manifest ファイルを使って指定する様々なオプションを WebUI から設定できるようになっています。このようなコンソール操作をしなくてもよい利便性が Rancher の特徴の一つになっています。

もう一つの特徴が複数の Kubernetes クラスタを管理できるというマルチクラウドの機能です。このマルチクラウドには、オンプレミスの Kubernetes クラスタも含まれ、クラウドやオンプレミス、その他様々なディストリビューションの複数の Kubernetes を一元的に管理することができます。

現在では複数の Kubernetes を管理できるというツールは非常に数多くありますが、開発当初から設置場所を問わず、マルチクラウドでマルチディストリビューション対応を打ち出しているツールは多くありません。

また、Kubernetes クラスタを企業向けのニーズにマッチさせるということも Rancher の大きなミッションの一つになっています。その一つはユーザー認証やセキュリティポリシーの設定です。今日の企業においてユーザー管理やセキュリティポリシーの設定は非常に重要な要素となっています。もし、ユーザーをクラスタ毎にバラバラで管理しなければならない場合、その手間は非常に大きなものとなるでしょう。

そういった認証システムと統合し、Active Directory や GitHub、Azure AD、FreeIPA、OpenLDAP といった認証システムとの連携ができます。認証ばかりではなく、ネットワークのポリシー、Kubernetes に則った RBAC の設定などを Kubernetes のコマンドを利用することなく Web ブラウザーの UI を通して設定することができます。

他にも、企業利用において重要な稼働監視やロギング、モニタリング、アラート通知機能があります。様々なクラウド上で動いているマネージドサービスの Kubernetes クラスタは管理についてあまり気にする必要はありません。しかし、オンプレミスの Kubernetes クラスタではサーバーが止まったり、リソースが足りなくなる事が考えられます。当然それらを無視することはできず、何らかの対応をしなければなりません。しかし、その状況が通知されなければ、そもそも対応することも出来ません。その為には稼働状況を常時記録し、問題があった場合には複数の手段で担当者へ連絡する必要があります。Rancher にはそれらの機能も備えていて、アラート発報方法、通知方法を設定し、障害対応の為にログをログサーバーへの転送することができます。

最後にアプリケーションの開発者向けの機能についてご紹介しておきます。WebUI の操作でコンテナを操作できると書きましたが、毎回 WebUI で全てのコンテナのパラメータを指定して操作するのは非常に大変です。そこで、Kubernetes のパッケージマネージャーとして知られている Helm を利用して、それらの操作を軽減することができますようになっています。

それらは、Rancher Catalog という名前で利用できますが、通常の Helm のカタログと Rancher 独自のカスタマイズ可能な Rancher Chart を利用したカタログの 2 種類を利用することができます。これは運用する

人にとって便利な機能となっているのみならず、開発者にとっても事前に作成しておいた環境を簡単に動かすことができるというメリットをもたらしています。

これらの機能を No Vendor Lock-In つまり **どこのベンダーにも依存しない** という一貫したポリシーにより開発されています。Rancher 社のどのベンダーからも、どのクラウドにも、ツールにも依存しないという思想は逆にどこにでも置いて、どこにでもつなげて、Kubernetes の中立的な立ち位置で利用できるという製品のポジションを確立しているのです。

## 1.4. Rancher のインストール

### 1.4.1. Rancher インストールに必要な構成

Rancher 2.1.1 がサポートしている Docker のバージョンは以下の通りです。

- Ubuntu 16.04 (64-bit)
  - Docker 17.03.2
- Red Hat Enterprise Linux (RHEL)/CentOS 7.5 (64-bit)
  - RHEL Docker 1.13
  - Docker 17.03.2
- RancherOS 1.4 (64-bit)
  - Docker 17.03.2
- Windows Server version 1803 (64-bit)
  - Docker 17.06

基本的に Kubernetes がサポートしている Docker のバージョンが対応しているバージョンとなります。

### 1.4.2. Docker のインストール

Docker のバージョンは、上記のものをインストールしますが、インストールする方法は、Rancher Labs 社のインストールスクリプトを使うのが便利です。ここでは、Ubuntu 16.04 で動かします。

```
$ curl https://releases.rancher.com/install-docker/17.03.sh | sh
```

curl コマンドが入っていない場合は、curl コマンドをインストールします



```
$ sudo apt install curl
```

### 1.4.3. Rancher のインストール

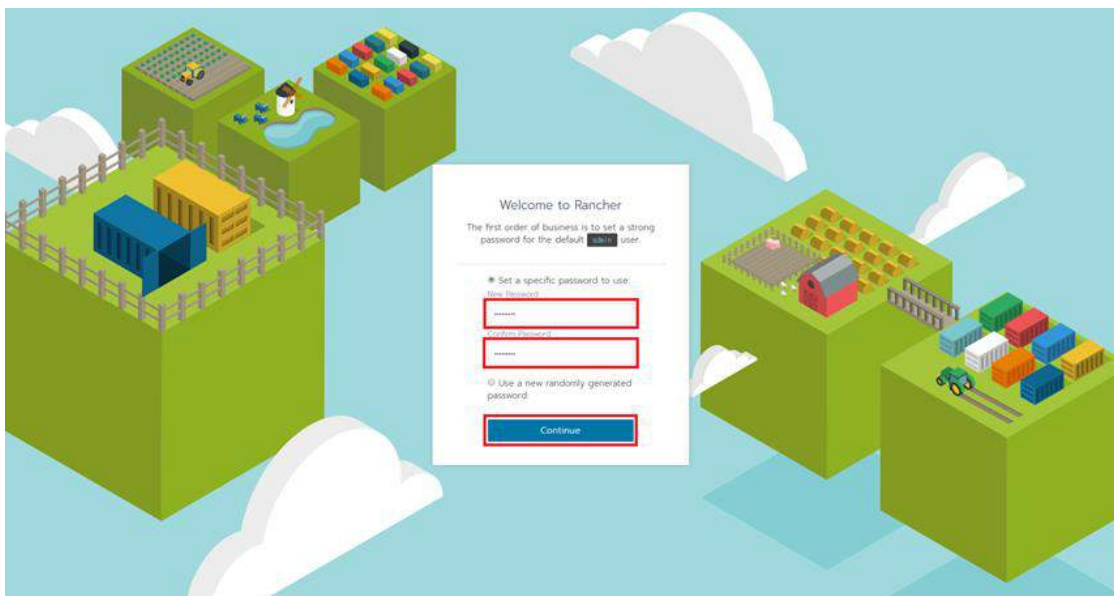
Rancher をインストールします。

```
$ sudo docker run -d --restart=unless-stopped -p 80:80 -p 443:443  
rancher/rancher
```

## 1.5. Rancher から Kubernetes のインポート

### 1.5.1. Rancher へのログイン

インストール後に Rancher をインストールしたサーバーをブラウザで表示します。以下のような画面が表示されます。(※SSL 証明書が自己証明書の場合は、承認を追加します)



パスワード設定後、Rancher 管理画面が表示されます。

### 1.5.2. Kubernetes クラスタ追加方法

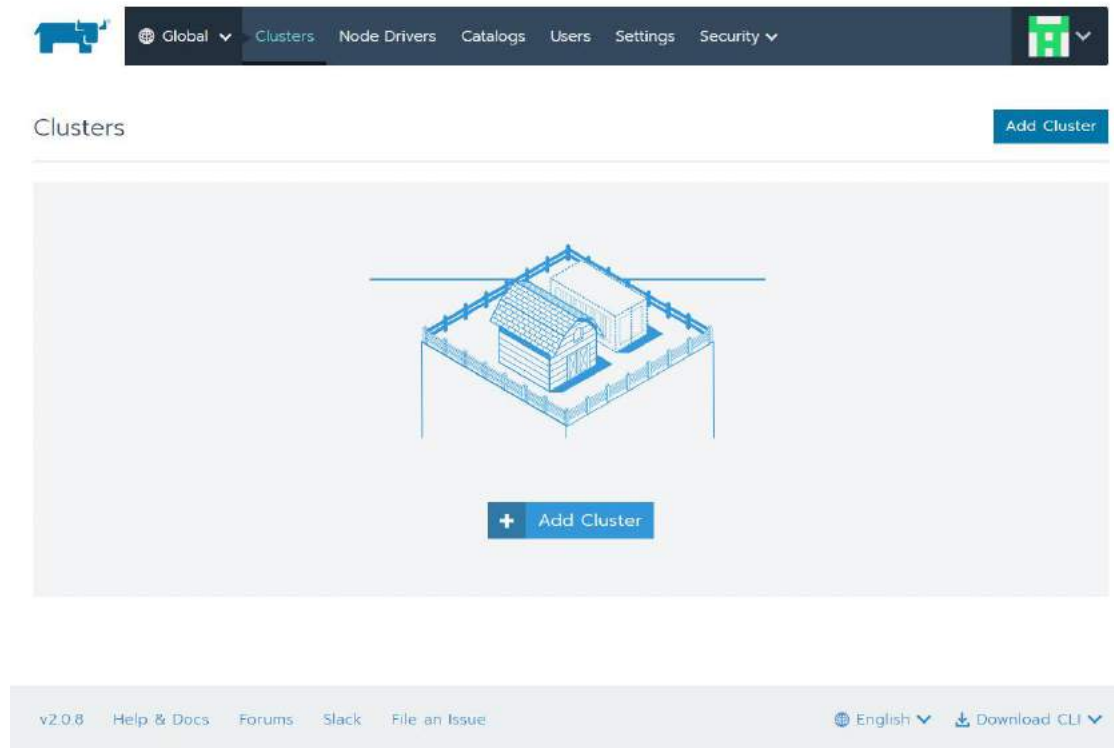
起動したばかりの Rancher には Kubernetes クラスタがありません。Kubernetes クラスタの追加方法にはいくつかの方法がありますが、ここでは事前に kubeadm で構築しておいた Kubernetes クラスタをインポートする方法をご紹介します。

