

# Retrieving Data Part 2 – Transfer Protocols

© Explore Data Science Academy

## Learning objectives

By the end of this train, you should be able to:

- explain what FTP is and how it functions;
- understand NFS and SMB protocols and their differences;
- understand the differences between SFTP and FTPS and how they function;
- describe and understand typical methods of integrating legacy systems; and
- set up an NFS system in AWS.

## 1. Introduction to FTP

### What is FTP?

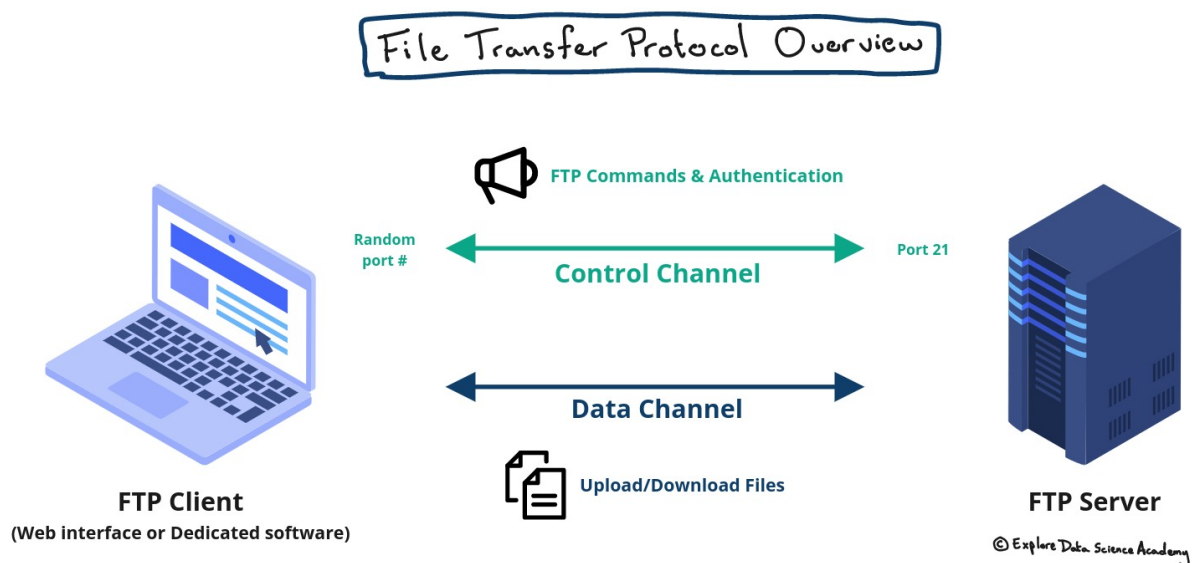


Figure 1. An overview of the File Transfer Protocol (FTP) with its two interacting entities: the FTP Client and FTP Server.

FTP stands for **File Transfer Protocol**, and it is one of the oldest protocols for transferring large files online. We can use FTP to transfer data between different computers or web applications. Generally, it enables the storage or retrieval of files securely.

In Figure 1, we can see how an FTP client transfers data to an FTP server. The FTP client is the front end of a web application and the FTP server is the back end, serving the content.

An FTP client could be a piece of software used to transfer files online securely, or it could be a simple web interface as in the example in Figure 1. An FTP server is a data store that listens for and sends specific data at the client's request. The FTP server can also receive data from an FTP client.

### History of FTP

FTP was developed in the early 1970s by an MIT student named Abhay Bhushan. It was initially created to allow files to be securely transferred between servers and host computers via the [ARPANET Network Control Program \(NCP\)](#) – a precursor to the modern-day internet.

The first FTP applications were developed as command line programs before operating systems had graphical user interfaces. They are still shipped with most operating systems today!

One critical downside of FTP is that data transmitted between client and server are not encrypted. This creates the potential for data to be hacked or tampered with while it is being transferred from one system to another. Due to security concerns, FTP has evolved into more secure protocols such as Secure File Transfer Protocol (SFTP) and File Transfer Protocol Secure (FTPS). We'll look at these protocols later on in the train to better understand the improved security measures.

