

# 1VMで作成するVanilla Kubernetes環境の構築とvSphere CSI Driverの導入 バックアップ環境構築- 手順書

## 目的

最小vSphere環境で、Kubernetesの環境構築、さらにバックアップ環境を作成する。

既設のvSphere環境に最小限の影響、最小限のリソース追加で実現することを目的とする  
また、loadbalancer環境を設置することで、ポートフォワードをすることなくアクセスができ、パブリッククラウドで使っていたデプロイメントの移植性を高める。

パブリッククラウドライクな構成を作成する（パブリッククラウドで利用できるほとんどの機能を実装）

いうまでもないが本番環境には向かない。少なくともKindやMinikubeのように、Docker in Dockerのような閉じた環境よりは便利

vSphere CSI driverを利用することによって、vSphere 6.7U3からサポートされたFCDというDiskオブジェクトの利用も体験する

ソリューションとして、vSphere CSI driverが利用できるのは、

VMware Tanzu/OpenShift on vSphereがあるが、それ以外にスクラッチでの作成するVanilla Kubernetesがある。今回は、Vanilla Kubernetesを利用する。

なお、Kubernetes自体の環境作成の解説（スクリプト自体のコード解説）は、この文章の対象外。

<https://github.com/masezou/k8s-study-vanilla>

---

## このスクリプトで構成されるもの

コンテナ環境

作成するKubernetesは最新(2021/11時点)より、1つ古いバージョン。また、Docker Shimではなく、Containerdを利用（Docker Shimが非推奨なため）

Kubernetes 1.21.06/containerd.io 1.4.11 (1ノード)

CNI: Metallb/Ingress

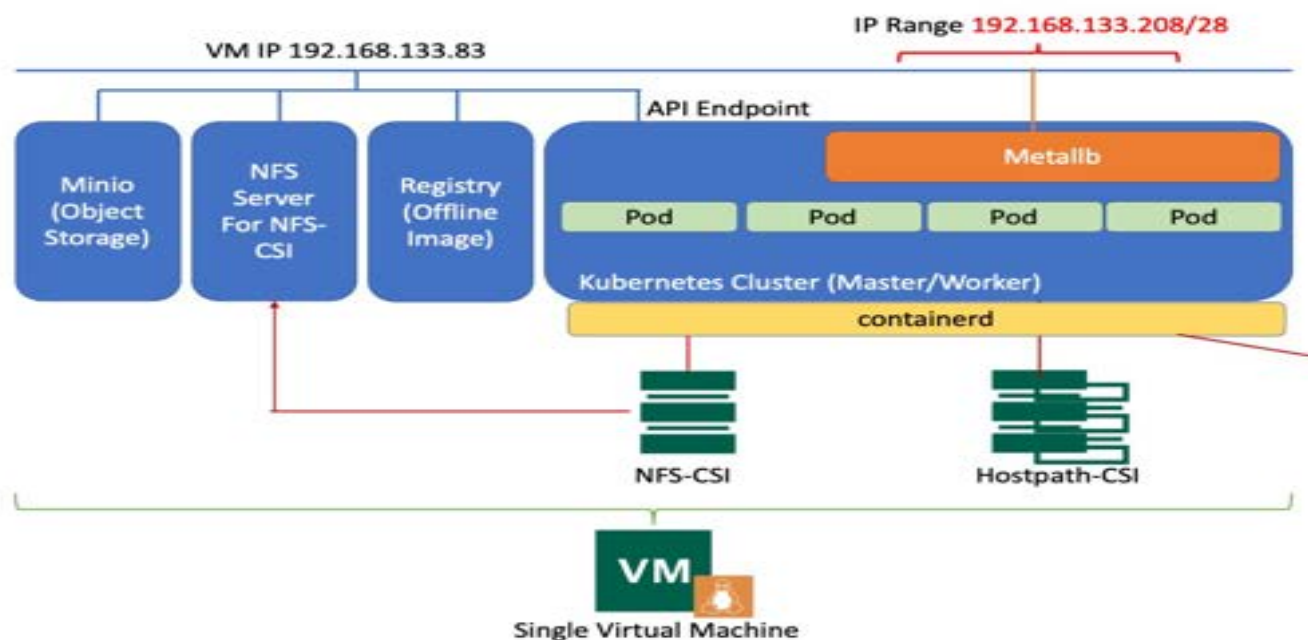
CSI: Hostpath-CSI/NFS-CSI/vSphere-CSI

付随環境

Minio オブジェクトストレージ

NFSサーバ (NFS-CSI用)

Docker Registryサーバ



ちなみにKasten DRをテストしたい場合は、オブジェクトストレージ用のUbuntu 20.04.3 VM (2vCPU/4GB 100GB HDD)を用意して、0-minio.shを実行する。

### 必要となる前提知識

vSphere (vCenter/ESX環境)でUbuntu VMがセットアップができて、Linuxの基本的な操作ができること

Kubernetesの知識があれば望ましいが、なくても構築ができる。

これからKubernetesを始めようとする方にもおすすめ

### 構築に関する所要時間

要件を満たすvSphere環境と1つのUbuntu VMがあれば、インターネットの速度とリソースのパワーに依存するが30分程度で終わる。最初にファイルにパラメータを入れる以外は基本的にスクリプトの実行とその待ちしかない。

## 必要なもの

- vSphere CSI Driverを使わない場合

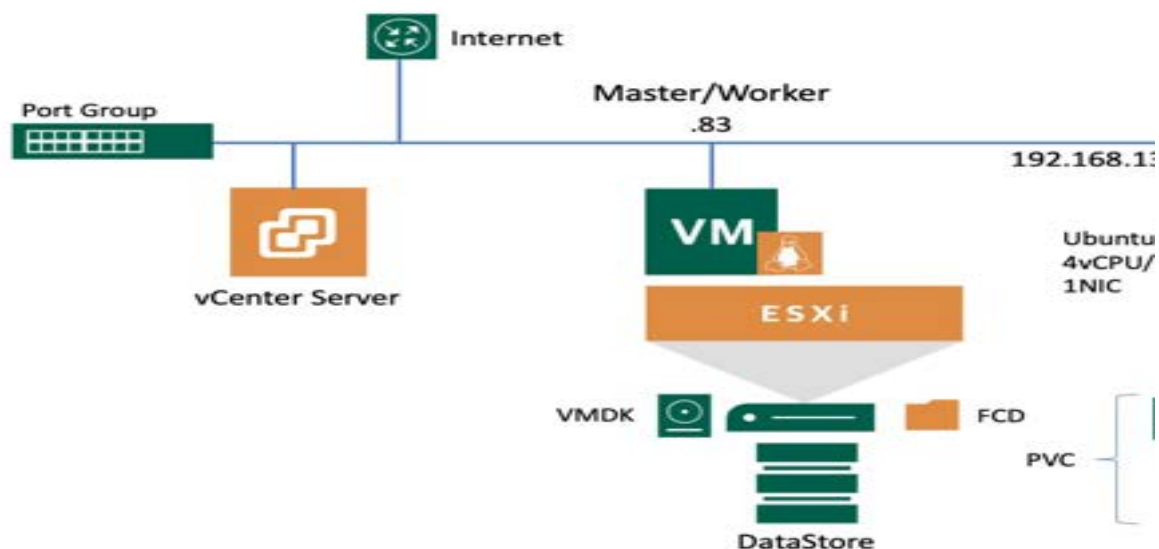
Ubuntu 20.04.3 (x86\_64) で仮想、物理サーバ1台（以後、このドキュメントのvSphereとVBRにまつわる記載は無視して問題なし。）

- vSphere CSI Driverを使う場合

vSphere 6.7U3以上のvCenter Server 1 台と最低1台のESX、データストア（なんでもいいが、vSANがあるとRWXで作成できる。今回は割愛）

- 仮想マシンスペック

最低でも4vCPU 16GB 100GB HDDが立ち上げられるリソース



vCenterの構成は、単純に以下の構成。vCenterにESXがぶら下がっているだけ。



---

- 必要なネットワーク

インターネット接続

以下同じセグメントでIPアドレスを用意

Ubuntu VM用のIP address 1 個

例 192.168.133.83/24

Load Balancer用のIP address 16個 (サブネット指定)

例 192.168.133.208/28

あるいは、連続したIPアドレス。

例 192.168.133.51-192.168.133.62

ー>簡単にいうと、IPアドレスが連続して20くらい。あるいは、単一のIPアドレス+いくつかの連続したIPアドレス。

vSphere側では、1つのポートグループしか使わないので、VLAN、DVSとかも要らない。ネットワーク要件を満たすならVM Networkだけあればいい。VBR連携でFCDリストアをする場合は、クラスタ（クラスタの構成は不問）の中にVMを配置する必要がある。

上位ルータもインターネットに出れるのであれば、特に設定不要。

(Edgerouter XやVyOSがあればLoadbalancerやDNS周りでのインテグレーションが可能になるが、今回は割愛)

---

## VMの構成

OS: Ubuntu Server 20.04.3のVMを1つ (Desktop版ではない。)

<https://ubuntu.com/download/server>

## VMの設定

4vCPU以上 (vCPUをケチるとアプリの展開ができなくなる。その場合は、csi-hostpathを利用していなければworkerノードを足す。)

8GBメモリ以上、16GB以上を推奨 (CPU同様。ケチるとアプリの展開ができなくなる。)

100GB HDD 1つ。(hostpath/vSphere CSI driverを使う場合)

Kubernetes環境 20GB、


残りは、MINIO、NFS-CSI領域、ローカルレポジトリ領域で利用。Ubuntu OSを作成したときに/diskに別のボリュームをマウントしておくと容量が拡張しやすく、再構築しても一部データが戻ってくる。

物理的な消費は以下が目安。そこそこCPUを消費する。メモリは、それほど消費していない。



VMの設定：DISKUUIDの設定を必ず有効にしておく。この設定がないと、vSphere CSI Driverの利用時にノードからFCDのボリュームをマウントできない。

### Configuration Parameters

 Modify or add configuration parameters as needed for experimental features or as instructed by technical support. Existing values will be removed (supported on ESXi 6.0 and later).

