How to setup AD login with Kerberos authentication from Linux server to Windows SQL Server instance

To set up **Active Directory (AD) login with Kerberos authentication** from a **Linux server to a Windows SQL Server instance**, you need to properly configure **Kerberos**, **SQL Server**, and **ODBC** on the Linux machine. Here's a step-by-step guide:

**✅ Overview**

You’re enabling a **Linux client (using Kerberos auth)** to connect to **SQL Server on Windows** using an **AD account**.

**🔧 1. Prerequisites**

* **SQL Server** running on **Windows**, joined to Active Directory.
* **Linux server** (e.g., Ubuntu, CentOS).
* An **Active Directory account** (e.g., domain\sqluser).
* Access to create SPNs and manage keytabs if needed.

**🧩 2. Set SPN for SQL Server (on the Windows side)**

Run on a domain controller (or with an account that can set SPNs):

bash

setspn -A MSSQLSvc/sqlserver.domain.com:1433 domain\sqlserviceaccount

* Replace sqlserver.domain.com with the **FQDN** of the SQL Server host.
* Replace domain\sqlserviceaccount with the service account running SQL Server.

📌 *Ensure the SQL Server instance is listening on port 1433 or adjust as needed.*

**🔐 3. Install Kerberos & ODBC dependencies on Linux**

**For Ubuntu/Debian:**

bash

sudo apt update sudo apt install krb5-user msodbcsql18 unixodbc

**For RHEL/CentOS:**

bash

sudo yum install krb5-workstation msodbcsql18 unixODBC

**🧾 4. Configure Kerberos (**/etc/krb5.conf**)**

Example config:

ini

[libdefaults] default\_realm = DOMAIN.COM dns\_lookup\_realm = true dns\_lookup\_kdc = true [realms] DOMAIN.COM = { kdc = your.ad.controller.domain.com admin\_server = your.ad.controller.domain.com } [domain\_realm] .domain.com = DOMAIN.COM domain.com = DOMAIN.COM

**🪪 5. Get a Kerberos Ticket**

Use your AD credentials:

bash

kinit youruser@DOMAIN.COM

It will prompt for the password. Verify with:

bash

klist

**💻 6. Test Connection with**sqlcmd**or ODBC**

Install sqlcmd if needed (part of mssql-tools):

bash

sqlcmd -S sqlserver.domain.com -d YourDB -G

* -G enables **Kerberos Integrated Authentication**.
* Ensure your ODBC DSN (if using) is configured with Authentication=ActiveDirectoryIntegrated.

**⚙️ 7. SQL Server Configuration**

Ensure SQL Server is in **Mixed Mode** or **Windows Authentication** mode and the login exists:

sql

*-- Run this on SQL Server* CREATE LOGIN [DOMAIN\youruser] FROM WINDOWS;

**🧪 Troubleshooting**

* Use klist to ensure a valid ticket exists.
* Use setspn -L youraccount to check SPNs.
* Check SQL Server logs for authentication failures.

Here’s a shell script that automates most of the Kerberos and ODBC setup on a Linux machine to connect to a Windows SQL Server using an Active Directory login and Kerberos authentication.

🔧 This script is targeted for Ubuntu/Debian-based systems. Let me know if you need one for RHEL/CentOS.

**🚀 setup\_sqlserver\_kerberos.sh**

bash

#!/bin/bash *# CONFIGURE THESE VARIABLES* AD\_REALM="DOMAIN.COM" AD\_USERNAME="youruser" SQL\_SERVER\_FQDN="sqlserver.domain.com" SQL\_DB="YourDatabase" KDC\_SERVER="ad.domain.com" echo "🔧 Installing dependencies..." sudo apt update && sudo apt install -y krb5-user msodbcsql18 unixodbc curl echo "🔐 Configuring /etc/krb5.conf..." sudo bash -c "cat > /etc/krb5.conf" <<EOF [libdefaults] default\_realm = $AD\_REALM dns\_lookup\_realm = true dns\_lookup\_kdc = true ticket\_lifetime = 24h forwardable = true [realms] $AD\_REALM = { kdc = $KDC\_SERVER admin\_server = $KDC\_SERVER } [domain\_realm] .$AD\_REALM = $AD\_REALM .${AD\_REALM,,} = $AD\_REALM EOF echo "🎫 Requesting Kerberos ticket for $AD\_USERNAME@$AD\_REALM" echo "You will be prompted for the AD password..." kinit "${AD\_USERNAME}@${AD\_REALM}" echo "📋 Verifying Kerberos ticket..." klist echo "🧪 Testing connection to SQL Server with sqlcmd..." sqlcmd -S "$SQL\_SERVER\_FQDN" -d "$SQL\_DB" -G -Q "SELECT SYSTEM\_USER, ORIGINAL\_LOGIN();" echo "✅ Setup complete."

