9 Sharing data volumes between machines: EFS

This chapter covers

- Creating a highly available network filesystem
- Mounting a network filesystem on multiple EC2 instances
- Sharing files between EC2 instances
- Tweaking the performance of your network filesystem
- Monitoring possible bottlenecks in your network filesystem
- Backing up your shared filesystem

Many legacy applications store state in a filesystem on disk. Therefore, using Amazon S3—an object store, described in chapter 7—isn't possible without modifying the application. Using block storage as discussed in the previous chapter might be an option, but it doesn't allow you to access files from multiple machines. Because block storage persists data in a single data center only, AWS promises an uptime of only 99.9%.

If you need to share a filesystem between virtual machines or require high availability, the Elastic File System (EFS) might be an option. EFS is based on the NFSv4.1 protocol, which allows you to mount and access the filesystem on one or multiple machines in parallel. EFS distributes data among multiple data centers, called availability zones, and promises an uptime of 99.99%. In this chapter, you learn how to set up EFS, tweak the performance, and back up your data.

EFS WORKS ONLY WITH LINUX At this time, EFS isn't supported by Windows EC2 instances. The Amazon FSx for Windows File Server is an alternative to EFS for Windows workloads. See http://mng.bz/zmq1 to learn more.

Let's take a closer look at how EFS works compared to Elastic Block Store (EBS) and the instance store, introduced in chapter 8. An EBS volume is tied to a data center and can be attached to only a single EC2 instance from the same data center. Typically, EBS volumes are used as the root volumes that contain the operating system, or, for relational database systems, to store the state. An instance store consists of a hard drive directly attached to the hardware the virtual machine is running on. An instance store can be regarded as ephemeral storage and is, therefore, used only for temporary data, caching, or for systems with embedded data replication, like many NoSQL databases, for example. In contrast, the EFS filesystem supports reads and writes by multiple EC2 instances from different data centers in parallel. In addition, the data on the EFS filesystem is replicated among multiple data centers and remains available even if a whole data center suffers an outage, which isn't true for EBS and instance stores. Figure 9.1 shows the differences: EBS is a virtual disk, instance store is a local disk, and EFS is a shared folder.

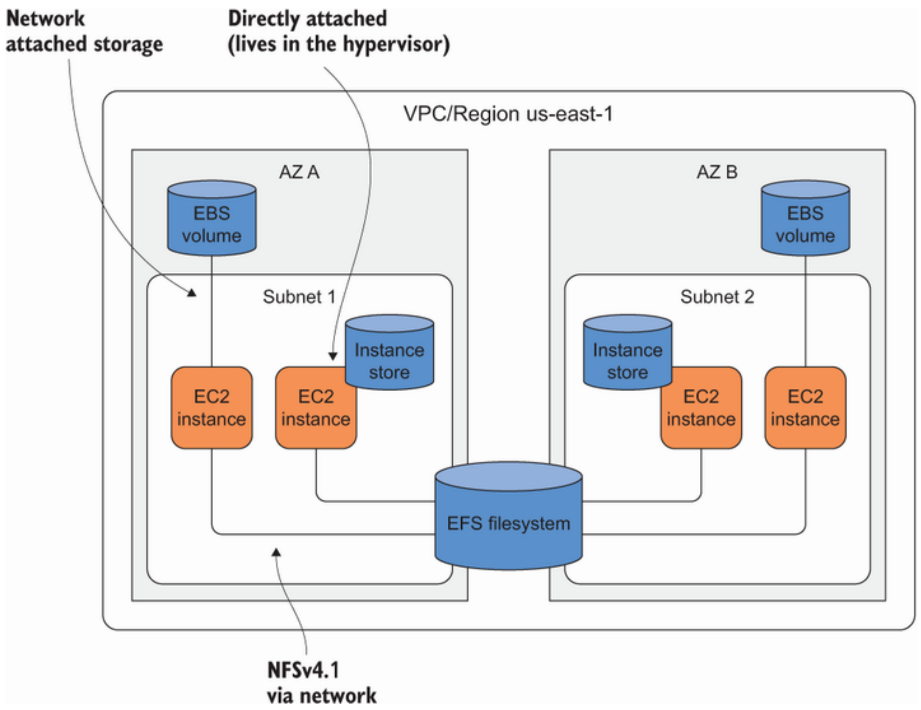

Figure 9.1 Comparing EBS, instance stores, and EFS

All examples are covered by the Free Tier

All examples in this chapter are covered by the Free Tier. As long as you follow the instructions and don't run them longer than a few days, you won't pay anything. You will find instructions to delete all resources at the end of the chapter.

EFS consists of two components:

- *Filesystem*—Stores your data
- *Mount target*—Makes your data accessible

The filesystem is the resource that stores your data in an AWS region, but you can't access it directly. To do so, you must create an EFS mount target in a subnet. The mount target provides a network endpoint that you can use to mount the filesystem on an EC2 instance via NFSv4.1. The EC2 instance must be in the same subnet as the EFS mount target, but you can create mount targets in multiple subnets. Figure 9.2 demonstrates how to access the filesystem from EC2 instances running in multiple subnets.

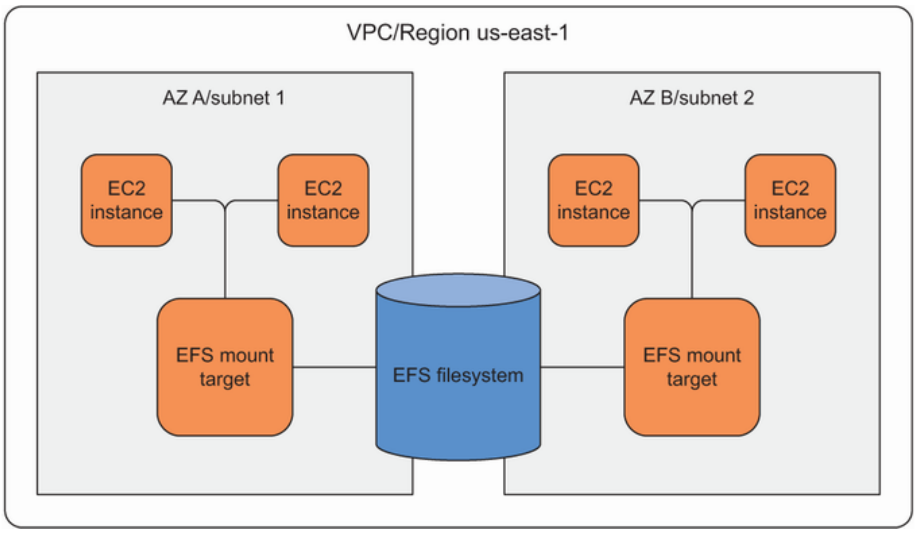

Figure 9.2 Mount targets provide an endpoint for EC2 instances to mount the filesystem in a subnet.

