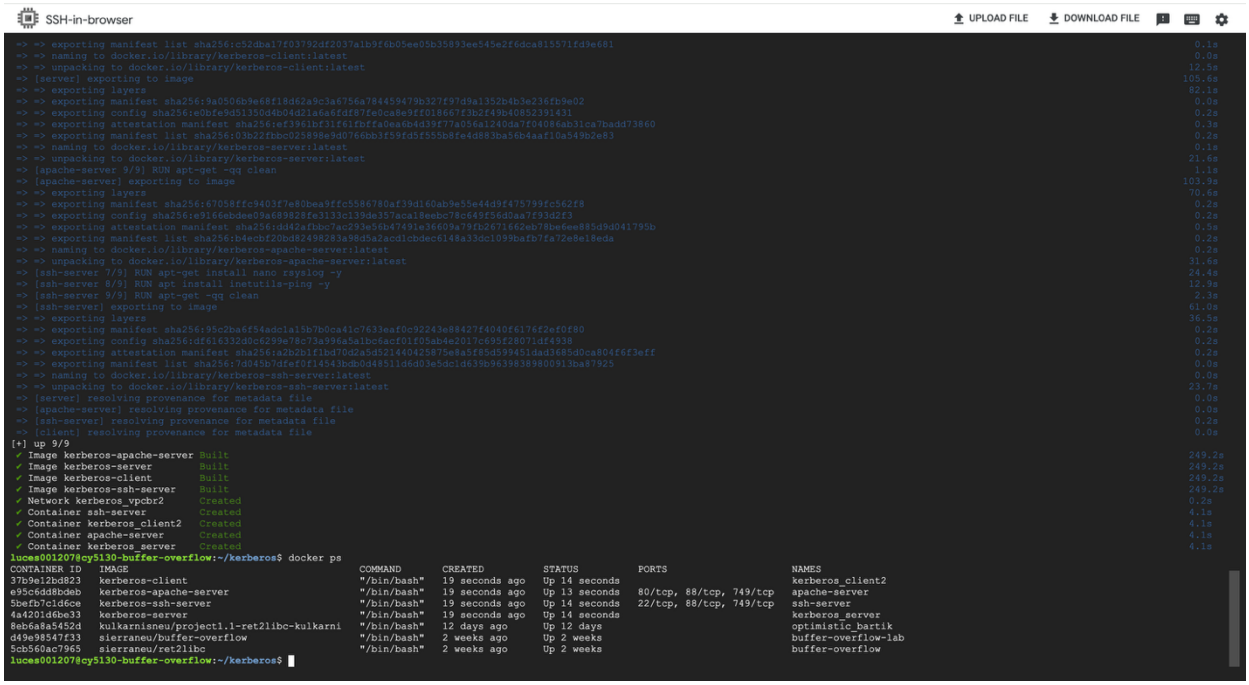
Project 2: Enterprise Authentication

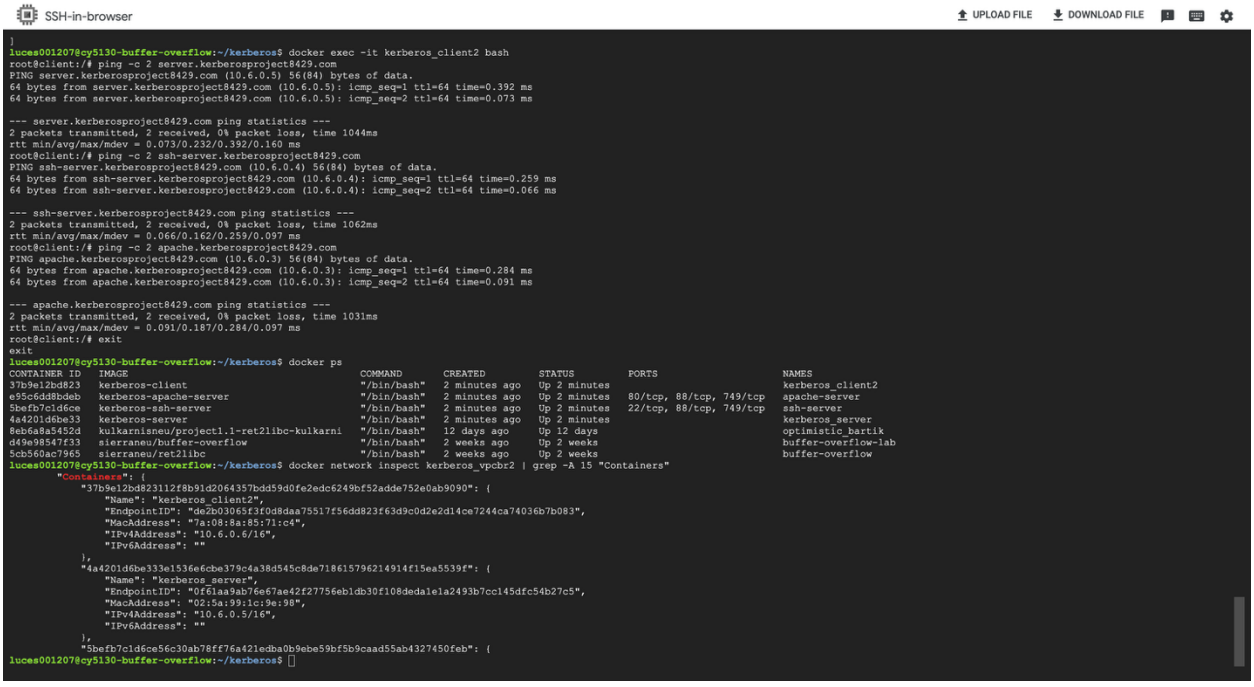
PART 1: SETTING UP DOCKER CONTAINERS

Screenshot 1: This screenshot shows I have successfully made a new directory called “Kerberos” and inside have the subdirectory like “server”, “client”, “ssh-server”, “apache-server” and “docker-compose.yaml”.All these will need to use it next to build the separate container.

