8 Storing data on hard drives: EBS and instance store

This chapter covers

- Attaching persistent storage volumes to an EC2 instance
- Using temporary storage attached to the host system
- Backing up volumes
- Testing and tweaking volume performance
- Differences between persistent (EBS) and temporary volumes (instance store)

Imagine your task is to migrate an enterprise application being hosted on-premises to AWS. Typically, legacy applications read and write files from a filesystem. Switching to object storage, as described in the previous chapter, is not always possible or easy. Fortunately, AWS offers good old block-level storage as well, allowing you to migrate your legacy application without the need for expensive modifications.

Block-level storage with a disk filesystem (FAT32, NTFS, ext3, ext4, XFS, and so on) can be used to store files as you would on a personal computer. A *block* is a sequence of bytes and the smallest addressable unit. The OS is the intermediary between the application that needs to access files and the underlying filesystem and block-level storage. The disk filesystem manages where (at what block address) your files are stored. You can use block-level storage only in combination with an EC2 instance where the OS is running.

The OS provides access to block-level storage via open, write, and read system calls. The simplified flow of a read request goes like this:

1. An application wants to read the file /path/to/file.txt and makes a read system call.
2. The OS forwards the read request to the filesystem.
3. The filesystem translates /path/to/file.txt to the block on the disk where the data is stored.

Applications like databases that read or write files by using system calls must have access to block-level storage for persistence. You can't tell a MySQL database to store its files in an object store because MySQL uses system calls to access files.

Not all examples are covered by the Free Tier

The examples in this chapter are not all covered by the Free Tier. A warning message appears when an example incurs costs. Nevertheless, as long as you don't run all other examples longer than a few days, you won't pay anything for them. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

AWS provides two kinds of block-level storage:

- *A* persistent block-level storage *volume connected via network*—This is the best choice for most problems, because it is independent of your virtual machine's life cycle and replicates data among multiple disks automatically to increase durability and availability.
- *A* temporary block-level storage *volume physically attached to the host system of the virtual machine*—This is interesting if you're optimizing for performance, because it is directly attached to the host system and, therefore, offers low latency and high throughput when accessing your data.

The next three sections will introduce and compare these two solutions by connecting storage with an EC2 instance, doing performance tests, and exploring how to back up the data.

8.1 Elastic Block Store (EBS): Persistent block-level storage attached over the network

Elastic Block Store (EBS) provides persistent block-level storage with built-in data replication. Typically, EBS is used in the following scenarios:

- Operating a relational database system on a virtual machine
- Running a (legacy) application that requires a filesystem to store data on EC2
- Storing and booting the operating system of a virtual machine

An EBS volume is separate from an EC2 instance and connected over the network, as shown in figure 8.1. EBS volumes have the following characteristics:

- They aren't part of your EC2 instances; they're attached to your EC2 instance via a network connection. If you terminate your EC2 instance, the EBS volumes remain.

- They are either not attached to an EC2 instance or attached to exactly one EC2 instance at a time.
- They can be used like typical hard disks.
- They replicate your data on multiple disks to prevent data loss due to hardware failures.

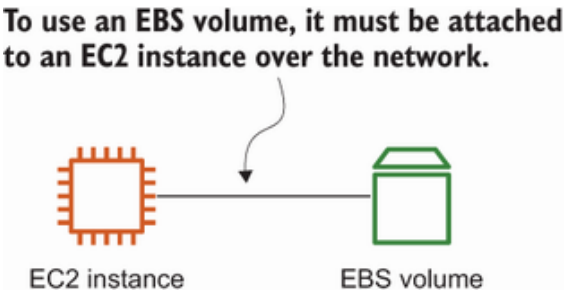

Figure 8.1 EBS volumes are independent resources but can be used only when attached to an EC2 instance.

EBS volumes have one big advantage: they are not part of the EC2 instance; they are an independent resource. No matter whether you stop your virtual machine or your virtual machine fails because of a hardware defect, your volume and your data will remain.