Equipped with the knowledge about filesystems and mount targets, we will now continue to practice.

Linux is a multiuser operating system. Many users can store data and run programs isolated from each other. Each user has a home directory, which is usually stored under /home/$username. For example, the user michael owns the home directory /home/michael. Of course, only user michael is allowed to read and write in /home/michael. The `ls -d -l /home/*` command lists all home directories, as shown next:

```
$  ls -d -l /home/*                          ①
drwx------ 2 andreas    andreas    4096
➥ Jul 24 06:25 /home/andreas                ②
drwx------ 3 michael    michael    4096
➥ Jul 24 06:38 /home/michael                ③
```

① Lists all home directories with absolute paths

② /home/andreas is accessible by the user and group andreas only.

③ /home/michael can only be accessed by user and group michael.

When using multiple EC2 instances, your users will have a separate home folder on each EC2 instance. If a Linux user uploads a file on one EC2 instance, they can't access the file on another EC2 instance. To solve this problem, create a filesystem and mount EFS on each EC2 instance under /home. The home directories are then shared across all your EC2 instances, and users will feel at home no matter which machine they log in to. In the following sections, you will build this solution step-by-step. First, you will create the filesystem.

9.1 Creating a filesystem

The filesystem is the resource that stores your files, directories, and links. Like S3, EFS grows with your storage needs. You don't have to provision the storage up front. The filesystem is located in an AWS region and replicates your data under the covers across multiple data centers. You will use CloudFormation to set up the filesystem next.

9.1.1 Using CloudFormation to describe a filesystem

First, configure the filesystem. The next listing shows the CloudFormation resource.

Listing 9.1 CloudFormation snippet of an EFS filesystem resource

```
Resources:                                  ①
  [...]
  FileSystem:
    Type: 'AWS::EFS::FileSystem'
    Properties:
      Encrypted: true                       ②
      ThroughputMode: bursting              ③
      PerformanceMode: generalPurpose       ④
      FileSystemPolicy:                     ⑤
        Version: '2012-10-17'
        Statement:
        - Effect: 'Deny'
          Action: '*'
          Principal:
            AWS: '*'
          Condition:
            Bool:
              'aws:SecureTransport': 'false'
```

① Specifies the stack resources and their properties

② We recommend to enable encryption at rest by default.

③ The default throughput mode is called bursting. We'll cover more on the throughput mode for I/O intensive workloads later.

④ The default performance mode is general purpose. We'll cover more on the performance mode later.

⑤ It's a security best practice to encrypt data in transit. The filesystem policy ensures that all access uses secure transport.

That's it. The filesystem is ready to store data. Before we do so, let's talk about the costs.

9.1.2 Pricing

Estimating costs for storing data on EFS is not that complicated. The following three factors affect the price:

- The amount of stored data in GB per month
- The frequency that the data is accessed
- Whether you want to trade in availability for cost

When accessing data frequently choose the Standard Storage or One Zone Storage storage class, which comes with the lowest latency. When accessing data less than daily, consider Standard–Infrequent Access Storage or One Zone–Infrequent Access Storage to reduce storage costs. Keep in mind that doing so increases the first-byte latency. Also, when using the Infrequent Access Storage classes, accessing the data comes with a fee.

If you do not need the high availability of 99.99% provided by the Standard Storage and Standard–Infrequent Access Storage, consider choosing the One Zone Storage or One Zone–Infrequent Access Storage storage classes. Those storage classes do not replicate your data among multiple data centers, which reduces the promised availability to 99.9%. It is worth mentioning that all storage classes are designed for a durability of 99.999999999%. However, you should back up your data stored with One Zone to be able to recover in the event of a data center destruction. You will learn more about backing up an EFS filesystem at the end of the chapter.

Table 9.1 shows EFS pricing when storing data in US East (N. Virginia), also called us-east-1.

Table 9.1 EFS storage classes affect the monthly costs for storing data.

Storage class	Price per GB/month	Access requests per GB transferred
Standard Storage	$0.30	$0.00
Standard–Infrequent Access Storage	$0.025	$0.01
One Zone Storage	$0.16	$0.00
One Zone–Infrequent Access Storage	$0.0133	$0.01

Let’s briefly estimate the costs for storing 5 GB of data on EFS. Assuming the data is accessed multiple times per day and high availability is required, we choose the Standard Storage class. So, the cost for storing data is 5 GB × $0.30, which results in a total of $1.50 a month.

Please note: the first 5 GB (Standard Storage) per month are free in the first year of your AWS account (Free Tier). For more details about EFS pricing go to https://aws.amazon.com/efs/pricing/.

After configuring an EFS filesystem with the help of CloudFormation and estimating costs, you will learn how to mount the NFS share on an EC2 instance. To do so, you need to create a mount target first.

9.2 Creating a mount target

An EFS mount target makes your data available to EC2 instances via the NFSv4.1 protocol in a subnet. The EC2 instance communicates with the mount target via a TCP/IP network connection. As you learned in section 6.4, you control network traffic on AWS using security groups. You can use a security group to allow inbound traffic to an EC2 instance or an RDS database, and the same is true for a mount target. Security groups control which traffic is allowed to enter the mount target. The NFS protocol uses port 2049 for inbound communication. Figure 9.3 shows how mount targets are protected.

