

Kubernetes Fundamental - Deployment Local to Production Part 4

Detail Materi



Persistence Volume

Indikator
Memahami Konsep dan Arsitektur
Persistent Volume
Dapat mengimplementasikan pada
service AWS



Implementasi EBS

Modul Sekolah DevOps Cilsy

Hak Cipta © 2020 **PT. Cilsy Fiolution Indonesia**

Hak Cipta dilindungi Undang-Undang. Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronik maupun mekanis, termasuk mecopy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis dan Penerbit.

Penulis : Estu Fardani

Editor: Taufik Maulana, Iqbal Ilman Firdaus & Muhammad Fakhri A

Revisi Batch 9

Penerbit : **PT. Cilsy Fiolution Indonesia**

Web Site : <https://cilsyfiolution.com> , <https://devops.cilsy.id>

Sanksi Pelanggaran Pasal 113 Undang-undang Nomor 28 Tahun 2014 tentang Hak Cipta

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam pasal 9 ayat (1) huruf i untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan atau pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).
2. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak cipta melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf c, huruf d, huruf f, dan atau huruf h, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah)
3. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf a, huruf b, huruf e, dan atau huruf g, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah)



Daftar Isi

Modul Sekolah DevOps Cilsy.....	2
11. Persistent Volume.....	5
Learning Outcomes.....	5
Outline Materi.....	5
11.1. Konsep Container.....	6
11.2. Persistent Data.....	6
11.2.1. Persistent Data : Volume.....	6
11.2.1.1. Praktek Docker Volume.....	8
11.2.2. Persistent Data : Bind Mount.....	10
11.2.2.1. Praktek Bind <i>Mount</i>	12
11.3. Konsep Persistent Volume dalam Kubernetes.....	14
11.3.1. Membuat Persistent Volume.....	15
11.3.2. Membuat Persistent Volume Claim.....	17
11.3.3. Menggunakan PVC pada Pod.....	19
11.4. Implementasi di AWS.....	21
11.5. Membuat EFS.....	23
11.5.1. Deklarasi Persistent Volume.....	24
11.5.2. Deklarasi Persistent Volume Claim.....	25
11.5.3. Deploy Aplikasi.....	25
11.5.4. Testing Aplikasi.....	26
11.6. Implementasi EBS.....	27
11.6.1. Membuat EBS.....	27
11.6.2. Mengaitkan EBS ke Node-01.....	27
11.6.3. Memformat EBS.....	27
11.6.4. Implementasi Persistent Volume EBS.....	27
11.6.5. Implementasi PVC EBS.....	28
11.7. Dynamic Volume Provisioning.....	31
11.7.1. Dynamic Volume Provisioning Best Practice.....	32



11.7.1.1. Minikube setup.....	32
11.7.1.2. Cek StorageClass.....	32
11.7.1.3. Membuat PVC.....	33
11.7.1.4. Membuat Pod.....	34
11.8. Referensi.....	36

11.

Persistent Volume

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Memahami Konsep dan Arsitektur Persistent Volume
2. Dapat mengimplementasikan pada service AWS

Outline Materi

1. Konsep Container
2. Persistent Data
3. Konsep Persistent Volume dalam Kubernetes
4. Implementasi di AWS
5. Membuat EFS
6. Dynamic Provisioning



11.1. Konsep Container

1. *Container* harus immutable. Artinya: *Container* tidak boleh diubah-ubah secara langsung. Jika kita ingin mengubah suatu data atau sistem di *container*, caranya dengan mematikan container, lalu nyalakan *container* dengan versi yang baru yang sudah dimodifikasi.
2. *Container* harus *temporary*. Artinya tidak ada *Container* yang spesial.

11.2. Persistent Data

Dua konsep *container* di atas menghasilkan masalah baru. Jika isi *Container* tidak boleh diubah-ubah, lalu bagaimana dengan data-data yang unik dan selalu berubah-ubah seperti database atau aset aplikasi? Dimana kita akan menyimpannya? Data-data yang unik dan selalu berubah ini disebut sebagai Persistent Data. Dan Docker memiliki 2 solusi untuk ini, yaitu Volume dan Bind Mount.

11.2.1. Persistent Data : Volume

Volume adalah cara terbaik untuk menyimpan persistent data pada *Container* karena dibuat dan ditangani langsung oleh sistem Docker. Ini untuk memastikan agar secara manajemennya lebih mudah, aman, dan minim error.

Volume memiliki *menu command* tersendiri yaitu `docker volume`. Dimana disana kita bisa melakukan berbagai kebutuhan seperti :

1. Membuat, menghapus volume : ``docker volume create, docker volume rm``
2. Menampilkan daftar volume yang ada : `docker volume ls`
3. Menggunakan berbagai tipe volume dari service cloud seperti S3 pada AWS dan Block Storage pada Digitalocean dengan opsi driver : `docker volume create -driver`

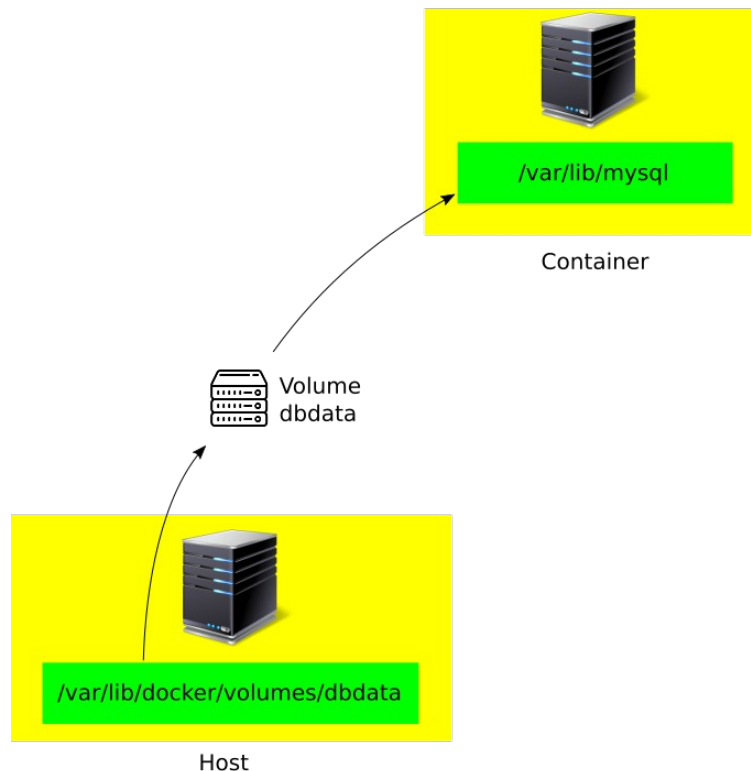


Data-data pada volume akan disimpan secara default di *folder* `/var/lib/docker/volumes` pada Host yang akan dibungkus kedalam sebuah wadah (*volume*) untuk kemudian folder ini akan di mount ke dalam Container.