kubeadm を使った Kubernetes クラスタの構築方法については下記公式ドキュメントを参照してください。

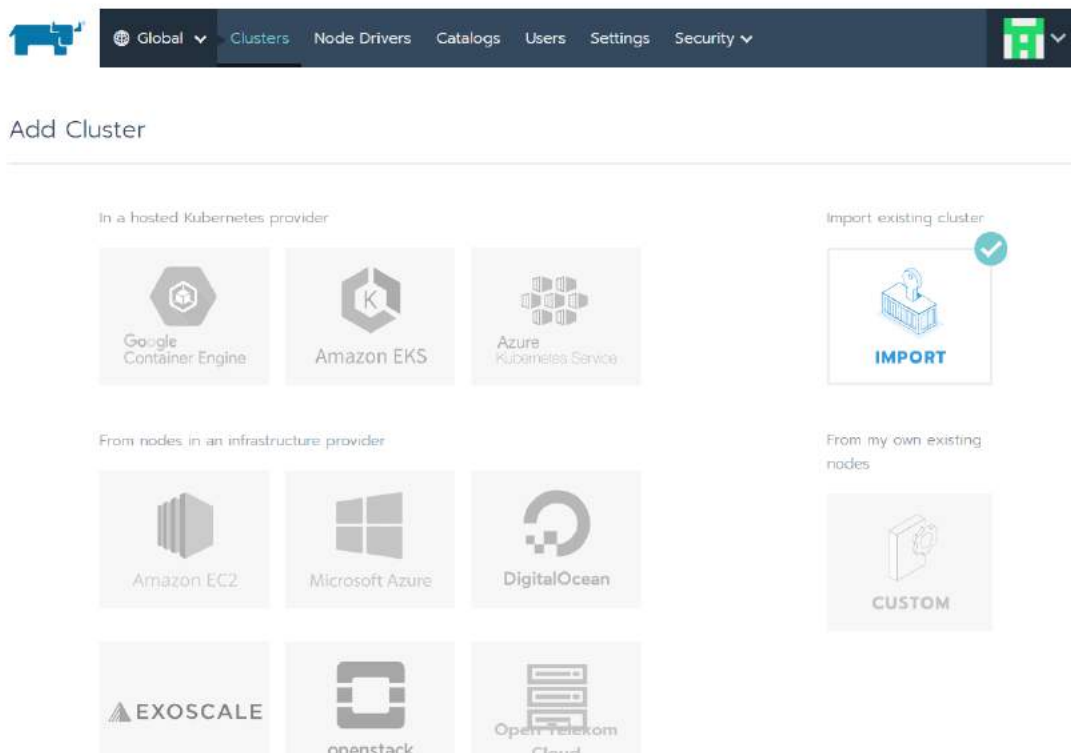
## Installing kubeadm - Kubernetes

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

次に、作った Kubernetes クラスターを Rancher から認識できるようにインポートします。Global から **Add Cluster** ボタンを押します。



クラスター追加画面が出てきますが、右上の **IMPORT** ボタンを押します。



次に、Cluster Name を指定して **Create** ボタンを押します(Member は自分一人で使う分には追加する必要はありません)。

Cluster Name \* Add a Description

e.g. sandbox

**Member Roles**  
Control who has access to the cluster and what permission they have to change it.

Name	Role
Default Admin	Cluster Owner

+ Add Member

Create Cancel

以下のページで表示されたコマンドを実行します。kubectl コマンドは事前にインストールし、kubernetes に接続できるように設定しておいてください。

Global Clusters Node Drivers Catalogs Users Settings Security

### Add Cluster: test-cluster

If your existing Kubernetes cluster already has a **cluster-admin** role defined, you must have this **cluster-admin** privilege to import the cluster into Rancher. In order to apply the privilege, you need to run the command below before running the command to import the cluster:

```
kubectl create clusterrolebinding cluster-admin-binding --clusterrole cluster-admin --user [USER_ACCOUNT]
```

Run the kubectl command below on an existing Kubernetes cluster running a supported Kubernetes version to import it into Rancher:

```
kubectl apply -f https://...:yaml
```

If you get an error about 'certificate signed by unknown authority' because your Rancher installation is running with an untrusted/self-signed SSL certificate, run the command below instead to bypass the certificate check:

```
curl --insecure -sL https://...:yaml | kubectl apply -f -
```

Done

v2.0.8 Help & Docs Forums Slack File an Issue English Download CLI

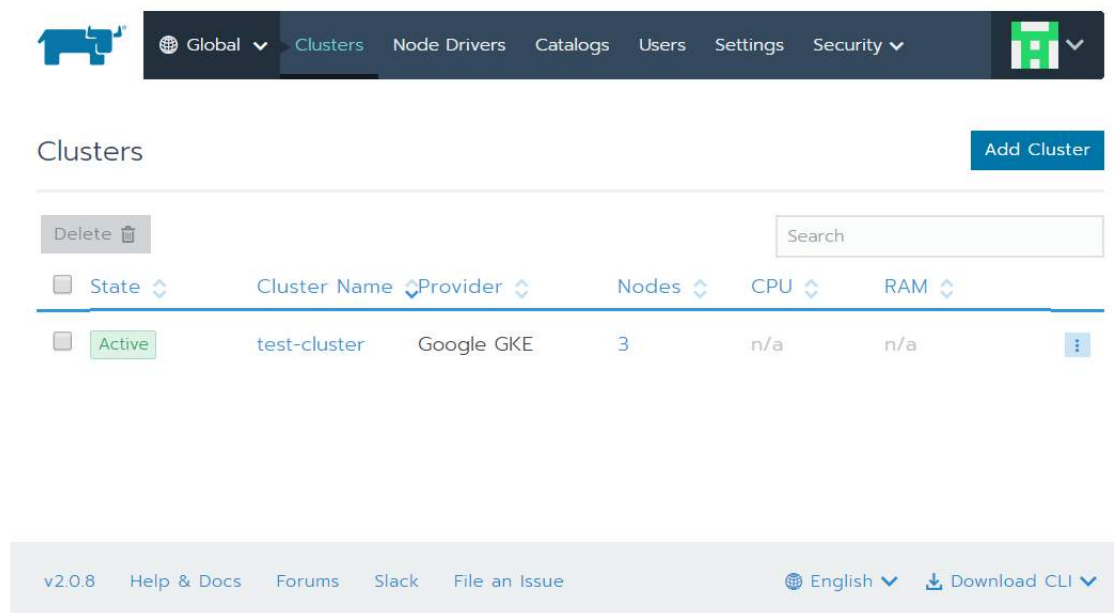
```
kubectl create clusterrolebinding cluster-admin-binding --clusterrole cluster-admin --user [USER_ACCOUNT]
```

上記の [USER\_ACCOUNT] は上記コマンドを実行するユーザーID を指定します。

上記のコマンドで証明書の問題のエラーが発生する場合は、以下のコマンドを実行して下さい。

```
curl --insecure -sL https://xxxxxxxxxxxxx.com/v3/import/XXXXXXXXXXXXXXXXXXXXXXXXX.yaml | kubectl apply -f -
```

Kubernetes クラスターが Rancher にインポートされると以下のように Global の Cluster ダッシュボードにインポートされたクラスターが表示されます。



The screenshot shows the Rancher Clusters dashboard. At the top, there is a navigation bar with the Rancher logo, a 'Global' dropdown menu, and links for 'Clusters', 'Node Drivers', 'Catalogs', 'Users', 'Settings', and 'Security'. A user profile icon is visible on the right. Below the navigation bar, the 'Clusters' section is displayed, featuring an 'Add Cluster' button. A table lists the clusters with columns for 'State', 'Cluster Name', 'Provider', 'Nodes', 'CPU', and 'RAM'. A 'Delete' button and a search input are also present. The table contains one entry: 'test-cluster' with 'Google GKE' provider, 3 nodes, and n/a for CPU and RAM. The footer includes version 'v2.0.8', links for 'Help & Docs', 'Forums', 'Slack', and 'File an Issue', along with a language dropdown set to 'English' and a 'Download CLI' link.

State	Cluster Name	Provider	Nodes	CPU	RAM
Active	test-cluster	Google GKE	3	n/a	n/a

## 2. Kubernetes のストレージについて

ここでは Kubernetes におけるデータを保管するコンポーネントについて記載します。

データを保管するもの、データを保管する領域を作成する方法やサービスカタログ化する kubernetes のストレージの章となります。

### 2.1. PersistentStorage の基本

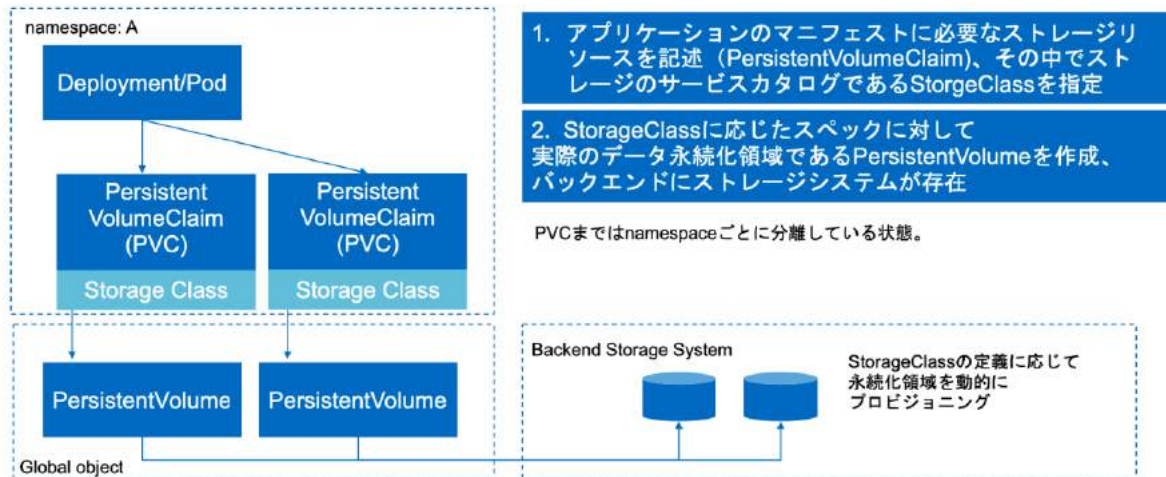
Kubernetes ではストレージに関するコンポーネントは 3 つあります。

- StorageClass
  - サービスカタログを定義するもの。
- PersistentVolumeClaim
  - アプリケーションが必要なストレージリソースを定義するオブジェクト、ここで前述の StorageClass を指定しアプリケーションに適したストレージをプロビジョニングする。
- PersistentVolume
  - 実際にデータを保管する場所を定義しているオブジェクト

それぞれの関係を図示し、ここに namespace の概念を加えたものは以下の図になります。

## 1. PersistentStorageの基本

システム観点



## 2.2. DynamicProvisioning とは

コンテナにおいてデータ永続化を実現する上でストレージは重要なコンポーネントになります。

Dynamic volume provisioning はオンデマンドにストレージをプロビジョニングするためのものです。

Static provisioning、Dynamic provisioning それぞれを比較します。

Static provisioning の場合、クラスターの管理者がストレージをプロビジョニングして、PersistentVolume オブジェクトを作成し Kubernetes に公開する必要があります。