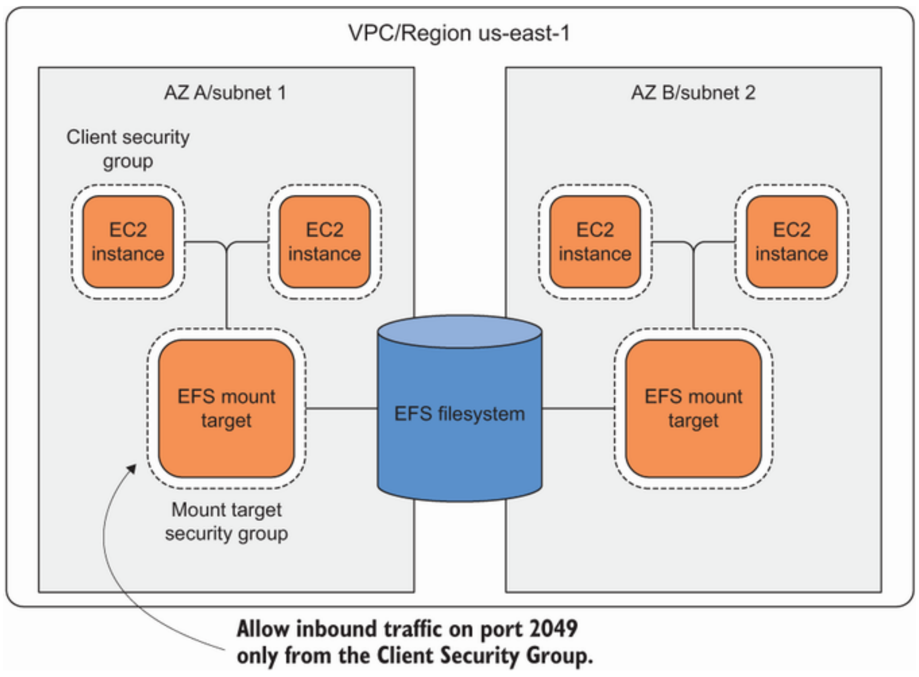

Figure 9.3 EFS mount targets are protected by security groups.

In our example, to control traffic as tightly as possible, you won't grant traffic based on IP addresses. Instead, you'll create two security groups. The client security group will be attached to all EC2 instances that want to mount the filesystem. The mount target security group allows inbound traffic on port 2049 only for traffic that comes from the client security group. This way, you can have a dynamic fleet of clients who are allowed to send traffic to the mount targets.

EFS IS NOT ONLY ACCESSIBLE FROM EC2 INSTANCES In this chapter, we are focusing on mounting an EFS filesystem on EC2 instances. In our experience, that's the most typical use case for EFS. However, EFS can also be used in the following other scenarios:

- Containers (ECS and EKS)
- Lambda functions
- On-premises servers

Next, use CloudFormation to manage an EFS mount target. The mount target references the filesystem, needs to be linked to a subnet, and is also protected by at least one security group. You will first describe the security groups, followed by the mount target, as shown in the following listing.

Listing 9.2 CloudFormation snippet of an EFS mount target and security groups

```
Resources:
  [...]
  EFSClientSecurityGroup:                                            ①
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'EFS Mount target client'
      VpcId: !Ref VPC
  MountTargetSecurityGroup:                                          ②
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'EFS Mount target'
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 2049                                               ③
        ToPort: 2049
        SourceSecurityGroupId: !Ref EFSClientSecurityGroup           ④
      VpcId: !Ref VPC
  MountTargetA:
    Type: 'AWS::EFS::MountTarget'
    Properties:
      FileSystemId: !Ref FileSystem                                  ⑤
      SecurityGroups:
      - !Ref MountTargetSecurityGroup                                ⑥
      SubnetId: !Ref SubnetA                                         ⑦
```

① This security group needs no rules. It's just used to mark outgoing traffic from EC2 instances.

② This security group is linked to the mount target.

③ Allows traffic on port 2049

④ Only allows traffic from the security group linked to the EC2 instances

⑤ Attaches the mount target to the filesystem

⑥ Assigns the security group

⑦ Links the mount target with subnet A

Copy the `MountTargetA` resource and also create a mount target for `SubnetB` as shown in listing 9.3.

Listing 9.3 CloudFormation snippet of an EFS mount target and security groups

```
Resources:
  [...]
  MountTargetB:
    Type: 'AWS::EFS::MountTarget'
    Properties:
      FileSystemId: !Ref FileSystem
      SecurityGroups:
      - !Ref MountTargetSecurityGroup
      SubnetId: !Ref SubnetB            ①
```

① Attaches the mount target to subnet B

Next, you will finally mount the /home directory on an EC2 instance.

9.3 Mounting the EFS filesystem on EC2 instances

EFS creates a DNS name for each filesystem following the schema `$FileSystemID .efs.$Region.amazonaws.com`. From an EC2 instance, this name resolves to the mount target of the instance's subnet.

We recommend using the EFS mount helper to mount EFS filesystems because the tool comes with two features: secure transport with TLS and authentication with IAM. Besides that, the EFS mount helper applies the recommended defaults for mounting EFS filesystems.

On Amazon Linux 2, which we are using for our examples, installing the EFS mount helper is quite simple, as shown here:

```
$ sudo yum install amazon-efs-utils
```

With the EFS mount util installed, the following command mounts an EFS filesystem. This snippet shows the full mount command:

```
$ sudo mount -t efs -o tls,iam $FileSystemID $EFSMountPoint
```

Replace `$FileSystemID` with the EFS filesystem, such as `fs-123456`, and `$EFSMountPoint` with the local path where you want to mount the filesystem. The following code snippet shows an example:

```
$ sudo mount -t efs -o tls,iam fs-123456 /home
```

1. `tls` initiates a TLS tunnel from the EC2 instance to the EFS filesystem to encrypt data in transit.
2. `iam` enables authentication via IAM using the IAM role attached to the EC2 instance.