[ADD CONFIGURATION PARAMETERS](#)

Name	Value
answer.msg.uuid.altered	I copied it
disk.enableUUID	TRUE
ethernet0.pciSlotNumber	160
guestinfo.detailed.data	architecture="X86" bitne

AirGapインストールをする場合は、別途、パソコン端末が必要。  
詳しくは、Kastenのオフラインインストール (AirGap)方法を参照

---

## Ubuntuのインストール

100GB HDDをファイルシステムをxfsにして、LVMを構築してデフォルトインストール。（何も考えずにext4 LVMでも動く。）

IPアドレスは、1 ノードなのでDHCPでも構わないし、問題なく動作するが、Loadbalancerでサブネットを指定するので、VMのIPアドレスは固定にしておいたほうが無難

パッケージは、OpenSSH Serverだけをインストール。Dockerはインストールしてはいけない。

ユーザでログインをしたら、`sudo -i`で rootになる。パッケージをインストールするので、基本的には構築の間はずっとrootで作業を行う。

---

## コンテナ基盤の作成

### スクリプトパッケージの入手

```
git clone https://github.com/masezou/k8s-study-vanilla
cd k8s-study-vanilla
```

#### スクリプトの設定変更

いきなり、実行してはならない。ただし、設定を一度してしまえば、引数などもなく、完全自動構築される。

viで設定変更をする場合、`""`を消さないように注意すること。viの色付けを見れば正しく設定されているかわかるはず。

正しく修正できている場合の例

```
#!/usr/bin/env bash

#####
# Pre-requirement
# vCenter is 6.7U3 above
# Each VM need to have set DISKUUID
#####

#For vSphere CSI/Tanzu
VSPHEREUSERNAME="administrator@vsphere.local"
VSPHEREPASSWORD="PASSWORD"
VSPHERESERVER="vc.example.com"
VSPHERESERVERIP="192.168.1.10"
VSPHEREDATASTORE="Datastore"

VSPHERECSI=2.3.0

if [ ${EUID:-${UID}} != 0 ]; then
    echo "This script must be run as root"
    exit 1
else
    echo "I am root user."
fi

-- INSERT --
```

修正に問題がある場合の例 (PASSWORDのあとに"がない。)

```
#!/usr/bin/env bash

#####
# Pre-requirement
# vCenter is 6.7U3 above
# Each VM need to have set DISKUUID
#####

#For vSphere CSI/Tanzu
VSPHEREUSERNAME="administrator@vsphere.local"
VSPHEREPASSWORD="PASSWORD"
VSPHERESERVER="vc.example.com"
VSPHERESERVERIP="192.168.1.10"
VSPHEREDATASTORE="Datastore"

VSPHERECSI=2.3.0

if [ ${EUID:-${UID}} != 0 ]; then
    echo "This script must be run as root"
    exit 1
else
    echo "I am root user."
fi

-- INSERT --
```

## 設定変更内容

Loadbalancerに振られるIPアドレスレンジの指定

vSphereの設定(vSphere CSI Driverを使う場合のみ)

具体的には以下を設定する。

```
vi 3-configk8s.sh
```

ファイル先頭にある IPRANGE="**192.168.133.208/28**"を設定変更

このサブネットがLoadBalancerが降り出すIPレンジとなる。もちろん、このサーバど同じセグメントで、このサブネット範囲のIPアドレスを持っているホストがないことが前提。

サブネット計算 <http://jodies.de/ipcalc>

もし、サブネットレベルでIPアドレスが振り出せない場合は、以下のような範囲指定も可能。

IPRANGE="**192.168.133.51-192.168.133.62**"

vSphere CSIドライバを使いたい場合は以下の2つも設定する



```
vi 5-csi-vsphere.sh
```

以下にvCenter/ESXの情報を入れる。指定したDataStoreにvSphere CSI Driverで作成されるFCDが作成される。複数のデータストアに入れたい場合は、後でvSphereクライアントからデータストアにタグを追加すればいい。

#For vSphere CSI/Tanzu

VSPHEREUSERNAME="**administrator@vsphere.local**"

VSPHEREPASSWORD="**PASSWORD**"

VSPHERESERVER="**YOUR\_VCENTER\_FQDN**"

VSPHERESERVERIP="**YOUR\_VCENTE\_IP**"

VSPHEREDATASTORE="**YOUR\_DATASTORE**"

```
vi K3-kasten-vsphere.sh
```

以下にvCenter/ESXの情報を入れる

VSPHEREUSERNAME="**administrator@vsphere.local**"

VSPHEREPASSWORD="**PASSWORD**"

VSPHERESERVER="**YOUR\_VCENTER\_FQDN**"

---

## スクリプトの実行（その1：Kubernetes環境を作成）

これでCNIとCSIが構成されたKubernetesの環境が作成される。(vSphereドライバー以外の全て)

以下でminioとサーバ側に必要なツールがインストールされる。以下の2つを実行したところでシャットダウンして、仮想マシンスナップショットを取っても構わない。

```
./0-minio.sh ; ./1-tools.sh
```

以下でKubernetes環境を作成する。

```
./2-buildk8s-lnx.sh ; ./3-configk8s.sh ; ./4-csi-storage.sh
```

画面と睨めっこしながら、コーヒーでもすすめることをおすすめする。

スクリプトが終わってプロンプトが帰ってきたら、一度ログアウトして、再度ログインをする。

---

## 出来上がったかどうかの確認

Note: MACのGoogle Chromeからアクセスする場合、証明書エラーが表示されるとその先へ進めない。その場合、"**This is unsafe**"とタイプ（空打ち）すると先に進める。

## minioの確認

```
mc admin info local
```

- 192.168.133.83:9000  
Uptime: 9 minutes  
Version: 2021-11-03T03:36:36Z  
Network: 1/1 OK  
Drives: 4/4 OK