Dynamic provisioning の場合、Static provisioning で手動で行っていたステップを自動化し、管理者がおこなっていたストレージの事前のプロビジョニング作業をなくすることができます。

StorageClass オブジェクトで指定したプロビジョナを使用し、動的にストレージリソースをプロビジョニングすることができます。

StorageClass には様々なパラメータを指定することができアプリケーションに適したストレージカタログ、プロファイルを作成することができ、物理的なストレージを抽象化するレイヤとなります。

ネットアップは Dynamic provisioning を実現するための NetApp Trident という provisioner を提供しています。

## 2.3. StorageClass の基本

StorageClass はストレージ管理者へストレージの **クラス** を定義する方法を提供します。クラスごとにサービスの内容をマッピングします、設定できる属性としては、性能におけるポリシー、データ保護に関連するポリシー、その他のポリシーを Kubernetes クラスターの管理者が設定できます。

ストレージクラスは AWS や Azure などのサービスカタログを定義するものと考えるとわかりやすいかと思います。たとえば 汎用用途の GP2、性能特価型の Provisioned IOPS、容量効率に特価した sc1 といったようなものです。

このコンセプト自体はストレージシステムではプロファイルと呼ばれることもあります。

- <https://kubernetes.io/docs/concepts/storage/storage-classes/>



### 3. Trident のコンポーネント・構築/構成

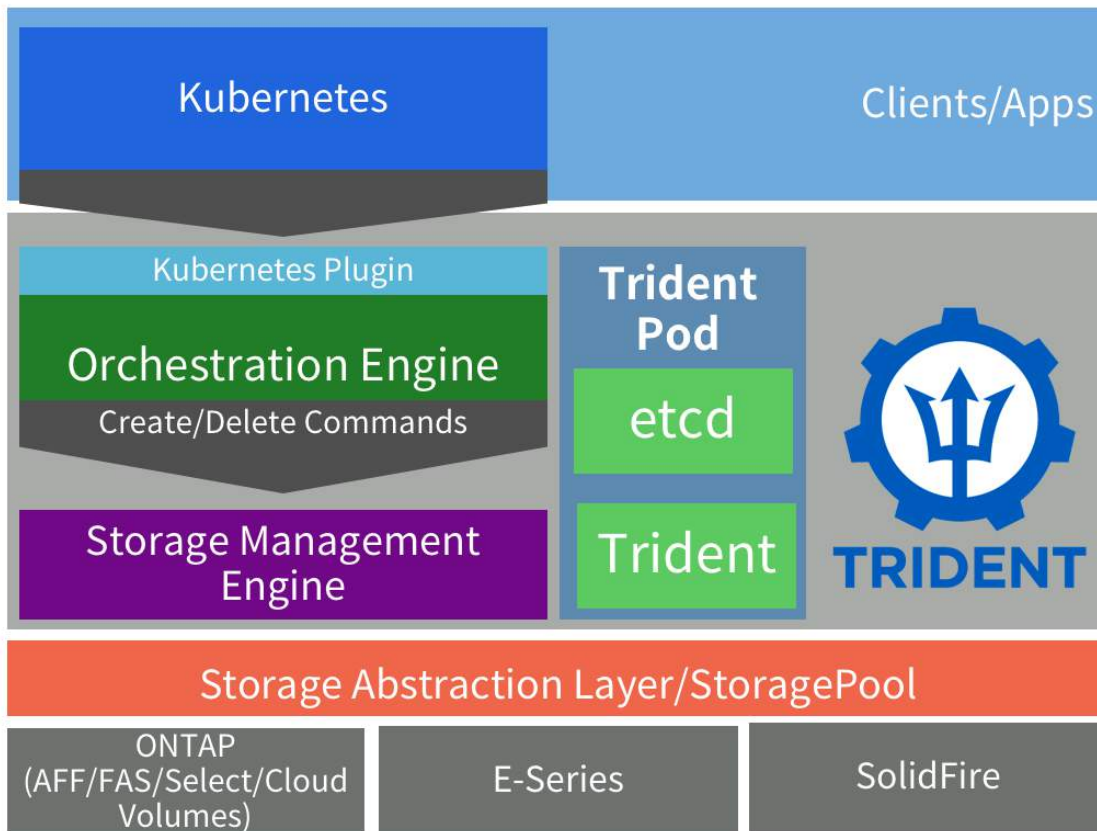
NetApp が提供している Trident は OSS として GitHub で公開しています。オンプレミスでクラウド同等のダイナミックプロビジョニングの仕組みを提供し、特徴的なストレージ機能が使用可能なものです。

- <https://github.io/netapp/trident>

#### 3.1. Trident の概要

Dynamic storage provisioning を実現するための NetApp Trident を紹介します。Trident は Pod としてデプロイされ通常のアプリケーションと同様に稼働します。

#### 動的にストレージリソースをプロビジョニングする仕組み



## 3.2. ホスト OS 設定(iscsi、nfs クライアントのインストール)

Trident のバックエンドには NFS, iSCSI が利用可能です。外部ストレージをホストにマウントするため、すべての Kubernetes クラスターに存在するノードに設定します。

RHEL、CentOS については以下の URL を確認ください。

### Worker preparation

参考までに最近の CoreOS は以下の NFS, iSCSI のパッケージがデフォルトでインストールされています。

#### 3.2.1. NFS

```
sudo apt-get install -y nfs-common
```

#### 3.2.2. iSCSI

パッケージの導入

```
sudo apt-get install -y open-iscsi lsscsi sg3-utils multipath-tools scsitools
```

マルチパスの有効化

```
$ sudo tee /etc/multipath.conf <<-'EOF'  
defaults {  
    user_friendly_names yes  
    find_multipaths yes  
}  
EOF  
  
$ sudo systemctl enable multipath-tools.service  
$ sudo service multipath-tools restart
```

open-iscsi と multipath-tools を有効化し、起動設定とします。

```
sudo systemctl status multipath-tools  
sudo systemctl enable open-iscsi.service  
sudo service open-iscsi start  
sudo systemctl status open-iscsi
```

### 3.3. バックエンドストレージ設定

#### 3.3.1. ONTAP の設定

ONTAP の事前準備は以下の通りです。

- SVM 作成・設定
- LIF 作成・設定
- export policy の作成・設定
- 使用するプロトコルのサービス有効化

以下の一連のコマンドラインを実行すると Trident からバックエンド設定する準備が整います。

```
vserver create -vserver [SVM 名] -ipSPACE [IPSpace 名] -aggregate [アグリゲート] -language C.UTF-8 -rootvolume root -rootvolume-security-style unix
```

```
nfs create -vserver [SVM 名] -access true -v3 enabled -v4.0 enabled -tcp enabled
```

```
export-policy rule create -vserver [SVM 名] -policyname default -clientmatch 0.0.0.0/0 -rorule any -rwrule any -superuser any
```

```
network interface create -vserver [SVM 名] -lif [データ LIF 名] -role data -data-protocol nfs -home-node [ホームノード名] -home-port [ホームポート] -subnet-name [サブネット名] -status-admin up -failover-policy system-defined -firewall-policy data -auto-revert true
```

```
network interface create -vserver [SVM 名] -lif [管理 LIF 名] -role data -data-protocol none -home-node [ホームノード名] -home-port [ホームポート] -subnet-name [サブネット名] -status-admin up -failover-policy system-defined -firewall-policy mgmt -auto-revert true
```

```
security login password -username vsadmin -vserver [SVM 名]
```

```
security login unlock -vserver [SVM 名] -username vsadmin
```

```
vserver modify -vserver [SVM 名] -aggr-list [アグリゲート名 (vsadmin で変更可能な aggregate を定義)]
```

#### 3.3.2. SolidFire の設定

SolidFire についての事前準備は今回の構成では不要です。定義ファイルに定義したユーザーなどは、なければ自動で作成します。

## 3.4. Trident のインストール

Trident のインストールで k8s クラスターの管理者権限が必要になります。

```
kubectl auth can-i '*' '*' --all-namespaces
```

バックエンドに登録するマネジメント IP に k8s クラスターのコンテナから疎通が取れるかを確認します。

```
kubectl run -i --tty ping --image=busybox --restart=Never --rm -- ping [ip アドレス]
```

バイナリをダウンロードしてインストールします。バックエンドストレージのための `setup/backend.json` を編集します。以下はサンプルとなります。

```
wget https://github.com/NetApp/trident/releases/download/v18.01.0/trident-installer-18.01.0.tar.gz
tar xzf trident*.tar.gz && cd trident-installer
cp sample-input/backend-ontap-nas.json setup/backend.json
```

trident のインストールや管理を簡易化するユーティリティの `tridentctl` が同一フォルダに存在します。今後のオペレーションは `tridentctl` を使用しておこないます。

### 3.4.1. Trident の構成

Trident では Trident 自体の設定(登録されているバックエンドストレージなど)を保管する必要があります。そのためインストールの時点で前節でコピーした `setup/backend.json` で構成情報を保管するボリュームを作成し、Trident のインストールプロセスでデータが保管されます。

今回は準備した ONTAP を Trident 構成情報のデータ永続化先とします。以下の構成ファイルを使用して

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "SVM 管理 IP",
  "dataLIF": "データ通信用の IP",
  "svm": "SVM 名",
  "username": "vsadmin",
  "password": "パスワード"
}
```

`tridentctl`

```
tridentctl -n trident create backend -f setup/backend.json
```

### 3.4.2. NFS バックエンドの場合(ONTAP バックエンド)

setup ディレクトリに `ontap-nas-backend.json` を作成します。修正する箇所は主に接続する IP とユーザー、パスワードです。

`setup/backend.json` を以下のように変更します。

```
{
  "version": 1,
  "storageDriverName": "ontap-nas",
  "managementLIF": "SVM 管理 IP",
  "dataLIF": "データ通信用の IP",
  "svm": "SVM 名",
  "username": "vsadmin",
  "password": "パスワード"
}
```

`backend.json` のを編集後、以下のコマンドを実行しストレージバックエンド登録を行います。

```
tridentctl -n trident create backend -f setup/backend.json
```

### 3.4.3. iSCSI バックエンドの場合(SolidFire バックエンド)

Trident では複数のストレージバックエンドを登録可能です。NFS バックエンドと同様に iSCSI バックエンドとして SolidFire を登録します。