By default, AWS sets the `DeleteOnTermination` attribute to `true` for the root volume of each EC2 instance. This means, whenever you terminate the EC2 instance, the EBS volume acting as the root volume gets deleted automatically. In contrast, AWS sets the `DeleteOnTermination` attribute to `false` for all other EBS volumes attached to an EC2 instance. When you terminate the EC2 instance, those EBS volumes will remain. If you need to modify the default behavior, it is possible to override the initial value for the `DeleteOnTermination` attribute.

WARNING You can't easily attach the same EBS volume to multiple virtual machines! If you are still interested in attaching an EBS volume to many EC2 instances, read **http://mng.bz/p68w** carefully and see if the many limitations still support your workload. See chapter 9 if you are looking for a network filesystem.

8.1.1 Creating an EBS volume and attaching it to your EC2 instance

Let's return to the example from the beginning of the chapter. You are migrating a legacy application to AWS. The application needs to access a filesystem to store data. Because the data contains business-critical information, durability and availability are important. Therefore, you create an EBS volume for persistent block storage. The legacy application runs on a virtual machine, and the volume is attached to the EC2 instance to enable access to the block-level storage.

The following bit of code demonstrates how to create an EBS volume and attach it to an EC2 instance with the help of CloudFormation:

```
Instance:                                    ①
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]                                  ②
Volume:                                      ③
  Type: 'AWS::EC2::Volume'
  Properties:
    AvailabilityZone: !Sub ${Instance.AvailabilityZone}
    Size: 5                                  ④
    VolumeType: gp2                          ⑤
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 8 (EBS)
VolumeAttachment:                            ⑥
  Type: 'AWS::EC2::VolumeAttachment'
  Condition: Attached
  Properties:
    Device: '/dev/xvdf'                      ⑦
    InstanceId: !Ref Instance                ⑧
    VolumeId: !Ref Volume                    ⑨
```

① Defines the EC2 instance

② We are skipping the properties of the EC2 instance in this example.

③ Defines the EBS volume

④ Creates a volume with a 5 GB capacity

⑤ Default volume type based on SSD

⑥ Attaches the EBS volume to the EC2 instance

⑦ Name of the device used by the EC2 instance

⑧ References the EC2 instance

⑨ References the EBS volume

An EBS volume is a standalone resource. This means your EBS volume can exist without an EC2 instance, but you need an EC2 instance to access the EBS volume.

8.1.2 Using EBS

To help you explore EBS, we've prepared a CloudFormation template located at **https://s3.amazonaws.com/awsinaction-code3/chapter08/ebs.yaml**. Create a stack based on that template by clicking the CloudFormation Quick-Create Link (**http://mng.bz/O6la**), select the default VPC and a random subnet, and set the `AttachVolume` parameter to `yes`. Don't forget to check the box marked "I Acknowledge That AWS CloudFormation Might Create IAM Resources." After creating the stack, use the SSM Session Manager to connect to the instance.

You can see the attached EBS volumes using `lsblk`. Usually, EBS volumes can be found somewhere in the range of /dev/xvdf to /dev/xvdp. The root volume (/dev/xvda) is an exception—it's based on the AMI you choose when you launch the EC2 instance and contains everything needed to boot the instance (your OS files), as shown here:

```
$ lsblk
NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   8G  0 disk             ①
└─xvda1 202:1    0   8G  0 part /
xvdf    202:80   0   5G  0 disk             ②
```