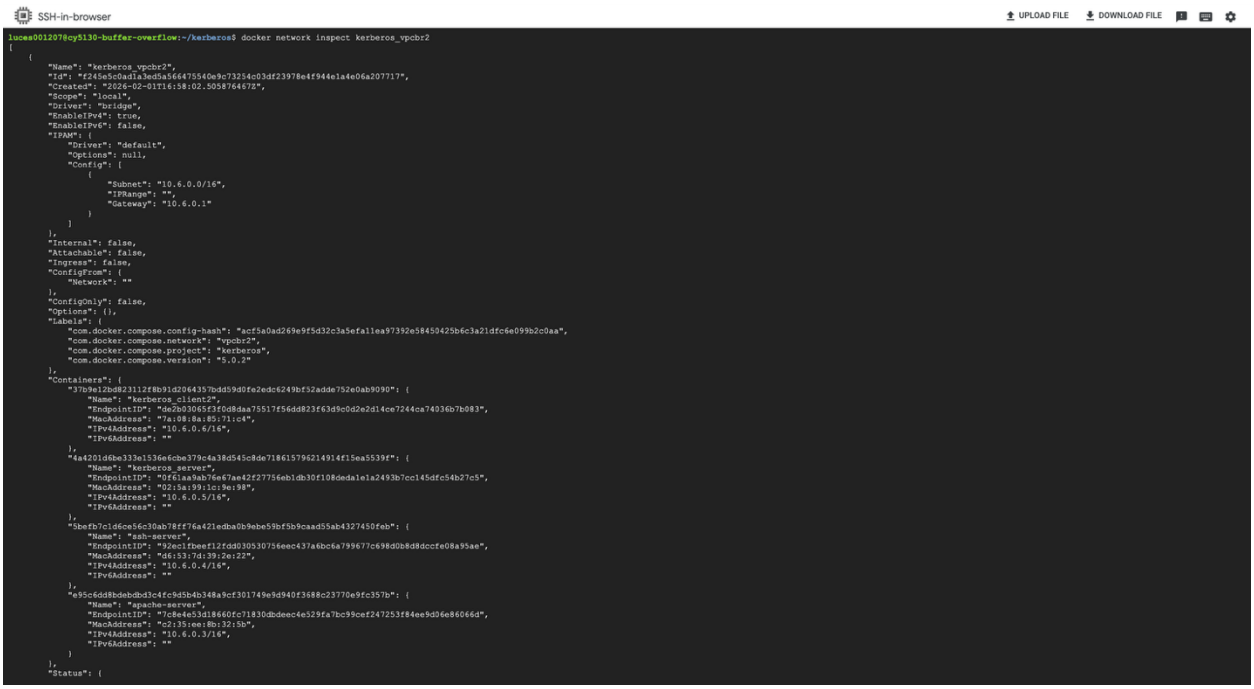
Screenshot 2: Docker compose building all four containers (server, kerberos_client2, ssh-server, apache-server) from Ubuntu 18.04 base images with required packages (preassigned static IP address and DNS resolution). This also shows successful creation of Docker network "kerberos_vpcbr2" and all four containers starting with status "done".



Screenshot 3: This shows an output of ***docker ps*** showing all four containers running with preassigned IP addresses (10.6.0.3-6) and exposed ports. I also have done the ping tests from kerberos_client2 container to other three containers and it shows successfully reaching all servers, confirming network connectivity and DNS resolution.



Screenshot 4: This shows the successful inspection of kerberos_vpcbr2 in Docker network.



PART 2: KERBEROS SERVER CONFIGURATION IN SERVER CONTAINER

Screenshot 5: KDC configuration file ***/etc/krb5kdc/kdc.conf*** showing that the realm name in the file is the same as the desired realm name (kerberosproject8429.com) for the project and granted with 10-hour ticket lifetime.

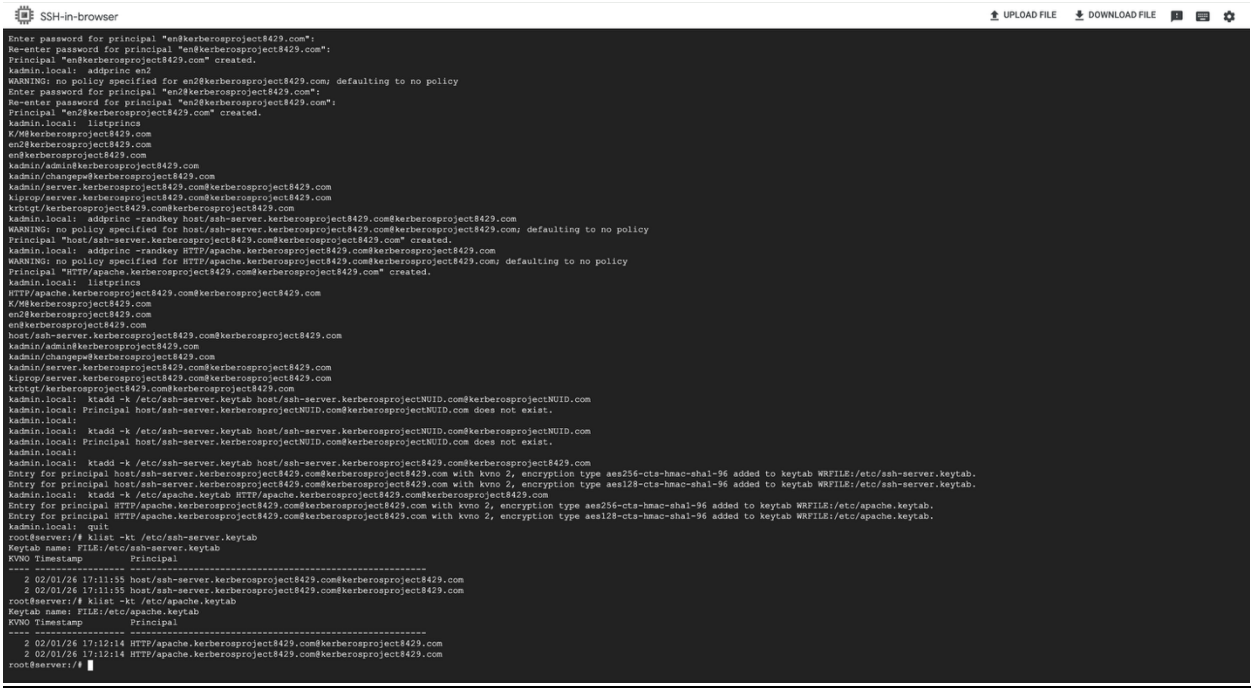
```
root@server:/# cat /etc/krb5kdc/kdc.conf
[kdcdefaults]
    kdc_ports = 750,88

[realms]
    kerberosproject8429.com = {
        database_name = /var/lib/krb5kdc/principal
        admin_keytab = FILE:/etc/krb5kdc/kadm5.keytab
        acl_file = /etc/krb5kdc/kadm5.acl
        key_stash_file = /etc/krb5kdc/stash
        kdc_ports = 750,88
        max_life = 10h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        master_key_type = des3-hmac-sha1
        #supported_enctypes = aes256-cts:normal aes128-cts:normal
        default_principal_flags = +preauth
    }
root@server:/#
```