Misalnya saja *folder* `/var/lib/docker/volumes/dbdata` akan dibungkus kedalam sebuah *volume* bernama `dbdata`. *Container* yang akan menggunakan *volume* `dbdata` ini adalah container database mysql (biasanya letak folder database mysql ada di `/var/lib/mysql`).

Maka *volume* `dbdata` (yang isinya adalah *folder* `/var/lib/docker/volumes/dbdata`) akan di *mounting* ke *folder* `/var/lib/mysql` pada *container* tersebut. Di dalam *container* mysql folder `/var/lib/mysql` akan berjalan seperti layaknya folder biasa. Agar lebih jelasnya perhatikan gambar berikut :





Ilustrasi *mounting volume* ke *container mysql*

Sistem *mounting* ini yang menyebabkan ketika *container* dihapus atau dimatikan tidak akan membuat *data* hilang, karena data masih tersimpan dengan baik di *volume dbdata* yang tersimpan pada *host*.

11.2.1.1. Praktek **Docker Volume**

Format untuk bisa menggunakan volume saat menjalankan *container* adalah menambahkan opsi berikut pada perintah docker container *run*:

```
-v namavolume:/letak/folder/mount/pada/container
```

Misalnya seperti ini jika kita ingin membuat *volume* bernama *mysql-data* dan diarahkan ke */var/lib/mysql* pada *container* yang menggunakan *image mysql*:

```
docker container run -d --name mysqlserver -e MYSQL_ALLOW_EMPTY_PASSWORD=True -v mysql-data:/var/lib/mysql mysql:latest
```



Atau misalnya seperti ini jika kita ingin membuat *volume* bernama *web-data* dan diarahkan ke */usr/share/nginx/html* untuk *Container* yang menggunakan *image* *nginx*:

```
$ docker container run -d --name webserver -p 80:80 -v
web-data:/usr/share/nginx/html nginx:latest
```

Sekarang mengujicoba apakah *volume* sudah dibuat dan datanya sudah *termounting* dengan baik ke dalam *container*.

```
docker volume ls
```

```
rizal@rizal-Inspiron-5468 ~ $ docker volume ls
DRIVER          VOLUME NAME
local          mysql-data
local          web-data
```

Terlihat bahwa ada 2 buah *volume* yang telah dibuat. *Drive local* artinya *volume* ini hanya menggunakan *folder* biasa dari *Host* ini.

Sekarang kita coba lihat dimanakan *volume* ini menyimpan data :

```
docker volume inspect mysql-data
```

```
docker volume inspect web-data
```

```
rizal@rizal-Inspiron-5468 ~ $ docker volume inspect mysql-data
[{"CreatedAt": "2018-10-03T16:50:36+07:00",
  "Driver": "local",
  "Labels": null,
  "Mountpoint": "/var/lib/docker/volumes/mysql-data/_data",
  "Name": "mysql-data",
  "Options": null,
  "Scope": "local"}]

Sekarang kita coba buktikan apakah betul sudah terbuat volume tersebut dan datanya sudah termounting dengan baik ke dalam Container :
```

```
rizal@rizal-Inspiron-5468 ~ $ docker volume inspect web-data
[{"CreatedAt": "2018-10-03T16:50:53+07:00",
  "Driver": "local",
  "Labels": null,
  "Mountpoint": "/var/lib/docker/volumes/web-data/_data",
  "Name": "web-data",
  "Options": null,
  "Scope": "local"}]
```

Bagian yang ditandai anak panah diatas adalah tempat dimana masing-masing *volume* ini menyimpan data. Jika kita melakukannya dengan benar, seharusnya isi dari `/var/lib/docker/volumes/mysql-data/_data` pada *Host* akan sama dengan isi `/var/lib/mysql` pada *container* *mysqlserver*. Begitupun yang terjadi pada *Container* *webserver*. Kita coba buktikan saja pada *Container* *mysqlserver*:

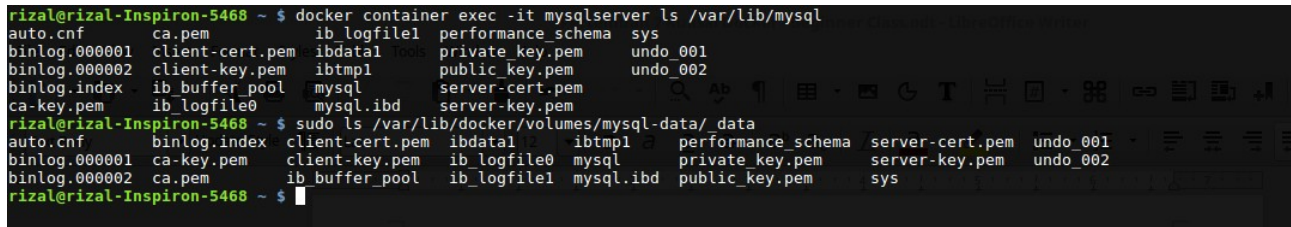


Perintah untuk cek isi `/var/lib/mysql` dari *container* `mysqlserver`

```
docker container exec -it mysqlserver ls /var/lib/mysql
```

Perintah untuk cek isi `/var/lib/docker/volumes/mysql-data/_data` pada Host

```
sudo ls /var/lib/docker/volumes/mysql-data/_data
```