**📝 Instructions**

1. Replace these variables in the script:
   * AD\_REALM → Your AD domain in uppercase (e.g. EXAMPLE.COM)
   * AD\_USERNAME → Your AD login username
   * SQL\_SERVER\_FQDN → The full domain name of your SQL Server
   * SQL\_DB → The name of a database you can access
   * KDC\_SERVER → The hostname or IP of your AD Domain Controller
2. Save the script:

bash

nano setup\_sqlserver\_kerberos.sh

1. Make it executable and run:

bash

chmod +x setup\_sqlserver\_kerberos.sh ./setup\_sqlserver\_kerberos.sh

Here’s the CentOS/RHEL-compatible version of the setup script for Kerberos authentication to SQL Server via Active Directory.

🚩 Compatible with RHEL 7/8/9 and CentOS 7. Assumes sudo/root privileges.

**🔧 setup\_sqlserver\_kerberos\_rhel.sh**

bash

#!/bin/bash *# CONFIGURE THESE VARIABLES* AD\_REALM="DOMAIN.COM" AD\_USERNAME="youruser" SQL\_SERVER\_FQDN="sqlserver.domain.com" SQL\_DB="YourDatabase" KDC\_SERVER="ad.domain.com" echo "🔧 Installing required packages..." sudo yum install -y krb5-workstation unixODBC curl *# Optional: Install MS ODBC Driver 18* if ! command -v sqlcmd &>/dev/null; then echo "📦 Installing Microsoft ODBC Driver 18 and sqlcmd..." *# Add Microsoft repo* sudo rpm -Uvh https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm *# Install the driver* sudo yum install -y msodbcsql18 mssql-tools18 *# Add to PATH* echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bash\_profile source ~/.bash\_profile fi echo "🔐 Configuring /etc/krb5.conf..." sudo bash -c "cat > /etc/krb5.conf" <<EOF [libdefaults] default\_realm = $AD\_REALM dns\_lookup\_realm = true dns\_lookup\_kdc = true ticket\_lifetime = 24h forwardable = true [realms] $AD\_REALM = { kdc = $KDC\_SERVER admin\_server = $KDC\_SERVER } [domain\_realm] .$AD\_REALM = $AD\_REALM .${AD\_REALM,,} = $AD\_REALM EOF echo "🎫 Requesting Kerberos ticket for $AD\_USERNAME@$AD\_REALM" echo "You will be prompted for the AD password..." kinit "${AD\_USERNAME}@${AD\_REALM}" echo "📋 Verifying Kerberos ticket..." klist echo "🧪 Testing connection to SQL Server with sqlcmd..." sqlcmd -S "$SQL\_SERVER\_FQDN" -d "$SQL\_DB" -G -Q "SELECT SYSTEM\_USER, ORIGINAL\_LOGIN();" echo "✅ Setup complete."

**🧪 Usage**

1. Customize the variables:
   * AD\_REALM: Your domain, e.g. EXAMPLE.COM
   * AD\_USERNAME: Your Active Directory username
   * SQL\_SERVER\_FQDN: FQDN of the SQL Server
   * SQL\_DB: Target database name
   * KDC\_SERVER: Hostname of the domain controller
2. Save and execute:

bash

nano setup\_sqlserver\_kerberos\_rhel.sh chmod +x setup\_sqlserver\_kerberos\_rhel.sh ./setup\_sqlserver\_kerberos\_rhel.sh

Would you like to automate this for multiple users or create a system-wide DSN configuration as well?

Great — below is a complete solution for automating:

1. Kerberos configuration for multiple users.
2. SQL Server DSN setup using ODBC.
3. Optionally, creating a shared system-wide DSN (for apps like isql, Python pyodbc, etc.).

This version supports RHEL/CentOS-based systems and includes reusable logic.

