TCP CLIENT

```
import java.net.*;
import java.io.*;
public class Client {
    private Socket socket = null;
    private DataInputStream input = null;
    private DataOutputStream out = null;
    public Client(String address, int port) {
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");
            input = new DataInputStream(System.in);
            out = new DataOutputStream(socket.getOutputStream());
        } catch (UnknownHostException u) {
            System.out.println(u);
        } catch (IOException i) {
            System.out.println(i);
        }
        String line = "";
        while (!line.equals("Over")) {
            try {
                line = input.readLine();
                out.writeUTF(line);
            } catch (IOException i) {
                System.out.println(i);
            }
        }
        try {
            input.close();      out.close();        socket.close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }
    public static void main(String args[]) {
        Client client = new Client("127.0.0.1", 5000);
    }
}
```

TCP SERVER:

```
import java.net.*;
import java.io.*;
public class Server {
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;
    public Server(int port) {
        try {
            server = new ServerSocket(port);
            System.out.println("Server started");
            System.out.println("Waiting for a client ...");
            socket = server.accept();
            System.out.println("Client accepted"); in = new
DataInputStream(new
bufferedInputStream(socket.getInputStream()));
            String line = "";
            while (!line.equals("Over")) {
                try {
                    line = in .readUTF();
                    System.out.println(line);
                } catch (IOException i) {
                    System.out.println(i);
                }
            }
            System.out.println("Closing connection");
            socket.close(); in .close();
        } catch (IOException i) {
            System.out.println(i);
        }
    }
    public static void main(String args[]) {
        Server server = new Server(5000);
    }
}
```

UDP CLIENT

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String args[]) throws
Exception {
        BufferedReader inFromUser = new
BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new
DatagramSocket();
        InetAddress IPAddress =
InetAddress.getByName("localhost");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new
String(receivePacket.getData());
        System.out.println("FROM SERVER:" +
modifiedSentence);
        clientSocket.close();
    }
}
```

UDP SERVER:

```
import java.io.*;
import java.net.*;
class UDPServer {
    public static void main(String args[]) throws
Exception {
        DatagramSocket serverSocket = new
DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while (true) {
            DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new
String(receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress =
receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence =
sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length,
IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

MD5WebService.java :

```
package vce.webservices.server;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.jws.WebMethod;
import javax.jws.WebService;
@WebService
public class MD5WebService {
    @WebMethod
    public String hashString(String input) {
        try {
            MessageDigest msgDigest =
MessageDigest.getInstance("MD5");
            byte[] inputBytes = input.getBytes();
            byte[] hashedBytes =
msgDigest.digest(inputBytes);
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < hashedBytes.length; i++) {
                sb.append(Integer.toString((hashedBytes[i]
& 0xff) + 0x100, 16)
                    .substring(1));
            }
            return sb.toString();
        } catch (NoSuchAlgorithmException ex) {
            ex.printStackTrace();
            return "";
        }
    }
}
```

WebServiceServer.java :

```
package vce.webservices.server;
import javax.xml.ws.Endpoint;
public class WebServiceServer {
    /*Starts a simple server to deploy the web service*/
    public static void main(String[] args) {
        String bindingURI =
"http://localhost:9898/md5WebService";
        MD5WebService webService = new
MD5WebService();
        Endpoint.publish(bindingURI, webService);
        System.out.println("Server started at: " +
bindingURI);
    }
}
```

WebServiceClient.java :

```
package vce.webservices.client;
public class WebServiceClient {
    /** * Starts the web service client. */
    public static void main(String[] args) {
        MD5WebServiceService client = new
MD5WebServiceService();
        MD5WebService md5Webservice =
client.getMD5WebServicePort();
        String hash =
md5Webservice.hashString("hyderabad");
        System.out.println("MD5 hash string: " + hash);
    }
}
```