PART 3: KERBEROS DATABASE CONFIGURATION

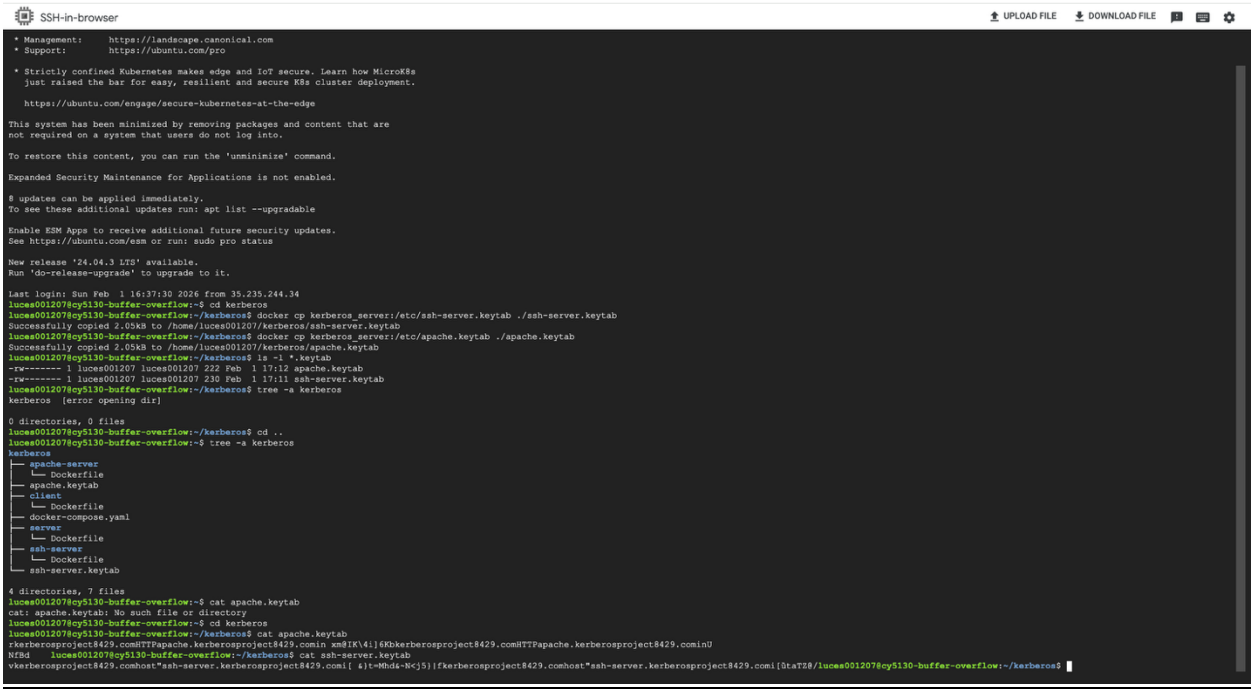
Screenshot 6: Creating Kerberos database and adding user's principals (en (user1), en2 (user2)) and host principal (SSH service, Apache service) using ***kadmin.local.***

```
SSH-in-browser                                                                                    UPLOAD FILE  DOWNLOAD FILE
kadmin.local:  listprincs
K/M@kerberosproject8429.com
en2@kerberosproject8429.com
en@kerberosproject8429.com
kadmin/admin@kerberosproject8429.com
kadmin/changepw@kerberosproject8429.com
kadmin/server.kerberosproject8429.com@kerberosproject8429.com
kiprop/server.kerberosproject8429.com@kerberosproject8429.com
krbtgt/kerberosproject8429.com@kerberosproject8429.com
kadmin.local:  addprinc -randkey host/ssh-server.kerberosproject8429.com@kerberosproject8429.com
WARNING: no policy specified for host/ssh-server.kerberosproject8429.com@kerberosproject8429.com; defaulting to no policy
Principal "host/ssh-server.kerberosproject8429.com@kerberosproject8429.com" created.
kadmin.local:  addprinc -randkey HTTP/apache.kerberosproject8429.com@kerberosproject8429.com
WARNING: no policy specified for HTTP/apache.kerberosproject8429.com@kerberosproject8429.com; defaulting to no policy
Principal "HTTP/apache.kerberosproject8429.com@kerberosproject8429.com" created.
kadmin.local:  listprincs
HTTP/apache.kerberosproject8429.com@kerberosproject8429.com
K/M@kerberosproject8429.com
en2@kerberosproject8429.com
en@kerberosproject8429.com
host/ssh-server.kerberosproject8429.com@kerberosproject8429.com
kadmin/admin@kerberosproject8429.com
kadmin/changepw@kerberosproject8429.com
kadmin/server.kerberosproject8429.com@kerberosproject8429.com
kiprop/server.kerberosproject8429.com@kerberosproject8429.com
krbtgt/kerberosproject8429.com@kerberosproject8429.com
kadmin.local:  ktadd -k /etc/ssh-server.keytab host/ssh-server.kerberosprojectNUID.com@kerberosprojectNUID.com
kadmin.local: Principal host/ssh-server.kerberosprojectNUID.com@kerberosprojectNUID.com does not exist.
kadmin.local:
kadmin.local:  ktadd -k /etc/ssh-server.keytab host/ssh-server.kerberosprojectNUID.com@kerberosprojectNUID.com
kadmin.local: Principal host/ssh-server.kerberosprojectNUID.com@kerberosprojectNUID.com does not exist.
kadmin.local:
kadmin.local:  ktadd -k /etc/ssh-server.keytab host/ssh-server.kerberosproject8429.com@kerberosproject8429.com
Entry for principal host/ssh-server.kerberosproject8429.com@kerberosproject8429.com with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/ssh-server.keytab.
Entry for principal host/ssh-server.kerberosproject8429.com@kerberosproject8429.com with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/etc/ssh-server.keytab.
kadmin.local:  ktadd -k /etc/apache.keytab HTTP/apache.kerberosproject8429.com@kerberosproject8429.com
Entry for principal HTTP/apache.kerberosproject8429.com@kerberosproject8429.com with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/apache.keytab.
Entry for principal HTTP/apache.kerberosproject8429.com@kerberosproject8429.com with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/etc/apache.keytab.
kadmin.local:  quit
root@server:/# klist -kt /etc/ssh-server.keytab
Keytab name: FILE:/etc/ssh-server.keytab
KVNO Timestamp         Principal
---- ----------------- --------------------------------------------------------
   2 02/01/26 17:11:55 host/ssh-server.kerberosproject8429.com@kerberosproject8429.com
   2 02/01/26 17:11:55 host/ssh-server.kerberosproject8429.com@kerberosproject8429.com
root@server:/# klist -kt /etc/apache.keytab
Keytab name: FILE:/etc/apache.keytab
KVNO Timestamp         Principal
---- ----------------- --------------------------------------------------------
   2 02/01/26 17:12:14 HTTP/apache.kerberosproject8429.com@kerberosproject8429.com
   2 02/01/26 17:12:14 HTTP/apache.kerberosproject8429.com@kerberosproject8429.com
root@server:/# service krb5-kdc restart
 * Restarting Kerberos KDC krb5kdc                                                                [ OK ]
root@server:/# service krb5-admin-server restart
 * Restarting Kerberos administrative servers kadmind                                             [ OK ]
root@server:/# service krb5-kdc status
 * krb5kdc is running
root@server:/# service krb5-admin-server status
 * kadmind is running
root@server:/#
```

Screenshot 7: Generating keytab files for SSH and Apache services with AES256/AES128 encryption keys.