① The root volume, an EBS volume with a size of 8 GiB

② An additional volume, an EBS volume with a size of 5 GiB

The first time you use a newly created EBS volume, you must create a filesystem. You could also create partitions, but in this case, the volume size is only 5 GB, so you probably don't want to split it up further. Because you can create EBS volumes in any size and attach multiple volumes to your EC2 instance, partitioning a single EBS volume is uncommon. Instead, you should create volumes at the size you need (1 GB to 16 TB); if you need two separate scopes, create two volumes. In Linux, you can create a filesystem on the additional volume with the help of `mkfs`. The following example creates an XFS filesystem:

```
$ sudo mkfs -t xfs /dev/xvdf
meta-data=/dev/xvdf              isize=512    agcount=4, agsize=327680
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=0
data     =                       bsize=4096   blocks=1310720, imaxpct=2
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
```

```
          =                       sectsz=512   sunit=0 blks, lazy-count=
realtime =none                    extsz=4096   blocks=0, rtextents=0
```

After the filesystem has been created, you can mount the device as follows:

```
$ sudo mkdir /data
$ sudo mount /dev/xvdf /data
```

To see mounted volumes, use `df` like this:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        484M     0  484M   0% /dev
tmpfs           492M     0  492M   0% /dev/shm
tmpfs           492M  348K  491M   1% /run
tmpfs           492M     0  492M   0% /sys/fs/cgroup
/dev/xvda1      8.0G  1.5G  6.6G  19% /                ①
/dev/xvdf       5.0G   38M  5.0G   1% /data            ②
```

① Root volume

② Additional volume

EBS volumes are independent of your virtual machine. To see this in action, as shown in the next code snippet, you will save a file to a volume, unmount, and detach the volume. Afterward, you will attach and mount the volume again. The data will still be available!

```
$ sudo touch /data/testfile    ①
$ sudo umount /data            ②
```

① Creates testfile in /data

② Unmounts the volume

Open the AWS Management Console and update the CloudFormation stack named `ebs`. Keep the current template, but update the `AttachVolume` parameter from `yes` to `no` and update the stack. This will detach the EBS volume from the EC2 instance. After the update is completed, only your root device is left, as shown here:

```
$ lsblk
NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
```

```
xvda    202:0    0   8G  0 disk
└─xvda1 202:1    0   8G  0 part /
```

The testfile in /data is also gone, illustrated next:

```
$ ls /data/testfile
ls: cannot access /data/testfile: No such file or directory
```

Next, attach the EBS volume again. Open the AWS Management Console and update the CloudFormation stack named `ebs`. Keep the current template but set the `AttachVolume` parameter to `yes`. Then, update the CloudFormation stack and wait for the changes to be applied. The volume `/dev/xvdf` is available again, as shown here:

```
$ sudo mount /dev/xvdf /data    ①
$ ls /data/testfile             ②
/data/testfile
```

① Mounts the attached volume again

② Checks whether testfile is still in /data

Voilà! The file testfile that you created in /data is still there.

8.1.3 Tweaking performance

Performance testing of hard disks is divided into read and write tests. To test the performance of your volumes, you will use a simple tool named `dd`, which can perform block-level reads and writes between a source `if=/path/to/source` and a destination `of=/path/to/destination`, shown next. For comparison, you'll run a performance test for a temporary block-level storage volume in the following section:

```
$ sudo dd if=/dev/zero of=/data/tempfile bs=1M count=1024 \      ①
➥ conv=fdatasync,notrunc
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 15.8331 s, 67.8 MB/s          ②

$ echo 3 | sudo tee /proc/sys/vm/drop_caches                    ③
3
$ sudo dd if=/data/tempfile of=/dev/null bs=1M                  ④
➥ count=1024
1024+0 records in
```