2PC Server.java:

```
import java.io.*;
import java.net.*;
class Clients {
    static int n;
    static String[] status = new String[2];
    Clients(int num) {
        n = num;
        for (int j = 0; j < n; j++) {
            status[j] = new String("NotPrepared");
        }
    }
}
class Coordinator implements Runnable {
    public static int i = -1;  int flag = 1;
    Socket s;  Thread t;
    MulticastSocket ms = null; InetAddress group;
    Coordinator(Socket c) {
        try {
            ms = new MulticastSocket(8899);
            group = InetAddress.getByName("228.5.6.7");
            ms.joinGroup(group);
        } catch (Exception e) {
            e.printStackTrace();
        }
        s = c;  t = new Thread(this);  t.start();  i++;
    }
    public void run() {
        int index = i;  String clientSattus;
        try {
            DataInputStream input = new
            DataInputStream(s.getInputStream());
            DataOutputStream output = new
            DataOutputStream(s.getOutputStream());
            while (true) {
                clientSattus = input.readUTF();
                System.out.println("Client " + index + " " +
clientSattus);
                Clients.status[index] = new String(clientSattus);
                for (int k = 0; k < Clients.n; k++) {
                    System.out.println(Clients.status[k]);
                    if (Clients.status[k].equalsIgnoreCase("prepared"))
                        continue;
                    else
                        flag = 0;
                    if (flag == 1) {
                        byte[] msg = new String("commit").getBytes();
                        DatagramPacket msgpack = new
                        DatagramPacket(msg, msg.length, group, 8899);
                        ms.send(msgpack);
                        System.out.println("BroadCasted msg " + new
String(msg));
                    }
                    flag = 1;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    class Server {
        public static ServerSocket ss;
        Server() {}
        public static void main(String args[]) throws Exception {
            ss = new ServerSocket(8088);  int num;
            num = Integer.parseInt(args[0]);
            new Clients(num);
            while (true) {
                System.out.println("Server waiting: ");
                Socket s = ss.accept();  new Coordinator(s);
            }
        }
    }
```

2PC Client.java:

```
import java.io.*;
import java.net.*; import java.awt.*; import java.awt.event.*;
import javax.swing.*; import java.sql.*; import java.util.*;
class Client extends JFrame implements ActionListener {
    JButton b1, b2, b4, b5; JTextField t1; JLabel l1;
    ServerSocket ss; Socket s; DataOutputStream output;
    DataInputStream input; Connection con; Statement stmt;
    String serverMessage = "Prepared";
    static int port;
    Client() {
        b1 = new JButton("Prepared");
        b2 = new JButton("NotPrepared");
        b4 = new JButton("Execute");
        b5 = new JButton("Exit");
        t1 = new JTextField("", 35);
        l1 = new JLabel("SQL");
        p1 = new JPanel(); p2 = new JPanel();
        p1.setLayout(new FlowLayout());
        p1.add(l1); p1.add(t1); p2.add(b2); p2.add(b4); p2.add(b5);
        add(p1); add(p2, "South"); setSize(600, 300);
        setTitle("DNS Client");
        setVisible(true);
        b1.addActionListener(this);  b2.addActionListener(this);
        b4.addActionListener(this);  b5.addActionListener(this);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        MulticastSocket ms = null;
        InetAddress group;
        try {
            s = new Socket("localhost", 8088);
            System.out.println("Client Connected");
            output = new DataOutputStream(s.getOutputStream());
            input = new DataInputStream(s.getInputStream());
            con = DBConnector.getDBConnection("mydb");
            stmt = con.createStatement();
            con.setAutoCommit(false);
            ms = new MulticastSocket(8899);
            group = InetAddress.getByName("228.5.6.7");
            ms.joinGroup(group);
            byte[] buffer = new byte[1024];
            while (true) {
                DatagramPacket serMsg = new DatagramPacket(buffer,
buffer.length);
                ms.receive(serMsg);
                String commitMsg = new
String(serMsg.getData()).trim();
                if (commitMsg.equals("commit")) {
                    System.out.println("Received " + commitMsg);
                    con.commit();
                    System.out.println("Transactions Committed");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void actionPerformed(ActionEvent ae) {
        try {
            String str = ae.getActionCommand();
            if (str.equals("Execute")) {
                String query = t1.getText();
                System.out.println(stmt.executeUpdate(query));
                t1.setText("Query Executed(NotPrepared)");
                output.writeUTF("NotPrepared");
            }
            if (str.equals("Prepared")) {
                output.writeUTF("Prepared");
            }
        }
    }
}
```

