# Tech Assessment - Integration Engineer - IT Operations - DataStealth

**Task 1: Network Debugging**

Here’s a step-by-step guide to diagnose the issue:

**1. Open Developer Tools**

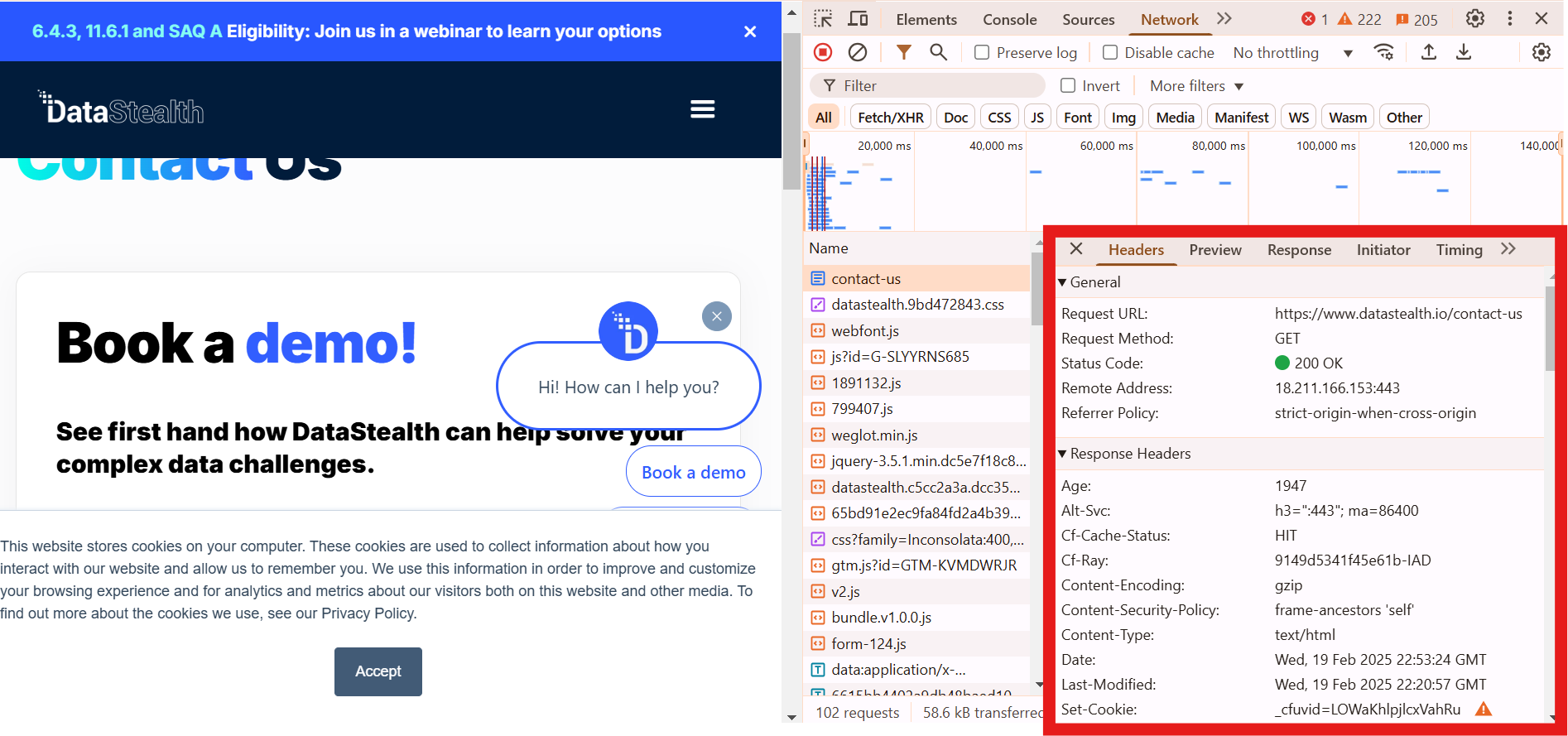
* **Shortcut**: Press F12 or Ctrl + Shift + I (Windows) / Cmd + Option + I (Mac).
* Navigate to the **Network** tab.



**2. Check HTTP Request and Response Status Codes**

* **Reproduce the Issue**: Submit the form while the Network tab is open.
* **Filter Requests**: Look for the form submission request (usually a POST request).
* **Status Codes**:
  + **2xx (e.g., 200)**: Success.
  + **4xx (e.g., 400, 403, 404)**: Client-side error (e.g., invalid data, unauthorized).
  + **5xx (e.g., 500)**: Server-side error (e.g., server crash, misconfiguration).
* **Intermittent Failures**: Look for requests that alternate between success (2xx) and failure (4xx/5xx).

**3. Inspect Headers to Confirm Data is Sent Correctly**

****

* **Select the Request**: Click on the form submission request in the Network tab.
* **Headers Tab**:
  + **Request Headers**: Check if Content-Type matches the data being sent
  + **Response Headers**: Check for errors or warnings in the server’s response.