```
1024+0 records out
1073741824 bytes (1.1 GB) copied, 15.7875 s, 68.0 MB/s ⑤
```

① Writes 1 MB 1,024 times

② 67.8 MB/s write performance

③ Flushes caches

④ Reads 1 MB 1,024 times

⑤ 68.0 MB/s read performance

Keep in mind that performance can be different depending on your actual workload. This example assumes that the file size is 1 MB. If you're hosting websites, you'll most likely deal with lots of small files instead.

EBS performance is more complicated. Performance depends on the type of EC2 instance as well as the EBS volume type. EC2 instances with EBS optimization benefit by having dedicated bandwidth to their EBS volumes. Table 8.1 gives an overview of EC2 instance types that are EBS-optimized by default. Some older instance types can be optimized for an additional hourly charge whereas others do not support EBS optimization at all. Input/output operations per second (IOPS) are measured using a standard 16 KB I/O operation size.

Table 8.1 What performance can be expected from modern instance types? Your mileage may vary.

Use case	Instance type	Baseline bandwidth (Mbps)	Maximum bandwidth (Mbps)
General purpose	m6a.large–m6a.48xlarge	531–40,000	6,666–40,000
Compute optimized	c6g.medium–c6g.16xlarge	315– 9,000	4,750–19,000
Memory optimized	r6i.large–r6i.32xlarge	650–40,000	10,000–40,000
Memory and network optimized	x2idn.16xlarge–x2idn.32xlarge	40,000–80,000	40,000–80,000

WARNING Performance depends heavily on your workload: read versus write, as well as the size of your I/O operations (a bigger operation size equates to more throughput).

Depending on your storage workload, you must choose an EC2 instance that can deliver the bandwidth you require. Additionally, your EBS volume must be balanced with the amount of bandwidth. Table 8.2 shows the different EBS volume types and how they perform.

Table 8.2 How EBS volume types differ

Volume type	Volume size	MiB/s	IOPS	Performance burst	Price
General Purpose SSD (gp3)	1 GiB–16 TiB	1,000	3,000 per default, plus as much as you provision (up to 500 IOPS per GiB or 16,000 IOPS)	n/a	$$$$
General Purpose SSD (gp2)	1 GiB–16 TiB	250	3 per GiB (up to 16,000)	3,000 IOPS	$$$$$
Provisioned IOPS SSD (io2 Block Express)	4 GiB–64 TiB	4000	As much as you provision (up to 500 IOPS per GiB or 256,000 IOPS)	n/a	$$$$$$
Provisioned IOPS SSD (io2)	4 GiB–16 TiB	1000	As much as you provision (up to 500 IOPS per GiB or 64,000 IOPS)	n/a	$$$$$$
Provisioned IOPS SSD (io1)	4 GiB–	1000	As much as you provision	n/a	$$$$$$$

	16 TiB		(up to 50 IOPS per GiB or 64,000 IOPS)		
Throughput Optimized HDD (st1)	125 GiB–16 TiB	40 per TiB (up to 500)	500	250 MiB/s per TiB (up to 500 MiB/s)	$$
Cold HDD (sc1)	125 GiB–16 TiB	12 per TiB (up to 250)	250	80 MiB/s per TiB (up to 250 MiB/s)	$
EBS Magnetic HDD (standard)	1 GiB–1 TiB	40-90	40-200 (100 on average)	Hundreds	$$$

Here are typical scenarios for the different volume types:

- Use General Purpose SSD (gp3) as the default for most workloads with medium load and a random access pattern. For example, use this as the boot volume or for all kinds of applications with low to medium I/O load.
- I/O-intensive workloads access small amounts of data randomly. Provisioned IOPS SSD (io2) offers throughput guarantees, for example, for large and business-critical database workloads.
- Use Throughput Optimized HDD (st1) for workloads with sequential I/O and huge amounts of data, such as Big Data workloads. Don't use this volume type for workloads in need of small and random I/O.
- Cold HDD (sc1) is a good fit when you are looking for a low-cost storage option for data you need to access infrequently and sequentially. Don't use this volume type for workloads in need of small and random I/O.
- EBS Magnetic HDD (standard) is an older volume type from a previous generation. It might be a good option when you need to access your data infrequently.

Gibibyte (GiB) and Tebibyte (TiB)

The terms gibibyte (GiB) and tebibyte (TiB) aren't used often; you're probably more familiar with gigabyte and terabyte. But AWS uses them in some places. Here's what they mean:

1 TiB = 240 bytes = 1,099,511,627,776 bytes	1 TB = 1012 bytes = 1,000,000,000,000 bytes
1 GiB = 230 bytes = 1,073,741,824 bytes	1 GB = 109 bytes = 1,000,000,000 bytes

Or, in other words, 1 TiB is 1.0995 TB and 1 GiB is 1.074 GB.

EBS volumes are charged based on the size of the volume, no matter how much data you store in the volume. If you provision a 100 GiB volume, you pay for 100 GiB, even if you have no data on the volume. If you use EBS Magnetic HDD (standard) volumes, you must also pay for every I/O operation you perform. Provisioned IOPS SSD (io1) volumes are additionally charged based on the provisioned IOPS. Use the AWS Simple Monthly Calculator at **https://calculator.aws/** to determine how much your storage setup will cost.

We advise you to use general-purpose (SSD) volumes as the default. If your workload requires more IOPS, then go with provisioned IOPS (SSD). You can attach multiple EBS volumes to a single EC2 instance to increase overall capacity or for additional performance.

8.1.4 Backing up your data with EBS snapshots

EBS volumes replicate data on multiple disks automatically and are designed for an annual failure rate (AFR) of 0.1% and 0.2%. This means on average you should expect to lose 0.5–1 of 500 volumes per year. To plan for an unlikely (but possible) failure of an EBS volume, or more likely a human failure, you should create backups of your volumes regularly. Fortunately, EBS offers an optimized, easy-to-use way to back up EBS volumes with EBS snapshots. A *snapshot* is a block-level incremental backup. If your volume is 5 GiB in size, and you use 1 GiB of data, your first snapshot will be around 1 GiB in size. After the first snapshot is created, only the changes will be persisted, to reduce the size of the backup. EBS snapshots are charged based on how many gigabytes you use.

You'll now create a snapshot using the CLI. Before you can do so, you need to know the EBS volume ID. You can find it as the `VolumeId` output

of the CloudFormation stack, or by running the following:

```
$ aws ec2 describe-volumes \
> --filters "Name=size,Values=5" --query "Volumes[].VolumeId" \
> --output text
vol-043a5516bc104d9c6      ①
```

① The output shows the $VolumeId.

With the volume ID, you can go on to create a snapshot like this:

```
$ aws ec2 create-snapshot --volume-id $VolumeId     ①
{
  "Description": "",
  "Encrypted": false,
  "OwnerId": "163732473262",
  "Progress": "",
  "SnapshotId": "snap-0babfe807decdb918",           ②
  "StartTime": "2022-08-25T07:59:50.717000+00:00",
  "State": "pending",                               ③
  "VolumeId": "vol-043a5516bc104d9c6",
  "VolumeSize": 5,
  "Tags": []
}
```

① Replace $VolumeId with the ID of your volume.

② Note the ID of the snapshot: $SnapshotId.

③ EBS is still creating your snapshot.

Creating a snapshot can take some time, depending on how big your volume is and how many blocks have changed since the last backup. You can see the status of the snapshot by running the following:

```
$ aws ec2 describe-snapshots --snapshot-ids $SnapshotId  ①
{
  "Snapshots": [
    {
      "Description": "",
      "Encrypted": false,
      "OwnerId": "163732473262",
      "Progress": "100%",                               ②
      "SnapshotId": "snap-0babfe807decdb918",
      "StartTime": "2022-08-25T07:59:50.717000+00:00",
      "State": "completed",                             ③
      "VolumeId": "vol-043a5516bc104d9c6",
```