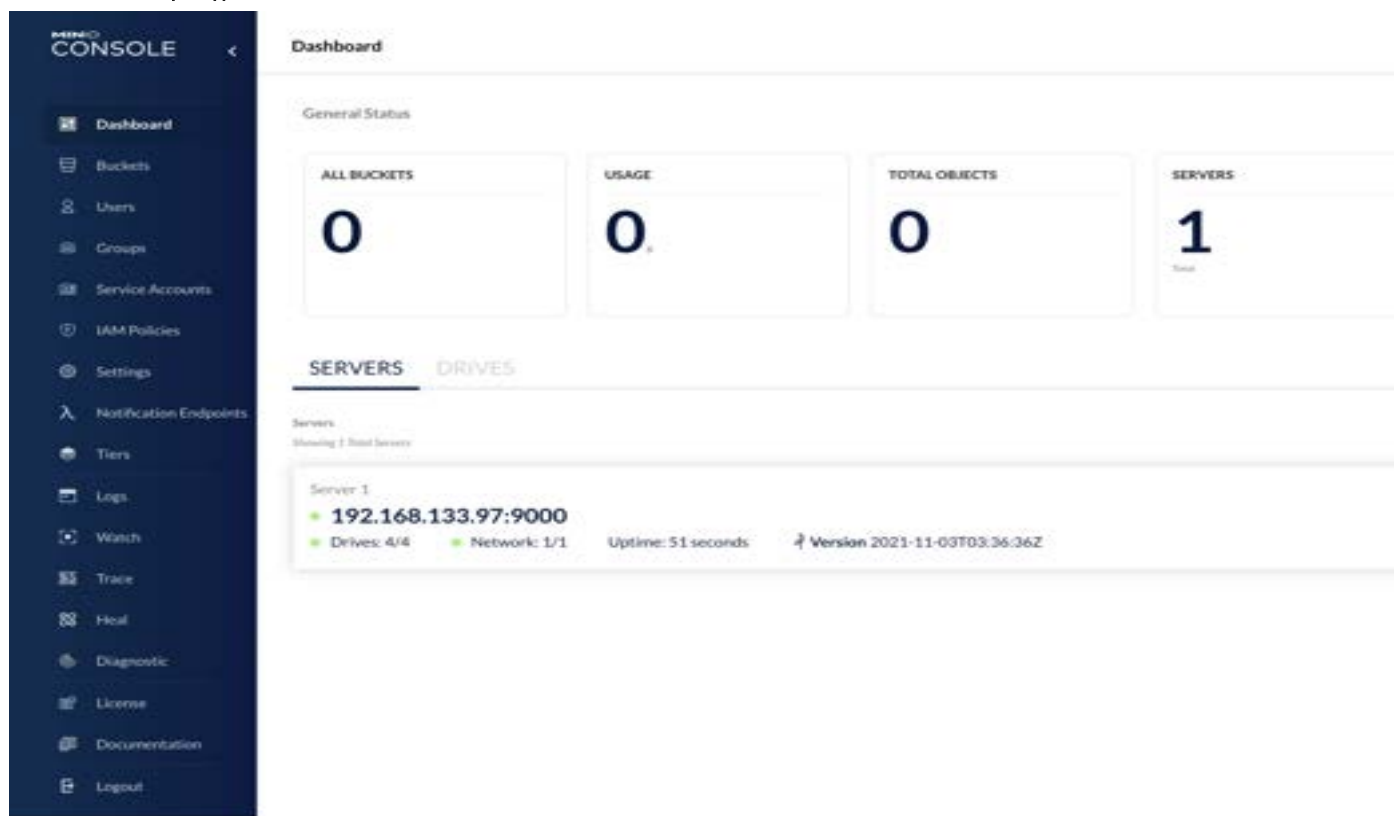
4 drives online, 0 drives offline

上記で4Driveと出ているのは、Immutable設定が利用できるようにしたため。実際は単なる4つのディレクトリである。

minioのコンソール: <https://<このVMのIP>:9001>

Username/Passwordともに **minioadminuser**

APIは、<https://<このVMのIP>:9000>で、クレデンシシャルは同じ



## NFSの確認

```
showmount -e
```

Export list for k8s-demo1:

```
/disk/k8s_share 127.0.0.1/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
```

/disk/k8s\_share がエクスポートされていること

## Local Registryの確認

レジストリが正しく動作しているかの確認

```
curl -X GET http://<このVMのIP>:5000/v2/_catalog
```

```
{"repositories":["bitnami-shell","mongodb","mysql","postgresql"]}
```

## kubernetesノードの確認

1 台だけノードがあり、よくみるとmaster/worker兼務になっている。

```
kubectl get node -o wide
```

NAME	STATUS	ROLES	AGE	VERSION
INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	
CONTAINER-RUNTIME				
k8s-demo1	Ready	control-plane, <b>master,worker</b>	99m	v1.21.6
192.168.133.83	<none>	Ubuntu 20.04.3 LTS	5.4.0-90-generic	
containerd://1.4.11				

```
kubectl get pod -A
```

量が多いので省略。csi-nfs-controllerだけPendingになっているが、後は全てRunningあるいはCompletedになっているはず。

## Kubernetesのロードバランサーの確認

```
kubectl get svc -A
```

NAMESPACE	NAME	TYPE	
CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	hostpath-service	NodePort	
10.102.140.234	<none>	10000:31501/TCP	13m
default	kubernetes	ClusterIP	
10.96.0.1	<none>	443/TCP	13m
ingress-nginx	ingress-nginx-controller		
LoadBalancer	10.97.26.75	<b>192.168.133.209</b>	
80:32047/TCP,443:32106/TCP		13m	
ingress-nginx	ingress-nginx-controller-admission	ClusterIP	
10.106.102.47	<none>	443/TCP	13m

kube-system	kube-dns	ClusterIP
10.96.0.10	<none>	53/UDP,53/TCP,9153/TCP 13m
kube-system	metrics-server	ClusterIP
10.105.98.161	<none>	443/TCP 13m
kubernetes-dashboard	dashboard-metrics-scraper	ClusterIP
10.109.199.14	<none>	8000/TCP 13m
kubernetes-dashboard	dashboard-service-lb	
LoadBalancer	10.109.37.4	192.168.133.208 443:30085/TCP 13m
kubernetes-dashboard	kubernetes-dashboard	ClusterIP
10.109.235.233	<none>	443/TCP 13m

Kubernetes DashboardにIPRANGEで指定した範囲内のIPアドレスが振られているはず。

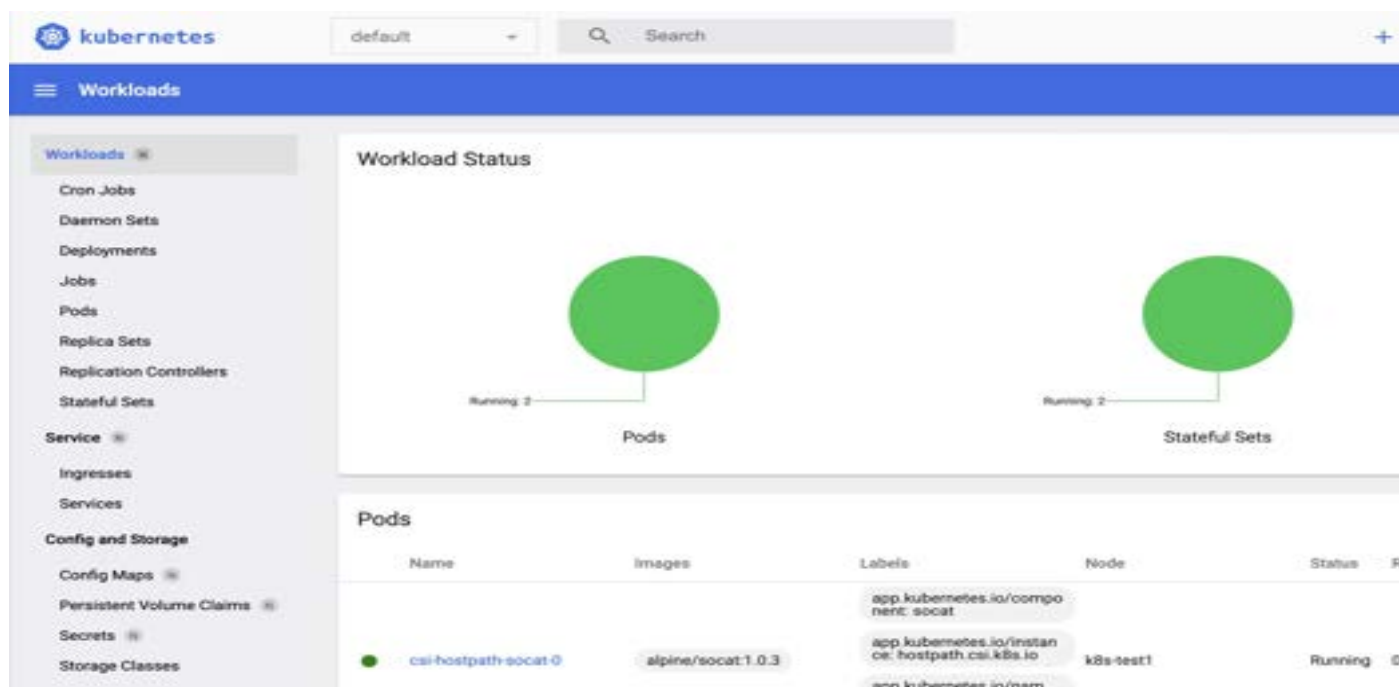
ダッシュボードへのアクセス

ダッシュボードへアクセスする必要がなくてもアクセス確認をしておくことをお勧めしておく。

<https://<このVMのIP>/>

ログインに必要なtokenは、dashboard.tokenに記載されている

```
cat dashboard.token
```



ダッシュボードを使うと、kubectlのコマンドを知らなくてもKubernetesが利用できる。  
今回は、単にLoadbalancerの動作確認目的でいっただけなので利用はしない。

## ストレージクラスの確認

今の時点で、以下の3つのドライバが表示されている

```
kubectl get sc
```

NAME	PROVISIONER	RECLAIMPOLICY
VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
csi-hostpath-sc (default)	hostpath.csi.k8s.io	Delete
Immediate	true	57s
local-path	rancher.io/local-path	Delete
WaitForFirstConsumer	false	43s
nfs-csi	nfs.csi.k8s.io	Delete
Immediate	false	44s