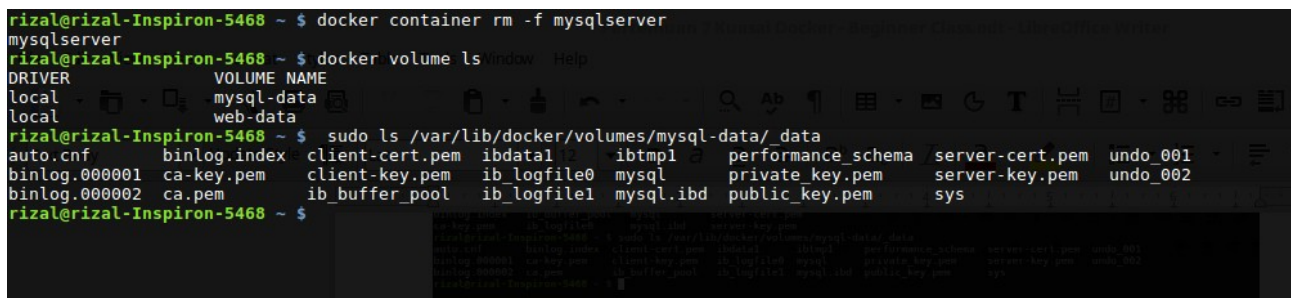
Pada gambar diatas terlihat hasilnya bahwa kedua *folder* tersebut memiliki isi data yang sama. Sekarang kita coba untuk menghapus *container* `mysql server` ini:

```
docker container rm -f mysqlserver
```

Lalu lihatlah bahwa *volume* `mysql-data` masih ada beserta isinya untuk membuktikan persistent data tetap terjaga.

```
docker volume ls
```

```
sudo ls /var/lib/docker/volumes/mysql-data/_data
```



Terbukti bahwa *volume* masih ada. Berikutnya kita bisa menjalankan *container* baru maupun menjalankan beberapa *container* menggunakan *volume* tersebut kembali.

11.2.2. Persistent Data : Bind Mount

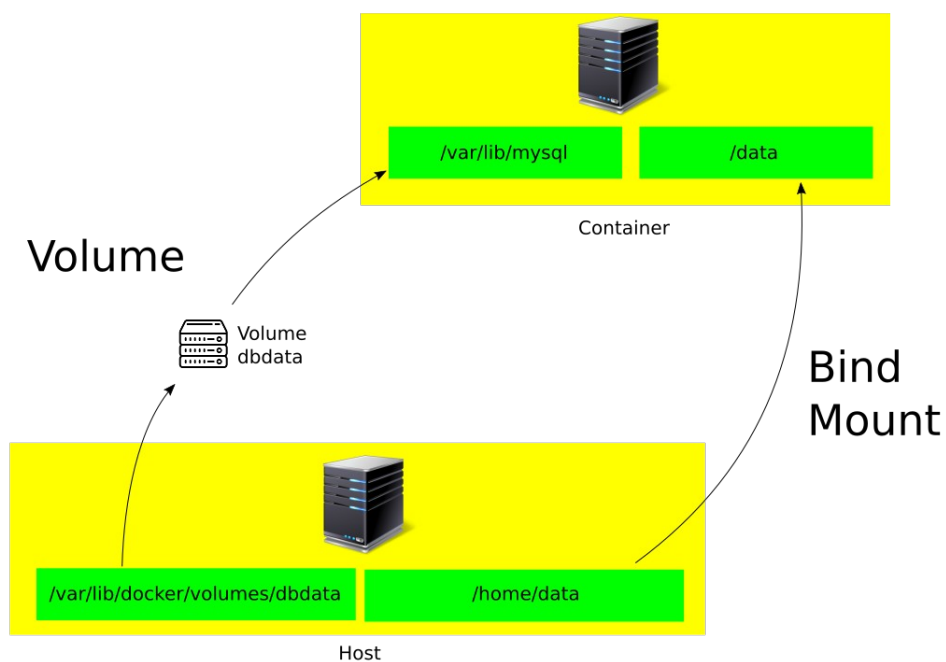
Secara konsep antara *Volume* dengan Bind Mount sebenarnya sama saja. Yaitu *me-mounting* suatu *folder* pada *Host* ke suatu *folder* di dalam *Container*.



Namun berbeda dengan *Volume*, Bind Mount sama sekali tidak ditangani oleh sistem Docker. Sehingga file/folder yang di mount ke dalam container dapat berada di folder manapun pada Host. Misalnya kita dapat *mounting folder* `/var/log/`, `/home/`, bahkan *folder* sistem seperti `/etc/`.

Jika pada *Volume* konsep *mounting folder*-nya seperti di bungkus ke dalam suatu wadah terlebih dahulu (itulah kenapa disebut *Volume*) baru di *mount*, jika pada Bind Mount maka si *folder*-nya langsung-lah yang di *mounting* ke dalam *Container*.

Maka pada Bind Mount tidak ada istilah nama seperti pada *Volume*. Bind Mount akan langsung merujuk pada alamat lengkap *folder*-nya seperti misalnya `/home/data`, tidak seperti pada *Volume* yang folder `/home/data` ini akan dibungkus kedalam suatu *volume* terlebih dahulu misalnya `dbdata`.



Ilustrasi bind mount

Sekilas terlihat sama, dan memang secara fungsi dan penggunaan akan tersama sama, namun ada kelebihan dan kekurangannya masing-masing secara implementasinya. Berikut kira-kira tabel perbandingannya :



	Kelebihan	Kekurangan
Volume	<ul style="list-style-type: none"> • Tidak tergantung letak persis dari <i>file/folder Host</i>. Sehingga cocok untuk kebutuhan <i>scale</i> dan <i>Fail over</i>. • Lebih mudah untuk kebutuhan <i>backup data</i> dan <i>migrasi</i>. • Lebih cocok untuk <i>production</i>. • Lebih cocok untuk <i>mounting</i> data yang akan terus tumbuh besar, misalnya <i>database</i>, penyimpanan <i>data user</i>, penyimpanan <i>data web app</i>. 	<ul style="list-style-type: none"> • Relatif lebih lambat, karena harus ada pemrosesan lebih banyak (melewati sistem <i>Docker</i> dulu) • Relatif sedikit rumit dalam penggunaan karena perlu belajar <i>volume</i> terlebih dahulu.
Bind Mount	<ul style="list-style-type: none"> • Relatif lebih cepat, karena langsung dari <i>filesystem host</i>. • Relatif lebih sederhana secara penggunaan. Tinggal mount suatu <i>file/folder</i>, selesai. • Lebih cocok untuk <i>local development</i>. • Lebih cocok untuk <i>mounting</i> file-file kecil atau file-file konfigurasi. Misalnya <i>mounting /etc/nginx/nginx.conf</i> saja atau <i>/etc/php/php.ini</i> saja. 	<ul style="list-style-type: none"> • Sangat tergantung dengan letak persis <i>file/folder</i> dari <i>Host</i>. Misal <i>folder /home/data</i> di <i>host A</i> pasti tidak sama dengan <i>host B</i>. Maka data tentunya perlu disinkronkan dulu antara <i>host A</i> dan <i>Host B</i> yang tentunya cukup repot dan bisa terjadi banyak <i>error</i>. Tidak cocok untuk kebutuhan <i>scale</i> dan <i>Fail over</i>.