Of course, it is also possible to use the /ets/fstab config file to automatically mount on startup. Again, you need to replace `$FileSystemID` and `$EFSMountPoint` as described in the previous example:

```
$FileSystemID:/ $EFSMountPoint efs _netdev,noresvport,tls,iam 0 0
```

You are already familiar with the options `tls` and `iam`. The two other options have the following meanings:

- `_netdev`—Identifies a network filesystem
- `noresvport`—Ensures that a new TCP source port is used when reestablishing a connection, which is required when recovering from network problems

It's now time to add two EC2 instances to the CloudFormation template. Each EC2 instance should be placed in a different subnet and mount the filesystem to /home. The /home directory will exist on both EC2 instances, and it will also contain some data (such as the folder ec2-user). You have to ensure that you're copying the original data the first time before you mount the EFS filesystem, which is empty by default. The following listing describes the EC2 instance that copies the existing /home folder before the shared home folder is mounted.

Listing 9.4 Using CloudFormation to launch an EC2 instance and mount an EFS filesystem

```
Resources:
  [...]
  EC2InstanceA:                                                            ①
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref 'AWS::Region', AMI]
      InstanceType: 't2.micro'
      IamInstanceProfile: !Ref IamInstanceProfile                          ②
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref EFSClientSecurityGroup                                      ③
        SubnetId: !Ref SubnetA                                             ④
      UserData:                                                            ⑤
        'Fn::Base64': !Sub |
          #!/bin/bash -ex
          trap '/opt/aws/bin/cfn-signal -e 1 --stack ${AWS::StackName}
➥ --resource EC2InstanceA --region ${AWS::Region}' ERR

          # install dependencies
          yum install -y nc amazon-efs-utils
          pip3 install botocore

          # copy existing /home to /oldhome
          mkdir /oldhome
          cp -a /home/. /oldhome                                            ⑥

          # wait for EFS mount target
          while ! (echo > /dev/tcp/${FileSystem}.efs.${AWS::Region}.
➥ amazonaws.com/2049) >/dev/null 2>&1; do sleep 5; done                   ⑦

          # mount EFS filesystem
          echo "${FileSystem}:/ /home efs _netdev,noresvport,tls,iam 0 0"
➥ >> /etc/fstab                                                           ⑧
          mount -a                                                          ⑨

          # copy /oldhome to new /home
          cp -a /oldhome/. /home                                            ⑩

          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName}
➥ --resource EC2InstanceA --region ${AWS::Region}                         ⑪
      Tags:
      - Key: Name
        Value: 'efs-a'
    CreationPolicy:                                                         ⑫
      ResourceSignal:
        Timeout: PT10M
    DependsOn:                                                              ⑬
    - VPCGatewayAttachment
    - MountTargetA
```

① Creates an EC2 instance

② The IAM role grants access to the EFS filesystem (see the next listing).

③ Attaches the security group, which is used to identify outgoing traffic to the filesystem

④ Places the EC2 instance into subnet A

⑤ Adds a Bash script to the user data of the virtual machine, which is executed automatically at the end of the first boot process

⑥ Backs up all home directories to /oldhome

⑦ After creating a new filesystem, it takes a few minutes until its DNS name resolves to the mount targets.

⑧ Adds an entry to fstab, which makes sure the filesystem is mounted automatically on each boot

⑨ Mounts all entries—most importantly the EFS filesystem—defined in fstab without the need of rebooting the system

⑩ Copies the old home directories to the EFS filesystem mounted under /home

⑪ Sends a success signal to CloudFormation

⑫ Tells CloudFormation to wait until it receives a success signal from the EC2 instance

⑬ The EC2 instance requires internet connectivity as well as a mount target. Because both dependencies are not apparent to CloudFormation, we add them manually here.

To prove that the EFS filesystem allows you to share files across multiple instances, we are adding a second EC2 instance, as shown next.

Listing 9.5 Mounting an EFS filesystem from a second EC2 instance

```
Resources:
  [...]
  EC2InstanceB:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref 'AWS::Region', AMI]
      InstanceType: 't2.micro'
      IamInstanceProfile: !Ref IamInstanceProfile
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref EFSClientSecurityGroup
        SubnetId: !Ref SubnetB                    ①
      UserData:
        'Fn::Base64': !Sub |
          #!/bin/bash -ex
          trap '/opt/aws/bin/cfn-signal -e 1 --stack ${AWS::StackName}
➥ --resource EC2InstanceB --region ${AWS::Region}' ERR

          # install dependencies
          yum install -y nc amazon-efs-utils
          pip3 install botocore                   ②

          # wait for EFS mount target
          while ! (echo > /dev/tcp/${FileSystem}.efs.${AWS::Region}
➥ .amazonaws.com/2049) >/dev/null 2>&1; do sleep 5; done

          # mount EFS filesystem
          echo "${FileSystem}:/ /home efs _netdev,noresvport,tls,iam 0 0"
```

```
➥ >> /etc/fstab
          mount -a

          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName}
➥ --resource EC2InstanceB --region ${AWS::Region}
      Tags:
      - Key: Name
        Value: 'efs-b'
    CreationPolicy:
      ResourceSignal:
        Timeout: PT10M
    DependsOn:
    - VPCGatewayAttachment
    - MountTargetB
```

① Places the EC2 instance into subnet B

② The old /home folder is not copied here. This is already done on the first EC2 instance in subnet A.

To make things easier, you can also add outputs to the template to expose the IDs of your EC2 instances like this:

```
Outputs:
  EC2InstanceA:
    Value: !Ref EC2InstanceA
    Description: 'Id of EC2 Instance in AZ A (connect via Session Manager)'
  EC2InstanceB:
    Value: !Ref EC2InstanceB
    Description: 'Id of EC2 Instance in AZ B (connect via Session Manager)'
```

The CloudFormation template is now complete. It contains the following:

- The EFS filesystem
- Two mount targets in subnet A and subnet B
- Security groups to control traffic from the EC2 instances to the mount targets
- EC2 instances in both subnets, including a `UserData` script to mount the filesystem

Where is the template located?

You can find the template on GitHub. Download a snapshot of the repository at **https://github.com/AWSinAction/code3/archive/main.zip**. The file we're talking about is located at chapter09/efs.yaml. On S3, the same file is located at **https://s3.amazonaws.com/awsinaction-code3/chapter09/efs.yaml**.