- csi-hostpath-sc

/var/lib/docker/volumes/にPVCが作成される。

1ノードでしか利用できない、ローカルストレージドライバ。ただし、スナップショットが使える。

一見便利そうだが、bitnamiとかのhelmでインストールをする場合は、  
volumePermissions.enabled=trueが必要になる。パーミッションの問題に遭遇しやすい。

- local-path

/opt/local-path-provisioner/にPVCが作成される。

csi-hostpath-scと似ているが**CSIドライバではない**。ダイナミックプロビジョニング  
をしてくれる。(KINDに標準でついていたもの)

<<オマケでつけてみた。>>

- nfs-csi

/disk/k8s\_share/にPVCが作成される。

NFSでPVCを作ってくれる。このドライバは、ReadWriteMany(RWX)でPVCを作れる  
ので、AutoScaleなどのテストでも使える。スナップショットは未サポート。

- ~~longhorn (Longhornを選択した場合)~~

~~— /var/lib/longhorn/replicas/にPVCが作成される。~~

~~——csi-hostpathと違って、iSCSIでアクセスするためにマルチノードでアクセスでき、パ  
ーミッションの問題はない。また、スナップショットが使える~~

~~——ただし、csi-hostpathより、かなりCPUなどのリソースを使うので、8vCPU以上の  
シングルノードか、workerを追加することを推奨~~

- rook-ceph-block/rook-cephfs (experimentalのスクリプトを実行した場合)  
LVM領域にPVCが作成される。

RWOで動作するブロックとRWXでも動作するファイルシステムの2つがある。スナ  
ップショットが使える

experimentalのスクリプトでは、csi-hostpathの変わりにcephが入る。また、シング  
ルノード用に記載しているので、このスクリプトではマルチノードでは動かないほう  
がいい。

CPUパワーをかなり酷使するので、注意。

ちなみに、csi-hostpath-sc (default) となっているのは、PVCを作る時にStorage  
Classを指定しないと使われるという意味でデフォルト。

## PVCの動作確認

実際にPVCが作成されて、podでマウントできることを確認することをおすすめする。

以下はPVCだけを作る例

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: [Storage Class名]
  accessModes:
    - ${AM}
  resources:
    requests:
      storage: 1Gi
EOF
kubectl get pvc
```

vSphere上で構築していない、あるいはvSphere環境に触りたくない、触れない場合は、  
ここで基盤作成完了。

<ホスト名>-cl-<ホスト名>\_kubeconfigというファイルを手元のPCに持ってくれば、手  
元の端末でkubectl が扱える。

## スクリプトの実行（その2：vSphere CSI Driverのインストール）

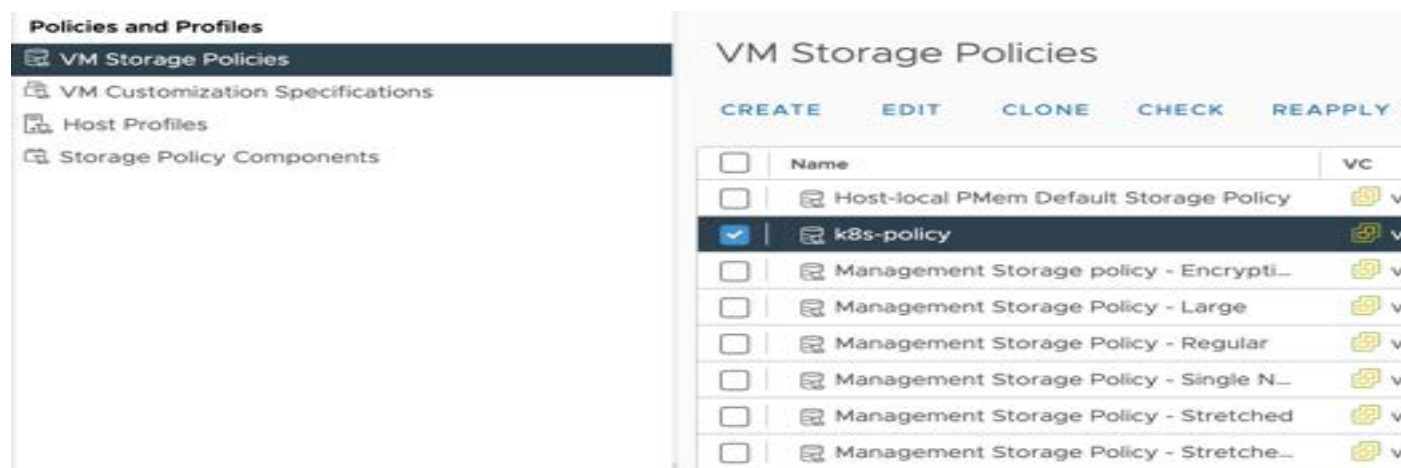
スクリプト内でvSphereの設定が正しくなされているかを必ず確認のこと。

```
./5-csi-vsphere.sh
```

vSphere CSI Driverのインストール確認

vCenterでのストレージポリシーの確認

vCenterで以下の確認をする。（vCenterでのポリシーは、スクリプトが自動で作成する。正常であれば、ポリシーはあるはず。）



ポリシーk8s-policyができていて、対象のデータストアが指定できていることを確認する。

### Edit VM Storage Policy

- 1 Name and description
- 2 Policy structure**
- 3 Tag based placement
- 4 Storage compatibility
- 5 Review and finish

#### Policy structure

##### Host based services

Create rules for data services provided by hosts. Available data services could include encryption. Host based services will be applied in addition to any datastore specific rules.

☐ Enable host based rules

##### Datastore specific rules

Create rules for a specific storage type to configure data services provided by the datastores. VMs are placed on the specific storage type.

☐ Enable rules for "vSAN" storage

☐ Enable rules for "vSANDirect" storage

☒ Enable tag based placement rules

### Edit VM Storage Policy

- 1 Name and description
- 2 Policy structure
- 3 Tag based placement**
- 4 Storage compatibility
- 5 Review and finish

#### Tag based placement

Add tag rules to filter datastores to be used for placement of VMs.

##### Rule 1

Tag category

Usage option

Tags  [BROWSE TAGS](#)

[ADD TAG RULE](#)

以下でデータストアが正しく表示されることを確認する。

### Edit VM Storage Policy

- 1 Name and description
- 2 Policy structure
- 3 Tag based placement
- 4 Storage compatibility**
- 5 Review and finish

#### Storage compatibility

**COMPATIBLE** **INCOMPATIBLE**

☐ Expand datastore clusters

Compatible

Name	Datacenter	Type	Free Space	Capacity
qnap2_iscsi	Datacenter	VMFS 6	1.23 TB	1.46 TB

ノードでの確認

vsphere-scで指定されているpolicyとvCenterで指定しているポリシー名が合致していることを確認する。

```
kubectl describe sc vsphere-sc
```

Name: vsphere-sc

IsDefaultClass: Yes

Annotations: kubectl.kubernetes.io/last-applied-configuration=  
{"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":  
{"annotations":{"storageclass.kubernetes.io/is-default-  
class":"true"},"name":"vsphere-sc"},"parameters":



```

{"fstype":"ext4","storagepolicyname":"k8s-
policy"},"provisioner":"csi.vsphere.vmware.com"}
,storageclass.kubernetes.io/is-default-class=true
Provisioner:          csi.vsphere.vmware.com
Parameters:          fstype=ext4,storagepolicyname=k8s-policy <-これが
vSphereのstorage Policyと同じ名前になるはず。
AllowVolumeExpansion: <unset>
MountOptions:        <none>
ReclaimPolicy:       Delete
VolumeBindingMode:   Immediate
Events:              <none>

```

```
ls -l /dev/disk/by-id/wwn-*
```

ここに少なくともsdaが表示されている必要がある。表示されていない場合は、DISKUUIDの設定の確認を。多分設定を忘れている。

これでvSphere CSIドライバの導入が完了した。今サポートしているStorageClassは、4つ。

```
kubectl get sc
```

NAME	PROVISIONER	RECLAIMPOLICY
VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
csi-hostpath-sc	hostpath.csi.k8s.io	Delete
Immediate	true	4m26s
local-path	rancher.io/local-path	Delete
WaitForFirstConsumer	false	4m12s
nfs-csi	nfs.csi.k8s.io	Delete
Immediate	false	4m13s
vsphere-sc (default)	csi.vsphere.vmware.com	Delete
Immediate	false	5s

となる。

デフォルトはvSphereに変更になっているので注意。StorageClassを指定しないとvSphere Driver でPVCが作成される。

- vsphere-sc

データストアのfcdディレクトリにpvcの実態をvmdkでつくる。フォーマットはExt4。また、kubernetesでのスナップショットは利用できない代わりにVMDKのスナッ

プショットが作成される。但しVMDKスナップショットを含むPVCとスナップショットは削除できないので手動で削除する必要がある。

また、vSAN DatastoreでないとRWXのPVCは作成できない。（このケースは、シングルノードなので影響はないが。）

ちなみにvsphere-scでPVCをつくると