11.2.2.1. Praktek *Bind Mount*

Bind Mount sangat cocok digunakan untuk proses testing-testing di local. Misalnya ketika kita membuat container berisi webserver *nginx* lalu kita perlu mengedit-edit tampilan website di dalamnya secara live.

Format dari Bind Mount sama dengan *volume* yaitu menggunakan *-v*, tetapi harus menggunakan alamat lengkap dari folder/file yang ingin di mount :



```
-v /alamat/lengkap/folder/sumber:/letak/folder/mount/pada/container
```

Bisa juga jika kita malas menulis alamat lengkap sumber di laptop/host kita, maka bisa menggunakan format berikut :

```
-v $(pwd):/letak/folder/mount/pada/container
```

Misalnya kita coba skenario berikut :

1. Kita akan praktek menggunakan file index.html, kita coba buat direktori praktekbind pada direktori home lalu buat file index pada direktori tersebut dengan isi sebagai berikut :

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Testing Bind Mount</title>
  </head>
  <body>
    <h1>Ini adalah tampilan awal dari index.html</h1>
  </body>
</html>
```

2. Kita akan menjalankan sebuah container nginx dengan mengaitkan folder praktek bind ke root directory web server dari nginx yaitu di /usr/share/nginx/html.
3. Saat kondisi container masih berjalan, kita dapat mengubah-ubah isi dari index.html secara live.

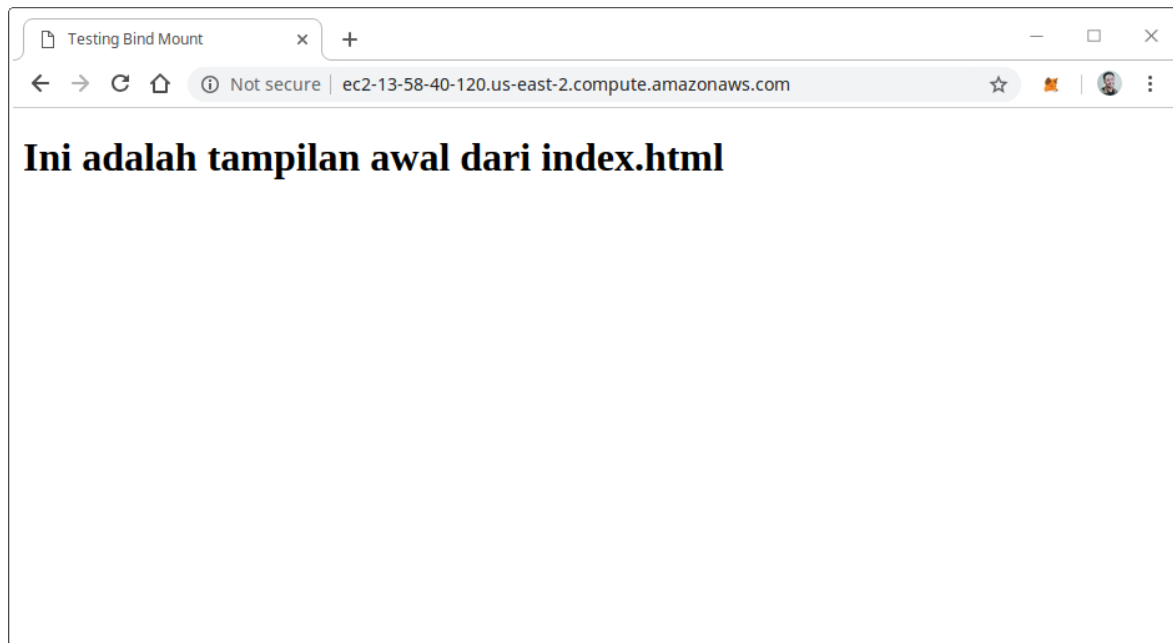
Kita coba dipraktekkan :

Pertama-tama masuklah ke dalam folder praktekbind melalui terminal masing-masing. Lalu ketikkan perintah berikut :

```
docker container run --rm -p 80:80 -v $(pwd):/usr/share/nginx/html nginx:latest
```

Setelah *container* berhasil berjalan, lihatlah tampilan awal dari nginx :





Hasil dari bind mount

Kira-kira seperti itu penggunaan dari Bind Mount. Kita dapat menggunakan Bind Mount untuk contoh-contoh lain misalnya isi dari file konfigurasi `nginx.conf` atau `php.ini` yang perlu diubah-ubah secara live.

11.3. Konsep Persistent Volume dalam Kubernetes

Salah satu fitur yang disediakan oleh Kubernetes adalah adanya **Persistent Volume (PV)**. Ini berguna untuk menyediakan resource volume pada kluster yang dapat digunakan sesuai kebutuhan. **Persistent Volume** adalah volume yang disediakan oleh administrator dengan file sistem tertentu, ukuran, dan pengenal seperti ID dan nama dari volume tersebut.

Namun, persistent volume tidak dapat langsung digunakan untuk pod. Oleh karena itu, ada fitur lain yang Kubernetes berikan, yaitu adalah **Persistent Volume Claim (PVC)**. **Persistent Volume Claim** adalah sebuah permintaan penggunaan Persistent Volume dari pengguna. Claim ini ditujukan ke sebuah pod yang mengkonsumsi resource dari **Persistent Volume** sesuai dengan permintaan dari **Persistent Volume Claim**. Dalam persistent volume claim, kita bisa menyebutkan spesifikasi volume yang diperlukan untuk pod misalnya adalah ukuran dari volume yang dibutuhkan.



Tipe Storage

- gcePersistentDisk
- awsElasticBlockStore
- Cinder

Untuk praktikum selanjutnya, kita akan mencoba membuat **Persistent Volume**, **Persistent Volume Claim** dan menggunakannya pada pod. Dalam praktik ini, kita bisa menggunakan **minikube** sebagai bahan percobaan dengan driver docker. Tujuannya adalah agar aplikasi pada pod dapat menggunakan volume yang ada pada host fisik yang dijadikan Persistent Volume.

11.3.1. Membuat Persistent Volume

Langkah awal adalah untuk menjalankan minikube dan docker dengan perintah **minikube start --driver=docker**.