```
{
  "version": 1,
  "storageDriverName": "solidfire-san",
  "Endpoint": "https://[ユーザ名]:[パスワード]@[MVIP のアドレス]/json-rpc/8.0",
  "SVIP": "192.168.0.240:3260", # SVIP の IP を指定
  "TenantName": "テナント名",
  "InitiatorIFace": "default",
  "UseCHAP": true,
  "Types": [
    {
      "Type": "Bronze",
      "Qos": {
        "minIOPS": 1000,
        "maxIOPS": 2000,
        "burstIOPS": 4000
      }
    },
    {
      "Type": "Silver",
      "Qos": {
        "minIOPS": 4000,
        "maxIOPS": 6000,

```

```

        "burstIOPS": 8000
    },
    {
        "Type": "Gold",
        "Qos": {
            "minIOPS": 6000,
            "maxIOPS": 8000,
            "burstIOPS": 10000
        }
    }
]
}

```

NFS バックエンド登録時との大きな違いはストレージのサービスレベルを定義しています。最小、最大、バーストの性能値をそれぞれ指定し StorageClass で性能値を指定された場合に自動でボリュームに QoS を設定することが可能となります。

### 3.5. StorageClass の定義例

前述のバックエンドごとに StorageClass を作成します。

NFS バックエンドの ONTAP 用のストレージクラスを定義します。以下のマニフェストが ONTAP 用のサンプルとなります。

`parameters` で性能面におけるサービスとプロビジョニング時のボリュームオプション、データ保護について指定しています。これらの組み合わせでサービスカタログを作成します。

- 指定可能なオプション: <https://netapp-trident.readthedocs.io/en/latest/kubernetes/operations/tasks/backends/ontap.html#ontap-aff-fas-select-cloud>

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ontap-gold
provisioner: netapp.io/trident
parameters:
  backendType: "ontap-nas"
  media: "ssd"
  provisioningType: "thin"
  snapshots: "true"
  clones: "true"

```

`tridentctl` ユーティリティを使用してバックエンド登録実施します。

```
$ ./tridentctl -n trident create backend -f setup/backend.json
```

```
+-----+-----+-----+-----+
| NAME   | STORAGE DRIVER | ONLINE | VOLUMES |
+-----+-----+-----+-----+
| svm12  | ontap-nas      | true   | 0        |
+-----+-----+-----+-----+
```

同様に SolidFire のバックエンドを登録します。

こちらのサービスカタログでは期待する IOPS 値とブロックストレージをフォーマットするファイルシステムを指定しています。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: solidfire-gold
provisioner: netapp.io/trident
parameters:
  backendType: "solidfire-san"
  IOPS: "8000"
  fsType: "ext4"
```

以下のコマンドでストレージクラスを作成します。

```
$ ./tridentctl -n trident create backend -f setup/solidfire.json
```

```
+-----+-----+-----+-----+
|          NAME          | STORAGE DRIVER | ONLINE | VOLUMES |
+-----+-----+-----+-----+
| solidfire_192.168.0.240 | solidfire-san  | true   | 0        |
+-----+-----+-----+-----+
```

これまでに作成したストレージクラスを確認します。

```
$ kubectl get sc
```

```
NAME             PROVISIONER          AGE
ontap-gold       netapp.io/trident    2m
solidfire-gold   netapp.io/trident    21s
```

`ontap-gold` ストレージクラスをデフォルトのストレージクラスに変更します。デフォルトのストレージクラスに設定すると PVC にストレージクラスを設定せずとも Dynamic Provisioning で使用されるストレージクラスが設定され利便性が向上します。

```
$ kubectl patch storageclass ontap-gold -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

```
storageclass.storage.k8s.io/ontap-gold patched
```

```
$ kubectl get sc
```

NAME	PROVISIONER	AGE
ontap-gold (default)	netapp.io/trident	5m
solidfire-gold	netapp.io/trident	3m



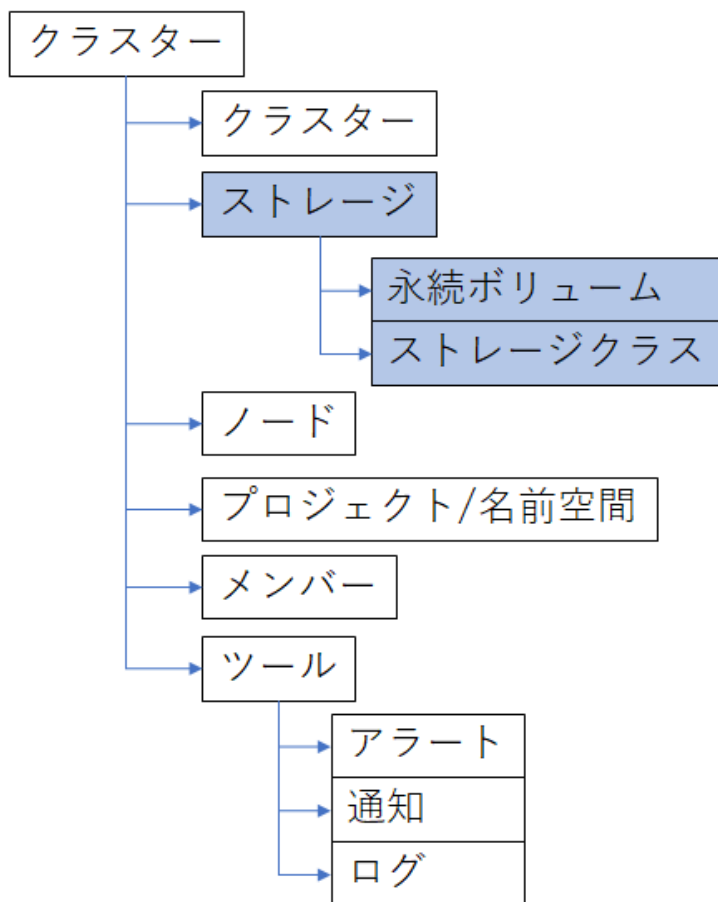
## 4. コンテナでの利用(Rancher での利用方法)

### 4.1. Dynamic Provisioning でのコンテナへのストレージの割り当て

#### 4.1.1. Rancher のメニュー構成について

まず、Rancher で Dynamic Provisioning を使う為の画面へたどり着くために Rancher UI 画面のメニュー構成について確認しておきます。クラスターのメニュー構成は以下ようになっており、ストレージ関連はクラスターに紐付いていることが分かります。

#### Rancherメニュー構成



#### 4.1.2. クラスターのストレージ

次にクラスターのストレージクラスで、第 3 章 StorageClass の定義例 で定義した ontap-gold があるのを確認します。

ストレージクラス

永続ボリューム  
ストレージクラス

クラスを追加

削除

検索

状態	名前	プロビジョナー	デフォルト
Active	ontap-gold	netapp.io/trident	

### デフォルトストレージクラスへの変更

しかし上記で確認したストレージクラスはデフォルトになっていません。これは UI 上から変更可能です。デフォルトになっていないストレージクラスの右側の縦三つをクリックして「デフォルトに設定」をクリックします。

ストレージクラス

クラスを追加

削除

検索

状態	名前	プロビジョナー	デフォルト
Active	ontap-gold	netapp.io/trident	

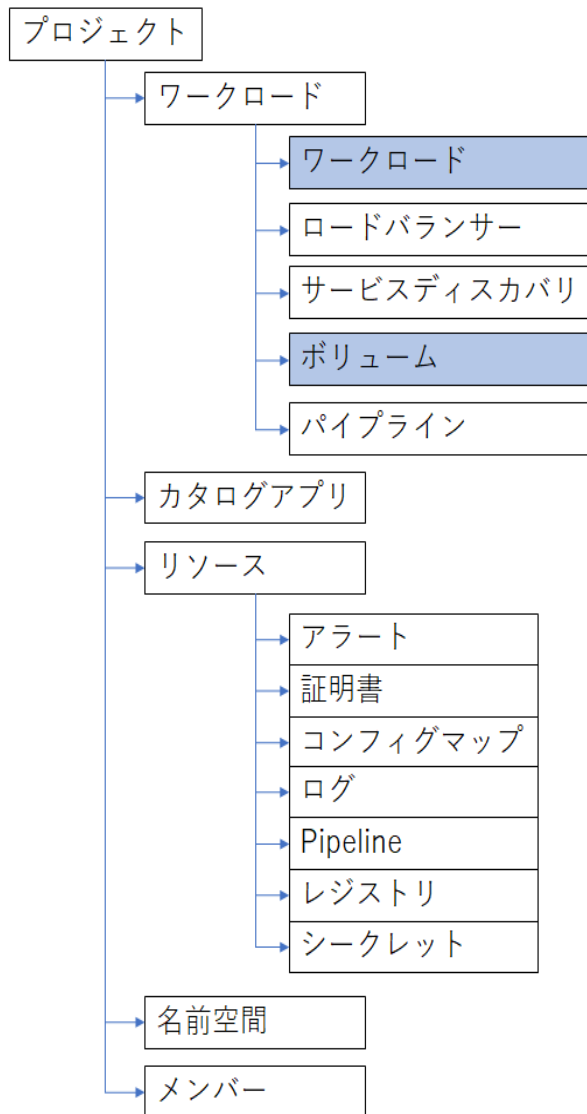
- 編集
- デフォルトに設定
- API を見る
- 削除

#### 4.1.3. プロジェクトのメニュー構成について

Rancher のメニュー構成で確認したクラスターのメニューからは、コンテナをデプロイすることはできませんでした。コンテナをデプロイするには、Rancher のプロジェクトメニューから行う必要があります。

まず、Rancher のプロジェクトのメニュー構成について確認しておきます。以下ようになっており、今回利用するのは網掛けのワークロードとボリュームです。

## プロジェクトメニュー構成



プロジェクトメニューにいっには、グローバルメニューからデフォルトで用意されている「Default」プロジェクトに切り換えます。



#### 4.1.4. コンテナをデプロイする

コンテナをデプロイして、ストレージを割り当てます。ワークロードからワークロードを選択し、「デプロイ」ボタンを押します。



次の画面でワークロードデプロイに名前とイメージを指定します。

## ワークロードをデプロイ

名前 \* [詳細情報を追加](#) ワークロードタイプ [詳細オプションを表示](#)  
例: myapp  スケーラブルデプロイメント:  ポッド

Docker イメージ \*  [新しい名前空間を追加](#)  
名前空間 \*  [新しい名前空間を追加](#)

### ポートマッピング

+ [ポートを追加](#)

[すべて開く](#)