```
        "VolumeSize": 5,
        "StorageTier": "standard"
      }
    ]
}
```

① Replace $VolumeId with the ID of your snapshot.

② Progress of your snapshot

③ The snapshot has reached the state completed.

Creating a snapshot of an attached, mounted volume is possible but can cause problems with writes that aren't flushed to disk. You should either detach the volume from your instance or stop the instance first. If you absolutely must create a snapshot while the volume is in use, you can do so safely as follows:

1. Freeze all writes by running `sudo fsfreeze -f /data` on the virtual machine.
2. Create a snapshot and wait until it reaches the pending state.
3. Unfreeze to resume writes by running `sudo fsfreeze -u /data` on the virtual machine.
4. Wait until the snapshot is completed.

Unfreeze the volume as soon as the snapshot reaches the state `pending`. You don't have to wait until the snapshot has finished.

With an EBS snapshot, you don't have to worry about losing data due to a failed EBS volume or human failure. You are able to restore your data from your EBS snapshot.

To restore a snapshot, you must create a new EBS volume based on that snapshot. Execute the following command in your terminal, replacing `$SnapshotId` with the ID of your snapshot:

```
$ aws ec2 create-volume --snapshot-id $SnapshotId \     ①
- --availability-zone us-east-1a                        ②
{
    "AvailabilityZone": "us-east-1a",
    "CreateTime": "2022-08-25T08:08:49+00:00",
    "Encrypted": false,
    "Size": 5,
    "SnapshotId": "snap-0babfe807decdb918",
    "State": "creating",
    "VolumeId": "vol-0bf4fdf3816f968c5",                ③
    "Iops": 100,
```

```
    "Tags": [],
    "VolumeType": "gp2",
    "MultiAttachEnabled": false
}
```

① The ID of your snapshot used to create the volume

② Choose data center.

③ The $RestoreVolumeId of the volume restored from your snapshot

Cleaning up

Don't forget to delete the snapshot, the volume, and your stack. The following code will delete the snapshot and volume. Don't forget to replace `$SnapshotId` with the ID of the EBS snapshot you created earlier and `$RestoreVolumeId` with the ID of the EBS volume you created by restoring the snapshot:

```
$ aws ec2 delete-snapshot --snapshot-id $SnapshotId
$ aws ec2 delete-volume --volume-id $RestoreVolumeId
```

Also delete your CloudFormation stack named `ebs` after you finish this section to clean up all used resources as follows:

```
$ aws cloudformation delete-stack --stack-name ebs
```

8.2 Instance store: Temporary block-level storage

An *instance store* provides block-level storage directly attached to the physical machine hosting the virtual machine. Figure 8.2 shows that the instance store is part of an EC2 instance and available only if your instance is running; it won't persist your data if you stop or terminate the instance. You don't pay separately for an instance store; instance store charges are included in the EC2 instance price.

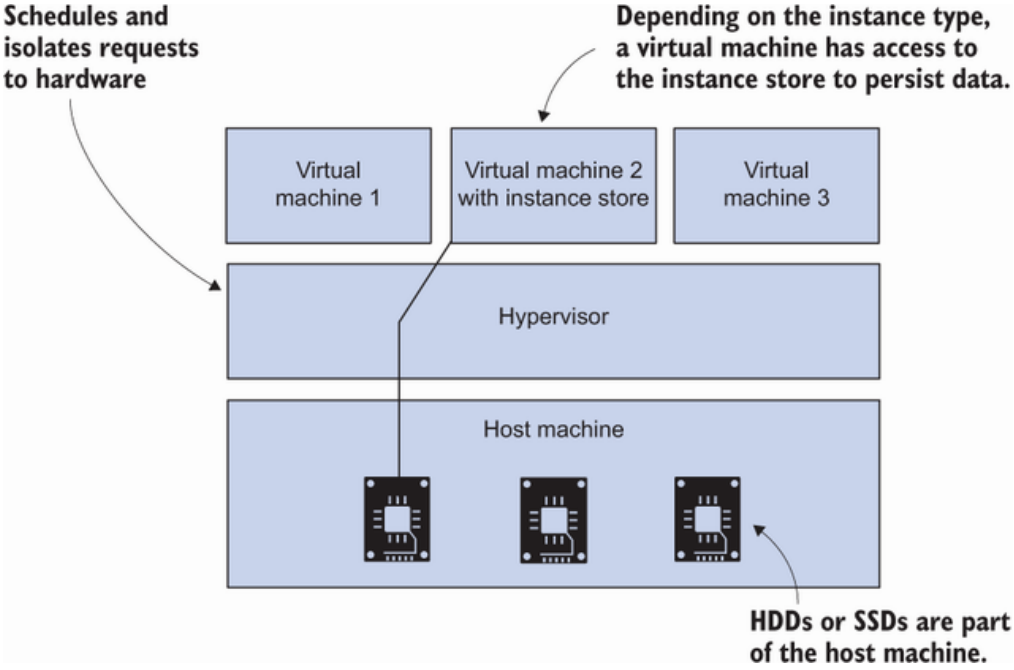

Figure 8.2 The instance store is part of your EC2 instance and uses the host machine's HDDs or SSDs.

In comparison to an EBS volume, which is connected to your EC2 instance over the network, the instance store depends upon the EC2 instance and can't exist without it. The instance store will be deleted when you stop or terminate the virtual machine.

Don't use an instance store for data that must not be lost; use it for temporary data only. For example, it is not a big deal to cache data on instance store, because you are able to restore the data from its origin anytime.

In rare cases, it is also advisable to persist data on instance storage, such as whenever the application is replicating data among multiple machines by default. For example, many NoSQL database systems use a cluster of machines to replicate data. The benefit of doing so is low latency and high throughput. But be warned, this is for experts in the field of distributed systems and storage only. If in doubt, use EBS instead.

WARNING If you stop or terminate your EC2 instance, the instance store is lost. *Lost* means all data is destroyed and can't be restored!

Note that most EC2 instance types do not come with instance storage. Only some instance families come with instance storage included. AWS offers SSD and HDD instance stores from 4 GB up to 335,520 GB. Table 8.3 shows a few EC2 instance families providing instance stores.

Table 8.3 Instance families with instance stores

Use case	Instance type	Instance store type	Instance store size in GB
General purpose	m6id.large m6id.32xlarge	SSD	118 7600
Compute optimized	c6id.large c6id.32xlarge	SSD	118 7600
Memory optimized	r6id.large r6id.32xlarge	SSD	118 7600
Storage optimized	i4i.large i4i.32xlarge	SSD	468 30000
Storage optimized	d3.xlarge d3.8xlarge	HDD	5940 47520
Storage optimized	d3en.xlarge d3en.12xlarge	HDD	27960 335520

The following listing demonstrates how to use an instance store with CloudFormation. Instance stores aren't standalone resources like EBS volumes; the instance store is part of your EC2 instance.

Listing 8.1 Using an instance store with CloudFormation

```
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    IamInstanceProfile: 'ec2-ssm-core'
    ImageId: 'ami-061ac2e015473fbe2'
    InstanceType: 'm6id.large'        ①
    SecurityGroupIds:
    - !Ref SecurityGroup
    SubnetId: !Ref Subnet
```

① Chooses an instance type with an instance store

Read on to see how you can use the instance store.

8.2.1 Using an instance store

To help you explore instance stores, we created the CloudFormation template located at **https://s3.amazonaws.com/awsinaction-code3/chapter08/instancestore.yaml**. Create a CloudFormation stack based on the template by clicking the CloudFormation Quick-Create Link (**http://mng.bz/YK5a**).

WARNING Starting a virtual machine with instance type `m6id.large` will incur charges. See **https://aws.amazon.com/ec2/pricing/on-demand/** if you want to find out the current hourly price.

Create a stack based on that template, and select the default VPC and a random subnet. After creating the stack, use the SSM Session Manager to connect to the instance and explore the available devices, as shown in the following code snippet.

WARNING You might run into the following error: "Your requested instance type (`m6id.large`) is not supported in your requested Availability Zone." Not all regions/availability zones support `m6id.large` instance types. Select a different subnet and try again.