```
taufik@hewlettpackard:~/shared$ minikube start --driver=docker
minikube v1.18.1 on Ubuntu 20.04
Using the docker driver based on user configuration
Starting control plane node minikube in cluster minikube
Creating docker container (CPUs=2, Memory=2200MB) ...
Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v4
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Pastikan minikube berjalan dengan perintah **minikube status**.

```
taufik@hewlettpackard:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
timeToStop: Nonexistent
```



Setelah minikube berjalan, kita cek docker untuk masuk ke container dari minikube dengan perintah **docker ps**.

```
taufik@hewlett-packard:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
88e15a807268	gcr.io/k8s-minikube/kicbase:v0.0.18	"/usr/local/bin/entr..."	5 minutes ago	Up 5 minutes	127.0.0.1:49177->22/tcp, 127.0.0.1:49176->2376/tcp, 127.0.0.1:49175->5000/tcp, 127.0.0.1:49174->8443/tcp, 127.0.0.1:49173->32443/tcp

Terlihat bahwa ada container minikube dengan container id **88e15a807268**. Kita masuk ke kontainer tersebut dengan perintah **docker exec -it 88e15a807268 bin/bash**. Tujuannya adalah untuk membuat direktori yang akan di bind untuk **Persistent Volume**. Mengingat kita menjalankan **minikube** diatas **docker**, maka persistent volume tidak dapat dibind langsung ke volume local (laptop atau komputer), melainkan ke volume dari container minikubanya itu sendiri.

```
taufik@hewlett-packard:~$ docker exec -it 88e15a807268 bin/bash
root@minikube:/#
```

Langkah selanjutnya yang dilakukan adalah untuk menyiapkan data fisik yang akan dijadikan **Persistent Volume**. Yaitu dengan membuat direktori **/mnt/data** dengan file **index.html** seperti berikut.

```
controlplane $ sudo mkdir /mnt/data
controlplane $ cd /mnt/data/
controlplane $ sudo sh -c "echo 'Hello from Kubernetes storage' > /mnt/data/index.html"
controlplane $ cat index.html
Hello from Kubernetes storage
```

Setelah itu kita bisa keluar dari kontainer minikube. Selanjutnya kita akan buat file konfigurasi persistent volume ini dengan nama **vp-devops.yaml**.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-devops
  labels:
```




```

    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"

```

script dapat dilihat di

<https://gist.github.com/sdcilsy/e4d13017f7d839c0a0e4c58307999986>

File konfigurasi tersebut memiliki **kapasitas** 10Gb dengan **access mode** Read Write Once, dan mounting pada `/mnt/data`. Access mode ini berarti persistent volume ini hanya dapat di read dan write oleh 1 node saja.

Selanjutnya kita eksekusi perintah **kubectl apply -f pv-devops.yaml** untuk membuat **persistent volume**.

```

controlplane $ kubectl apply -f pv-devops.yaml
persistentvolume/pv-devops created

```

kita bisa melihat **persistent volume** dengan perintah **kubectl get pv**.

```

controlplane $ kubectl get pv

```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE			
pv-devops	10Gi	RWO	Retain	Available	
manual		5m34s			

11.3.2. Membuat Persistent Volume Claim

Langkah selanjutnya adalah untuk membuat **Persistent Volume Claim** untuk kebutuhan pod yang akan dibuat nanti. Caranya hampir sama dengan membuat **persistent volume**, namun kita perlu mendefinisikan **persistent volumenya** dan alokasi ukuran volume yang diminta seperti pada file konfigurasi vpc berikut yang diberi nama **vpc-devops.yaml**.



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-devops
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```



script dapat dilihat di

<https://gist.github.com/sdcilsy/63405c8e10cdb9c70b2a76bd42747198>

Pada vpc tersebut, kita sudah mendefinisikan alokasi volume yang dibutuhkan.

Langkah selanjutnya adalah mengeksekusi perintah `kubectl apply -f pvc-devops.yaml` untuk membuat pvc.

```
controlplane $ kubectl apply -f vpc-devops.yaml
persistentvolumeclaim/pvc-devops created
```

Kita dapat melihat hasil pvc dengan perintah **kubectl get pv** dan **kubectl get pvc**.

```
controlplane $ kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM
STORAGECLASS  REASON    AGE
pv-devops     10Gi      RWO           Retain          Bound   default/pvc-
devops        manual    13m

controlplane $ kubectl get pvc
NAME          STATUS  VOLUME      CAPACITY  ACCESS MODES  STORAGECLASS  AGE
pvc-devops    Bound   pv-devops    10Gi      RWO           manual        59s
```

Dapat dilihat bahwa VP sekarang statusnya bound dan VPC secara otomatis mengklaim VP yang sudah dibuat sebelumnya.

11.3.3. Menggunakan PVC pada Pod

Lalu sekarang kita melakukan binding volume yang sudah diclaim ke pod. Kita akan membuat file konfigurasi pod dengan nama **pod-nginx.yaml** seperti berikut.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
```

```
- name: pv-devops-storage
  persistentVolumeClaim:
    claimName: pvc-devops
containers:
- name: pod-nginx
  image: nginx
  ports:
    - containerPort: 80
      name: "http-server"
  volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: pvc-devops-storage
```

script dapat dilihat di

<https://gist.github.com/sdcilsy/df19f357cd82aba77b4d9b093f636240>

Pastikan pod yang sudah dibuat berjalan dengan perintah **kubectl get pod pod-nginx**

```
controlplane $ kubectl get pod pod-nginx
NAME          READY   STATUS    RESTARTS   AGE
pod-nginx     1/1     Running   0           6m2s
```

Pengujian bisa dilakukan dengan mengecek aplikasi nginx dengan curl. Caranya adalah kita masuk ke pod yang sudah dibuat

```
controlplane $ kubectl exec -it pod-nginx -- /bin/bash
```

Setelah masuk ke pod, lakukan update dan install aplikasi **curl**, lalu mengeksekusi perintah **curl localhost**

```
apt update
apt install curl
curl http://localhost/
```

Jika pengujian berhasil, maka output dari perintah **curl** tadi adalah seperti yang dibuat pada awal praktikum.