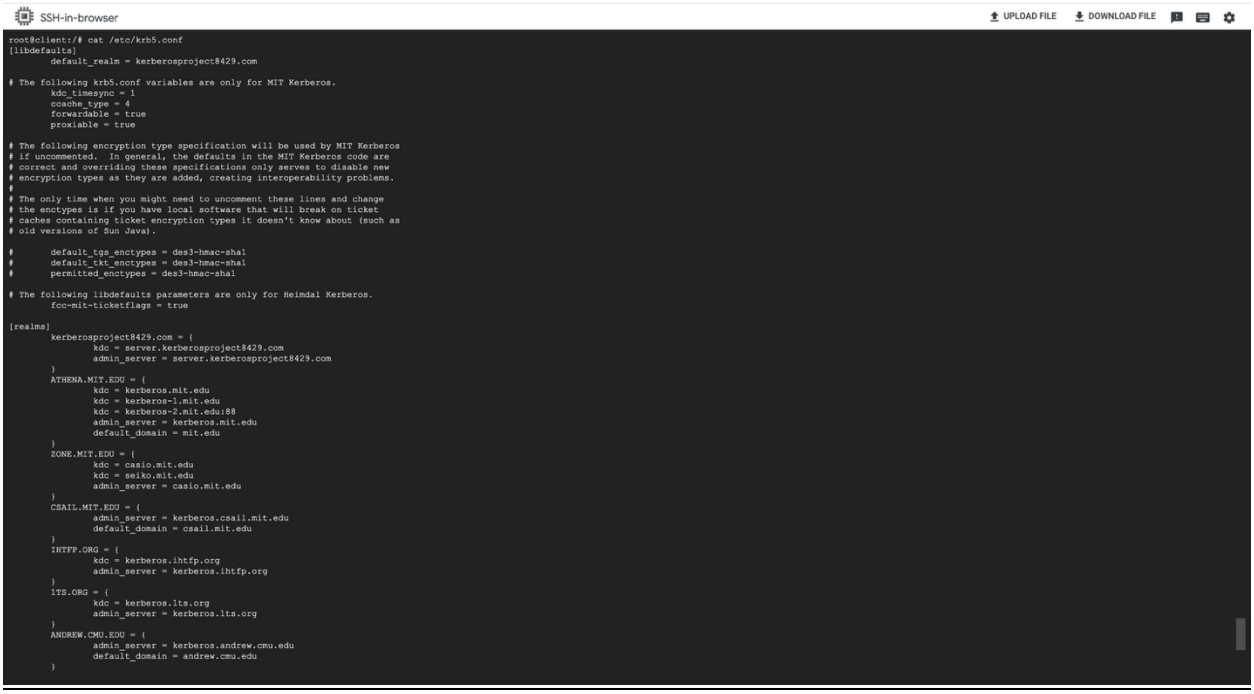


Screenshot 8: The Apache and SSH keytab files were successfully copied from the KDC container to the local host for transfer. Verification confirmed that the keytabs contained the correct service principals and that KDC services were running. Because the keytabs are generated inside the kerberos_server container but must be installed on the SSH and Apache server containers, the host machine serves as an intermediary to move the files between containers.

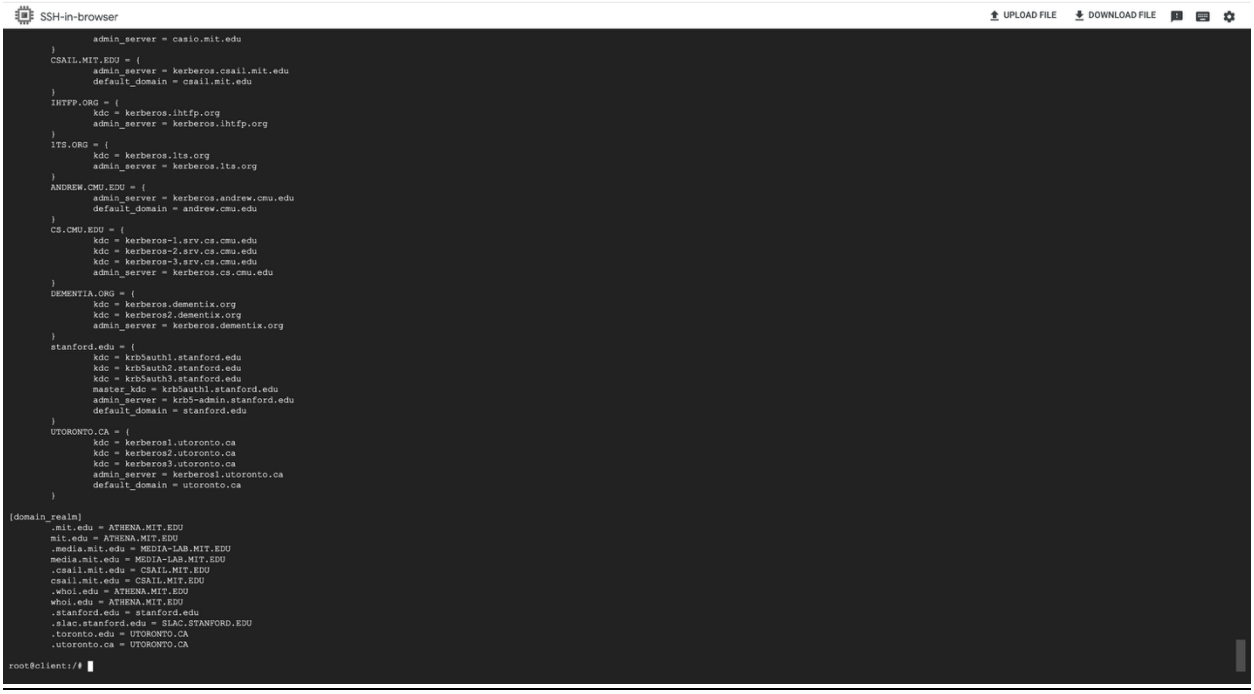


PART 4: CLIENT SETUP WITHIN THE CONTAINER

Screenshot 9: Client's ***/etc/krb5.conf*** configuration showing default realm (kerberosproject8429.com) and KDC/admin server information is the same as those desired for this project.



Screenshot 10: Continuation of client ***/etc/krb5.conf*** showing domain-to-realm mappings for automatic realm detection.



PART 5: SSH-SERVER SETUP WITHIN THE CONTAINER

Screenshot 11: I created a user account (en) on the ssh-server container and set a password for this account. After creating the user, I tested it by logging into the server container as this user. I then edited the /etc/ssh/sshd_config file to enable GSSAPI/Kerberos authentication, with KerberosTicketCleanup set to "no."

- I configure /etc/ssh/sshd_config to enable Kerberos authentication because SSH doesn't use Kerberos tickets by default. Without these settings (GSSAPIAuthentication yes, KerberosAuthentication yes), SSH will ignore the user's Kerberos ticket and always prompt for a password, defeating Single Sign-On. The configuration tells SSH to check for and validate Kerberos tickets first using the server's keytab, enabling passwordless authentication when a valid ticket is presented.