- 環境変数**  
シーケレットなどの他のリソースから投入する任意のコンテナから多数的に環境変数を設定します。
- ノード スケジューリング**  
ポッドがどのノード上でデプロイされるかを設定します。
- ヘルスチェック**  
定期的にコンテナヘルスチェックを送り、コンテナが生存して正しく応答するかどうかを確認します。
- ボリューム**  
個々のコンテナのライフサイクルから分離され、永続化やデータ共有に利用されます。
- スケーリング/アップグレード ポリシー**  
アップグレードを実行した際のポッドの置換方法を設定します。

[詳細オプションを表示](#)

#### 4.1.5. ボリュームの割り当て

次にボリュームを割り当てます。「ボリューム」をクリックします。「ボリュームを追加」をクリックして、「新しい永続ボリューム(要求)を作成」をクリックします。



名前とストレージクラスを指定して、容量を入力します。ソースは、「新しい永続ボリュームの作成にストレージクラスを使用」を選択します。

## ボリューム要求を追加

名前 詳細情報を追加

test-volume

ソース ストレージクラス

新しい永続ボリュームの作成にストレージクラスを使用 ontap-gold ▼

既存の永続ボリュームを使用

容量 \*

10 GiB

---

▼ カスタマイズ  
カスタマイズ用の拡張オプションです

アクセスモード

単一ノード読み取り/書き込み

複数ノード読み取り専用

複数ノード読み取り/書き込み

定義 キャンセル

「定義」ボタンをクリックします。

#### 4.1.6. ボリュームのマウント先を指定

ボリューム定義はできたので、それをコンテナのどこに割り当てるか設定します。

#### ボリューム要求を追加

名前 詳細情報を追加

test-volume

ソース ストレージクラス

新しい永続ボリュームの作成にストレージクラスを使用

既存の永続ボリュームを使用

ontap-gold

容量 <sup>\*</sup>

10 GiB

**カスタマイズ**  
カスタマイズ用の拡張オプションです

アクセスモード

単一ノード読み取り/書き込み

複数ノード読み取り専用

複数ノード読み取り/書き込み

定義 キャンセル

マウントポイントを指定します。これがコンテナ側から見られるパスになります。さらにその先にパスがある場合は、Sub Path in Volume を指定します。

ワークロードをデプロイの一番下にある「起動」を押してコンテナを起動します。

#### 4.1.7. マウントを確認

起動したコンテナに接続して、マウント状況を確認します。





## 4.2. StatefulSet を使ったダイナミックプロビジョニングの例

データが分散され保管されるアーキテクチャの場合、スケールするごとにストレージをプロビジョニングする必要があります。

このセクションでは MongoDB のレプリカ数に応じてストレージをプロビジョニングしデプロイします。

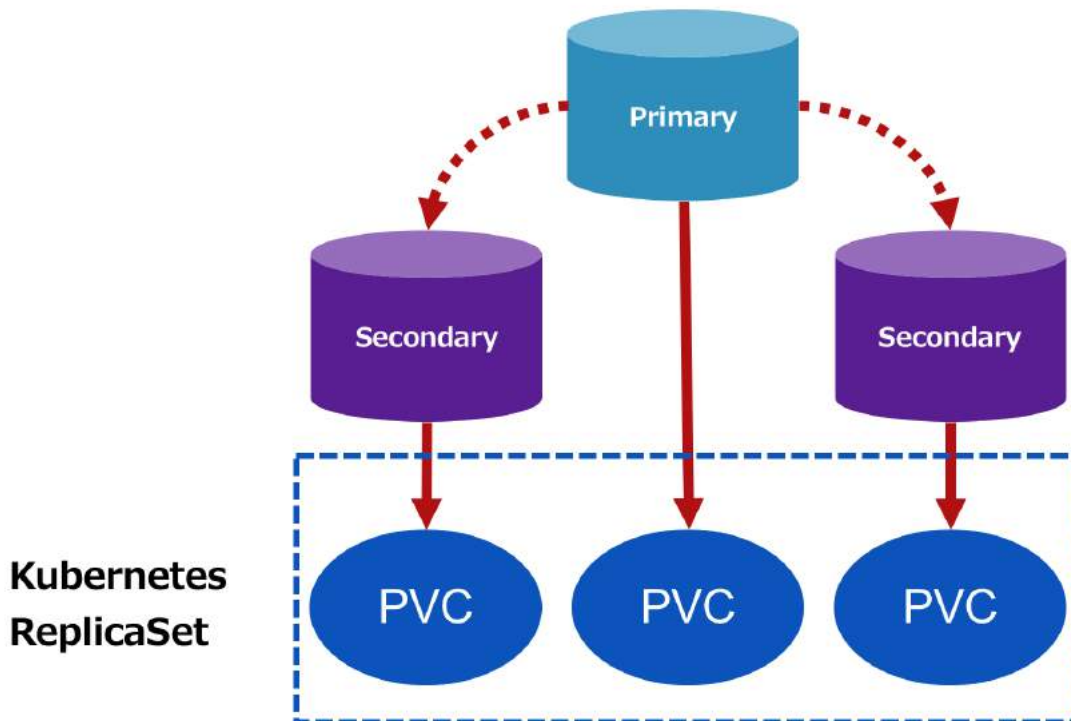
MongoDB のインストールには Helm を使用します。

### 4.2.1. MongoDB レプリカセット

MongoDB では高可用性やデータ可用性を実現する仕組みとして ReplicaSet を提供しています。

MongoDB クラスターでプライマリからセカンダリへデータをレプリケーションし、有事の際にはフェイルオーバーしデータへのアクセスを継続します。

Kubernetes 上では MongoDB のようなデータレプリケーションをするアプリケーション向けに PVC を割り当てる必要があります。Kubernetes の ReplicaSet を使用して Volume Claim template から動的に PVC を作成しコンテナに割り当てることができます。



- Kubernetes StatefulSet:  
<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

#### 4.2.2. Helm 用設定ファイルの準備

適応する `values.yaml` は以下の通りです。デフォルトからは dynamic provisioning をするため `storageClass` を変更しています。

```
replicas: 3
port: 27017

replicaSetName: rs0

podDisruptionBudget: {}
  # maxUnavailable: 1
  # minAvailable: 2

auth:
  enabled: false
  # adminUser: username
  # adminPassword: password
  # metricsUser: metrics
  # metricsPassword: password
  # key: keycontent
  # existingKeySecret:
  # existingAdminSecret:
  # existingMetricsSecret:

# Specs for the Docker image for the init container that establishes the replica
set
installImage:
  repository: k8s.gcr.io/mongodb-install
  tag: 0.6
  pullPolicy: IfNotPresent

# Specs for the MongoDB image
image:
  repository: mongo
  tag: 3.6
  pullPolicy: IfNotPresent

# Additional environment variables to be set in the container
extraVars: {}
# - name: TCMALLOC_AGGRESSIVE_DECOMMIT
#   value: "true"

# Prometheus Metrics Exporter
metrics:
  enabled: false
```

```

image:
  repository: ssalaues/mongodb-exporter
  tag: 0.6.1
  pullPolicy: IfNotPresent
port: 9216
path: "/metrics"
socketTimeout: 3s
syncTimeout: 1m
prometheusServiceDiscovery: true
resources: {}

# Annotations to be added to MongoDB pods
podAnnotations: {}

securityContext:
  runAsUser: 999
  fsGroup: 999
  runAsNonRoot: true

resources: {}
# limits:
#   cpu: 500m
#   memory: 512Mi
# requests:
#   cpu: 100m
#   memory: 256Mi

## Node selector
## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#nodeselector
nodeSelector: {}

affinity: {}

tolerations: []

extraLabels: {}

persistentVolume:
  enabled: true
  ## mongodb-replicaset data Persistent Volume Storage Class
  ## If defined, storageClassName: <storageClass>
  ## If set to "-", storageClassName: "", which disables dynamic provisioning
  ## If undefined (the default) or set to null, no storageClassName spec is
  ##   set, choosing the default provisioner. (gp2 on AWS, standard on
  ##   GKE, AWS & OpenStack)
  ##
  storageClass: "solidfire-gold"
  accessModes:
    - ReadWriteOnce

```

```

    size: 10Gi
    annotations: {}

# Annotations to be added to the service
serviceAnnotations: {}

tls:
  # Enable or disable MongoDB TLS support
  enabled: false
  # Please generate your own TLS CA by generating it via:
  # $ openssl genrsa -out ca.key 2048
  # $ openssl req -x509 -new -nodes -key ca.key -days 10000 -out ca.crt -subj
"/CN=mydomain.com"
  # After that you can base64 encode it and paste it here:
  # $ cat ca.key | base64 -w0
  # cacert:
  # cakey:

# Entries for the MongoDB config file
configmap:

# Readiness probe
readinessProbe:
  initialDelaySeconds: 5
  timeoutSeconds: 1
  failureThreshold: 3
  periodSeconds: 10
  successThreshold: 1

# Liveness probe
livenessProbe:
  initialDelaySeconds: 30
  timeoutSeconds: 5
  failureThreshold: 3
  periodSeconds: 10
  successThreshold: 1

```

#### 4.2.3. MongoDB デプロイ

MongoDB をインストールします。

MongoDB をデプロイする名前スペースを作成します。

```

$ kubectl create ns mongo-replica
namespace/mongo-replica created

```

以下の URL を参考に、replicaset の永続化のバックエンドストレージには iSCSI プロトコルを使用した SolidFire を使用しています。

- <https://github.com/helm/charts/tree/master/stable/mongodb-replicaset>

Helm を使い MongoDB をデプロイします。

```
$ helm install --name mongodb --namespace mongo-replica -f values.yaml
stable/mongodb-replicaset

NAME:    mongodb
LAST DEPLOYED: Fri Jul 27 20:43:15 2018
NAMESPACE: mongo-replica
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME                                DATA  AGE
mongodb-mongodb-replicaset-init    1      0s
mongodb-mongodb-replicaset-mongodb 1      0s
mongodb-mongodb-replicaset-tests   1      0s

==> v1/Service
NAME                                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
mongodb-mongodb-replicaset         ClusterIP      None          <none>        27017/TCP  0s

==> v1beta2/StatefulSet
NAME                                DESIRED  CURRENT  AGE
mongodb-mongodb-replicaset         3        1        0s

==> v1/Pod(related)
NAME                                READY  STATUS   RESTARTS  AGE
mongodb-mongodb-replicaset-0       0/1    Pending  0          0s

NOTES:
1. After the statefulset is created completely, one can check which instance is
primary by running:

    $ for ((i = 0; i < 3; ++i)); do kubectl exec --namespace mongo-replica
mongodb-mongodb-replicaset-$i -- sh -c 'mongo --
eval="printjson(rs.isMaster())"'; done

2. One can insert a key into the primary instance of the mongodb replica set by
running the following:
    MASTER_POD_NAME must be replaced with the name of the master found from the
previous step.
```

```
$ kubectl exec --namespace mongo-replica MASTER_POD_NAME -- mongo --eval="printjson(db.test.insert({key1: 'value1'}))"
```