```
root@pod-nginx:/# curl localhost
Hello from Kubernetes storage
```



11.4. Implementasi di AWS

Teorinya adalah kita memiliki deployment yang memerlukan volume berfungsi untuk menyimpan file/data dari service. Disini ada dua tipe storage yg bisa digunakan, masing-masing dengan kelebihan dan kekurangannya.

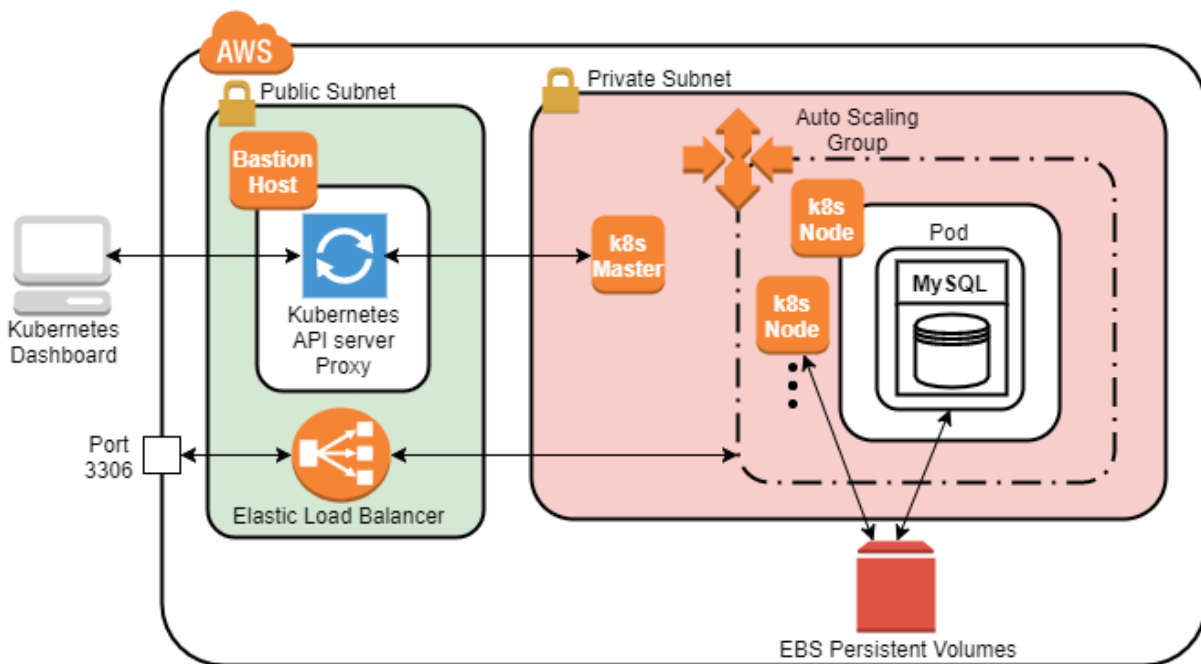
EBS (elastic block storage) vs EFS (Elastic File System)

EBS	EFS
<i>Amazon EBS is a cloud block storage service that provides direct access from a single EC2 instance to a dedicated storage volume.</i>	<i>Amazon EFS provides a shared file storage for use with compute instances in the AWS cloud and on premise servers.</i>
<i>Applications that require persistent dedicated block access for a single host can use EBS as a high available and low-latency block storage solution.</i>	<i>Applications that require shared file access can use Amazon EFS for reliable file storage delivering high aggregate throughput to thousands of clients simultaneously.</i>
<i>EBS PV provides only ReadWriteOnce access mode.</i>	<i>EFS PV provides ReadWriteMany access mode.</i>
<i>EBS can be accessed by the host it is connected within the zone. EBS volume is automatically replicated within its Availability Zone to protect you from component failure, offering high availability and durability.</i>	<i>AN EFS file system can be accessed from multiple availability zones and it is the valuable for multi-AZ cluster.</i>
<i>Automatic scaling is not available in EBS but can be scaled up/down based on the need.</i>	<i>It is better to choose EFS when it is difficult to estimate the amount of storage the application will use because EFS is built to elastically scale.</i>
<i>Cost-efficient.</i>	<i>More costly than EBS.</i>
<i>Can support all types of applications.</i>	<i>EFS is a file system; hence, it won't support some applications such as databases that require block storage.</i>



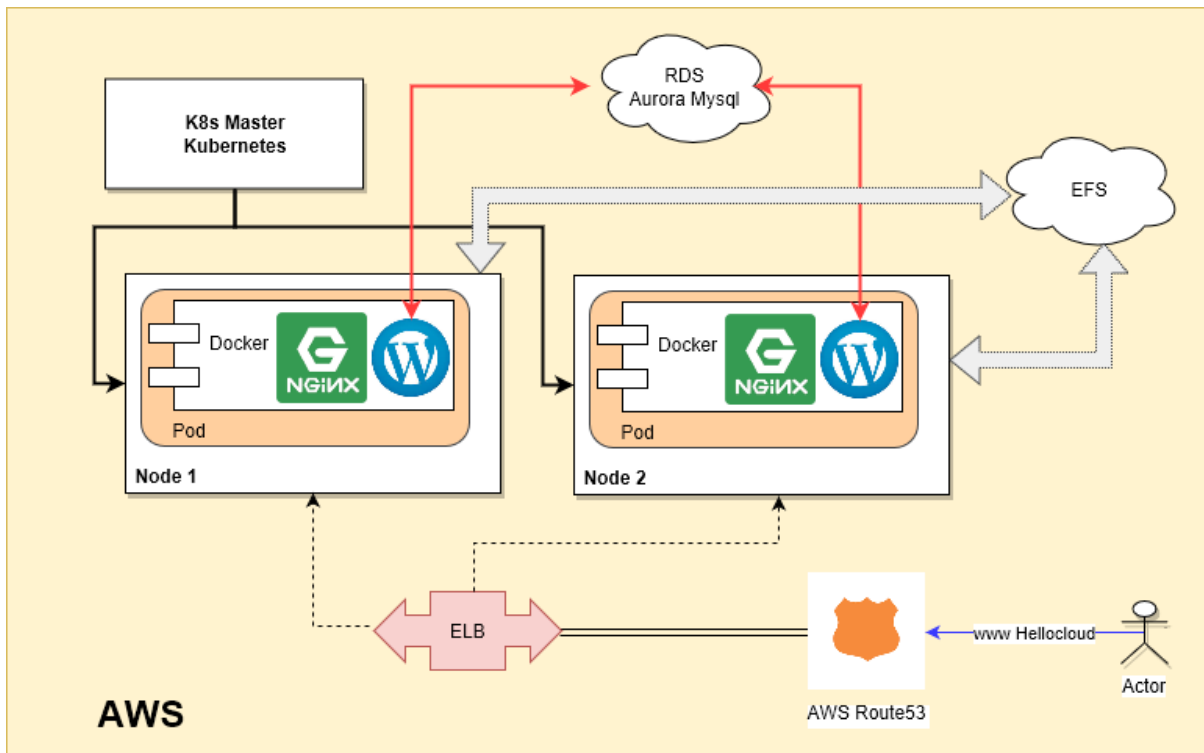
<i>EBS on the other hand provides point-in-time snapshots of EBS volumes, which are backed up to Amazon S3 for long-term durability.</i>	<i>EFS doesn't support any backup mechanism we need to setup backup manually.</i>
<i>Amazon EBS provides the ability to copy snapshots across AWS regions, enabling geographical expansion, data center migration, and disaster recovery providing flexibility and protecting for your business.</i>	<i>EFS doesn't support snapshots.</i>

Dengan singkat kata, EBS adalah hardisk yang akan kita tempel. Dan EFS adalah NAS.



Gambar EBS dalam kluster





Gambar EFS dalam kluster

Dalam implementasinya, EBS tidak serta merta dapat diimplementasikan seperti konsep diatas. Ini dikarenakan dalam proses menjamin keutuhan data, EBS yang dipasang di node1, harusnya ketika diakses memiliki konten yang sama ketika diakses dari node2 atau node3. Untuk menyelesaikan issue diatas, butuh implementasi CEPH sebagai salah satu Software-defined storage. Bisa dengan bantuan rock.io.

Di kelas cilsy materi ini belum akan diajarkan, namun siswa dipersilahkan riset mengenai teknologi ini.

Untuk praktek Persistent Volume kali ini, kita akan menggunakan EFS saja. Diagramnya menggunakan diagram EFS diatas.

11.5. Membuat EFS

Buatlah EFS dari menu aws <https://console.aws.amazon.com/efs/>. Beberapa pengaturan yg dibutuhkan:



- VPC pilih vpc dari cluster kubernetes
- Enable Lifecycle Management, select a Lifecycle policy
- Keep Bursting and General Purpose selected as your default performance and throughput modes.
- Security Group masukkan SG untuk master dan nodes kubernetes

Catat output yang dihasilkan (kira-kira seperti ini):

- Domain: fs-5ce9b11d.efs.ap-southeast-1.amazonaws.com

11.5.1. Deklarasi Persistent Volume