It's now time to create a stack based on your template to create all the resources in your AWS account. Use the AWS CLI to create the stack like this:

```
$ aws cloudformation create-stack --stack-name efs \
➥ --template-url https:/ /s3.amazonaws.com/awsinaction-code3/\
➥ chapter09/efs.yaml --capabilities CAPABILITY_IAM
```

Once the stack is in the state `CREATE_COMPLETE`, two EC2 instances are running. Both mounted the EFS share to /home. You also copied the existing home directories to the EFS share. It's time to connect to the instances via the Session Manager and do some tests to see whether users can really share files between the EC2 instances in their home directory.

9.4 Sharing files between EC2 instances

Use the Session Manager to connect to the virtual machine named `EC2InstanceA`. Use the following command to get the EC2 instance IDs of `EC2InstanceA` and `EC2InstanceB`:

```
$ aws cloudformation describe-stacks --stack-name efs \
➥ --query "Stacks[0].Outputs"
[{
  "Description": "[...]",
  "OutputKey": "EC2InstanceA",
  "OutputValue": "i-011a050b697d12e7a"
}, {
  "Description": "[...]",
  "OutputKey": "EC2InstanceB",
  "OutputValue": "i-a22b67b2a4d25a2b"
}]
```

Then, establish a second connection to the virtual machine `EC2InstanceB` with the help of the Session Manager and switch to the user's home directory, as shown in the next code snippet. By the way, the default user on Amazon Linux 2 is the ec2-user. But when using the Session Manager via the AWS Management Console, you are logged in with another user named ssm-user:

```
$ cd $HOME          ①
```

① Changes to the home directory of the current user

Also check whether there are any files or folders in your home directory as follows:

```
$ ls                ①
```

① If no data is returned, the folder /home/ec2-user is empty.

Now, create a file on one of the machines like this:

```
$ touch i-was-here ①
```

① The touch command creates an empty file named i-was-here.

On the other machine, confirm that you can see the new file as follows:

```
$ cd $HOME
$ ls
i-was-here          ①
```

① The file created on the other machine appears here.

This simple experiment proves that you have access to the same home directory on both machines. You could add hundreds of machines to this example. All would share the same home directory, and users would be able to access the same home directory on all EC2 instances. You can apply the same mechanism to share files between a fleet of web servers, for example, mounting the folder /var/www/html from an EFS filesystem. A similar example is building a highly available Jenkins server by putting /var/lib/jenkins on EFS.

To run the solution successfully, you also need to take care of backups, performance tuning, and monitoring. You will learn about this in the following sections.

9.5 Tweaking performance

EFS makes it easy for us to mount a network filesystem from many machines. However, we can say from our own experience that we repeatedly have problems with the performance of EFS in everyday practice. Therefore, in the following section, we describe what you should consider to get the maximum performance out of EFS.

The following factors affect latency, throughput, and I/O operations per second of an EFS filesystem:

- *The performance mode*—General Purpose or Max I/O
- *The throughput mode*—Bursting or Provisioned
- *The storage class*—Standard or One Zone

Let’s dive into the details.

9.5.1 Performance mode

EFS comes with two performance modes:

- *General Purpose mode*—Supports up to 35,000 IOPS
- *Max I/O mode*—Supports 500,000+ IOPS

IOPS stands for read or write operations per second. When reading or writing a file, EFS accounts for one I/O operation for every 4 KB with a minimum of one I/O operation.

So, when to choose which option? So far, you've used the General Purpose performance mode, which is fine for most workloads, especially latency-sensitive ones where small files are served most of the time. The /home directory is a perfect example of such a workload. Typical files like documents are relatively small, and users expect low latency when fetching files.

But sometimes, EFS is used to store massive amounts of data for analytics. For data analytics, latency is not important. Throughput is the metric you want to optimize instead. If you want to analyze gigabytes or terabytes of data, it doesn't matter if your time to first byte takes 1 ms or 100 ms. Even a small increase in throughput will decrease the time it will take to analyze the data. For example, analyzing 1 TB of data with 100 MB/second throughput will take 167 minutes. That's almost three hours, so the first few milliseconds don't really matter. Optimizing for throughput can be achieved using the Max I/O performance mode.

Please note: the performance mode being used by an EFS filesystem cannot be changed—you set it when the filesystem is created. Therefore, to change the performance mode, you have to create a new filesystem. We recommend you start with the General Purpose performance mode if you are unsure which mode fits best for your workload. You will learn how to check whether you made the right decision by monitoring data, which is covered in the following section.

As most AWS services do, EFS sends metrics to CloudWatch, allowing us to get insight into the performance of a filesystem. We recommend creating a CloudWatch alarm to monitor filesystems with General Purpose performance mode. The metric `PercentIOLimit` tells you whether a filesystem is approaching its I/O limit. By migrating your data to a filesystem with Max I/O mode, you can increase the I/O limit from 35,000 IOPS to 500,000+ IOPS. Keep in mind, however, that doing so increases read latency from around 600 microseconds to single-digit milliseconds. The next listing shows the CloudWatch alarm to monitor the `PercentIOLimit` metric.

Listing 9.6 Monitoring the `PercentIOLimit` metric using a CloudWatch alarm

```
PercentIOLimitTooHighAlarm:
  Type: 'AWS::CloudWatch::Alarm'                              ①
  Properties:
    AlarmDescription: 'I/O limit has been reached, consider ...'
    Namespace: 'AWS/EFS'                                      ②
    MetricName: PercentIOLimit                                ③
    Statistic: Maximum                                        ④
    Period: 600                                      ⑤
    EvaluationPeriods: 3                             ⑥
    ComparisonOperator: GreaterThanThreshold         ⑦
    Threshold: 95                                    ⑧
    Dimensions:                                      ⑨
    - Name: FileSystemId                             ⑩
      Value: !Ref FileSystem                         ⑪
```

① Creates a CloudWatch alarm

② EFS uses the AWS/EFS for all its metrics.

③ If the PercentIOLimit metric reaches 100%, we are hitting the limit of 35,000 read or 7,000 write operations. Switching to the Max I/O performance mode is a possible mitigation.