### How FTP works

FTP can run in one of two modes: *active* or *passive*. The choice of mode determines how the data connection is established between the FTP client and the FTP server. In both of these cases, the client creates a [TCP control](#) connection from a random, unprivileged [ephemeral port](#) to the FTP server, which has a command port of 21.

### Active mode

In Figure 2, we can see how the FTP client starts listening for incoming data connections from the FTP server on port  $x$ . Once set up, it sends an FTP command "PORT  $x$ " to the server to tell it which port it is listening on. Here, the client actively provides the server with details of how to connect, such as its port number and IP address.

- To indicate the successful exchange of communication details, the server then sends a success status code 200.
- A status code of 150 follows this step to indicate that the server is initiating a data transfer.
- Lastly, the server initiates a *data channel* to the FTP client from its port 20 (FTP server data port), and the transfer of files begins.
- Once the file transfer procedure is complete, the FTP server will send a status code 226 to indicate a successful transfer of files.

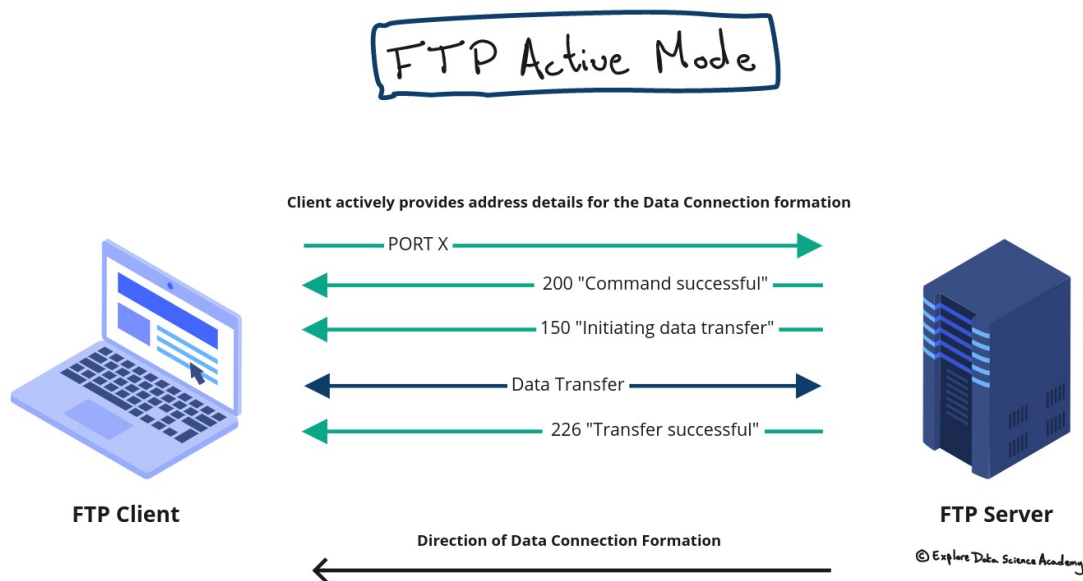


Figure 2. A representation of the control and data communication signals exchanged during an FTP 'active mode' session.

### Passive mode

This mode is usually used when the FTP client is behind a firewall and is therefore unable to accept incoming TCP connections.

- In Figure 3, we see that in passive mode the client will use a control connection to send the PASV command to the FTP server.
- The server responds by providing a port number and IP address for the client to connect over. Upon receipt of this information, a connection is opened from a random client port to the server IP address and server port.
- Similar to active mode, the server responds with a status code 150 to indicate that the data transfer is going to be initiated.
- Lastly, once the files have been transferred, the server sends a status code 226 to indicate a successful transfer.