**4. Debug Potential CORS Issues**

* **CORS Errors**: Look for requests blocked due to CORS policy (visible in the **Console** tab with errors like Access-Control-Allow-Origin).
* **Headers**:
  + Check if the server includes Access-Control-Allow-Origin in the response headers.
  + Ensure the origin in the request matches the allowed origins on the server.
* **Preflight Requests**: Look for OPTIONS requests (preflight checks) and verify their responses.

**5. Use the Console Tab**

* **Errors**: Check for JavaScript errors or warnings that might indicate issues with form submission logic.
* **Network Errors**: Look for failed network requests logged in the console.
* **Debugging**: Add console.log() statements in the form submission JavaScript to trace the flow and identify where it fails.

**Deliverables**

**Diagnosis Steps**

1. Open Developer Tools → **Network** tab.
2. Submit the request and inspect the request/response status codes.
3. Check request headers to ensure data is sent correctly.
4. Look for CORS errors in the **Console** tab or preflight requests in the **Network** tab.
5. Use the **Console** tab to debug JavaScript errors or network failures.

**Resolving the Issue**

* **4xx Errors**: Fix client-side issues (e.g., validate form data, check authentication).
* **5xx Errors**: Investigate server logs for crashes or misconfigurations.
* **CORS Issues**: Update server-side CORS policies to allow the correct origins.
* **Intermittent Failures**: Check for race conditions, network instability, or server load issues.

By combining insights from the **Network** and **Console** tabs, we can pinpoint the root cause and resolve the issue effectively.

**Task2: Network Configuration - Subnet Calculation (**192.168.0.0/24)

### ****1. Usable IP Addresses in 192.168.0.0/24****

* **Subnet Mask**: /24 means the first 24 bits are for the network, leaving 8 bits for hosts.
* **Total IPs**: 28=256 addresses.
* **Usable IPs**: Subtract 2 (network and broadcast addresses).
  + **Usable IPs**: 256−2=254.

### ****2. Dividing into Two Subnets (/25)****

* **New Subnet Mask**: /25 means the first 25 bits are for the network, leaving 7 bits for hosts.
* **Total IPs per Subnet**: 27=128 addresses.
* **Usable IPs per Subnet**: Subtract 2 (network and broadcast addresses).
  + **Usable IPs**: 128−2=126 in each network
* Subnet 1: 192.168.0.0/25 (126 usable IPs).
* Subnet 2: 192.168.0.128/25 (126 usable IPs)

**Note:** Each /25 subnet can accommodate **126 usable hosts**.

### ****3. Brief Explanation of Subnet****

**Subnet calculation** is the process of dividing a larger IP network into smaller, more manageable sub-networks (subnets). It involves calculating:

* **Subnet Mask**: Determines the network and host portions of an IP address (e.g /24, /25).
* **Network Address**: Identifies each subnet.
* **Broadcast Address**: Used to communicate with all devices in a subnet.
* **Range of IPs**: Assignable IP addresses within each subnet.

**Benefits**:

* + Improves network performance by reducing congestion.
  + Enhances security by isolating traffic.
  + Efficiently allocates IP addresses.

**Task 3: File Management**

**1. Commands**

**Archive and Compress a Directory**

We can use tar to create a compressed archive of the logs directory:

tar –czvf logs.tar.gz logs/

* -c: Create a new archive.
* -z: Compress using gzip.
* -v: Verbose output (optional).
* -f: Specify the archive filename.

1. **Securely Transfer to a Remote Server**

Use scp to transfer the archive to a remote server:

scp logs.tar.gz user@”remote\_server”:/”path-to-destination”

* Replace user with your remote server username.
* Replace remote\_server with the server's IP or hostname.
* Replace /path-to-destination/ with the target directory on the remote server.

Alternatively, use rsync for more robust transfers:

rsync -avz --progress logs.tar.gz user@”remote\_server”:/”path-to-destination”

* -a: Archive mode (preserves permissions, timestamps, etc.).
* -v: Verbose output.
* -z: Compress during transfer.
* --progress: Show transfer progress.

1. **Uncompress and Extract on the Remote Server**

SSH into the remote server and extract the archive:

ssh user@remote-server then do the below command to extract

tar –xzyf /path-to-destination/logs.tar.gz –C /path/customerlogs

* -x: Extract the archive.
* -C: Specify the target directory (customerlogs).

1. **Resuming an Interrupted Transfer**

We can use**rsync** if the transfer is interrupted, rsync can resume from where it left off:

rsync –avz –partial –progress logs.tar.gz user@remote\_server:/path-to-destination/

* --partial: Keeps partially transferred files, allowing resumption.
* -a: Archive mode (preserves permissions, timestamps, etc.).
* -v: Verbose output.
* -z: Compress during transfer.
* --progress: Show transfer progress.

**Summary**

1. **Compress**: tar -czvf logs.tar.gz logs/
2. **Transfer**: scp logs.tar.gz user@remote\_server:/path/to/destination/ or rsync -avz --progress logs.tar.gz user@remote\_server:/path/to/destination/
3. **Extract**: tar -xzvf /path/to/destination/logs.tar.gz -C /path/to/customerlogs/
4. **Resume**: Use rsync --partial for resumable transfers.