④ The alarm evaluates the PercentIOLimit metric by using the maximum.

⑤ The alarm monitors a period of 600 seconds.

⑥ The alarm takes into consideration the last three periods, which is 3x 600 seconds.

⑦ The alarm will trigger if the maximum PercentIOLimit was greater than the threshold.

⑧ The threshold is set to 95%.

⑨ Each filesystem reports its own PercentIOLimit metric. Therefore, we are configuring a dimension here.

⑩ The dimension is named FileSystemId.

⑪ References the ID of the filesystem

Next, you will learn about the second factor that affects the performance of an EFS filesystem: the throughput mode.

9.5.2 Throughput mode

Besides the performance mode, the throughput mode determines the maximum throughput of an EFS filesystem using the following modes:

- *Bursting Throughput mode*—Comes with a small baseline throughput but is able to burst throughput for short periods of time. Also, the throughput grows with the amount of data stored.
- *Provisioned Throughput mode*—Gives you a constant throughput with a maximum of 1 GiBps. You pay $6.00 per MB/s per month.

When using Bursting Throughput mode, the amount of data stored in a filesystem affects the baseline and bursting throughput. The baseline throughput for an EFS filesystem is 50 MiBps per TiB of storage with a minimum of 1 MiBps. For a limited period of time, reading or writing data may burst up to 100 MiBps per TiB of storage, with a minimum of 100 MiBps.

WARNING The discussed limits are labeled default limits by AWS. Depending on your scenario, it might be possible to increase those limits. However, there is no guarantee for that, and AWS does not provide any information on the maximum of a possible limit increase.

A filesystem with Bursting Throughput mode handles up to 1 or 3 GiBps, depending on the region (see http://mng.bz/09Av). Table 9.2 shows the baseline and bursting write throughput of an EFS filesystem in US East (N. Virginia) where the maximum bursting throughput is 3 GiBps.

Table 9.2 EFS throughput depends on the storage size.

Filesystem size	Baseline throughput	Bursting throughput	Explanation
<=20 GiB	1 MiBps	100 MiBps	Minimum baseline throughput is 1 MiBps, and minimum bursting throughput is 100 MiBps.
1 TiB	50 MiBps	100 MiBps	The bursting throughput starts to grow with filesystems larger than 1 TiB.
30 TiB	1500 MiBps	3000 MiBps	The bursting throughput hits the maximum for filesystems in US East (N. Virginia).
>=60 TiB	3000 MiBps	3000 MiBps	Both the baseline and bursting throughput hit the maximum for filesystems in US East (N. Virginia).

As long as you are consuming less than the baseline throughput, the filesystem accumulates burst credits. For every TiB of storage, your filesystem accrues credits of 50 MiB per second. For example, a filesystem with 500 GiB accumulates burst credits for accessing 2160 GiB within 24 hours. That’s bursting at the maximum of 100 MiBps for 6 hours.

WARNING EFS does not support more than 500 MiBps per client, which means an EC2 instance cannot transfer more than 500 MiBps to an EFS filesystem.

Be aware that the maximum credit balance is 2.1 TiB for filesystems smaller than 1 TiB and 2.1 TiB per TiB stored for filesystems larger than 1 TiB. This means filesystems can burst for no more than 12 hours.

It is important to mention that a discount exists for reading data compared to writing data. A filesystem allows 1.66 or three times more read throughput depending on the region. For US East (N. Virginia), the discount is 1.66, which means the maximum read throughput is 5 GiBps. However, be aware that the discount for reading data is not valid for requests returning less than 4 KB of data.

To make things even more complicated, it is worth mentioning that data stored in the Infrequent Access storage class is not taken into consideration when calculating the maximum baseline and burst throughput.

When using the Bursting Throughput mode, you should monitor the burst credits of the filesystem because if the filesystem runs out of burst credits, the throughput will decrease significantly. We’ve seen many outages caused by an unexpected drop in the baseline performance. The following listing shows how to create a CloudWatch alarm to monitor the `BurstCreditBalance` metric. We’ve configured the threshold of the alarm to a burst credit of 192 GB, which will allow the filesystem to burst at 100 MiBps for an hour.

Listing 9.7 Monitoring the `PercentIOLimit` metric using a CloudWatch alarm

```
BurstCreditBalanceTooLowAlarm:
  Type: 'AWS::CloudWatch::Alarm'
  Properties:
    AlarmDescription: 'Average burst credit balance over last ...'
    Namespace: 'AWS/EFS'
    MetricName: BurstCreditBalance        ①
    Statistic: Average
    Period: 600
    EvaluationPeriods: 1
    ComparisonOperator: LessThanThreshold
    Threshold: 192000000000               ②
    Dimensions:
    - Name: FileSystemId
      Value: !Ref FileSystem
```

① The alarm monitors the BurstCreditBalance metric.

② The threshold of 192000000000 bytes translates into 192 GB (lasts for ~30 minutes; you can burst at 100 MiBps).

If you are running out of burst credits because your workload requires a high and constant throughput, you could switch to the Provisioned Throughput mode or add data to your EFS filesystem to increase the baseline and bursting throughput.

Provisioned Throughput mode

It is possible to upgrade an EFS filesystem from Bursting Throughput mode to Provisioned Throghput mode at any time. When activating the Provisioned Throughput mode, you need to specify the provisioned throughput of the filesystem in MiBps.

A filesystem will deliver the provisioned throughput continuously. The maximum provisioned throughput is 1 GiBps. Also, you have to pay for the provisioned throughput. AWS charges $6.00 per MB/s per month.

For a filesystem in Provisioned Throughput mode, you pay for throughput only above the baseline performance. So, for example, when provisioning 200 MiBps for a filesystem with 1 TiB, you will pay for only 150 MiBps, because 50 MiBps would be the baseline performance of a filesystem in Bursting Throughput mode.