# FTP Passive Mode

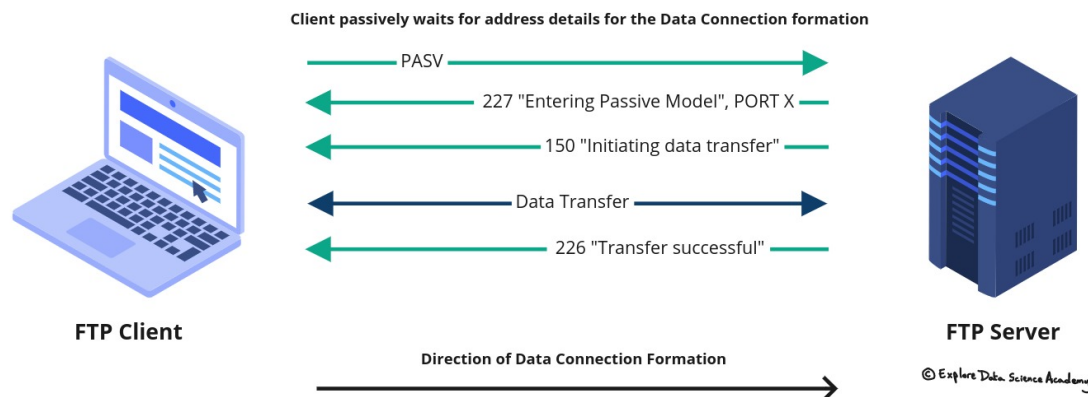


Figure 3. A representation of the control and data communication signals exchanged during an FTP 'passive mode' session.

In considering both modes described above, note that the FTP server always uses two ports: one for sending and one for receiving. This is because FTP was originally designed to operate on [NCP](#), a simple protocol that made use of two-port connections for two-way communication.

When [TCP](#) was standardised, the need to have a port for each application was removed. However, the FTP protocol was never altered to use only one port and therefore continues to use two ports for backward compatibility.

## How to use FTP

FTP, SFTP, and FTPS are useful tools for the transfer of files and data between networked computers and servers. Sometimes these protocols are embedded within an application, but in other cases, you can use an FTP client. The list below includes some of the most popular and reliable FTP clients used in the industry.

FTP client	Protocols	Extra features
FileZilla	FTP/FTPS/SFTP	Free and open source; provides white papers and tutorials
WinSCP	FTP/SFTP	Free and open source; Windows only; provides basic scripting and file management functions
Cyberduck	FTP/FTPS/SFTP	Free and open source; Windows and Mac
FireFTP	FTP/SFTP	Free; cross-platform (Firefox add-on)
Classic FTP	FTP/FTPS	Free; Windows and Mac

## 2. Types of sharing protocols

To frame our challenge appropriately, we need to understand that many businesses rely heavily on file-sharing tools to make files available across many different devices and platforms. This is a challenging problem, as most businesses use a blend of operating systems (Windows, Mac, and Linux) and machines that may all require access to a file simultaneously. The primary constraint here is that each operating system follows different protocols and logical semantics – hence this single file share is a non-trivial problem.

Let's consider some solutions that aim to provide seamless access to files across several systems.

### Network File System (NFS) protocol

NFS is a [distributed file system](#) protocol that enables a client machine to access, over a network connection, files stored on other machines on the network. NFS was designed and is primarily used for information exchange between Unix-based systems.

To provide access to the files located on a server, NFS allows a user or system administrator to mount the server file system onto a client computer. The administrator can enable the client to access a portion of the files or assign different types of privileges to each file. The requests between the clients and servers are made using [Remote Procedure Calls \(RPCs\)](#).

### Server Messaging Block (SMB) protocol

The SMB protocol has a similar function to NFS in that it also allows a client computer to access files on remote computing

infrastructure. The protocol is primarily used for Windows machines and utilises the TCP network protocol to communicate over the internet.

Both SMB and NFS protocols have evolved to become more secure and efficient. Primarily this is due to encryption, as well as authentication before allowing access via the protocols to the stored files. The [SMB protocol version](#) and [NFS protocol version](#) documentation illustrate this progression of security, along with the current features offered by these technologies.

## SFTP and FTPS

As mentioned earlier, the lack of security within FTP led to the emergence of newer protocols that enable secure file transfers across any network: SFTP and FTPS. These technologies differ subtly:

- **SFTP** is FTP over Secure Shell (SSH).
- **FTPS** is FTP over Secure Sockets Layer (SSL).

Both SSH and SSL encrypt the connections between client and server, but there are a few fundamental differences.