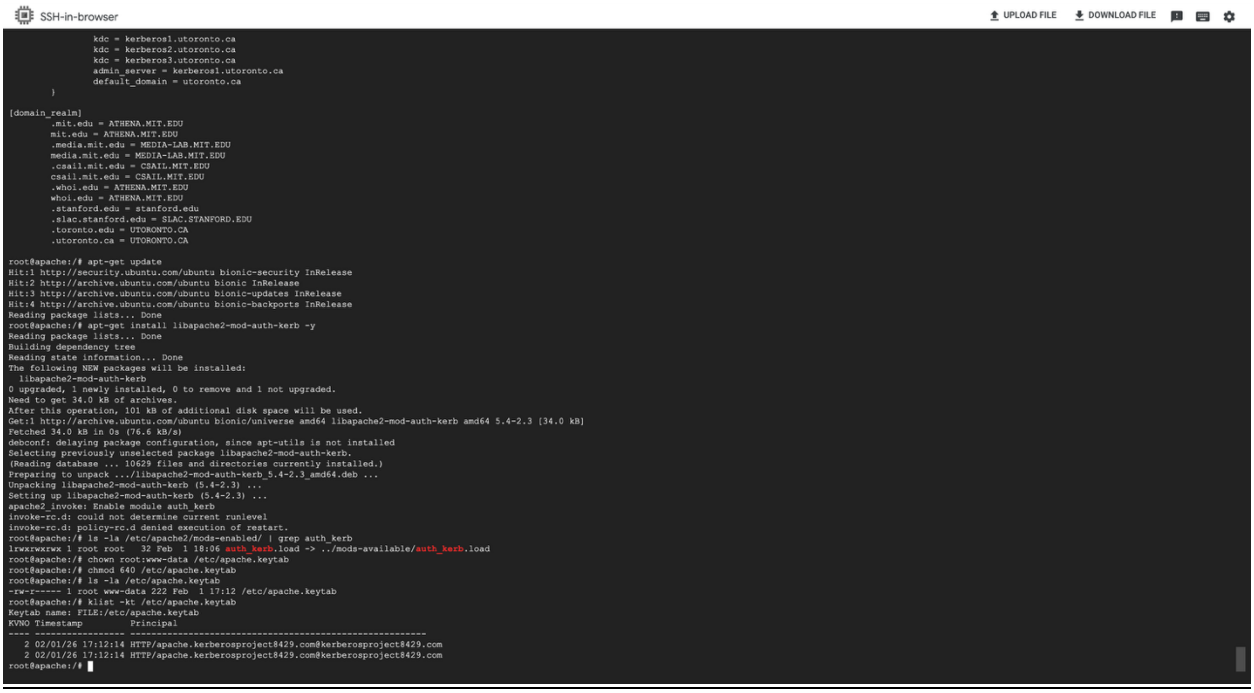
Screenshot 12: SSH keytab (***/etc/krb5.keytab***)file permissions showing 600 (owner read/write only) for security. The krb5.keytab file has 600 permissions (owner read/write only), owned by root:root. These restrictive permissions are critical for security because the krb5.keytab contains the SSH server's secret key used to decrypt Kerberos tickets. If this file were compromised, an attacker could impersonate the SSH server.

Screenshot 13: SSH keytab contents displaying host principal with AES256/AES128 encryption keys. Multiple encryption types ensure compatibility with various Kerberos clients. The KVNO

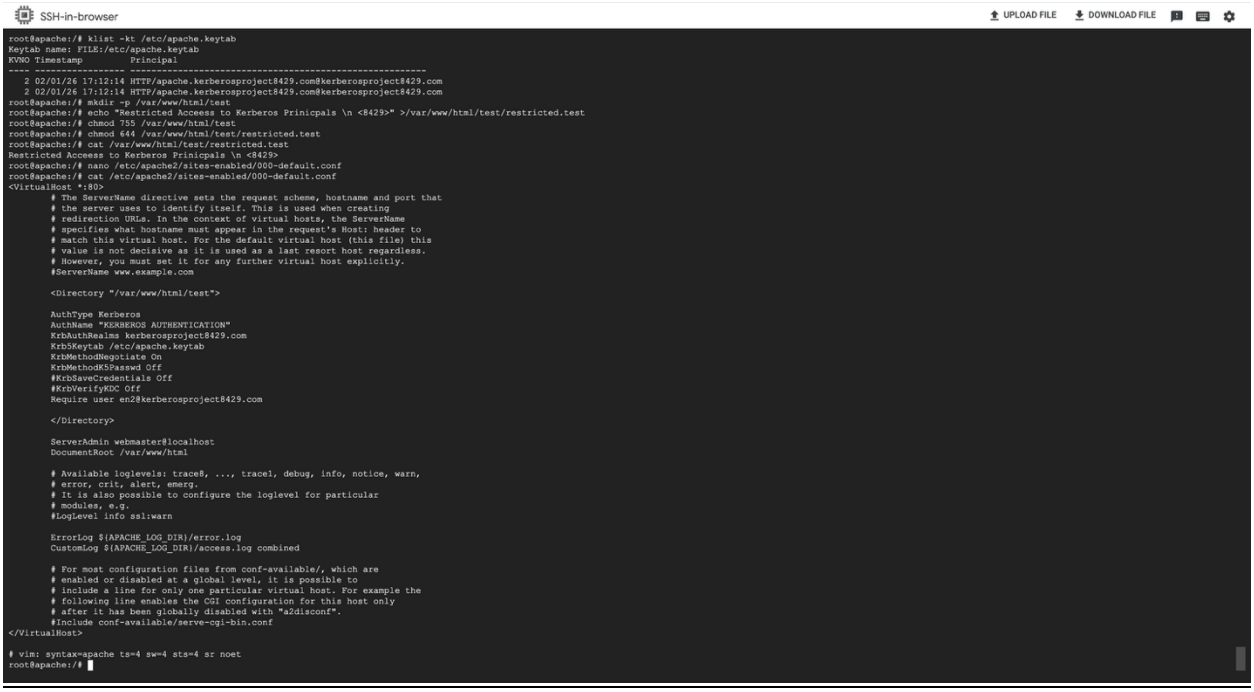
(Key Version Number) of 2 indicates this is the second version of the keytab, allowing for key rotation without breaking existing tickets.

PART 6: APACHE server SETUP WITHIN THE CONTAINER

Screenshot 14: Apache Kerberos module installation and keytab permissions showing 640 (root:www-data). The keytab has 640 permissions with root:www-data ownership because Apache runs as the www-data user and needs to read the keytab to validate tickets, but shouldn't be able to write or modify it for security.