The following code snippet shows how to provision throughput using CloudFormation. Use the efs-provisioned.yaml template to update your `efs` CloudFormation stack:

```
$ aws cloudformation update-stack --stack-name efs \
➥ --template-url https:/ /s3.amazonaws.com/awsinaction-code3/\
➥ chapter09/efs-provisioned.yaml --capabilities CAPABILITY_IAM
```

The next listing contains two differences to the template you used in the previous example: the `ThroughputMode` is set to `provisioned` instead of `bursting`, and the `ProvisionedThroughputInMibps` sets the provisioned throughput to 1 MiBps, which is free, because this is the baseline throughput of a filesystem with bursting throughput.

Listing 9.8 EFS filesystem with provisioned throughput

```
FileSystem:
  Type: 'AWS::EFS::FileSystem'
  Properties:
    Encrypted: true
    ThroughputMode: provisioned          ①
    ProvisionedThroughputInMibps: 1      ②
    PerformanceMode: generalPurpose
    FileSystemPolicy:
      Version: '2012-10-17'
      Statement:
      - Effect: 'Deny'
        Action: '*'
        Principal:
          AWS: '*'
        Condition:
          Bool:
            'aws:SecureTransport': 'false'
```

① Sets the throughput mode from bursting to provisioned

② Configures the provisioned throughput in MiBps. 1 MiBps is free, even for empty filesystems.

With Provisioned Throughput mode enabled, you should keep an eye on the utilization of the throughput that you provisioned. This allows you to increase the provisioned throughput to avoid performance problems caused by EFS becoming the bottleneck of your system. Listing 9.9 shows how to create a CloudWatch alarm to monitor the utilization of the maximum throughput of an EFS filesystem. The same CloudWatch alarm works for filesystems with Bursting Throughput mode as well. This combines the metrics `MeteredIOBytes`, the throughput to the filesystem, and `PermittedThroughput`, the maximum throughput of the filesystem, by using metric math (see **http://mng.bz/qdvA** for details).

Listing 9.9 Monitoring using a CloudWatch alarm and metric math

```
PermittedThroughputAlarm:
  Type: 'AWS::CloudWatch::Alarm'
  Properties:
    AlarmDescription: 'Reached 80% of the permitted throughput ...'
    Metrics:                                         ①
    - Id: m1                                         ②
      Label: MeteredIOBytes
      MetricStat:
        Metric:
          Namespace: 'AWS/EFS'
          MetricName: MeteredIOBytes                 ③
          Dimensions:
          - Name: FileSystemId
            Value: !Ref FileSystem
        Period: 60
        Stat: Sum
        Unit: Bytes
      ReturnData: false
    - Id: m2
      Label: PermittedThroughput
      MetricStat:
        Metric:
          Namespace: 'AWS/EFS'
          MetricName: PermittedThroughput            ④
          Dimensions:
          - Name: FileSystemId
```

```
            Value: !Ref FileSystem
          Period: 60
          Stat: Sum
          Unit: 'Bytes/Second'
        ReturnData: false
      - Expression: '(m1/1048576)/PERIOD(m1)'     ⑤
        Id: e1                                     ⑥
        Label: e1
        ReturnData: false
      - Expression: 'm2/1048576'                   ⑦
        Id: e2
        Label: e2
        ReturnData: false
      - Expression: '((e1)*100)/(e2)'              ⑧
        Id: e3
        Label: 'Throughput utilization (%)'
        ReturnData: true                           ⑨
      EvaluationPeriods: 10
      DatapointsToAlarm: 6
      ComparisonOperator: GreaterThanThreshold     ⑩
      Threshold: 80                                ⑪
```

① Uses metric math to combine multiple metrics

② Assigns the ID m1 to the first metric

③ The first metric taken into consideration is the MeteredIOBytes metric, which contains the current utilization.

④ The second metric is the PermittedThroughput metric, which indicates the maximum throughput.

⑤ The first metric, m1, returns the sum of bytes transferred within 60 seconds. This expression converts this into MiBps.

⑥ Assigns the ID e1 to the metric to reference it in the following line

⑦ Converts the second metric from bytes/second into MiBps

⑧ Calculates the throughput utilization in percent

⑨ The output of the third expression is the only metric/expression returning data used for the alarm.

⑩ The alarm triggers if the throughput utilization is greater than …

⑪ …80%.

If the CloudWatch alarm flips into the `ALARM` state, you might want to increase the provisioned throughput of your filesystem.

9.5.3 Storage class affects performance

The third factor that affects the performance of an EFS filesystem is the storage class used to persist data. By default, read latency to data persisted with standard storage class is as low as 600 microseconds. The latency for write requests is in the low single-digit milliseconds.

But if you choose the One Zone storage class to reduce costs, the read and write latency increases to double-digit milliseconds. Depending on your workload, this might have a significant effect on performance.

9.6 Backing up your data

AWS promises a durability of 99.999999999% (11 9s) over a given year for data stored on EFS. When using the Standard storage class, EFS replicates data among multiple data centers. Still, we highly recommend backing up the data stored on EFS, particularly so you can recover from a scenario where someone, maybe even you, accidentally deleted data from EFS.

Luckily, AWS provides a service to back up and restore data: AWS Backup. This service to centralize data protection supports EC2, EBS, S3, RDS, DynamoDB, EFS, and many more. The following three components are needed to create snapshots of an EFS filesystem with AWS Backup, as illustrated in figure 9.4:

- *Vault*—A container for grouping backups.
- *Plan*—Defines when and how to backup resources.
- *Recovery Point*—Contains all the data needed to restore a resource; you could also call it a snapshot.

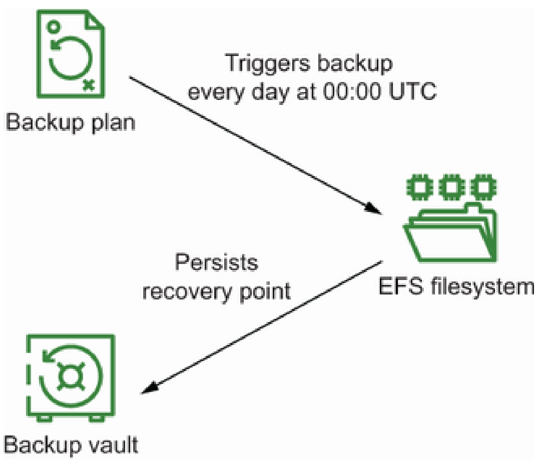

Figure 9.4 AWS Backup creates recovery points based on a schedule and stores them in a vault.