```
# default
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-log
  namespace: default
  labels:
    type: nfs-kubernetes
spec:
  storageClassName: aws-efs
  capacity:
    storage: 60Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /
    server: fs-5ce9b11d.efs.ap-southeast-1.amazonaws.com
    readOnly: false
```

Script dapat dilihat di <https://gist.github.com/sdcilisy/b01e7587a7abc10fe618973809cd770b>



11.5.2. Deklarasi Persistent Volume Claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: efs
  annotations:
    volume.beta.kubernetes.io/storage-class: "aws-efs"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
  selector:
    matchLabels:
      type: "nfs-kubernetes"
```

Script dapat dilihat di

<https://gist.github.com/sdcilsy/917c5412dd3fe205b8c09c0a42df3a65>

11.5.3. Deploy Aplikasi

```
# Deploy the pods
apiVersion: apps/v1
kind: Deployment
metadata:
  name: d-sandbox-deployment
  namespace: development
  labels:
    app: bbox-sandbox
    env: development
spec:
  selector:
    matchLabels:
      app: d-sandbox
      env: development
```



```
replicas: 1
template:
  metadata:
    labels:
      app: d-sandbox
      env: development
  spec:
    volumes:
      - name: d-sandbox-log
        persistentVolumeClaim:
          claimName: efs
    imagePullSecrets:
      - name: registry-secret
    containers:
      - name: d-sandbox
        image: tuanpembual/sandbox:latest
        imagePullPolicy: Always
        ports:
          - containerPort: 8080
        volumeMounts:
          - name: d-sandbox-log
            mountPath: "/log"
            subPath: "sandbox/development/"
```

script dapat dilihat di

<https://gist.github.com/sdcilsy/71c48a7cef36b4dad20b9857cafba28e>

11.5.4. Testing Aplikasi

Aplikasi bisa dicoba dengan exec ke pods, kemudian cek isi folder nya. Atau jalankan pods lain yang mengarah ke volume ini juga.



11.6. Implementasi EBS

11.6.1. Membuat EBS

Kita akan melakukan beberapa konfigurasi menggunakan aws cli.

```
aws ec2 create-volume --availability-zone=ap-southeast-1a --size=10 --volume-type=gp2
```

Akan mengeluarkan output seperti ini:

```
{
  "AvailabilityZone": "ap-southeast-1a",
  "CreateTime": "2020-03-13T03:16:27.000Z",
  "Encrypted": false,
  "Size": 10,
  "SnapshotId": "",
  "State": "creating",
  "VolumeId": "vol-05662622bb17b963d",
  "Iops": 100,
  "Tags": [],
  "VolumeType": "gp2"
}
```

11.6.2. Mengaitkan EBS ke Node-01

Ganti instance-id dengan instance-id node-01 masing-masing.

```
aws ec2 attach-volume --device /dev/xvdf --instance-id i-008418f1bc35b9144 --volume-id vol-05662622bb17b963d
```

11.6.3. Memformat EBS

SSH ke node-01. Lakukan format storage.

```
sudo mkfs -t ext4 /dev/xvdf
```

11.6.4. Implementasi Persistent Volume EBS

Ganti volume ID dengan output dari pembuatan EBS

```
apiVersion: v1
kind: PersistentVolume
```



```
metadata:
  name: pv0001
  labels:
    type: amazonEBS
spec:
  storageClassName: gp2
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: vol-05662622bb17b963d
    fsType: ext4
```

script dapat dilihat di

<https://gist.github.com/sdcilisy/e502f7458189227dcddf8d1e392f5b36>

11.6.5. Implementasi PVC EBS

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: sandbox-pvc
  labels:
    type: amazonEBS
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      type: "amazonEBS"
```

script dapat dilihat di

<https://gist.github.com/sdcilisy/15b9f007e7889c21660f52b02cd732d1>



Implementasi PVC EBS di Deployment