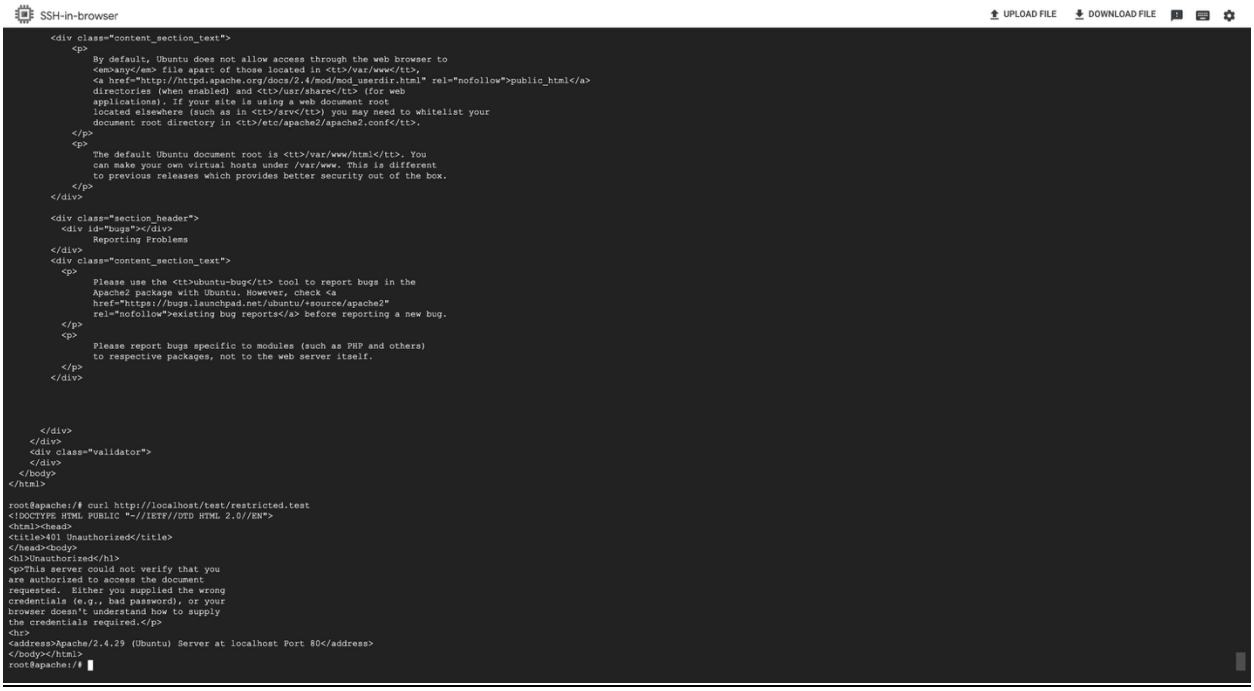


Screenshot 15: Complete Apache Kerberos configuration in ***/etc/apache2/sites-enabled/000-default.conf.*** Create a file to check if Kerberos authentication is working and restrict access to all kerberos principals and allow only en2 to access the website test directory because this user is the only one in the ACL. Setting permissions to 755 for directories and 644 for files ensures www-data can read and serve the content while maintaining security (only root can modify).



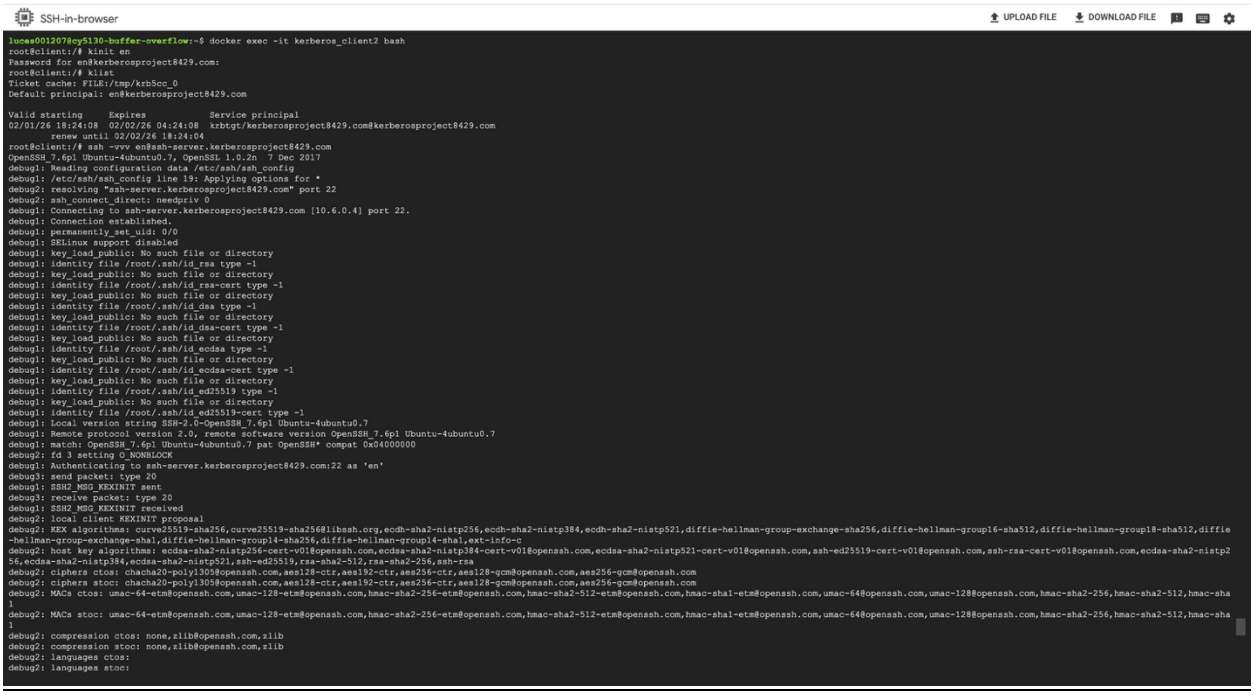
Screenshot 16: This shows a successful restart after the configuration was modified.

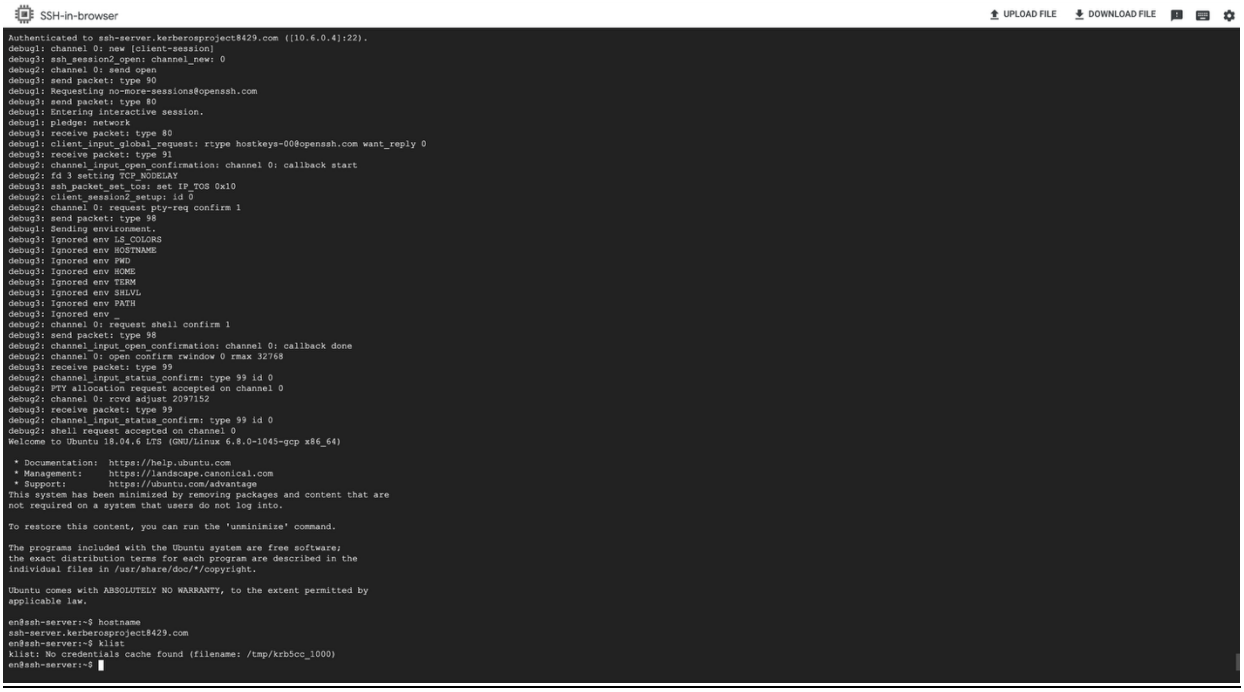
Screenshot 17: I use en (user1) to test the Kerberos authentication, although en(user1) granted the server ticket but because it's not in the ACL. Thus, en (user1) can't be successfully authorized and it returns access denied.