Firstly, it's important to understand that although the term is commonly used, SSL is a deprecated technology, and now almost always refers to [Transport Layer Security \(TLS\)](#). SSL uses certificates and [Public Key Infrastructure \(PKI\)](#) to create encrypted HTTPS connections between a website and its visitors. SSL is designed for the transmission of information and data, and everything that is sent via SSL is encrypted. Authentication is only required server-side, not client-side. SSL uses port 443, and as of July 2018, all websites are required to have SSL installed.

SSH is similar to SSL in that it is also PKI-based and forms an encrypted communication tunnel. The difference is that SSH is designed to *execute commands*, while SSL is intended for *information transmission*. Thus, we'd generally use SSH when we want to log into a network remotely. SSH uses port 22 and requires both client and server authentication. This makes sense if we think about it – to run commands such as those used when committing code to git, one would need certain user-specific permissions. Therefore the user or client identity must be known.

### Summary of the differences between SFTP and FTPS

Features	SFTP	FTPS
Implements strong encryption algorithms	Yes	Yes
Encrypts usernames and passwords	Yes	Yes
Supports key-based authentication	Yes	No
Supports certificates	No	Yes
Firewall-friendly	Yes	No

### Additional considerations

- Both SFTP and FTPS use strong encryption algorithms such as [AES](#) and [Triple DES](#) for data transfer.
- With SFTP, SSH keys can be used to authenticate connections in addition to or instead of passwords.
- With FTPS, we can authenticate connections using a user ID, password, and certificates.
- Due to SFTP only requiring a single port number (defaulting to 22), it is firewall-friendly. FTPS can be difficult to use with a tightly-secured firewall since it uses multiple port numbers.

## Necessity of file sharing protocols

Data are generated in high volumes and in many locations, and we need to be able to transfer them. File sharing protocols should enable:

- **Security:** Like Navy SEALs say, "Two is one and one is none". It's not sufficient to have data in just one location. We need to be able to back it up, and this involves transferring data securely. Enter SFTP and FTPS.
- **Collaboration:** Sharing files within a business for multiple users to work on and collaborate simultaneously.
- **Migration of historical data between cloud environments:** Increasingly, applications need to be migrated between cloud service environments (AWS to Azure, for example). This involves a large migration of historical data.
- **Hybrid cloud environments:** It is increasingly common for companies with on-premise data centres to incorporate the public cloud into their infrastructure. This means that data needs to be synced between the cloud provider and their on-premise storage.

## 3. Integrating legacy systems

## Integrating FTP with legacy systems

Typically, software systems exchange information through the use of an API. However, many legacy systems, and even some in development today, do not support API integration.

We inevitably encounter situations where we need to extract data from a system in a manner that is more complicated than a simple API call. Here we'll discuss a few solutions in this regard.

**1. Sharing files using FTP:** This solution is applicable when the systems share a common resource such as a server, database, or data lake. One system uploads the files to the shared resource, from which a scheduled job is run to ingest the new data and transform them according to requirements. While simple, this solution can introduce substantial latency as data cannot be transferred from one system to another in real-time.

**2. Sharing a database:** In this case, two applications share a common database. One application is granted read-only access and takes a passive role. This approach is simple to set up, though it can lead to impractical changes to the database structure and compromises data security.

**3. Full daily data dump:** If sharing an active database is infeasible or undesirable, an organisation can perform data dumps at fixed intervals (for example, daily or weekly). One drawback is that this causes data privacy issues, as large amounts of unfiltered data move around the organisation regularly.

**4. Webpage scraping:** Sometimes, web scraping is the only possible way to extract data from an application. However, with web scraping, we're at the mercy of the developers of the website we're scraping. For example, the front-end HTML of the webpage might be changed each week with updates. Clearly, our scraping application can only be as reliable as the code we are using it to scrape.

**5. Email:** If either the sending or receiving entity in a system has no support for legitimate integration, email may be considered a last resort. Here, an event triggered by the arrival of data in a mailbox could be used to pull the received data to the destination machine.

**6. Adapters:** Another approach is the development of a custom software module. This module links directly to the source system's database and pulls the required data to the destination system. Two complexities arise here:

1. Bespoke development is required to maintain the module; and
2. The state space of the source database must be fully understood.