```
kubectl get pvc -A
```

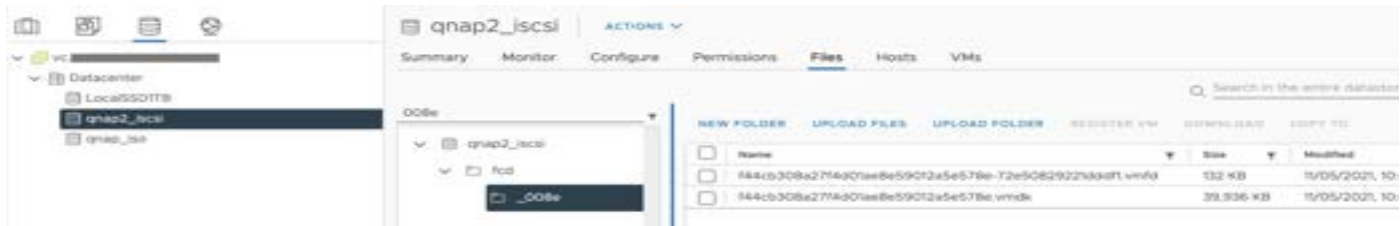
NAMESPACE	NAME	STATUS	VOLUME	
	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
test3	task-pv-claim	Bound	pvc-562b073c-4d5f-45b0-b076-a2a2faed7887	1Gi
		RWO	vsphere-sc	28s

<参考>PowerCLIでは、対応するvDISKが表示される。以下PowerShellのPowerCLIでの実行結果

```
get-vdisk | Format-Table -AutoSize -Wrap
```

Name	Disk Type	CapacityGB
	Filename	
----	-----	-----
	-----	
pvc-562b073c-4d5f-45b0-b076-a2a2faed7887	Flat	1.000
[qnap2_iscsi] fcd/_008e/f44cb308a27f4d01ae8e59012a5e578e.vmdk		

vCenterでみると  
FCDが作成されている



VMware Paravirtual SCSIが追加され、その下にPVCがマウントされている。

## Edit Settings

k8s-demo1



> Memory	16	GB
<b>Hard disks</b> 10 total   339.76 GB		
> Hard disk 1	100 GB   SCSI(0:0)	
> Hard disk 2	100 GB   SCSI(0:1)	
> Hard disk 3	8 GB   SCSI(1:0)	
> Hard disk 4	5 GB   SCSI(1:1)	
> Hard disk 5	5 GB   SCSI(1:2)	
> Hard disk 6	37.25 GB   SCSI(1:3)	
> Hard disk 7	8 GB   SCSI(1:4)	
> Hard disk 8	37.25 GB   SCSI(1:5)	
> Hard disk 9	37.25 GB   SCSI(1:6)	
> Hard disk 10	2 GB   SCSI(1:8)	
> SCSI controller 0	LSI Logic Parallel	
> SCSI controller 1	VMware Paravirtual	
> Network adapter 1	Ent6 Network	<input checked="" type="checkbox"/> Connected
> CD/DVD drive 1	Client Device	<input type="checkbox"/> Connected
> Video card	Specify custom settings	
> Security Devices	Not Configured	
VMCI device		

CANCEL

OK

Edit Settingsk8s-demo1

> Memory16GB

Hard disks10 total | 339.76 GB

> Hard disk 1100 GB | SCSI(0:0)

> Hard disk 2100 GB | SCSI(0:1)

> Hard disk 38GB

Maximum Size1.23 TB

VM storage policyk8s-policy

TypeThin Provision

SharingNo sharing

Disk File[qnap2\_iscsi] fcd/\_008e/df83b067081c4522a03b05681c7dcba4-000018.vmdk

SharesNormal1000

Limit - IOPsUnlimited

Disk ModeDependent

Virtual Device NodeSCSI controller 1SCSI(1:0) Hard disk 3

> Hard disk 45 GB | SCSI(1:1)

> Hard disk 55 GB | SCSI(1:2)

> Hard disk 637.25 GB | SCSI(1:3)

> Hard disk 78 GB | SCSI(1:4)

> Hard disk 837.25 GB | SCSI(1:5)

CANCEL

OK

これでvSphere CSIドライバ込みの環境構築が完了

## experimentalの解説

あくまでも実験的なスクリプト

## buildk8s-worker.sh

同じUbuntu VMを作成して、Workerノードを追加ができる。Workerノード用のUbuntu VMを作成したら、このスクリプト「だけ」実行した後に、クラスタに追加する。クラスタへの追加方法は割愛

## external-dns.sh

DNSを作成して、Loadbalancerに振られるIPアドレスに対してホスト名エントリを作成することができる。

スクリプトにドメイン名を設定する必要があります。

DNSのリゾルバをこのUbuntu VMにすることより、ホスト名アクセスができるようになる。

ちなみに隠しパラメータとして、OpenShiftで必要となるDNSの設定も記載されています。

#### 4-csi-storage-ceph.sh

csi-hostpathドライバではなく、Cephを使うことができる。

注意点

CPUパワーをものすごくラッシュするので、少なくとも8vCPUくらいが必要。また、物理サーバの構成も第10世代くらいのIntel CPUが必要（

シングルノード構成にしているのでマルチノード構成には向かない。

---

## バックアップ基盤の作成

### Kastenのインストール（オンライン）

IPアドレスが変わってしまうことを防ぐために、デモアプリを作成する前に、Kastenを入れてしまうことをおすすめする。

Kastenのインストールは、ドキュメントを参照しながらインストールするべきだが、このスクリプトでは自動でインストールができる。

Kastenは、PVCを利用する。利用されるPVCは指定もできるが、以下のコマンドを実行して、defaultとついているStorage Classにできる。

```
kubectl get sc
```

NAME	PROVISIONER	RECLAIMPOLICY
VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION	AGE
csi-hostpath-sc	hostpath.csi.k8s.io	Delete
Immediate	true	4m26s
local-path	rancher.io/local-path	Delete
WaitForFirstConsumer	false	4m12s
nfs-csi	nfs.csi.k8s.io	Delete
Immediate	false	4m13s
vsphere-sc ( <b>default</b> )	csi.vsphere.vmware.com	Delete
Immediate	false	5s

スクリプトは、Storage Classに応じたインストールをするようになっているので、今回は気にする必要はないが、手動でKastenをインストールする場合でcsi-hostpath-scを利用するには、追加のインストール設定をしないとグラフ表示がでない。

### Kastenのインストールの実行

```
./K0-kasten-tools.sh ; ./K1-kasten.sh
```

K0-kasten-tools.shで、k10コマンドとkubestrが自動でインストールされる

K1-kasten.shで、Kastenで必要となるスナップショットの注釈をした上で、Kastenをインストールをし、NFS-CSIがあればバックアップ用のストレージとしてNFS PVCを作成してくれる

CSI Provisioner doesn't have VolumeSnapshotClass - Error

と表示されるが、NFS-CSIはスナップショットが使えないので無視していい。

kasten-ioのPodが全てRunningになったら、

```
kubectl -n kasten-io get svc | grep gateway-ext
```

gateway-ext	LoadBalancer	10.99.97.237	192.168.133.210
80:32167/TCP		3m25s	

ここで表示されたIPアドレスを開く

**http://<EXTERNAL-IP>/k10/**

ログイン画面が開くので

k10-k10.tokenに記載されているトークンを入力する。

```
cat k10-k10.token
```

会社名とメールアドレスを登録するとログインができる。

バックアップストレージとvSphereをKastenのダッシュボードで設定してもいいのだが、以下のスクリプトで自動設定ができる。

バックアップストレージとしてminioとNFSの保存先が自動設定できる。

```
K2-kasten-storage.sh
```

vSphere CSIドライバを使う場合は、同様に、以下でvCenterの設定が自動設定できる。

```
K3-kasten-vsphere.sh
```

Blueprintは以下で、最新を一括でインストールしてくれる。

```
K4-kasten-blueprint.sh
```

RBACを設定したい場合のサンプル

```
K5-kasten-local-rbac.sh
```

## マルチクラスタ構成

一般的な環境だと、クラスタは、新旧環境だったり、開発、本番だったり複数のクラスタを持っている場合が多い。マルチクラスタ環境にすると、複数の環境で利用できる。

マルチクラスタにすると

- 1つの画面で複数のクラスタに対して集中管理ができる
- グローバルなポリシーを各クラスタに配布できるようになる。

マルチクラスタの構成

1つ目のクラスタ(Kastenの導入済み)をプライマリにして、残りのクラスタ(Kastenの導入済み)をセカンダリとして、登録する。

個々のクラスタへのKastenの導入は、スタンドアローン、つまり単体インストールをしておくだけで構わない。

このデプロイメントはKastenのURLが露出されているので、同じVMをもう一つ立てれば、Multiclusterを構築できる。

(IPRANGEの設定変更は忘れずに。)

また、クラスタが1台しかなくても利用は可能。(画面表示のみだが)

プライマリクラスタでの設定

まず、コンテキストを調べる。

```
kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO
	NAMESPACE		
*	kubernetes-admin@k8s-demo1-cl	k8s-demo1-cl	kubernetes-admin

以下のコマンドを実行

```
k10multicluster setup-primary \  
  --context= kubernetes-admin@k8s-demo1-cl \  
  --name=k8s-demo1-cl
```

Bootstrapping Primary Cluster...