PART 7: SSH KERBEROS AUTHENTICATION FROM CLIENT (DOCKER CONTAINER)

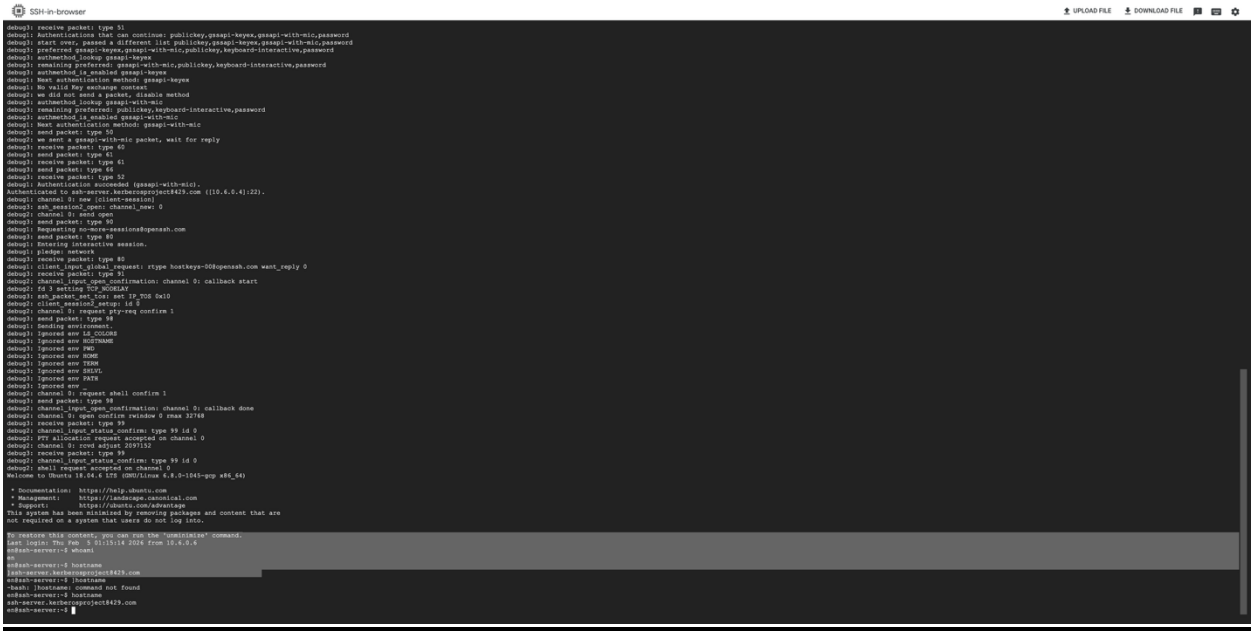
Screenshot 18: Request a TGT from the KDC for the user (en) principal, and verify if the TGT is being granted with klist. Once the TGT is verified, I initiate an SSH connection using ssh -vvv showing successful GSSAPI authentication without password prompt.





login in again on Feb 4 to provided a cleaner capture

I Connect to the server container from the client container using the ssh command and highlight the line which shows that Kerberos/GSSAPI authentication was successful.



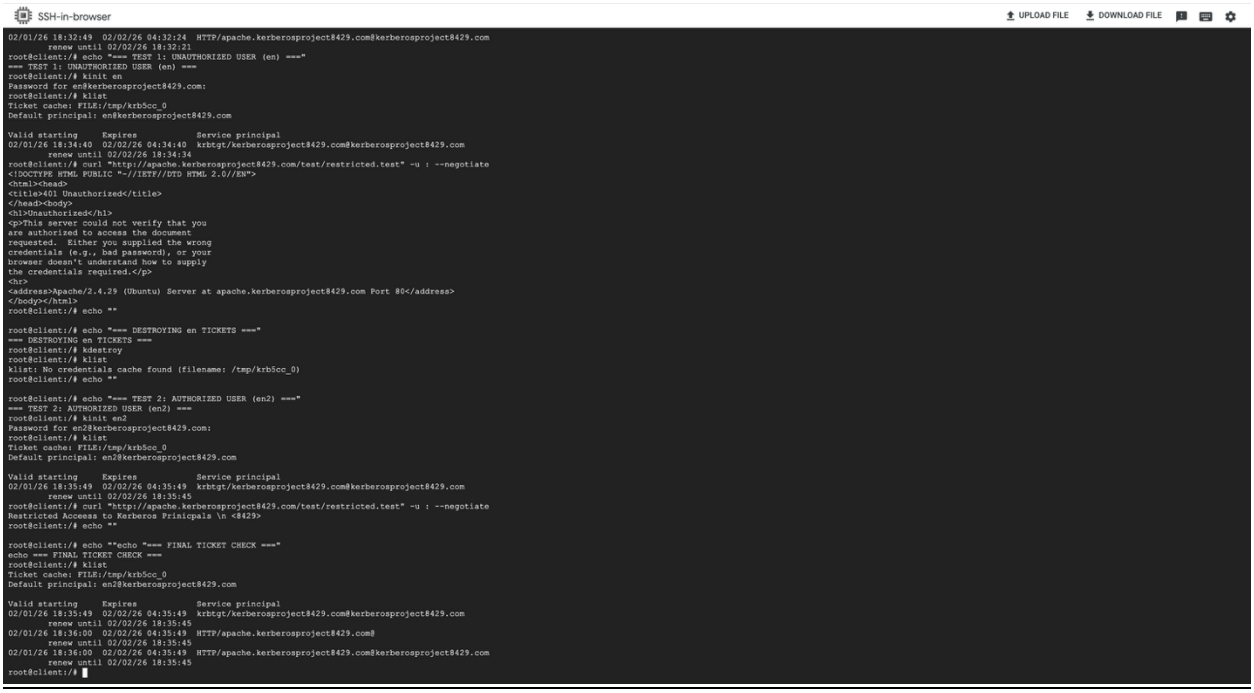
Screenshot 19: klist output showing both TGT and SSH service ticket after successful connection.