Query.txt :

```
update account set bal = 500 where accno = 1
```

2PC MulticastPeer.java:

```
import java.net.*;  import java.io.*;
public class MulticastPeer {
    public static void main(String args[]) {
        MulticastSocket s = null;
        try {
            InetAddress group = InetAddress.getByName(args[1]);
            s = new MulticastSocket(6789); s.joinGroup(group);
            byte[] m = args[0].getBytes();
            DatagramPacket messageOut = new
DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);  byte[] buffer = new byte[1024];
            for (int i = 0; i < 3; i++) {
                DatagramPacket messageIn = new
DatagramPacket(buffer, buffer.length); s.receive(messageIn);
                System.out.println("Received:" + new
String(messageIn.getData()));
            }
            s.leaveGroup(group);
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (s != null) s.close();
        }
    }
}
```

2PC DBConnector.java:

```
import java.sql.*;
class DBConnector {
    public static Connection getDBConnection(String dsn) throws
Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        return DriverManager.getConnection("jdbc:odbc:" + dsn);
    }  }
```

MULTICHAT Server.java

```
import java.io.*;
import java.util.*;
import java.net.*;
public class Server {
    static Vector < ClientHandler > ar = new Vector < > ();
    static int i = 0;
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(1234);
        Socket s;
        while (true) {
            s = ss.accept();
            System.out.println("New client request received : " + s);
            DataInputStream dis = new
DataInputStream(s.getInputStream());
            DataOutputStream dos = new
DataOutputStream(s.getOutputStream());
            System.out.println("Creating a new handler for this
client...");
            ClientHandler mtch = new ClientHandler(s, "client " + i,
dis, dos);
            Thread t = new Thread(mtch);
            System.out.println("Adding this client to active client
list");
            ar.add(mtch);  t.start();  i++;
        }
    }
}
class ClientHandler implements Runnable {
    Scanner scn = new Scanner(System.in);  private String
name;
    final DataInputStream dis;  final DataOutputStream dos;
    Socket s; boolean isloggedin;
    public ClientHandler(Socket s, String name,
        DataInputStream dis, DataOutputStream dos) {
        this.dis = dis; this.dos = dos; this.name = name;
        this.s = s; this.isloggedin = true;
    }
    @Override
    public void run() {
        String received;
        while (true) {
            try {
                received = dis.readUTF();
System.out.println(received);
                if (received.equals("logout")) {
                    this.isloggedin = false; this.s.close();
                    break;
                }
                StringTokenizer st = new StringTokenizer(received,
"#");
                String MsgToSend = st.nextToken(); String recipient =
st.nextToken();
                for (ClientHandler mc: Server.ar) {
                    if (mc.name.equals(recipient) && mc.isloggedin ==
true) {
                        mc.dos.writeUTF(this.name + " : " +
MsgToSend);
                        break;
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        try {
            this.dis.close(); this.dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MULTICHAT Client.java

```
import java.io.*; import java.net.*; import java.util.Scanner;
public class Client {
    final static int ServerPort = 1234;
    public static void main(String args[]) throws
UnknownHostException, IOException {
        Scanner scn = new Scanner(System.in);
        InetAddress ip = InetAddress.getByName("localhost");
        Socket s = new Socket(ip, ServerPort);
        DataInputStream dis = new
DataInputStream(s.getInputStream());
        DataOutputStream dos = new
DataOutputStream(s.getOutputStream());
        Thread sendMessage = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    String msg = scn.nextLine();
                    try {
                        dos.writeUTF(msg);
                    } catch (IOException e) {
                        e.printStackTrace(); } } }