To enable backups for your EFS, update your CloudFormation stack with the template efs-backup.yaml, which contains the configuration for AWS Backups, as shown here. We'll dive into that next:

```
$ aws cloudformation update-stack --stack-name efs \
➥ --template-url https:/ /s3.amazonaws.com/awsinaction-code3/\
➥ chapter09/efs-backup.yaml --capabilities CAPABILITY_IAM
```

Where is the template located?

You can find the template on GitHub. Download a snapshot of the repository at **https://github.com/AWSinAction/code3/archive/main.zip**. The file we're talking about is located at chapter09/efs-backup.yaml. On S3, the same file is located at **https://s3.amazonaws.com/awsinaction-code3/chapter09/efs-backup.yaml**.

The following listing is an excerpt from efs-backup.yaml and shows how to create a backup vault and plan. On top of that, it includes a backup selection, which references the EFS filesystem.

Listing 9.10 Backing up an EFS filesystem with AWS Backup

```
BackupVault:                                                ①
  Type: 'AWS::Backup::BackupVault'
  Properties:
    BackupVaultName: !Ref 'AWS::StackName'                  ②
BackupPlan:                                                 ③
  Type: 'AWS::Backup::BackupPlan'
  Properties:
    BackupPlan:
      BackupPlanName: 'efs-daily'
      BackupPlanRule:                                       ④
      - RuleName: 'efs-daily'
        TargetBackupVault: !Ref BackupVault                 ⑤
        Lifecycle:
          DeleteAfterDays: 30                               ⑥
        ScheduleExpression: 'cron(0 5 ? * * *)'             ⑦
        CompletionWindowMinutes: 1440                       ⑧
BackupSelection:                                            ⑨
  Type: 'AWS::Backup::BackupSelection'
  Properties:
    BackupPlanId: !Ref BackupPlan                           ⑩
    BackupSelection:
      SelectionName: 'efs'
      IamRoleArn: !GetAtt 'BackupRole.Arn'                  ⑪
      Resources:
      - !GetAtt  'FileSystem.Arn'                           ⑫
```

① Creates a backup vault to persist recovery points

② A backup vault requires a unique name; therefore, we are using the name of the CloudFormation stack here.

③ Creates a backup plan

④ Each backup plan includes at least one rule.

⑤ The rule defines which backup vault should be used to store backups.

⑥ The life cycle is configured in a way to delete backups older than 30 days.

⑦ The backup plan schedules a backup for 05:00 UTC every day.

⑧ The backup should complete within 24 hours; otherwise, it will be cancelled.

⑨ Selects the resources to include in the backup

⑩ References the backup plan

⑪ An IAM role is required to grant access to EFS (see the following listing).

⑫ Adds the filesystem to the backup selection by using its ARN

Besides this, an IAM role granting access to the EFS filesystem is needed, as shown in the next listing.

Listing 9.11 The IAM role granting AWS Backup access to EFS

```
BackupRole:
  Type: 'AWS::IAM::Role'
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
      - Effect: Allow
        Principal:
          Service: 'backup.amazonaws.com'                ①
        Action: 'sts:AssumeRole'
    Policies:
    - PolicyName: backup
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Action:
          - 'elasticfilesystem:Backup'                   ②
          - 'elasticfilesystem:DescribeTags'             ③
          Resource: !Sub 'arn:${AWS::Partition}:elasticfilesystem
➥ :${AWS::Region}:${AWS::AccountId}:
➥ file-system/${FileSystem}'                            ④
```

① Only the AWS Backup service is allowed to assume this IAM role.

② The role grants access to create backups of EFS filesystems...

③ ...and allows you to describe tags of EFS filesystems, which is needed in case you define a backup selection based on tags.

④ The policy restricts access to the filesystem we defined in the CloudFormation template earlier.

AWS Backup allows you to restore an entire filesystem or only some of the folders. It is possible to restore data to the original filesystem as well as to create a new filesystem to do so.

Unfortunately, it will take a while until AWS Backup creates a recovery point for your EFS filesystem. If you are interested in the results, wait 24 hours and check the results as follows:

1. Open the AWS Management Console.
2. Go to the AWS Backup service.
3. Open the backup vault named `efs`.
4. You'll find a list with available recovery points.
5. From here, you could restore an EFS filesystem in case of an emergency.

That's all you need to back up data stored on an EFS filesystem. We should briefly talk about the costs. You are paying for the data stored and if you need to restore your data. The following prices are valid for US East (N. Virginia). For further details, visit **https://aws.amazon.com/backup/pricing/**:

- Storage: $0.05 per GB and month
- Restore: $0.02 per GB

Cleaning up

First, make sure that the backup vault `efs` does not contain any recovery points like this:

```
$ aws backup list-recovery-points-by-backup-vault --backup-vault-name e
➥ --query "RecoveryPoints[].RecoveryPointArn"
```

If you are getting a list of recovery points, delete each of them using the following command. forget to replace `$RecoveryPointArn` with the ARN of the recovery point you are trying to

```
$ aws backup delete-recovery-point --backup-vault-name efs \
➥ --recovery-point-arn $RecoveryPointArn
```

After that, use the following command to delete the CloudFormation stack named `efs` that y while going through the examples within this chapter:

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

Summary

- EFS provides a NFSv4.1-compliant filesystem that can be shared between Linux EC2 instances in different availability zones.
- EFS mount targets are bound to an availability zone and are protected by security groups.
- Data that is stored in EFS is replicated across multiple data centers.
- EFS comes with two performance modes: General Purpose and Max I/O. Use General Purpose if your workload accesses many small files, and Max I/O if you have a few large files.
- By default, the throughput of an EFS filesystem is capable of bursting for a certain amount of time. If you are expecting a high and constant throughput, you should use provisioned throughput instead.
- You need at least two mount targets in different data centers (also called availability zones) for high availability.
- Encryption of data in transit (TLS) and IAM authentication help to protect sensitive data stored on EFS.
- AWS Backup allows you to create and restore snapshots of EFS filesystems.