3. One can fetch the keys stored in the primary or any of the slave nodes in the following manner.

POD\_NAME must be replaced by the name of the pod being queried.

```
$ kubectl exec --namespace mongo-replica POD_NAME -- mongo --eval="rs.slaveOk(); db.test.find().forEach(printjson)"
```

生成された Kubernetes オブジェクトを確認します。

```
$ kubectl get all -n mongo-replica
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mongodb-mongodb-replicaset-0	1/1	Running	0	3m
pod/mongodb-mongodb-replicaset-1	1/1	Running	0	2m
pod/mongodb-mongodb-replicaset-2	1/1	Running	0	1m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
service/mongodb-mongodb-replicaset	ClusterIP	None	<none>
27017/TCP	3m		

NAME	DESIRED	CURRENT	AGE
statefulset.apps/mongodb-mongodb-replicaset	3	3	3m

```
$ kubectl get statefulset -n mongo-replica
```

NAME	DESIRED	CURRENT	AGE
mongodb-mongodb-replicaset	3	3	3m

```
$ kubectl describe statefulset -n mongo-replica
```

```
Name:          mongodb-mongodb-replicaset
Namespace:     mongo-replica
CreationTimestamp: Fri, 27 Jul 2018 20:43:15 +0900
Selector:      app=mongodb-replicaset,release=mongodb
Labels:        app=mongodb-replicaset
               chart=mongodb-replicaset-3.5.2
               heritage=Tiller
               release=mongodb
Annotations:   <none>
Replicas:      3 desired | 3 total
Update Strategy: RollingUpdate
Pods Status:   3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:       app=mongodb-replicaset
               release=mongodb
  Annotations: prometheus.io/path=/metrics
               prometheus.io/port=9216
               prometheus.io/scrape=true
```

```

Init Containers:
copy-config:
  Image:      busybox
  Port:       <none>
  Host Port:  <none>
  Command:
    sh
  Args:
    -c
    set -e
set -x

cp /configdb-readonly/mongod.conf /data/configdb/mongod.conf

  Environment: <none>
  Mounts:
    /configdb-readonly from config (rw)
    /data/configdb from configdir (rw)
    /work-dir from workdir (rw)
install:
  Image:      k8s.gcr.io/mongodb-install:0.6
  Port:       <none>
  Host Port:  <none>
  Args:
    --work-dir=/work-dir
  Environment: <none>
  Mounts:
    /work-dir from workdir (rw)
bootstrap:
  Image:      mongo:3.6
  Port:       <none>
  Host Port:  <none>
  Command:
    /work-dir/peer-finder
  Args:
    -on-start=/init/on-start.sh
    -service=mongodb-mongodb-replicaset
  Environment:
    POD_NAMESPACE:  (v1:metadata.namespace)
    REPLICA_SET:    rs0
  Mounts:
    /data/configdb from configdir (rw)
    /data/db from datadir (rw)
    /init from init (rw)
    /work-dir from workdir (rw)
Containers:
mongodb-replicaset:
  Image:      mongo:3.6
  Port:       27017/TCP
  Host Port:  0/TCP

```



```

Command:
  mongod
Args:
  --config=/data/configdb/mongod.conf
  --dbpath=/data/db
  --replSet=rs0
  --port=27017
  --bind_ip=0.0.0.0
Liveness:   exec [mongo --eval db.adminCommand('ping')] delay=30s
timeout=5s period=10s #success=1 #failure=3
Readiness:   exec [mongo --eval db.adminCommand('ping')] delay=5s
timeout=1s period=10s #success=1 #failure=3
Environment: <none>
Mounts:
  /data/configdb from configdir (rw)
  /data/db from datadir (rw)
  /work-dir from workdir (rw)
Volumes:
config:
  Type:          ConfigMap (a volume populated by a ConfigMap)
  Name:          mongodb-mongodb-replicaset-mongodb
  Optional:     false
init:
  Type:          ConfigMap (a volume populated by a ConfigMap)
  Name:          mongodb-mongodb-replicaset-init
  Optional:     false
workdir:
  Type:          EmptyDir (a temporary directory that shares a pod's lifetime)
  Medium:
configdir:
  Type:          EmptyDir (a temporary directory that shares a pod's lifetime)
  Medium:
Volume Claims:
  Name:          datadir
  StorageClass:  solidfire-gold
  Labels:        <none>
  Annotations:   <none>
  Capacity:      10Gi
  Access Modes:  [ReadWriteOnce]
Events:
  Type           Reason             Age   From                    Message
  ----           -
  Normal        SuccessfulCreate    3m   statefulset-controller create Claim datadir-
mongodb-mongodb-replicaset-0 Pod mongodb-mongodb-replicaset-0 in StatefulSet
mongodb-mongodb-replicaset success
  Normal        SuccessfulCreate    3m   statefulset-controller create Pod mongodb-
mongodb-replicaset-0 in StatefulSet mongodb-mongodb-replicaset successful
  Normal        SuccessfulCreate    2m   statefulset-controller create Claim datadir-
mongodb-mongodb-replicaset-1 Pod mongodb-mongodb-replicaset-1 in StatefulSet
mongodb-mongodb-replicaset success

```

```

Normal SuccessfulCreate 2m statefulset-controller create Pod mongodb-
mongodb-replicaset-1 in StatefulSet mongodb-mongodb-replicaset successful
Normal SuccessfulCreate 1m statefulset-controller create Claim datadir-
mongodb-mongodb-replicaset-2 Pod mongodb-mongodb-replicaset-2 in StatefulSet
mongodb-mongodb-replicaset success
Normal SuccessfulCreate 1m statefulset-controller create Pod mongodb-
mongodb-replicaset-2 in StatefulSet mongodb-mongodb-replicaset successful

```

Kubernetes サイドより作成された PVC を確認します。

```
$ kubectl get pvc -n mongo-replica -o wide
```

NAME	CAPACITY	ACCESS MODES	STORAGECLASS	STATUS	AGE	VOLUME
datadir-mongodb-mongodb-replicaset-0				Bound		mongo-replica-datadir-mongodb-solidfire-gold
			10Gi	RWO	24m	
datadir-mongodb-mongodb-replicaset-1				Bound		mongo-replica-datadir-mongodb-solidfire-gold
			10Gi	RWO	23m	
datadir-mongodb-mongodb-replicaset-2				Bound		mongo-replica-datadir-mongodb-solidfire-gold
			10Gi	RWO	22m	

バックエンドのストレージに作成された実体のストレージボリュームを確認します。

CLASS	PROTOCOL	NAME	BACKEND	POOL	SIZE	STORAGE
mongo-replica-datadir-mongodb-mongodb-replicaset-2-7baf3-gold	block	solidfire_192.168.0.240	Gold		10 GiB	solidfire-gold
default-mysql-pv-claim-4c942	file	svm12	aggr1_01		20 GiB	ontap-gold
default-wp-pv-claim-7bb75	file	svm12	aggr1_01		20 GiB	ontap-gold
mongo-replica-datadir-mongodb-mongodb-replicaset-0-3f8e9-gold	block	solidfire_192.168.0.240	Gold		10 GiB	solidfire-gold
mongo-replica-datadir-mongodb-mongodb-replicaset-1-5f8c1-gold	block	solidfire_192.168.0.240	Gold		10 GiB	solidfire-gold

MongoDB のマスターノードを確認します、マスタが選出され正常稼働していることが確認できました。

```
$ for ((i = 0; i < 3; ++i)); do kubectl exec --namespace mongo-replica mongodb-
mongodb-replicaset-$i -- sh -c 'mongo --eval="printjson(rs.isMaster())"'; done
```

```

MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.6

```

```

{
  "hosts" : [
    "mongodb-mongodb-replicaset-0.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-1.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-2.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017"
  ],
  "setName" : "rs0",
  "setVersion" : 3,
  "ismaster" : true,
  "secondary" : false,
  "primary" : "mongodb-mongodb-replicaset-0.mongodb-mongodb-
replicaset.mongo-replica.svc.cluster.local:27017",
  "me" : "mongodb-mongodb-replicaset-0.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
  "electionId" : ObjectId("7fffffff0000000000000002"),
  "lastWrite" : {
    "opTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "lastWriteDate" : ISODate("2018-07-27T11:53:23Z"),
    "majorityOpTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "majorityWriteDate" : ISODate("2018-07-27T11:53:23Z")
  },
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 100000,
  "localTime" : ISODate("2018-07-27T11:53:25.350Z"),
  "logicalSessionTimeoutMinutes" : 30,
  "minWireVersion" : 0,
  "maxWireVersion" : 6,
  "readOnly" : false,
  "ok" : 1,
  "operationTime" : Timestamp(1532692403, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1532692403, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
}
MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017

```

```

MongoDB server version: 3.6.6
{
  "hosts" : [
    "mongodb-mongodb-replicaset-0.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-1.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-2.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017"
  ],
  "setName" : "rs0",
  "setVersion" : 3,
  "ismaster" : false,
  "secondary" : true,
  "primary" : "mongodb-mongodb-replicaset-0.mongodb-mongodb-
replicaset.mongo-replica.svc.cluster.local:27017",
  "me" : "mongodb-mongodb-replicaset-1.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
  "lastWrite" : {
    "opTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "lastWriteDate" : ISODate("2018-07-27T11:53:23Z"),
    "majorityOpTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "majorityWriteDate" : ISODate("2018-07-27T11:53:23Z")
  },
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 100000,
  "localTime" : ISODate("2018-07-27T11:53:25.705Z"),
  "logicalSessionTimeoutMinutes" : 30,
  "minWireVersion" : 0,
  "maxWireVersion" : 6,
  "readOnly" : false,
  "ok" : 1,
  "operationTime" : Timestamp(1532692403, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1532692403, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017

```