        });
        Thread readMessage = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    try {
                        String msg = dis.readUTF();
System.out.println(msg);
                    } catch (IOException e) {
                        e.printStackTrace(); } } }
        });
        sendMessage.start();  readMessage.start();
    }
}
```

Lab Experiment

Hosting a Static Website

Accessing the AWS Management Console

1. At the top of these instructions, choose Start Lab to launch your lab.

   A **Start Lab** panel opens, and it displays the lab status.

2. Wait until the **Start Lab** panel displays the message *Lab status: ready*, then close the panel by choosing the **X**.
3. At the top of these instructions, choose AWS.

   This action opens the AWS Management Console in a new browser tab. The system automatically logs you in.

4. Arrange the **AWS Management Console** tab so that it displays alongside these instructions. Ideally, you will have both browser tabs open at the same time so that you can follow the lab steps more easily.

   Do not change the Region unless specifically instructed to do so.

Task 1: Creating a bucket in Amazon S3

In this task, you will create an S3 bucket and configure it for static website hosting.

In the **AWS Management Console**, on the **Services** menu, choose **S3**.

5. Choose **Create bucket**

   An S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. After you create a bucket, the name of that bucket cannot be used by another AWS account in any AWS Region unless you delete the bucket.

   Thus, for this lab, you will use a bucket name that includes a random number, such as: *website-123*

6. For **Bucket name**, enter: website-<123> (replace *<123>* with a random number)

   Public access to buckets is blocked by default. Because the files in your static website will need to be accessible through the internet, you must permit public access.

   ◦ Verify the **AWS Region** is set to **us-east-1** (if it is not, choose the us-east-1 Region)
7. In the **Object Ownership** section, select **ACLs enabled**, then verify **Bucket owner preferred** is selected.

8. Clear **Block all public access**, then select the box that states **I acknowledge that the current settings may result in this bucket and the objects within becoming public**.
9. Choose **Create bucket**.

   You can use tags to add additional information to a bucket, such as a project code, cost centre, or owner.

10. Choose the name of your new bucket.
11. Choose the **Properties** tab.
12. Scroll to the **Tags** panel.
13. Choose Edit then Add tag and enter:

- **Key:** Department
- **Value:** Marketing

14. Choose **Save changes** to save the tag.

    Next, you will configure the bucket for static website hosting.

15. Stay in the **Properties** console.
16. Scroll to the **Static website hosting** panel.
17. Choose Edit
18. Configure the following settings:
    ◦ **Static web hosting:** Enable
    ◦ **Hosting type:** Host a static website
    ◦ **Index document:** index.html
      ▪ **Note:** You must enter this value, even though it is already displayed.
    ◦ **Error document:** error.html
19. Choose **Save changes**
20. In the **Static website hosting** panel, choose the link under **Bucket website endpoint**.

    You will receive a *403 Forbidden* message because the bucket permissions have not been configured yet. Keep this tab open in your web browser so that you can return to it later.

    Your bucket has now been configured to host a static website.

Task 2: Uploading content to your bucket

In this task, you will upload the files that will serve as your static website to the bucket.

21. Right-click each of these links and download the files to your computer:

    Ensure that each file keeps the same file name, including the extension.

    ◦ index.html

    ◦ script.js
    ◦ style.css
22. Return to the Amazon S3 console and in the website-<123> bucket you created earlier, choose the **Objects** tab.
23. Choose **Upload.**
24. Choose Add files
25. Locate and select the three files that you downloaded.
26. If prompted, choose I acknowledge that existing objects with the same name will be overwritten.
27. Choose **Upload.**

    Your files are uploaded to the bucket.

    ◦ Choose **Close**

Task 3: Enabling access to the objects

Objects that are stored in Amazon S3 are private by default. This ensures that your organization's data remains secure.

In this task, you will make the uploaded objects publicly accessible.

First, confirm that the objects are currently private.

28. Return to the browser tab that showed the *403 Forbidden* message.
29. Refresh the webpage

    You should still see a *403 Forbidden* message.

    Analysis: This response is expected! This message indicates that your static website is being hosted by Amazon S3, but that the content is private.

    You can make Amazon S3 objects public through two different ways:

    ◦ To make either a whole bucket public, or a specific directory in a bucket public, use a *bucket policy*.
    ◦ To make individual objects in a bucket public, use an *access control list (ACL)*.
30. Return to the web browser tab with the Amazon S3 console (but do not close the website tab).
31. Select all three objects.
32. In the Actions menu, choose **Make public via ACL**.

    A list of the three objects is displayed.

33. Choose Make public

Your static website is now publicly accessible.

34. Return to the web browser tab that has the *403 Forbidden* message.
35. Refresh the webpage.

You should now see the static website that is being hosted by Amazon S3.

Task 4: Updating the website

You can change the website by editing the HTML file and uploading it again to the S3 bucket.

36. On your computer, load the **index.html** file into a text editor (for example, Notepad or TextEdit).
37. Find the text **Served from Amazon S3** and replace it with Created by <YOUR-NAME>, substituting your name for *<YOUR-NAME>* (for example, *Created by Jane*).
38. Save the file.
39. Return to the Amazon S3 console and upload the **index.html** file that you just edited.