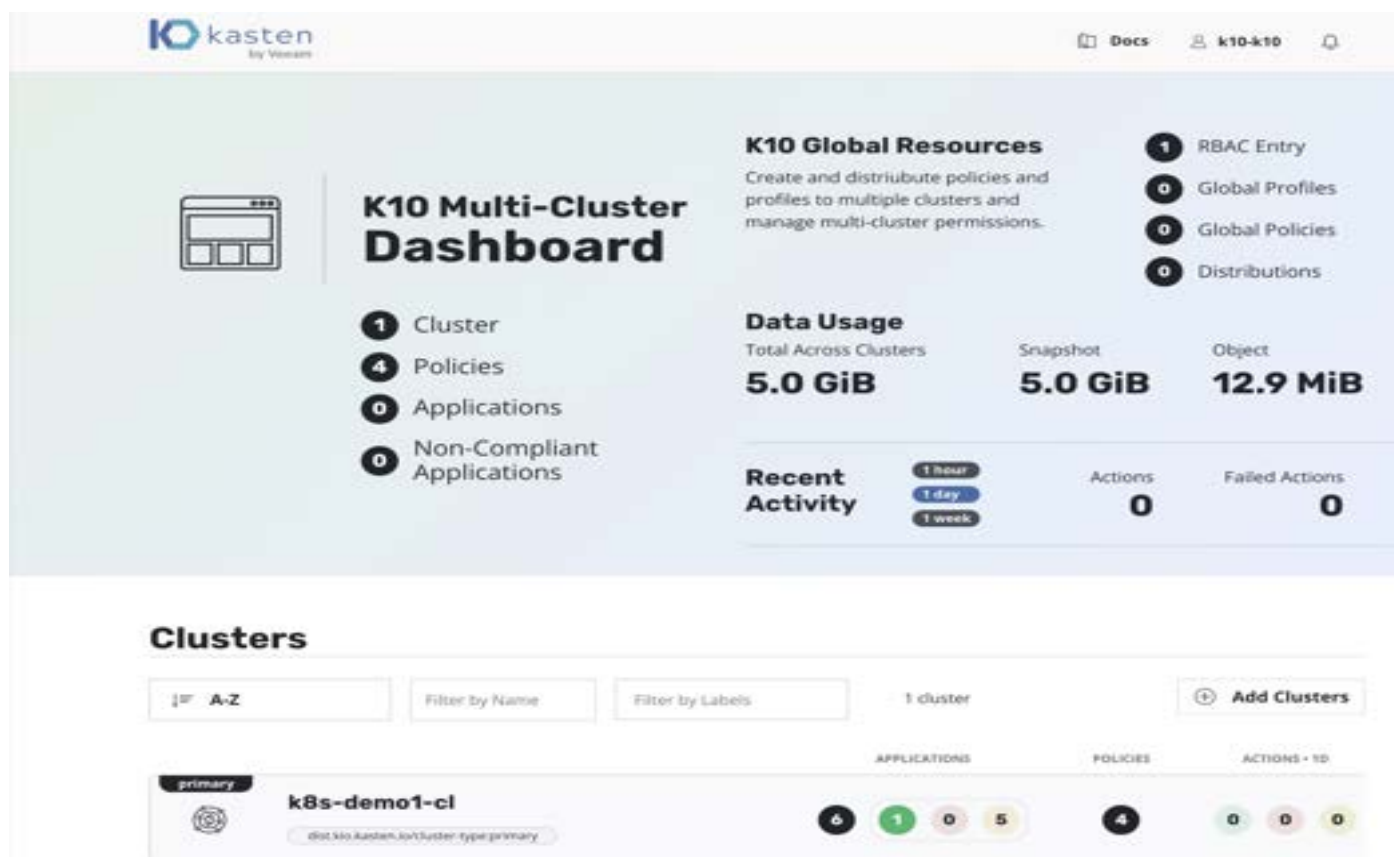
Getting Primary Cluster Config...

Verifying cluster parameters: k8s-demo1-cl

Setting up primary multicluster configuration: k8s-demo1-cl

Setting up Primary Cluster Complete!

ダッシュボードをインストールするとクラスタの画面が表示できる。



セカンダリの登録は以下の要件を満たすこと

- 登録元、先は、オンプレのKubernetesでもパブリッククラウドのKubernetesでも構わない
- プライマリのサーバからダッシュボードに対してIPで到達できること (Kind環境はNG)
- Ingress/Loadbalancerでhttpsあるいはhttpでアクセスができること (ポートフォワードの環境はNG)
- トークン認証がされていること (認証なし、ベーシック認証はNG)

クラスタ登録に必要な情報

~/.kube/configファイルの中身

2台目のKatenダッシュボードのURL (例 http://192.168.20.13/K10/)

Kubeconfigを貼り付けるかロードをして、登録をする。

セカンダリ側で

```
cat ~/.kube/config
```

これをコピーする。



コピーすると自動的に設定が読み込まれるので、クラスタを指定して、URLを登録、SSLのVerifyを無効にする。クラスター名は、kubernetes-admin@の部分を削除する。

## Add Clusters



```
kuhH21EbmV5a1J5d1MrdV130V2ZQnpRcV045VV1bXE3VHLSHGswVE9oCnQ2NMN3QnnpYX06S0RCeEUH50NoQ3dLQmdGVkhZ
dnW0WZNY2RKY0t3NmN1T1FQSnMyWEFK0VawWtHZGhyQkUKQkdaVWV0sJdUeUJ2MHNVR2s2MUR5VmN0eTdZD0VKU2x5YU1j
Q8p6REZR55tsd0cRn2t0c183cEFuVnVWVGXbgoVUs0MwRqQUVQbJdMcTF0VUpMaHZXR24z0LV45hZsQWEyZj3K2UfVME1t
eUNHbUNDQnVxR1EwZzF5NG94OU02C1dYdjd8b0dBSent2Hfo2dW1GHjRmaHB15nZ1SUJ8anVMNEpOREVyenMrR1Z3RWV0d2tP
TDJbeHFIr3ZhcWfYT2sKT01Xd2s0cE03d0xpbkt001YTmdoNS12QkIva3F2aUlpZKzHb05udktXUmf55m5GcFBLcGZyW0d3
ZXF2NnsxVAprNHFPd0M2T0Rxc51N4S2RyUkFqNkp0ND1wQXdyGxaNhh0Ny9ZVwRTb0hNC9jHepK0Kc9C10tLS0tRUSE1FJT
```

Or select your kubeconfig file

Choose file...

Browse



Successful Import

Found 1 cluster contexts in this kubeconfig.

### Select Clusters

Select clusters you wish to add to this multi-cluster deployment.

Available Options (0)

Select All

Selected (1)

Deselect All

No Unselected Options



kubernetes-admin@k8s-de...



We'll need some additional information for each of the clusters you've selected.

**kubernetes-admin@k8s-demo2-cl**

#### Cluster Display Name

Name to be displayed in K10 Dashboard

k8s-demo2-cl

#### Ingress URL

URL for the K10 instance deployed in this cluster

Example <https://cluster1.example.com/k10/>

http://192.168.134.211/k10/

#### Other Settings

##### K10 Namespace

In most cases, the K10 namespace is `kasten-io`, but if it has been changed on this cluster, edit it here.

kasten-io

##### Helm Release Name

In most cases, this is `k10`, but if it has been changed on this cluster, edit it here.

k10

#### Insecure TLS

Disable TLS verification so that any certificate will be accepted. This should only be used for testing.