**Task 4: HTTPS Configuration**

To ensure your web service supports only **TLS 1.2** and disables older, insecure protocols (e.g., SSLv2, SSLv3, TLS 1.0, TLS 1.1), follow these steps:

**1. Configure the Server (e.g., Nginx)**

**Steps**

1. **Edit Nginx Configuration**:
   * Locate the Nginx configuration file (usually /etc/nginx/nginx.conf or /etc/nginx/sites-available/default).
   * Modify the server block handling HTTPS traffic.
2. **Set SSL Protocols**:
   * Use the ssl\_protocols directive to specify TLSv1.2 and disable older protocols.
3. **Set Strong Cipher Suites**:
   * Use the ssl\_ciphers directive to enforce strong encryption algorithms.
4. **Reload Nginx**:
   * Apply the changes by reloading the Nginx configuration.

**2. Example Nginx Configuration**

server {

listen 443 ssl;

server\_name yourdomain.com;

# SSL Certificate and Key

ssl\_certificate /etc/ssl/certs/yourdomain.crt;

ssl\_certificate\_key /etc/ssl/private/yourdomain.key;

# Enforce TLS 1.2 only

ssl\_protocols TLSv1.2;

# Strong cipher suites

ssl\_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256';

ssl\_prefer\_server\_ciphers on;

# Enable HSTS (Optional but recommended)

add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

# Other configurations

location / {

root /var/www/html;

index index.html;

}

}

**3. Reload Nginx**

After making changes, reload Nginx to apply the new configuration:

sudo nginx -t # Test configuration for syntax errors

sudo systemctl reload nginx # Reload Nginx

**4. Verify Configuration**

Use tools like **SSL Labs** (<https://www.ssllabs.com/ssltest/>) or the openssl command to verify that only TLS 1.2 is supported:

openssl s\_client -connect yourdomain.com:443 -tls1\_2

**Key Points**

* **ssl\_protocols TLSv1.2;**: Enforces TLS 1.2 only.
* **ssl\_ciphers**: Specifies strong encryption algorithms.
* **HSTS**: Adds an HTTP header to enforce HTTPS for all future requests.

This setup ensures your web service is secure and compliant with modern encryption standards.

**Task 5: Troubleshooting Error Messages**

**Troubleshooting Error: curl: (7) Failed to connect to remote\_server.com port 443: No route to host**

This error indicates that curl cannot establish a connection to the specified host (remote\_server.com) on port 443 (HTTPS). Below are the probable causes and steps to resolve the issue:

**Probable Causes**

1. **Network Connectivity Issues**:
   * The host (remote\_server.com) is unreachable from your machine.
   * There may be a network misconfiguration, firewall rule, or routing issue.
2. **DNS Resolution Failure**:
   * The domain name (remote\_server.com) cannot be resolved to an IP address.
3. **Firewall Blocking**:
   * A firewall (on your machine, network, or the server) is blocking traffic to port 443.
4. **Server Down or Misconfigured**:
   * The server hosting remote\_server.com is down or not listening on port 443.
5. **Incorrect Host or Port**:
   * The hostname or port number is incorrect.

**Steps to Identify and Resolve the Issue**

**i. Check Network Connectivity**

* **Ping the Host**:

ping remote\_server.com

* + If the host is unreachable, there may be a network issue.
* **Traceroute**:

traceroute remote\_server.com

* + Identify where the connection is failing.

**2. Verify DNS Resolution**

* Use nslookup or dig to check if the domain resolves to an IP:

nslookup remote\_server.com

* + If no IP is returned, there may be a DNS issue.

**3. Check Firewall Rules**

* **Local Firewall**:
  + Ensure your machine allows outbound traffic on port 443.
  + For example, on Linux:

sudo ufw allow 443

* **Remote Firewall**:
  + Ensure the server allows inbound traffic on port 443.

**4. Test Port Connectivity**

* Use telnet or nc to check if port 443 is open:

telnet remote\_server.com 443

* + If the connection fails, the server may not be listening on port 443.

**5. Verify Server Status**

* Check if the server hosting remote\_server.com is up and running.
* If you have access to the server, verify that the web service (e.g., Nginx, Apache) is running and configured to listen on port 443.

**6. Check for Typos**

* Ensure the hostname (remote\_server.com) and port (443) are correct.

**Example Debugging Workflow**

1. **Ping the Host**:

ping remote\_server.com

* + If unreachable, check your network connection.

1. **Resolve DNS**:

nslookup remote\_server.com

* + If no IP is returned, check your DNS settings.

1. **Test Port 443**:

telnet remote\_server.com 443

* + If the connection fails, check the server's firewall and service status.

1. **Check Local Firewall**:

sudo ufw status

* + Ensure port 443 is allowed.

**Resolution**

* Fix the underlying issue (e.g., network connectivity, DNS, firewall, or server configuration).
* Retry the curl command:

curl https://remote\_server.com