```
# Deploy the pods
apiVersion: apps/v1
kind: Deployment
metadata:
  name: d-sandbox-deployment
  namespace: development
  labels:
    app: bbox-sandbox
    env: development
spec:
  selector:
    matchLabels:
      app: d-sandbox
      env: development
  replicas: 1
  revisionHistoryLimit: 5
  template:
    metadata:
      labels:
        app: d-sandbox
        env: development
    spec:
      volumes:
        - name: sandbox-data
          persistentVolumeClaim:
            claimName: sandbox-pvc
      containers:
        - name: d-sandbox
          image: tuanpembual/sandbox:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          volumeMounts:
```



```
- name: sandbox-data  
  mountPath: /data/
```

script dapat dilihat di

<https://gist.github.com/sdcilsy/66441b0e3dee4d4ec0ed96ca91c25927>



11.7. Dynamic Volume Provisioning

Dynamic Volume Provisioning memungkinkan volume penyimpanan dibuat sesuai permintaan. Tanpa Dynamic Volume Provisioning, administrator cluster harus secara manual memanggil cloud atau penyedia penyimpanan mereka untuk membuat volume penyimpanan baru, lalu membuat objek PersistentVolume di Kubernetes. Fungsi dinamis ini menghilangkan kebutuhan administrator klaster untuk melakukan pra-konfigurasi penyimpanan. Sebaliknya, ini akan secara otomatis mengatur ruang penyimpanan ketika pengguna memintanya.

Dengan menggunakan Dynamic Volume Provisioning, apabila kita ingin menggunakan volume pada kubernetes, kita hanya mendeklarasikan saja **VPC** dengan object **StorageClass** yang sudah dibuat oleh platform yang kita gunakan. Si **StorageClass** ini lah yang akan mengatur request volume yang kita minta.

Singkatnya, dengan menggunakan Dynamic Volume Provisioning, secara otomatis, request volume kita akan dihandle oleh **StorageClass**.

Ada beberapa StorageClass yang tersedia diberbagai Cloud Provider contohnya seperti pada tabel berikut

Cloud Provider	Default StorageClass Name	Default Provisioner
AWS	gp2	aws-ebs
Microsoft Azure	standard	azure-disk
Google Cloud Platform	standard	gce-pd
OpenStack	standard	cinder
VMware vSphere	thin	vsphere-volume



11.7.1. Dynamic Volume Provisioning Best Practice

Sebetulnya, **Minikube** juga sudah mempunyai StorageClass dengan nama **standard** dengan provisioner, yaitu **HostPath**. Oleh karena itu, kita akan melakukan praktikum menggunakan Minikube lagi.

11.7.1.1. Minikube setup

Seperti biasa, kita akan menjalankan Minikube dengan perintah seperti dibawah.

```
Controlplane $ minikube start --driver=docker
minikube v1.18.1 on Ubuntu 20.04
Using the docker driver based on user configuration
Starting control plane node minikube in cluster minikube
Creating docker container (CPUs=2, Memory=2200MB) ...
Preparing Kubernetes v1.20.2 on Docker 20.10.3 ...
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v4
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```

11.7.1.2. Cek StorageClass

Kita bisa mengecek storage class pada cluster kita menggunakan perintah **kubectl get sc**.




```
taufik@hewlettpackard:~$ kubectl get sc
NAME                PROVISIONER          RECLAIMPOLICY    VOLUMEBINDINGMODE  ALLOWVOLUMEEXPANSION  AGE
standard (default)  k8s.io/minikube-hostpath  Delete           Immediate           false                 33s
```

Bisa dilihat bahwa Minikube sudah memiliki **StorageClass** default seperti yang sudah disebutkan diawal.

11.7.1.3. Membuat PVC

Langkah selanjutnya adalah kita membuat file konfigurasi **PVC** bernama **pvc-dynamic.yaml** dengan storage **standard** dengan request volume 1GB.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-volume
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi
```

script dapat dilihat di

<https://gist.github.com/sdcilsy/ce5d0be085d5fbb02ba1a2958af188f9>

Kita bisa mengeksekusi perintah berikut untuk membuat **PVC**

```
controlplane $ kubectl apply -f pvc-dynamic.yaml
persistentvolumeclaim/dynamic-volume created
```

Kita bisa melihat hasil dari **PVC** ini menggunakan perintah **kubectl get PVC**.

```
taufik@hewlettpackard:~$ kubectl get pvc
NAME                STATUS    VOLUME                                     CAPACITY  ACCESS MODES  STORAGECLASS  AGE
dynamic-volume      Bound     pvc-cb623150-4822-4f55-920f-1b14c022b5a7  1Gi       RWO           standard      7m23s
```

Kita bisa melihat, bahwa PVC sudah dibuat.



11.7.1.4. Membuat Pod

Selanjutnya, kita akan membuat pod dengan volume yang sudah dibuatkan VPC nya. Misalnya kita membuat pod dari image nginx. Kita buat terlebih dahulu file konfigurasi pod dengan nama **pod-dynamic.yaml**.



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
spec:
  volumes:
    - name: volume-1
      persistentVolumeClaim:
        claimName: dynamic-volume
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: volume-1
```

script dapat dilihat di

<https://gist.github.com/sdcilisy/a038d44a160d3792fba028305413607b>

Kita bisa mengeksekusi perintah berikut untuk membuat **pod**

```
controlplane $ kubectl apply -f pod-dynamic.yaml
pod/pod-nginx created
```

Cobalah untuk menyimpan file ke container, lalu coba mengakses service dari kontainer yang sudah dibuat tadi.



11.8. Referensi

- <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
- <https://containerjournal.com/topics/container-networking/using-ebs-and-efs-as-persistent-volume-in-kubernetes/>
- <https://www.nebulaworks.com/blog/2019/08/27/leveraging-aws-ebs-for-kubernetes-persistent-volumes/>
- <https://portworx.com/tutorial-kubernetes-persistent-volumes/>
- <https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/>
- <https://kubernetes.io/blog/2017/03/dynamic-provisioning-and-storage-classes-kubernetes/>
- <https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/>
- <https://platform9.com/blog/tutorial-dynamic-provisioning-of-persistent-storage-in-kubernetes-with-minikube/>