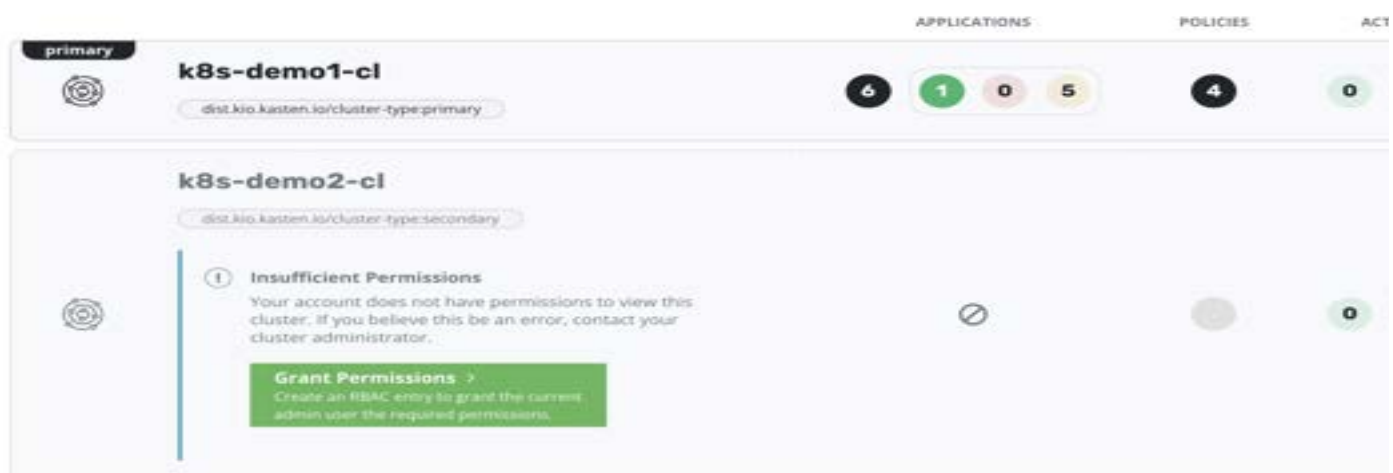
☒ TLS Verify Off

☐ TLS Verify On

Add Clusters

Cancel

Add Clusterをクリックすると、パーミッションの警告がでるので、クリック



何も設定せずにSaveをクリック

## Add K10 Cluster Role Binding



### Name

The Kubernetes custom resource name for the K10ClusterRoleBinding.

### K10 Cluster Role

Choose a role that defines the permissions you want to grant.

### Clusters

Choose which clusters the permissions should apply to

☒ All Clusters☐ Select Clusters

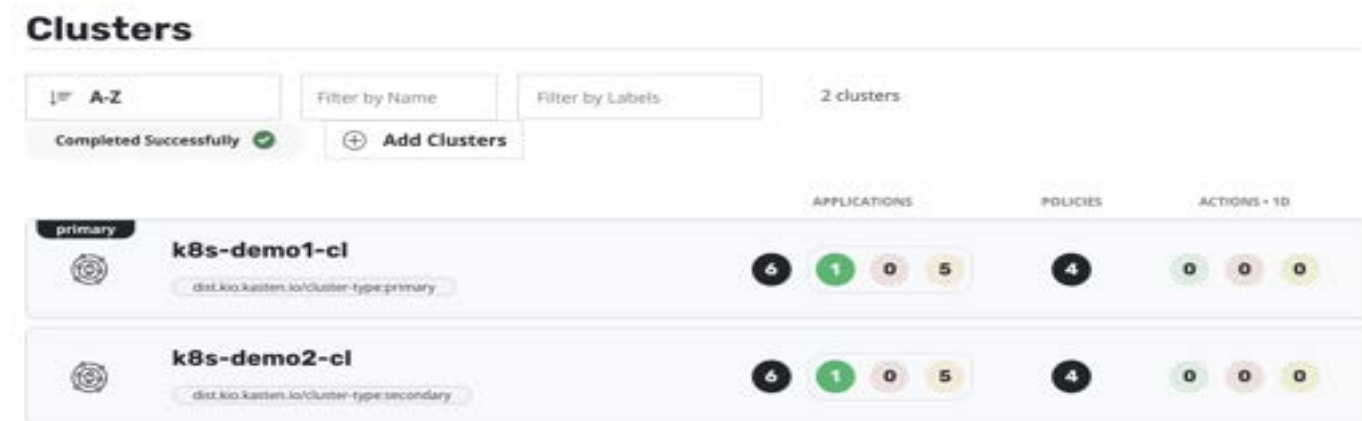
### Subjects

Define which users and/or groups you're giving permissions to.

KIND	NAME
User	system:serviceaccount:kasten-io:k10-k10 

☒ User☐ Group

クラスタの登録が完了



両方のクラスタにアクセスできるだけでなく、クラスタ共通のProfileやPolicyを作成して配布することが可能。

## Kastenのオフラインインストール (AirGap)方法 （オンラインでインストールをした場合は不要）

AirGapができると何がいい？

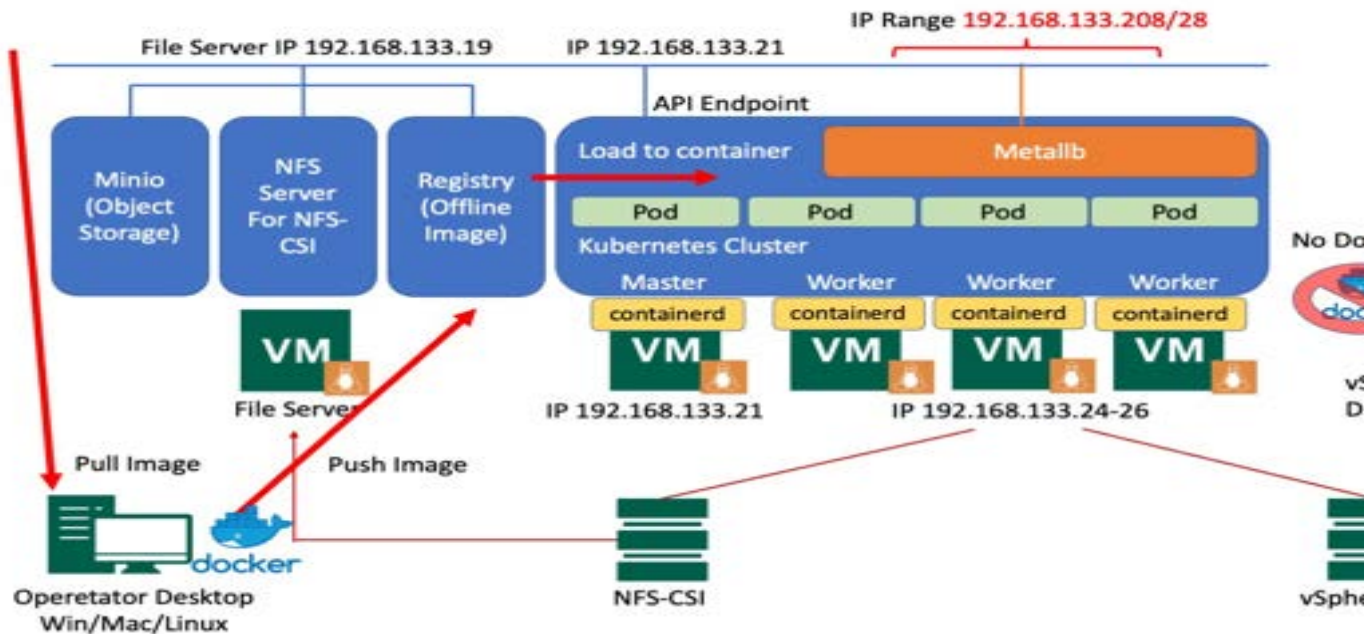
- インターネットのダイレクト接続が不要

- インターネットに通じていない環境や通じていない場合でも利用できる。

- 開発環境でテストしてから、本番環境へ適用ができる。（同じイメージでの動作補償）

K1-Kasten.shは、オンラインインストールだが、以下に書かれているAir Gapインストールも利用できる。もし、試したい場合は、一度以下のコマンドを実行する（Kastenがアンインストールされる。NFSに保存したバックアップデータも消えるので注意）

```
bash ./K-uninstall.sh
```



## Dockerが使える端末で必要な環境

このスクリプトでは、dockerをインストールしないので、dockerコマンドは入っていない。

インターネットに接続可能かつ、サーバに接続可能であるもの

空き容量は、Docker インストール後、最低でも5GBは必要

Windows/MAC/Linuxでdocker/helm/kubectが動くのであればプラットフォーム不問。

簡単に言えば、このサーバVMではなく、普通の手元のパソコン。

## 必要なソフトウェア

Docker

Windows/Macの場合は、Docker Desktop

<https://www.docker.com/products/docker-desktop>

Linuxの場合は、

`apt -y install docker.io`

をインストールしておく

Chocolatey/homebrewからkubectlとhelmをインストール

Windowsの場合は、

Chocolatey <https://chocolatey.org>

Mac/Linuxの場合は、

homebrew [https://brew.sh/index\\_ja](https://brew.sh/index_ja)