**🛠️ Script: configure\_sql\_kerberos\_dsn.sh**

bash

#!/bin/bash *# Edit these to fit your environment* AD\_REALM="DOMAIN.COM" KDC\_SERVER="ad.domain.com" SQL\_SERVER\_FQDN="sqlserver.domain.com" SQL\_PORT="1433" DSN\_NAME="MSSQL\_KERBEROS" SQL\_DB="YourDatabase" *# Ensure required packages are installed* echo "📦 Installing required packages..." sudo yum install -y krb5-workstation unixODBC *# Install MS ODBC Driver 18 and tools (if not already)* if ! command -v sqlcmd &>/dev/null; then echo "⬇ Installing MS ODBC Driver 18..." sudo rpm -Uvh https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm sudo yum install -y msodbcsql18 mssql-tools18 echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bash\_profile source ~/.bash\_profile fi *# Configure Kerberos (global)* echo "🔐 Writing /etc/krb5.conf..." sudo bash -c "cat > /etc/krb5.conf" <<EOF [libdefaults] default\_realm = $AD\_REALM dns\_lookup\_realm = true dns\_lookup\_kdc = true ticket\_lifetime = 24h forwardable = true [realms] $AD\_REALM = { kdc = $KDC\_SERVER admin\_server = $KDC\_SERVER } [domain\_realm] .$AD\_REALM = $AD\_REALM .${AD\_REALM,,} = $AD\_REALM EOF *# Configure system-wide DSN in /etc/odbc.ini* echo "🧾 Creating system-wide DSN: $DSN\_NAME" sudo bash -c "cat > /etc/odbc.ini" <<EOF [$DSN\_NAME] Driver=ODBC Driver 18 for SQL Server Server=$SQL\_SERVER\_FQDN,$SQL\_PORT Database=$SQL\_DB Authentication=ActiveDirectoryIntegrated TrustServerCertificate=yes Encrypt=yes EOF *# Register the driver (if not already listed)* if ! grep -q "ODBC Driver 18" /etc/odbcinst.ini; then echo "🛠 Registering ODBC Driver 18 in /etc/odbcinst.ini" sudo bash -c "cat >> /etc/odbcinst.ini" <<EOF [ODBC Driver 18 for SQL Server] Description=Microsoft ODBC Driver 18 for SQL Server Driver=/opt/microsoft/msodbcsql18/lib64/libmsodbcsql-18.X.so UsageCount=1 EOF fi *# List user instructions* cat <<EOF ✅ Setup Complete To connect using Kerberos: 1. Log in as any user and run: kinit youruser@${AD\_REALM} 2. Test using: sqlcmd -S $SQL\_SERVER\_FQDN -d $SQL\_DB -G 3. Or use isql with DSN: isql -v $DSN\_NAME -k EOF

**🔧 To Use**

1. Save the script:  
   nano configure\_sql\_kerberos\_dsn.sh
2. Make it executable:  
   chmod +x configure\_sql\_kerberos\_dsn.sh
3. Run it:  
   sudo ./configure\_sql\_kerberos\_dsn.sh
4. Then, as any user:
   * Run kinit to get a Kerberos ticket.
   * Use sqlcmd or isql with the configured DSN.

here’s a sample Python script using pyodbc to connect to SQL Server via Kerberos using the system-wide DSN you configured earlier.

✅ This example assumes:

* You ran the script above and created a DSN (e.g. MSSQL\_KERBEROS).
* The user has run kinit and has a valid Kerberos ticket.
* Python 3 and pyodbc are installed.

**🐍 Sample: connect\_sqlserver\_kerberos.py**

python

import pyodbc *# Replace with your actual DSN name* DSN\_NAME = "MSSQL\_KERBEROS" try: print("🔗 Connecting to SQL Server using Kerberos and DSN...") conn = pyodbc.connect(f"DSN={DSN\_NAME};Authentication=ActiveDirectoryIntegrated", autocommit=True) cursor = conn.cursor() cursor.execute("SELECT SYSTEM\_USER, ORIGINAL\_LOGIN();") row = cursor.fetchone() print(f"✅ Connected as: {row[0]} (Original Login: {row[1]})") cursor.close() conn.close() except pyodbc.Error as e: print("❌ Connection failed.") print(e)

**🧪 How to Run**

1. Make sure you have a Kerberos ticket:

bash

kinit youruser@DOMAIN.COM

1. Install pyodbc if not already:

pip install pyodbc

1. Save and run the script:

python3 connect\_sqlserver\_kerberos.py

Would you like this example extended to handle query parameters or error logging?