```

MongoDB server version: 3.6.6
{
  "hosts" : [
    "mongodb-mongodb-replicaset-0.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-1.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
    "mongodb-mongodb-replicaset-2.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017"
  ],
  "setName" : "rs0",
  "setVersion" : 3,
  "ismaster" : false,
  "secondary" : true,
  "primary" : "mongodb-mongodb-replicaset-0.mongodb-mongodb-
replicaset.mongo-replica.svc.cluster.local:27017",
  "me" : "mongodb-mongodb-replicaset-2.mongodb-mongodb-replicaset.mongo-
replica.svc.cluster.local:27017",
  "lastWrite" : {
    "opTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "lastWriteDate" : ISODate("2018-07-27T11:53:23Z"),
    "majorityOpTime" : {
      "ts" : Timestamp(1532692403, 1),
      "t" : NumberLong(2)
    },
    "majorityWriteDate" : ISODate("2018-07-27T11:53:23Z")
  },
  "maxBsonObjectSize" : 16777216,
  "maxMessageSizeBytes" : 48000000,
  "maxWriteBatchSize" : 100000,
  "localTime" : ISODate("2018-07-27T11:53:26.068Z"),
  "logicalSessionTimeoutMinutes" : 30,
  "minWireVersion" : 0,
  "maxWireVersion" : 6,
  "readOnly" : false,
  "ok" : 1,
  "operationTime" : Timestamp(1532692403, 1),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1532692403, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}

```

#### 4.2.4. ノード障害発生時の動作確認

##### 確認事項

ノード障害が発生し、Deployment で管理されているポッドが別ポッドとして起動した場合にもデータが永続化されていることを確認します。

以下のフローで永続化の確認をします。

1. 障害発生前にデータを保存
2. Deployment で管理されている Pod をすべて削除
3. 再度 Deployment で Pod が起動され、別ノードで起動時に「1.」で保存したデータを参照できることを確認

確認のためのテストデータを投入します。

```
$ kubectl exec mongodb-mongodb-replicaset-0 -n mongo-replica -- mongo --eval="printjson(db.test.insert({key1: 'trident fail test'}))"
MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.6
{ "nInserted" : 1 }
```

投入したデータを確認します。

```
$ kubectl exec mongodb-mongodb-replicaset-0 -n mongo-replica -- mongo --eval="rs.slaveOk(); db.test.find({key1: {\$exists:true}}).forEach(printjson)"
MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.6
{
  "_id" : ObjectId("5b5b0d43e521f72e61cf2f9f"),
  "key1" : "trident fail test"
}
```

Replica Set が各ノードに配置されています。

```
mongodb-mongodb-replicaset-0 1/1 Running 0 5m
10.244.3.6 node2
mongodb-mongodb-replicaset-1 1/1 Running 0 5m
10.244.4.8 node3
mongodb-mongodb-replicaset-2 1/1 Running 0 4m
10.244.1.5 node0
```

ここで Helm Charts で付与されているラベルで Pod をすべて削除します。

```
$ kubectl delete po -l "app=mongodb-replicaset,release=mongodb" -n mongo-replica
pod "mongodb-mongodb-replicaset-0" deleted
pod "mongodb-mongodb-replicaset-1" deleted
pod "mongodb-mongodb-replicaset-2" deleted
```

Deployment により管理されているため、Pod が停止したのを検知して再度 Pod が起動されます。

```
$ kubectl get po --watch-only -n mongo-replica -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
mongodb-mongodb-replicaset-0 10.244.3.6 node2	1/1	Terminating	0	7m	
mongodb-mongodb-replicaset-1 10.244.4.8 node3	1/1	Terminating	0	6m	
mongodb-mongodb-replicaset-2 10.244.1.5 node0	1/1	Terminating	0	6m	
mongodb-mongodb-replicaset-0 10.244.3.6 node2	1/1	Terminating	0	7m	
mongodb-mongodb-replicaset-1 10.244.4.8 node3	1/1	Terminating	0	6m	
mongodb-mongodb-replicaset-2 10.244.1.5 node0	1/1	Terminating	0	6m	
mongodb-mongodb-replicaset-0 10.244.3.6 node2	0/1	Terminating	0	7m	
mongodb-mongodb-replicaset-1 <none> node3	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-2 <none> node0	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-2 <none> node0	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-1 <none> node3	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-2 <none> node0	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-1 <none> node3	0/1	Terminating	0	6m	
mongodb-mongodb-replicaset-0 10.244.3.6 node2	0/1	Terminating	0	7m	
mongodb-mongodb-replicaset-0 10.244.3.6 node2	0/1	Terminating	0	7m	
mongodb-mongodb-replicaset-0 <none>	0/1	Pending	0	0s	<none>
mongodb-mongodb-replicaset-0 node3	0/1	Pending	0	0s	<none>
mongodb-mongodb-replicaset-0 node3	0/1	Init:0/3	0	0s	<none>
mongodb-mongodb-replicaset-0 10.244.4.9 node3	0/1	Init:1/3	0	25s	

```

mongodb-mongodb-replicaset-0 0/1      Init:2/3  0      26s
10.244.4.9 node3
mongodb-mongodb-replicaset-0 0/1      Init:2/3  0      27s
10.244.4.9 node3
mongodb-mongodb-replicaset-0 0/1      PodInitializing  0      30s
10.244.4.9 node3
mongodb-mongodb-replicaset-0 0/1      Running    0      31s
10.244.4.9 node3
mongodb-mongodb-replicaset-0 1/1      Running    0      44s
10.244.4.9 node3
mongodb-mongodb-replicaset-1 0/1      Pending    0      0s      <none>
<none>
mongodb-mongodb-replicaset-1 0/1      Pending    0      0s      <none>
node2
mongodb-mongodb-replicaset-1 0/1      Init:0/3   0      0s      <none>
node2
mongodb-mongodb-replicaset-1 0/1      Init:1/3   0      17s
10.244.3.7 node2
mongodb-mongodb-replicaset-1 0/1      Init:2/3   0      18s
10.244.3.7 node2
mongodb-mongodb-replicaset-1 0/1      Init:2/3   0      19s
10.244.3.7 node2
mongodb-mongodb-replicaset-1 0/1      PodInitializing  0      22s
10.244.3.7 node2
mongodb-mongodb-replicaset-1 0/1      Running    0      23s
10.244.3.7 node2
mongodb-mongodb-replicaset-1 1/1      Running    0      35s
10.244.3.7 node2
mongodb-mongodb-replicaset-2 0/1      Pending    0      0s      <none>
<none>
mongodb-mongodb-replicaset-2 0/1      Pending    0      0s      <none>
node1
mongodb-mongodb-replicaset-2 0/1      Init:0/3   0      0s      <none>
node1
mongodb-mongodb-replicaset-2 0/1      Init:1/3   0      9s
10.244.2.6 node1
mongodb-mongodb-replicaset-2 0/1      Init:2/3   0      10s
10.244.2.6 node1
mongodb-mongodb-replicaset-2 0/1      Init:2/3   0      11s
10.244.2.6 node1
mongodb-mongodb-replicaset-2 0/1      PodInitializing  0      18s
10.244.2.6 node1
mongodb-mongodb-replicaset-2 0/1      Running    0      19s
10.244.2.6 node1
mongodb-mongodb-replicaset-2 1/1      Running    0      27s
10.244.2.6 node1

```

最終的に起動した replicaset が Pod 停止前と別のノードで起動されています。

```
$ kubectl get po -n mongo-replica -o wide
```



NAME	READY	STATUS	RESTARTS	AGE	IP
mongodb-mongodb-replicaset-0	1/1	Running	0	2m	10.244.4.9 node3
mongodb-mongodb-replicaset-1	1/1	Running	0	1m	10.244.3.7 node2
mongodb-mongodb-replicaset-2	1/1	Running	0	1m	10.244.2.6 node1

テスト前に入れたデータベースの値を確認します。

```
$ kubectl exec mongodb-mongodb-replicaset-0 -n mongo-replica -- mongo --
eval="rs.slaveOk(); db.test.find({key1:{$exists:true}}).forEach(printjson)"

MongoDB shell version v3.6.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.6
{
  "_id" : ObjectId("5b5b0d43e521f72e61cf2f9f"),
  "key1" : "trident fail test"
}
```

ポッドが削除され、再作成された上でもデータは永続化している状態が確認できました。

### 4.3. Trident のバックエンドストレージに設定した ONTAP での使い方

ストレージ機能としてストレージ全体を遠隔地にレプリケーションする機能があります。(SVM-DR)

ここでは Heptio ark を使用し、Kubernetes の構成情報をバックアップ/リストアし、実データを SVM-DR を用いデータ転送後、別 Kubernetes 環境でリストアする方法について説明します。

なお、ここでは2つの Kubernetes クラスターのプライマリクラスター、DR クラスターが稼働している前提です。

#### 4.3.1. Heptio Ark

Heptio 社が提供している Kubernetes のデファクトのバックアップ・リストアツールです。

- <https://github.com/heptio/ark>

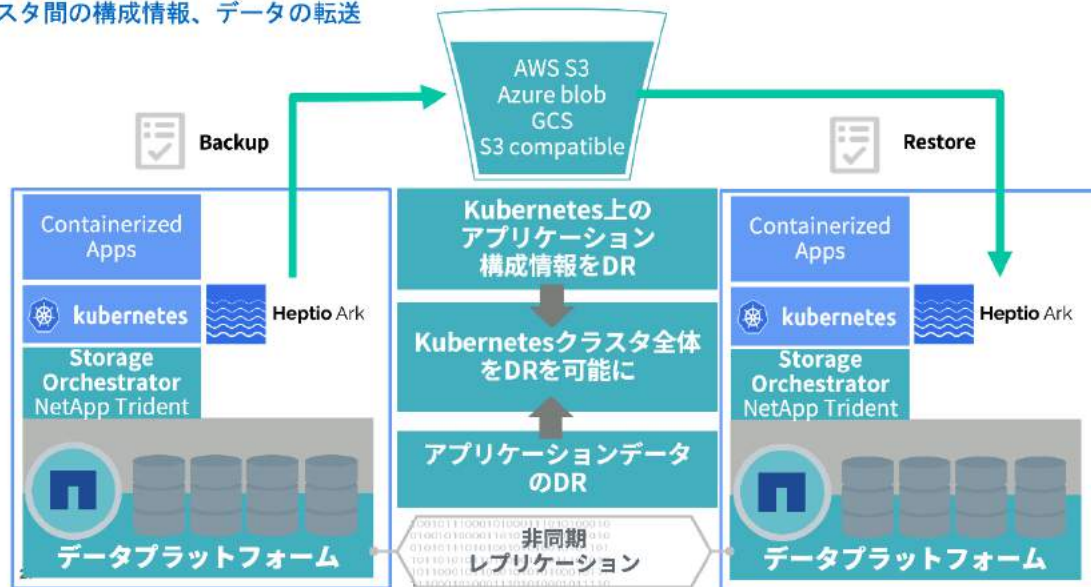
Heptio ark を使うことでクラスター全体や Label で指定した Kubernetes オブジェクトをバックアップすることができます。