## Legacy system example

Here we'll consider a real-world scenario where systems need to be connected.

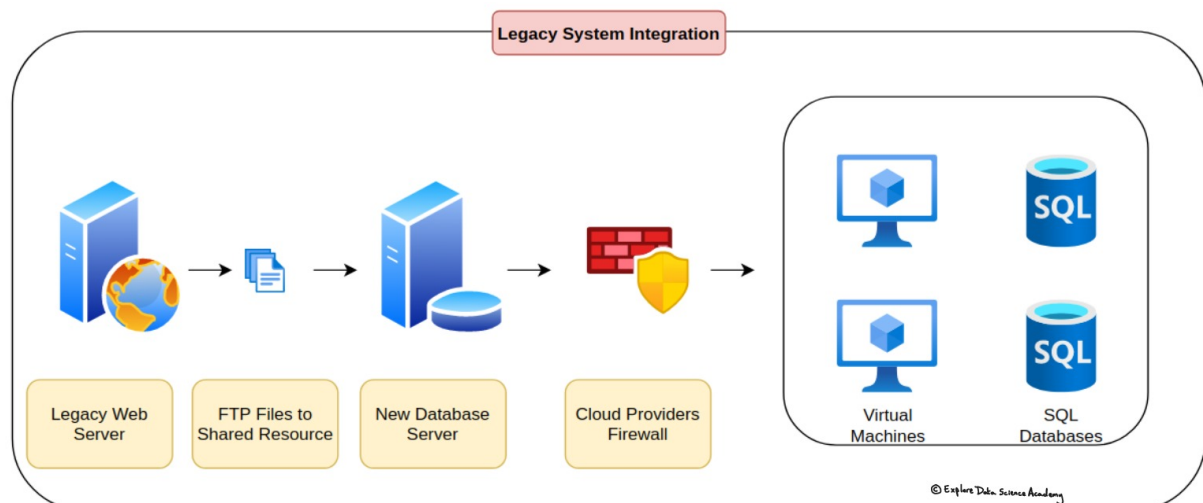


Figure 4. An illustration of how various file sharing solutions can be used to retrieve data from a legacy system without API support.

Figure 4 shows a legacy web server which contains the information we need, but which does not have an API for integration. Our goal is to move data from this system onto a public cloud (shown right) where it can be transformed and stored. To perform the ingestion:

1. Our first step is to use an FTP connection to move the required files to a newer system server.
2. We configure this server with an appropriate database, which we access via an API.
3. Using this API we can pull the data into our cloud environment, where it first passes through various firewalls before being written to a SQL database.

## 4. Exercise: Setting up an NFS file share

Let's get our hands dirty with a practical exercise – setting up an NFS mount in the cloud. To do this, we'll be leveraging AWS services including Amazon Elastic File System (EFS) and Amazon EC2.

### Step 1: Configure the security group

Before we can set up the various services used in our NFS mount, we'll need to create a new *security group* which contains appropriate inbound/outbound communication rules.

1. Log in to your AWS management console.
2. Open up the **Security Groups** dashboard by navigating to *Services > EC2 > Network & Security > Security Groups*.
3. Create a new security group by clicking on *Create security group*.
4. Give your security group an easily identifiable name, for example, *DE-NFS-Exercise*, as well as a simple description.
5. Keep the **VPC** selection as the default value for your account.
6. Under **Inbound rules**, use the *Add rule* button to add the following two rules to the security group:

Type	Port Range	Source	Value
NFS	2049	Custom	0.0.0.0/0
SSH	22	Custom	0.0.0.0/0

7. Leave the **Outbound rules** section set to the default *All traffic* value. The finalised inbound and outbound rules should mirror those shown in Figure 5.

The screenshot displays the AWS Security Groups console. The 'Inbound rules' section is highlighted with a red box and contains two rules: one for NFS (Type: NFS, Protocol: TCP, Port range: 2049, Source: Custom, Value: 0.0.0.0/0) and one for SSH (Type: SSH, Protocol: TCP, Port range: 22, Source: Custom, Value: 0.0.0.0/0). The 'Outbound rules' section below it shows a single rule for 'All traffic' (Type: All traffic, Protocol: All, Port range: All, Destination: Custom, Value: 0.0.0.0/0). Each rule has a 'Delete' button to its right.