PART 8: APACHE KERBEROS AUTHENTICATION FROM CLIENT (DOCKER CONTAINER)

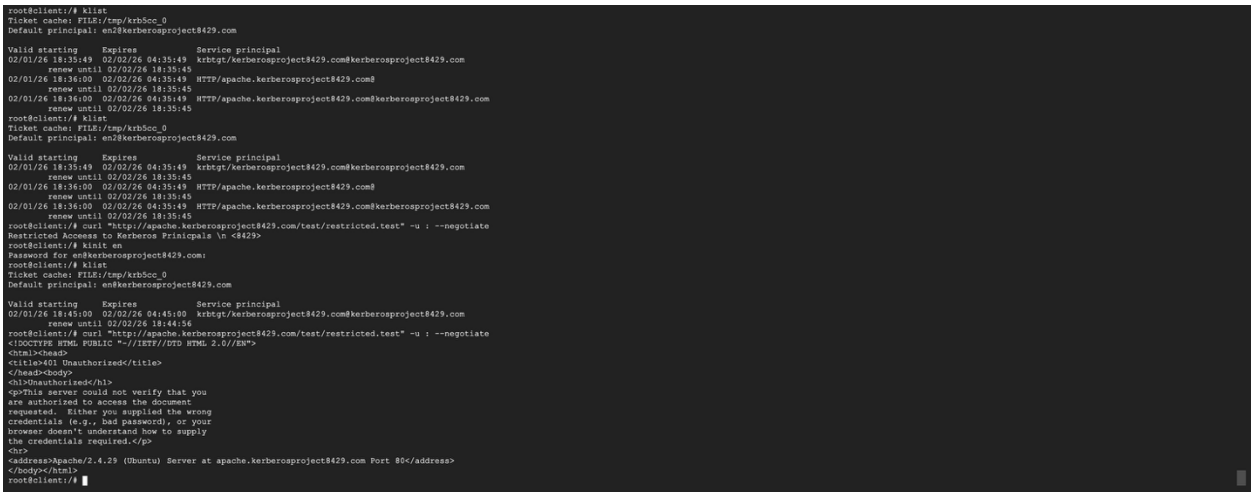
Screenshot 20: Authorized access succeeded for *en2*, and klist displays the final tickets. The curl command successfully accessed the restricted content. Content displayed: “Restricted

Access to Kerberos Principals \n <8429>.”The klist output shows both the TGT and the HTTP service ticket for the user *en2*.

Screenshot 21: This screenshot shows the complete authentication and authorization flow for Apache Kerberos access. First, en (user1) obtains a TGT via kinit and attempts to access the restricted directory, but receives a 401 Unauthorized response because en is not in the Apache ACL, demonstrating that authentication alone does not grant authorization. After using kdestroy to clear all tickets (confirmed by klist showing no credentials), en2 (user2) obtains a fresh TGT and successfully accesses the restricted content, returning "Restricted Access to Kerberos Principals <8429>." A final klist confirms that both the TGT and the HTTP service ticket were issued for en2.



Screenshot 22: en2 successfully accesses restricted content showing "NUID: 8429", with klist displaying both TGT and HTTP service ticket.



QUESTIONS:

Why do we use the docker network? How does the network help us to imitate the described scenario?

- We use Docker network to create an isolated environment for four containers to communicate like separate servers on an internal network. It assigns static IPs (10.6.0.3-6) and DNS resolution, necessary for Kerberos's hostname-based service principals. This setup mimics an organization with multiple Linux servers (via SSH and Apache containers) managed by administrators (client container) needing centralized KDC authentication. Without Docker network, containers couldn't communicate, preventing Kerberos authentication and Single Sign-On demonstration. The network enables containers to function as an enterprise infrastructure, allowing administrators to authenticate once and access all services, fulfilling the organization's goal of centralized Kerberos authentication.

You generated a ssh-server.keytab file on the kerberos KDC server.

(A) Why did you need to copy it into the SSH server?

- The SSH server needs the keytab file locally to validate Kerberos tickets without contacting the KDC each time. When a client presents a service ticket, the SSH server decrypts it uses the secret key in the keytab to verify and extract the username. The keytab contains the same key used by the KDC, creating a shared secret between them. Without the keytab on the SSH server, ticket validation would be impossible, forcing password fallback and weakening security. Local validation is safer (passwords aren't transmitted) and more efficient (no network to KDC).

(B) How many of these .keytab files are required in an architecture like ours and where are they located?

- Our architecture requires one keytab per service, totaling 2: one for the SSH server at /etc/krb5.keytab, with the principal host/ssh server.kerberosproject8429.com@ kerberosproject8429.com, and one for the Apache server at /etc/apache.keytab, with the principal HTTP/apache.kerberosproject8429.com@ kerberosproject8429.com. Scaling to 7 SSH and 7 Apache servers would need 14 keytabs, as each has a unique hostname and principal. User principals (en, en2) do not have keytabs, as humans authenticate by password, unlike services.

(C) How are they protected?

- Keytab files are protected with restrictive permissions because they contain secret keys that could allow impersonation. The SSH server's keytab has 600 permissions (-rw-------) with root: root ownership, so only root can access it, as the SSH daemon runs as root. The Apache server's keytab has 640 permissions (-rw-r-----) with root:www-data ownership, allowing the www-data user to read it while preventing non-root modifications. Additional protections include physical security, network isolation, SELinux/AppArmor, and regular keytab rotation. If stolen, an attacker could decrypt tickets and impersonate the server, so permissions enforce the principle of least privilege; only authorized services have access to the key.

The organization described in the scenario has 8 new administrators and extends their apache web servers' architecture from 2 to 7. What should they do?You can use the names of the parts of this assignment to simplify your explanation (e.g.: Part 2 should be completed but including the new host in the configuration file such as: …)

- To accommodate 8 new administrators and scale from 2 to 7 Apache servers, the organization should first update the infrastructure (Part 1) by adding 5 new Apache server containers with unique static IPs, hostnames, and DNS entries required for Kerberos principal resolution. The KDC configuration (Part 2) should be verified to ensure the new hostnames are resolvable. Following Part 3, use kadmin.local to create 8 new administrator principals (addprinc admin3 through admin10) and 5 new service principals (addprinc -randkey HTTP/apache3.company.com through apache7.company.com), generating individual keytabs for each new server. Each administrator's client (Part 4) should have /etc/krb5.conf configured to point to the KDC for TGT retrieval. For Part 6, each new Apache server needs Kerberos packages installed (krb5-user, libapache2-mod-auth-kerb), /etc/krb5.conf configured to point to the KDC, the appropriate keytab copied with 640 permissions (root:www-data), the VirtualHost configured with Kerberos authentication directives, and the ACL updated (Require user) to include the new administrators. Finally, local user accounts should be created with

adduser on servers where shell access is needed, though passwords are unnecessary since Kerberos handles authentication centrally.

Assume Northeastern University is adopting Kerberos when providing services. Explain how it works.

- If Northeastern University adopts Kerberos, it would establish a Kerberos realm (e.g., NEU.EDU) with centralized Key Distribution Center (KDC) servers that store credentials for all users. Students log in once with kinit, obtaining a Ticket Granting Ticket (TGT) valid for the day. This TGT enables seamless access to email, Canvas, library systems, Wi-Fi, SSH, and printing, without re-entering passwords, because service-specific tickets are automatically requested from the KDC. Each service has a keytab file to verify tickets, confirming user identity without revealing passwords. When accessing a service like Canvas, the browser requests a ticket from the KDC using the TGT, and Canvas validates the ticket with its keytab, granting access transparently. This Single Sign-On system reduces password management, enhances security by transmitting only passwords to the KDC, and centralizes access control, so revoking a single account disables all access.