40. Select **index.html** and use the **Actions** menu to choose the **Make public via ACL** option again.
41. Return to the web browser tab with the static website and refresh the page.

    Your name should now be on the page.

Your static website is now accessible on the internet. Because it is hosted on Amazon S3, the website has high availability and can serve high volumes of traffic without using any servers.

You can also use your own domain name to direct users to a static website that is hosted on Amazon S3. To accomplish this, you could use the Amazon Route 53 Domain Name System (DNS) service in combination with Amazon S3.

Submitting your work

42. At the top of these instructions, choose **Submit** to record your progress and when prompted, choose **Yes**.
43. If the results don't display after a couple of minutes, return to the top of these instructions, and choose Grades
44. To find detailed feedback on your work, choose Details followed by **View Submission Report**.

Lab complete

45. Choose End Lab at the top of this page, and then select **Yes** to confirm that you want to end the lab.

    A panel indicates that *DELETE has been initiated... You may close this message box now.*

46. Select the **X** in the top right corner to close the panel.

Lab Experiment

Introducing Amazon Elastic File System (Amazon EFS)

Accessing the AWS Management Console

1. At the top of these instructions, choose Start Lab to launch your lab.

   A **Start Lab** panel opens, and it displays the lab status.

   Tip: If you need more time to complete the lab, restart the timer for the environment by choosing the Start Lab button again.

2. Wait until the **Start Lab** panel displays the message *Lab status: ready*, then close the panel by choosing the **X**.
3. At the top of these instructions, choose AWS.
4. Arrange the **AWS Management Console** tab so that it displays alongside these instructions. Ideally, you will have both browser tabs open at the same time so that you can follow the lab steps more easily.

Task 1: Creating a security group to access your EFS file system

5. In the **AWS Management Console**, on the Services menu, choose **EC2**.
6. In the navigation pane on the left, choose **Security Groups**.
7. Copy the **Security group ID** of the *EFSClient* security group to your text editor.

   The Group ID should look similar to *sg-03727965651b6659b*.

8. Choose Create security group then configure:
   - **Security group name:** EFS Mount Target
   - **Description:** Inbound NFS access from EFS clients
   - **VPC:** *Lab VPC*
9. Under the **Inbound rules** section, choose **Add rule** then configure:
   - **Type:** *NFS*
   - **Source:**
     - *Custom*
     - In the *Custom* box, paste the security group's **Security group ID** that you copied to your text editor
   - Choose Create security group.

Task 2: Creating an EFS file system

10. On the Services menu, choose **EFS**.
11. Choose Create file system
12. In the **Create file system** window, choose **Customize**

13. On **Step 1**:
    - Uncheck Enable automatic backups.
    - **Lifecycle management:** Select *None*
    - In the **Tags** section, configure:
      - **Key:** Name
      - **Value:** My First EFS File System
14. Choose Next
15. For **VPC**, select *Lab VPC*.
16. Detach the default security group from each *Availability Zone* mount target by choosing the check box on each default security group.
17. Attach the **EFS Mount Target** security group to each *Availability Zone* mount target by:

- Selecting each **Security groups** check box.
- Choosing **EFS Mount Target**

  A mount target is created for each subnet

18. Choose Next
19. On **Step 3**, choose Next
20. On **Step 4:**
- Review your configuration.
- Choose Create

Proceed to the next step after the **Mount target state** for each mount target changes to *Available*. Choose the screen refresh button after 2–3 minutes to check its progress.

Task 3: Connecting to your EC2 instance via SSH

In this task, you will connect to your EC2 instance by using Secure Shell (SSH).

21. Above these instructions that you are currently reading, choose the Details dropdown menu, and then select Show

A **Credentials** window opens.

22. Choose the **Download PPK** button and save the **labsuser.ppk** file.

Note: Typically, your browser saves the file to the **Downloads** directory.

23. Note the **EC2PublicIP** address if it is displayed.
24. Exit the **Details** panel by choosing the **X**.
25. To use SSH to access the EC2 instance, you must use ***PuTTY***. If you do not have PuTTY installed on your computer, download PuTTY.
26. Open **putty.exe**.
27. To keep the PuTTY session open for a longer period of time, configure the PuTTY timeout:
- Choose **Connection**
- **Seconds between keepalives**: 30
28. Configure your PuTTY session by using the following settings.

- Choose **Session**
- **Host Name (or IP address):** Paste the **EC2PublicIP** for the instance you noted earlier
  - Alternatively, return to the Amazon EC2 console and choose **Instances**
  - Select the instance you want to connect to
  - In the *Description* tab, copy the **IPv4 Public IP** value
- Back in PuTTY, in the **Connection** list, expand **SSH**
- Choose **Auth** (but don't expand it)
- Choose **Browse**
- Browse to the *labsuser.ppk* file that you downloaded, select it, and choose **Open**
- Choose **Open** again

29. To trust and connect to the host, choose **Yes**.
30. When you are prompted with **login as**, enter: ec2-user.

    This action connects you to the EC2 instance.

Task 4: Creating a new directory and mounting the EFS file system

31. In your SSH session, make a new directory by entering `sudo mkdir efs`
32. Back in the **AWS Management Console**, on the Services menu, choose **EFS**.
33. Choose **My First EFS File System**.
34. In the **Amazon EFS Console**, on the top right corner of the page, choose Attach to open the Amazon EC2 mount instructions.
35. Copy the entire command in the **Using the NFS client** section.

    The mount command should look similar to this example:

```
sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport fs-
bce57914.efs.us-west-2.amazonaws.com:/ efs
```

    The provided `sudo mount...` command uses the default Linux mount options.

36. In your Linux SSH session, mount your Amazon EFS file system by:
    - Pasting the command
    - Pressing ENTER

37. Get a full summary of the available and used disk space usage by entering:

```
sudo df -hT
```

Task 5: Examining the performance behavior of your new EFS file system

38. Examine the write performance characteristics of your file system by entering:

```
              sudo fio --name=fio-efs --filesize=10G --filename=./efs/fio-efs-test.img --bs=1M --nrfiles=1 --direct=1 -
-sync=0 --rw=write --iodepth=200 --ioengine=libaio
```

Monitoring performance by using Amazon CloudWatch

39. In the **AWS Management Console**, on the Services menu, choose **CloudWatch**.
40. In the navigation pane on the left, choose **Metrics**.
41. In the **All-metrics** tab, choose **EFS**.
42. Choose **File System Metrics**.
43. Select the row that has the **PermittedThroughput** Metric Name.

    You might need to wait 2–3 minutes and refresh the screen several times before all available metrics, including **PermittedThroughput**, calculate and populate.

44. On the graph, choose and drag around the data line. If you do not see the line graph, adjust the time range of the graph to display the period during which you ran the `fio` command.

45. Pause your pointer on the data line in the graph. The value should be *105M*.
46. In the **All-metrics** tab, *uncheck* the box for **PermittedThroughput**.
47. Select the check box for **DataWriteIOBytes**.

    If you do not see *DataWriteIOBytes* in the list of metrics, use the **File System Metrics** search to find it.

48. Choose the **Graphed metrics** tab.
49. On the **Statistics** column, select **Sum**.
50. On the **Period** column, select **1 Minute**.
51. Pause your pointer on the peak of the line graph. Take this number (in bytes) and divide it by the duration in seconds (60 seconds). The result gives you the write throughput (B/s) of your file system during your test.