Figure 5. Ports to configure during the security group setup step.

8. Complete the security group setup by clicking on *Create security group*.

### Step 2: Create an EFS file system

Security group configured, we can now set up Amazon EFS to serve as our NFS share. EFS enables a highly available and scalable network file system – find more information [here](#).

1. Within the AWS management console, navigate to the EFS dashboard: *Services > EFS*.
2. Create a new file system by clicking on *Create file system*, and then *Customise*.
3. Under **Step 1: File system settings**, give your file system a descriptive name, for example, *DE-NFS-Exercise-FS*. Leave all other configurations set to default values, and click *Next*.
4. For **Step 2: Network access**, ensure that the **default VPC** for your account is selected. Under **Mount targets**, assign each availability one (AZ) to the security group created in the [step 1](#), as shown in Figure 6. Hit *Next*.



Figure 6. Configure the EFS mount targets to have the same security group as the one configured in step 1.

5. Leave **Step 3: File system policy** set to default values. Click *Next*.
6. Finalise the EFS share by clicking on *Create* at the end of **Step 4: Review and Create**. Setup should complete in a matter of seconds.
7. Once fully set up, take note of the **File system ID** associated with your EFS file system. To display it, click on the *File systems* menu within the EFS dashboard and find the corresponding entry for your EFS file system. We will use this ID in [step 5](#).

### Step 3: Create an EC2 instance

Next, we need an EC2 instance so that our EFS resource has a mount target.

1. Navigate to the EC2 Dashboard in the AWS management console. Select *Instances* and then *Launch Instances*.
2. When prompted to choose an Amazon Machine Image (AMI), select *Amazon Linux 2 AMI (HVM)*.
3. Choose a *t2.micro* EC2 instance under the *Instance Type* selection.
4. Under **Configure Instance Details**, ensure that the *Default VPC* is selected. Leave all other settings set to default values.
5. Leave all options set to default values for the **Add Storage** selection.
6. Add a tag in the form of **{owner: firstname\_surname}**, for example, owner:dora\_explorer. Ensure that this tag is applied to all associated resources.
7. Under **Configure Security Group**, click on the *Select an existing security group* option and enable the security group we selected in [step 1](#).
8. Now, select *existing key pair* or *create a new key pair*. Save the resulting .pem to your machine for use in connecting to your instance.
9. Hit *Launch* to complete the setup.

### Step 4: Connect to your instance

First, we need to retrieve the public IPv4 address of the instance.

1. Navigate to *Services > EC2 > Instances (running)*. Find your instance launched in the previous step (you can use the owner tag you created to help your search).
2. Locate the correct instance and copy its IPv4 address to your clipboard.

On any OS, the connection is made using `ssh`. However, the specific method you use to connect to your EC2 instance from your local machine depends on your operating system.

### Windows

- Use a graphical client such as [PuTTY](#); or
- Use the `ssh` command via a Git Bash terminal.

## Mac/Linux

- Use the `ssh` command via a Terminal window.

### For both

- The SSH command syntax for connecting to a remote instance is as follows:

```
ssh {username}@{host_ip_address}
```

- Use `ec2-user` when specifying the username of the target machine (your EC2 instance).
- Use the IPv4 address you just copied above as the `host_ip_address`.
- Remember to include the `-i` argument to use the `.pem` authentication certificate downloaded previously.
- For example, if we had a security certificate named `nfs-train.pem`, and a machine IPv4 address of `105.0.0.5`, the `ssh` command would read:

```
ssh -i nfs-train.pem ec2-user@105.0.0.5
```

## Step 5: Mount EFS to your EC2 instance

To access our file system we will now mount EFS to our EC2 instance using the `mount` command and our EFS file system's DNS name.

1. Within the remote instance, install the `amazon-efs-utils` library to help mount the EFS file system. More information about this library [here](#).