をインストール

これらのツールでkubectlコマンドとhelmコマンドをインストールしておく。

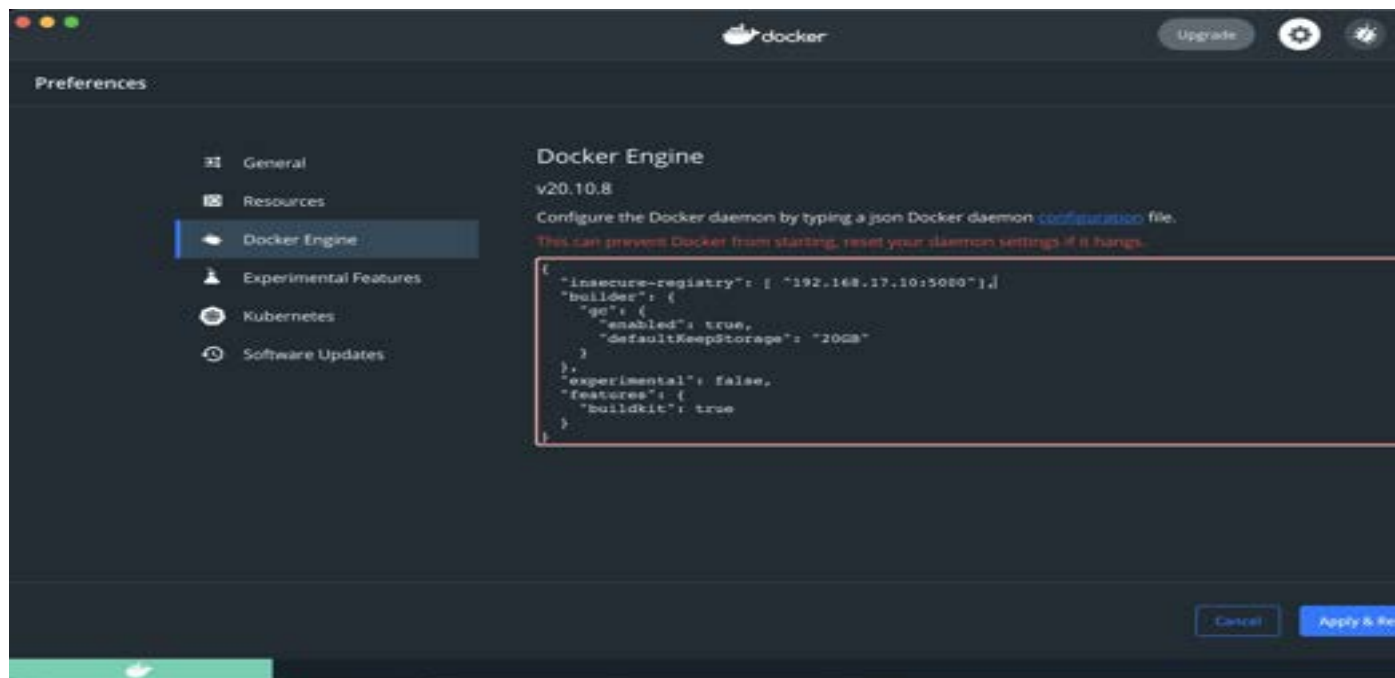
## Dockerの設定

Windows/MACの場合は、Docker Desktopをインストールしておく。Insecure Registryの設定を行なっておく。

"insecure-registries":["このVMのIP:5000"]

を追加しておく。

WindowsやMacでの設定例



Linuxの場合は、`~/docker/config.json`に記載のこと。

## kubectlの設定

Dockerの入っている端末でkubectl が動作できるようにする

<ホスト名>-cl-<ホスト名>\_kubeconfigのファイルが生成されているはずなので、ホームディレクトリ以下に.kube/configとしてコピーするか、コピペして.kube/configを作成する。以下を実行して動作確認

```
kubectl get nodes
```

## レジストリへ転送

<https://docs.kasten.io/latest/install/offline.html#air-gapped-k10-installation>

注意点は、**repo.example.com**と書かれているところを<このVMのIP>:5000として実行すること。

以下を実行。インターネットからのダウンロードとレジストリへのアップロードをするので、時間がかかる。

```
docker run --rm -it gcr.io/kasten-images/k10offline:<バージョン> list-images
docker run --rm -it -v /var/run/docker.sock:/var/run/docker.sock \
  gcr.io/kasten-images/k10offline:<バージョン> pull images
docker run --rm -ti -v /var/run/docker.sock:/var/run/docker.sock \
  -v ${HOME}/.docker:/root/.docker \
  gcr.io/kasten-images/k10offline:<バージョン> pull images --newrepo <このVMのIP>:5000
```

->プルまでは普通にできるが、プッシュができない場合は、Insecure-registryの設定が間違っているか、ホスト名、ポートが間違っているのいずれか。

プッシュした端末からはイメージのリストは見えるかもしれないが、以下のコマンドでプッシュされたことを確認。

```
curl -X GET http://<このVMのIP>:5000/v2/_catalog
```

```
{"repositories":["aggregatedapis","ambassador","auth","bitnami-shell","bloblifecyclemanager","catalog","cephtool","config","configmap-reload","crypto","dashboardbff","datamover","dex","executor","frontend","grafana","jobs","k10tools","kanister","kanister-tools","logging","metering","mongodb","mysql","postgresql","prometheus","restorectl","state","ubi-minimal","upgrade","vbrintegrationapi"]}
```

GUIで確認したい場合は、

Dockerが入っている端末で以下のDockerインスタンスを立ち上げて

```
docker run \
  -d \
  -e ENV_DOCKER_REGISTRY_HOST=<このVMのIP> \
  -e ENV_DOCKER_REGISTRY_PORT=5000 \
  --name registry-frontend \
  -p 18080:80 \
  konradkleine/docker-registry-frontend:v2
```

ブラウザで<http://localhost:18080>へアクセスするとDocker registry frontendが表示されて、Docker registryの中身が見える。



## Docker Registry Frontend

You are here: [Home](#) / [Repositories](#)

# Repositories



Filter repositories on this page

### aggregatedapis

[aggregatedapis](#)

### ambassador

[ambassador](#)

### auth

[auth](#)

### bitnami-shell

[bitnami-shell](#)

### bioblifecyclemanager

[bioblifecyclemanager](#)

### catalog

[catalog](#)

### cephtool

[cephtool](#)

プッシュが確認できたらDockerが入っている端末側のイメージを消しても構わない。  
Docker DesktopだとUIから一括で削除ができる。(Kastenだけで4GBもあるので)

## AirGapインストール

kastenのネームスペースを作っておく。また、スナップショットの注釈もつけておく。

```
kubectl create namespace kasten-io
kubectl annotate volumesnapshotclass csi-hostpath-snapclass \
    k10.kasten.io/is-snapshot-class=true
```

k10-<バージョン名>.tgzができていることを確認する。

```
helm repo update && \
    helm fetch kasten/k10
ls k10-<バージョン>.tgz
```

インストールオプションをつけて、インストールをする。Registryにイメージを落として  
いるのでいきなりコンテナの生成から始まる。

(以下は、vSphere CSI Driverにインストールする場合。詳しい手順やパラメータは、K1-kasten.shやドキュメントを参照のこと。)

```
helm install k10 k10-<バージョン>.tgz --namespace kasten-io \  
--set global.airgapped.repository=<このVMのIP>:5000 \  
--set gateway.insecureDisableSSLVerify=true \  
--set global.persistence.size=40G \  
--set auth.tokenAuth.enabled=true \  
--set externalGateway.create=true \  
--set ingress.create=true \  
--set grafana.enabled=true
```

---

## 実際にアプリケーションを動かす

### デモアプリの使い方

デモアプリは以下がある。

Wordpress

Postgress

Pacman

### Wordpressの説明

```
vi P-wordpress.sh
```

以下でnamespaceと作成するSCを指定できる。

NAMESPACE=wordpress-sample

SC=vsphere-sc

変更したら、以下を実行

```
./P-wordpress.sh
```

SC=csi-hostpath-sc とした場合のhelmに対する考慮は不要で、何も考えずに実行ができる。

スクリプトの最後にWordpressのURLが表示される。見逃したら

```
kubectl -n <NAMESPACE> get svc
```

でIPアドレスがわかる。

同様のデモアプリが、Postgresqlもある。

Blueprintでバックアップしたい場合は、

```
kubectl --namespace NAMESPACE annotate statefulset/mysql-release \  
kanister.kasten.io/blueprint=mysql-blueprint
```

注意点：

Blueprintが指定されているポリシーでVBRインテグレーションをするとバックアップに失敗する。

## PostgreSQL

P-postgresql-app.shでアプリケーション静止点のバックアップができる。Kastenをインストールした後に実行すること

P-postgresql-lb.shで、論理バックアップができる。（なぜかバックアップ失敗する。調査中）

## Pacman

P-pacman.shでMongodbベースのPacmanが起動する。svcでLoadbalancer IPを調べてWebブラウザで接続する。（音が鳴るので注意）

Blueprintは指定できない。

それ以外でもインターネットにある、Kubernetesのアプリやパブリッククラウド向けのアプリでも動作すると思われる。

以上で終わり