**You said:**

yes

**ChatGPT said:**

Here’s an extended and more production-ready version of the Python pyodbc script. It includes:

✅ Parameterized queries  
✅ Robust error logging  
✅ Optionally logs to a file  
✅ Query execution with input parameters

**🐍 Extended Script: connect\_sqlserver\_kerberos.py**

python

CopyEdit

import pyodbc import logging from datetime import datetime *# ====== Configuration ======* DSN\_NAME = "MSSQL\_KERBEROS" LOG\_FILE = "sql\_kerberos\_query.log" QUERY = "SELECT SYSTEM\_USER AS UserName, ORIGINAL\_LOGIN() AS OriginalLogin;" PARAM\_QUERY = "SELECT \* FROM YourTable WHERE Column1 = ?;" PARAM\_VALUE = "SomeValue" *# Replace with actual parameter if needed* *# ====== Set up logging ======* logging.basicConfig( filename=LOG\_FILE, level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s" ) def connect\_and\_query(): try: logging.info("Attempting to connect to SQL Server using DSN: %s", DSN\_NAME) print("🔗 Connecting to SQL Server using Kerberos and DSN...") conn = pyodbc.connect( f"DSN={DSN\_NAME};Authentication=ActiveDirectoryIntegrated", autocommit=True ) logging.info("Connection successful.") cursor = conn.cursor() *# Run a simple query* cursor.execute(QUERY) row = cursor.fetchone() print(f"✅ Connected as: {row.UserName} (Original Login: {row.OriginalLogin})") *# Optional: Run a parameterized query* logging.info("Running parameterized query: %s", PARAM\_QUERY) cursor.execute(PARAM\_QUERY, (PARAM\_VALUE,)) results = cursor.fetchall() print(f"\n📄 Results for parameter '{PARAM\_VALUE}':") for r in results: print(r) cursor.close() conn.close() except pyodbc.Error as e: error\_msg = f"❌ Connection or query failed: {str(e)}" print(error\_msg) logging.error(error\_msg) if \_\_name\_\_ == "\_\_main\_\_": start\_time = datetime.now() print("🚀 Starting SQL Server Kerberos test...") connect\_and\_query() print(f"🕒 Done in {datetime.now() - start\_time}")

**📝 Notes**

* Replace YourTable and Column1 with your actual table and column.
* PARAM\_VALUE can be dynamically populated from input or environment variables.
* The log file (sql\_kerberos\_query.log) will contain time-stamped logs of connection and query activity.
* The parameterized query prevents SQL injection.

Would you like this packaged into a CLI tool (e.g., using argparse for command-line parameters)?

**You said:**

example of command line sql connection from linux to sql server

**ChatGPT said:**

Here’s an example of a command-line SQL Server connection from a Linux machine using the official Microsoft sqlcmd tool and Kerberos authentication:

**✅ Example: Connect using Kerberos (Active Directory login)**

Assuming:

* You have a valid Kerberos ticket (via kinit)
* SQL Server supports Windows authentication
* The ODBC driver and sqlcmd are installed

bash

CopyEdit

sqlcmd -S sqlserver.domain.com -d YourDatabase -G

📌 Explanation:

* -S: SQL Server hostname or FQDN (optionally include port, e.g. sqlserver.domain.com,1433)
* -d: Target database name
* -G: Use Kerberos (integrated) authentication

You must have run:

bash

kinit youruser@DOMAIN.COM

before executing this.

**❗ Optional: Specify Query**

You can also run a single query:

bash

sqlcmd -S sqlserver.domain.com -d YourDatabase -G -Q "SELECT SYSTEM\_USER, CURRENT\_USER;"

**💡 Alternative (with ODBC DSN):**

If you've defined a DSN named MSSQL\_KERBEROS in /etc/odbc.ini:

bash

isql -v MSSQL\_KERBEROS -k

or via sqlcmd:

sqlcmd -S MSSQL\_KERBEROS -G

Here’s a one-liner you can use to test a Kerberos-authenticated SQL Server connection from Linux and log the output to a file for debugging:

📝 Make sure you’ve already authenticated with kinit.

🔧 One-liner command:

bash

sqlcmd -S sqlserver.domain.com -d YourDatabase -G -Q "SELECT SYSTEM\_USER, ORIGINAL\_LOGIN();" -o sql\_test\_output.txt

📌 Explanation:

* -S: SQL Server hostname (e.g., sqlserver.domain.com or sqlserver.domain.com,1433 if using a non-default port)
* -d: The database to connect to
* -G: Use Kerberos authentication (Active Directory Integrated)
* -Q: A query to run immediately
* -o: Output file to store results

🧪 To view the result:

bash

cat sql\_test\_output.txt

📁 To also capture errors, you can redirect stderr:

bash

sqlcmd -S sqlserver.domain.com -d YourDatabase -G -Q "SELECT SYSTEM\_USER;" -o sql\_test\_output.txt 2>&1