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

2. Create a folder named `efs` to act as the mount point for the file system:

```
sudo mkdir efs
```

3. We are now ready to mount the EFS file system on our instance. The generic command for this is as follows:

```
sudo mount -t efs -o tls {EFS_ID}:/ {MOUNT_TARGET}
```

Here, `EFS_ID` corresponds to the *File system ID* we recorded in [step 2.7](#). `MOUNT_TARGET` is the path to the mount point in which we'd like to place our NFS file share.

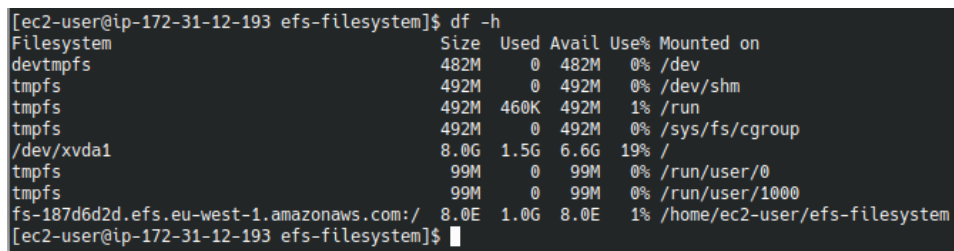
For example, if we had a File system ID = `fs-187d6d2d`, then we'd enter the following command to mount our EFS file system as an NFS access point:

```
sudo mount -t efs -o tls fs-187d6d2d:/ efs
```

4. After a couple of seconds, the above command (with the correct File system ID) should execute. We can verify that the file system has been successfully mounted by running:

```
df -h
```

Here, we should see an output similar to the image below:



```
[ec2-user@ip-172-31-12-193 efs-filesystem]$ df -h
Filesystem                Size      Used Avail Use% Mounted on
devtmpfs                  482M        0  482M   0% /dev
tmpfs                     492M        0  492M   0% /dev/shm
tmpfs                     492M    460K  492M   1% /run
tmpfs                     492M        0  492M   0% /sys/fs/cgroup
/dev/xvda1                 8.0G    1.5G   6.6G  19% /
tmpfs                     99M        0   99M   0% /run/user/0
tmpfs                     99M        0   99M   0% /run/user/1000
fs-187d6d2d.efs.eu-west-1.amazonaws.com/ 8.0E    1.0G   8.0E   1% /home/ec2-user/efs-filesystem
[ec2-user@ip-172-31-12-193 efs-filesystem]$
```

Figure 7. Terminal output after a successful mounting of Amazon EFS.

In the **Filesystem** column there should be an entry of the form:

```
{file_system_id}.efs.eu-west-1.amazonaws.com/
```

5. Next, create a 1GB text file called `demo.txt` in the newly mounted file system:

```
sudo dd if=/dev/zero of=~/.efs/demo.txt bs=1M count=1024 status=progress
```

6. Navigate (using `cd`) into the mount directory and check that the file exists (using `ls`).



7. We can also validate the write operation we've just performed by looking at our file system's metrics:

- Navigate to *Amazon EFS > File Systems* in the AWS management console.
- Click on your file system's ID.
- Open the *Monitoring* tab.
- Select a relevant period, and you should be able to obtain results similar to those in Figure 8.