取得したバックアップを使用することで以下のようなことが実現可能です。

- Kubernetes クラスターのバックアップ、DR
- 本番環境のクラスターから開発環境へ構成情報をリストア
- クラスターの引っ越し

# Kubernetes + NetApp + Heptio Ark

クラスタ間の構成情報、データの転送



稼働を確認した時点ではボリュームのレプリケーションはサポートされていないため、別の手段でバックアップ・リストアをする必要があります。Heptio ark 0.9 以降では restic インテグレーションがサポートされ、アプリケーションデータをオブジェクトストレージへ取得できるようになりました。ただし、サポートされているのは local persistent volume となっています。

- local persistent volume:  
<https://kubernetes.io/docs/concepts/storage/volumes/#local>

## 4.3.2. アプリケーションデータのレプリケーション

アプリケーションデータのバックアップ・DR はストレージレイヤーのレプリケーション機能で実現します。

NetApp ではデータレプリケーションの方法としては以下の3つの方法があります。

1. SnapMirror: ボリューム単位のデータレプリケーション、ブロック差分転送が可能
2. SVMDR: 仮想ストレージ単位のデータレプリケーション、ブロック差分転送が可能、かつストレージ情報(アカウントやデータ通信用の IP など)を維持した状態で転送
3. CloudSync: SaaS のデータ同期ツール、ファイル単位の差分転送が可能

今回は SVMDR を使用して実現しました。

## SVMDR を選択した理由

Heptio Ark を使用して取得するバックアップにはバックエンドストレージの IP や構成情報が含まれている状態でバックアップされます。そこから復元を行うと、ストレージレイヤーの構成情報(データ通信用の IP 等)が変更になると正常に稼働しなくなるため、仮想ストレージ単位ですべてを転送する仕組みを提供できる SVMDR を選択しました。

### 4.3.3. 実際の構成

#### Heptio Ark のインストール

インストールについては開発が活発なこともあり、公式のドキュメントを参考とすることをおすすめします。

- <https://heptio.github.io/ark/v0.9.0/quickstart>

サポートされるバックアップ先のストレージは以下のページに記載があります。

- <https://github.com/heptio/ark/blob/master/docs/support-matrix.md>

Heptio ark インストール時にバックアップの取得先のオブジェクトストレージを定義します。

この操作は両方のクラスターで実施する必要があります。

#### Heptio Ark バックアップスケジュール設定

ark ではスケジュール設定をすることで定期的に構成情報のバックアップを取得可能です。

スケジュールは `--schedule=` で設定でき、書式は crontab のものが使用可能です。

```
ark create schedule k8sbackup --schedule="*/5 * * * *" --exclude-namespaces kube-system
```

namespace の指定や、ラベルセクターを指定することも可能です。

#### ネットアップストレージの設定

ここではある程度 ONTAP を知っている前提で記載しています。SVMDR を構成するため、それぞれのクラスターで使用するストレージ同士でレプリケーションの設定を行います。実施する作業としては以下の流れです。2つのクラスターを行き来しながら操作します。

- Intercluster LIF 作成
- cluster peer 作成
- SVM Peer 作成

- SVMDR のリレーション作成
- SVMDR のデータシンク

以上でストレージレイヤーの事前設定が完了です。

#### 4.3.4. クラスタフェイルオーバーを実施

ここまでの状態で以下 2 点の定期的なバックアップが取得できている状態です。

- Heptio ark による Kubernetes の構成情報
- ストレージレイヤーにおけるアプリケーションデータの定期的なバックアップ・遠隔地への転送

ここからはストレージが停止したことを想定し、どのような操作が必要かを説明します。

DR 側のストレージで以下のコマンドを実行し、DR をプライマリとして扱うようにします。

SVMDR のリレーションを“ブレイク”します。

```
snapmirror break -destination-path [DR 用の SVM]:
```

仮想ストレージサーバを起動

```
vserver start -vserver [DR 用の SVM]
```

ここまでで DR 側ストレージの起動が完了しました。

続いて、Ark からバックアップデータをリストアして Kubernetes クラスタがプライマリと同じ状態になるかを確認します。

まずはバックアップデータの一覧を確認します。

```
$ ark backup get
```

NAME	STATUS	CREATED	EXPIRES	SELECTOR
test1	Completed	2018-07-06 16:27:30 +0900 JST	29d	<none>

リストアを実行します。

```
$ ark restort create restore1 --from-backup test1
```

```
Restore request "restore1" submitted successfully.
Run `ark restore describe restore1` for more details.
[root@dr-master ~]# ark get restore
```

NAME	BACKUP	STATUS	WARNINGS	ERRORS	CREATED
SELECTOR					
restore1	test1	InProgress	0	0	2018-07-06 16:39:40 +0900
JST	<none>				

実行が完了したら Kubernetes クラスターの確認を行います。

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

#### 4.3.5. 考慮点

- プライマリ、DR 側で同じネットワーク構成にしておくことで DR 時操作の簡易化が可能であるため DR 実施時にはネットワーク構成も考慮すること
- 切り替えのタイミングの自動化を検討
- 切り戻しの実施方法

## 4.4. Trident Fast PVC Cloning の使い方とユースケース

Trident の特徴的な機能の 1 つである、PVC Fast Cloning について紹介します。

### 4.4.1. PVC Fast Cloning とは

既存の PVC を高速にコピーする機能です。この機能を活用すると実現できることとしては以下の内容のものがあります。

- テストデータ設定済みの Database as a Service の実現
- テスト用途の環境を瞬時にコピー
- 本番環境からデータをコピーして、調査用の環境を作成し what-if 分析

また、ストレージ容量を消費しないというメリットもあります。

### 4.4.2. 実際の設定方法

ここでは実際に使用するマニフェストを提示し、設定すべき項目について説明いたします。

まずは mysql のパスワードを secret に入れます。

```
$ kubectl create secret generic mysql-pass --from-literal=password=yourpassword -n [ネームスペース]
```

mysql 自体のデプロイメントは以下のようなものをつかいます。ここではベースとなる PVC mysql-pv-claim を作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqldb
  labels:
    app: database
spec:
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      containers:
        - image: mysql:8
          name: mysqldb
```

```

    env:
      - name: PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-pass
            key: password
    ports:
      - containerPort: 3306
        name: mysqldatabase
    volumeMounts:
      - name: mysql-persistent-storage
        mountPath: /var/lib/mysql
    volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: database
  annotations:
    trident.netapp.io/reclaimPolicy: "Retain"
    trident.netapp.io/exportPolicy: "default"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: ontap-gold
---
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: database
spec:
  ports:
    - port: 3306
  selector:
    app: database
    tier: mysql
  type: NodePort

```

起動後にデータを投入し元となるデータベースを作成します。



クローン元の PVC には I/O が発生していないことを強く推奨します。

クローニングは PVC の annotation に `trident.netapp.io/cloneFromPVC` を設定し、コピーもとの PVC を指定することで実現できます。

以下の記述を追記します。

```
annotation:  
  trident.netapp.io/cloneFromPVC: mysql-pv-claim
```

上記の通りマニフェストを複数展開する方法もありますが、Helm を使うことでより簡単に展開することができます。以下の 2 つの項目を helm 実行時に設定します。

- `persistence.storageClass=ontap-gold`
- `persistence.annotations={trident.netapp.io/cloneFromPVC: XXX}`

```
$ helm install stable/mysql --name [リリース名] --namespace [ネームスペース] --set  
persistence.storageClass=ontap-gold --set  
persistence.annotations."trident\.netapp\.io/cloneFromPVC"=[コピー元の PVC 名]
```

マニフェストを随時書くのではなく、上記のように必要部分（今回はストレージクラス名と annotation）のみを定義し迅速に展開することができます。

今回の例はデータベースを例として説明しましたが、大量のデータ・セットを配るようなオペレーションをするときにも同様の手段で実施することができます。

## 5. 総括

コンテナプラットフォームの実現と運用上課題となる複雑性、データ永続化についてどのように解決できるかを本ホワイトペーパーで提示しました。技術的なスタックとしては Kubernetes, Rancher, Trident といったコンポーネントを使用しました。

具体的な構成としては、ストレージはエンタープライズストレージのネットアップストレージをコンテナプラットフォーム(kubernetes)へつなぐことができるストレージオーケストレーター Trident のデプロイから使い方までを説明し、デプロイされている Kubernetes を管理する役割として Rancher を使用しています。

Trident、そして Rancher により、コンテナへのボリュームのダイナミックプロビジョニングと障害時のレプリケーションによるバックアップ、データのクローニングまで確認することができました。

コンテナにおける複雑性、永続的なストレージの管理は運用をする上で重要な課題です。本ホワイトペーパーで提示した上記の組み合わせが、これら課題の解決策となり、日本でのコンテナ利用を推進するための一助となる事を願っています。

スタイルズ社・ネットアップ社は、本ホワイトペーパーを通じて、コンテナを利用したビジネスの発展を支援、ビジネスでのコンテナ利用をスムーズにしていく為の情報今後も提供していきたいと考えています。

ネットアップ合同会社/株式会社スタイルズ

## 6. 会社概要・問い合わせ先

ネットアップ合同会社

世界中の組織が、データを管理、保存するソリューションとして、ネットアップのソフトウェア、システム、サービスに信頼を寄せています。顧客は自社の現在、そして将来の成功のために、ネットアップのチームワーク、専門性、情熱を評価しています。ネットアップ合同会社は、米 NetApp, Inc.の日本法人です。製品、ソリューション、サービスの詳細に関しては、[www.netapp.com/jp](http://www.netapp.com/jp) をご覧ください。

E-mail : [ng-japan-openecosystem@netapp.com](mailto:ng-japan-openecosystem@netapp.com)

株式会社スタイルズ (Stylez Corp.)

<https://www.stylez.co.jp>

スタイルズは 2003 年の設立以来、企業が円滑な事業を行うのに必要な IT ソリューションを提供しているシステムインテグレーション企業です。最新技術を積極的に採用した IT サービスを展開しています。近年、サポート終了ソフトウェアや費用対効果が悪い WEB システムを、最適な環境下へ移植を行う「移行サービス」に注力し、企業の TCO 削減、デジタル化推進貢献を目指しています。

住所：〒101-0052 東京都千代田区神田小川町 1-2 風雲堂ビル 6 階

TEL : 03-5244-4111 (代表) / FAX : 03-5289-3141

E-mail : [web-contact@stylez.co.jp](mailto:web-contact@stylez.co.jp)