```
$ lsblk
NAME          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
nvme1n1       259:0    0 109.9G  0 disk    ①
nvme0n1       259:1    0     8G  0 disk    ②
├─nvme0n1p1   259:2    0     8G  0 part /
└─nvme0n1p128 259:3    0     1M  0 part
```

① The instance store device

② The EBS root volume device

To see the mounted volumes, use this command:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           3.9G  348K  3.9G   1% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/nvme0n1p1  8.0G  1.5G  6.6G  19% /      ①
```

① The EBS root volume contains the OS.

Your instance store volume is not yet usable. You have to format the device and mount it as follows:

```
$ sudo mkfs -t xfs /dev/nvme1n1     ①
$ sudo mkdir /data                  ②
$ sudo mount /dev/nvme1n1 /data     ③
```

① Formats by creating an XFS filesystem

② Creates a folder on which to mount the device

③ Mounts the device

You can now use the instance store by reading and writing to `/data` like this:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
[...]
/dev/nvme1n1    110G  145M  110G   1% /data      ①
```

① 110 GB are available.

Windows

For Windows instances, instance store volumes are NTFS formatted and mounted automatically.

8.2.2 Testing performance

Let's take the same performance measurements we took in section 8.1.3 to see the difference between the instance store and EBS volumes, as shown next:

```
$ sudo dd if=/dev/zero of=/data/tempfile bs=1M count=1024 \
➥ conv=fdatasync,notrunc
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 13.3137 s, 80.6 MB/s     ①
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3

$ sudo dd if=/data/tempfile of=/dev/null bs=1M count=1024
1024+0 records in
1024+0 records out
```

```
1073741824 bytes (1.1 GB) copied, 5.83715 s, 184 MB/s
sh-4.2$ echo 3 | sudo tee /proc/sys/vm/drop_caches
```
②

① 18% faster compared with EBS from section 8.1.3

② 170% faster compared with EBS from section 8.1.3

Keep in mind that performance can vary, depending on your actual workload. This example assumes a file size of 1 MB. If you're hosting websites, you'll most likely deal with lots of small files instead. The performance characteristics show that the instance store is running on the same hardware as the virtual machine. The volumes are not connected to the virtual machine over the network as with EBS volumes.

Cleaning up
Don't forget to delete your stacks after you finish this section to clean up all used resources. Otherwise, you'll be charged for the resources you use.

8.2.3 Backing up your data

There is no built-in backup mechanism for instance store volumes. Based on what you learned in section 7.3, you can use a combination of scheduled jobs and S3 to back up your data periodically, as shown here:

```
$ aws s3 sync /path/to/data s3://$YourCompany-backup/instancestore-back
```

If you need to back up data, you should probably use more durable, block-level storage like EBS. An instance store is better used for ephemeral persistence requirements.

You will learn about another option to store your data in the next chapter: a network filesystem.

Summary

- Block-level storage can be used only in combination with an EC2 instance because the OS is needed to provide access to the block-level storage (including partitions, filesystems, and read/write system calls).
- When creating an EBS volume, you need to specify the volume size. AWS charges you for the provisioned storage, no matter whether or not you are using all the storage. Also, it is possible to increase the size of a volume later.

- EBS volumes are connected to a single EC2 instance via network. Depending on your instance type, this network connection can use more or less bandwidth.
- There are different types of EBS volumes available: General Purpose SSD (gp3), Provisioned IOPS SSD (io2), Throughput Optimized HDD (st1), and Cold HDD (sc1) are the most common.
- EBS snapshots are a powerful way to back up your EBS volumes because they use a block-level, incremental approach.
- An instance store provides low latency and high throughput. However, as you are using storage directly attached to the physical machine running your virtual machine, data is lost if you stop or terminate the instance.
- Typically, an instance store is used only for temporary data that does not need to be persisted.