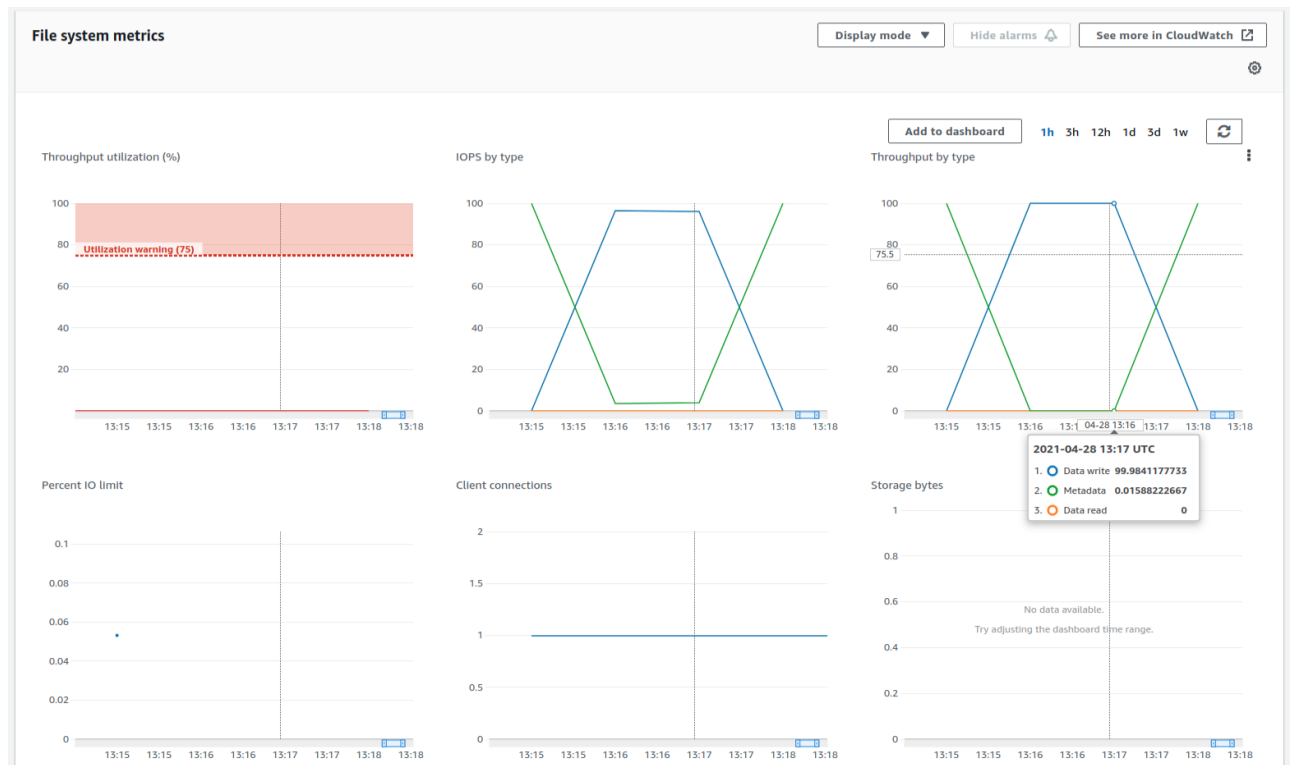


Figure 8. Read/Write metrics obtained from the EFS control dashboard.

If the commands for this step are executed successfully, and the file exists, along with metric data, pat yourself on the back. You've just created your first Network File Share system!

## Step 6: Delete resources

Even though we've used free tier resources in this tutorial, it's always good practice to delete them when we're finished using them.

1. First, terminate your Amazon EC2 instance. Navigate to the EC2 Console, select your instance, click *Instance state*, then click *Stop Instance*.
2. Now, delete your file system from Amazon EFS. Select your file system and then click *Delete*.
3. Finally, delete your security group under the *Security groups* menu within the EC2 dashboard.

## 5. Conclusion

This train has built on previous material surrounding data retrieval. We expanded upon this foundation by introducing common methods to transfer files over a network. We looked at:

- how the File Transfer Protocol originated;
- how FTP works and the progression of FTP into SFTP and FTPS owing to security concerns with the original protocol;
- the NFS and SMB protocols used for sharing files between Unix-based and Windows operating systems;
- typical methods for integrating legacy systems with new applications for data access; and
- how to set up an NFS system using common AWS services such as EC2 and Elastic File System.

Data retrieval skills have immense utility within the industry. We encourage you to practise and grow your skills in the concepts we've touched on and more.

## References

### FTP

- [The 5 Best OpenSource FTP Clients](#)
- [What Is the Difference Between Active and Passive FTP](#)

## **NFS and SMB**

- [How to Set Up Multiprotocol NFS and SMB File Share Access](#)
- [SFTP vs. FTPS: What's the Best Protocol for Secure FTP?](#)
- [The Difference Between SSH and SSL](#)

## **Legacy systems**

- [Seven Legacy Integration Patterns](#)
- [Integrating Data from Legacy Systems Using Object Linking and Embedding Technology](#)