

Bacula Mock System

End-User Guide

Functionality, Configuration & Implementation Guide

Version 2.0.0 | March 2026

Complete end-user guide for the Bacula Enterprise Mock Server system. Covers installation, configuration, the CLI tool, REST API usage, TLS setup, authentication, common workflows, external test client, BWeb integration, service management, and troubleshooting.

API Endpoints	40+
Client Profiles	30+
Catalog Schema	v1027 (EE)
CI/CD Tests	321
Authentication	OAuth2 + Basic
TLS Support	Yes (optional)

Copyright (c) 2026 faaleoleo dev team — License: BSD 2-Clause

"Bacula" is a registered trademark of Bacula Systems SA. This software is not affiliated with or endorsed by Bacula Systems SA.

Table of Contents

1. What Is the Bacula Mock System? 2. Getting Started

- Installation
- First Start
- Generating Test Data

3. TLS (HTTPS) Configuration

- Enabling TLS During Installation
- Enabling TLS After Installation
- Using the API with TLS
- Using a CA-Signed Certificate
- Disabling TLS

4. Authentication

- HTTP Basic Auth
- OAuth2 Tokens
- Changing Credentials

5. Using the CLI Tool

- Server Management
- Data Management
- Querying Data
- Exports and Testing

6. Using the REST API Directly

- Catalog Endpoints
- Command Endpoints
- Status and Resource Endpoints
- Mock Utilities

7. Configuration

- Server Settings
- Simulation Settings
- Client Profiles
- Applying Changes

8. Common Workflows

- Building a Monitoring Dashboard
- Error Analysis
- Load Testing
- Test Scenarios

9. External Test Client (Binary)

- Overview
- Building from Source
- Running Tests
- Output Modes
- Test Groups
- CI/CD Integration

10. BWeb Integration 11. Service Management

- Systemd Commands
- Log Files
- Firewall Configuration

12. Upgrading 13. Troubleshooting 14. Uninstallation 15. Quick Reference

1. What Is the Bacula Mock System?

The Bacula Mock System is a complete simulation of the Bacula Enterprise REST API. It provides a fully functional API server that behaves identically to a real Bacula Enterprise Director — same endpoints, same response formats, same authentication — but without requiring any actual Bacula infrastructure.

Use cases:

- Developing and testing monitoring dashboards
- Training new administrators on the Bacula API
- Running integration tests in CI/CD pipelines
- Demonstrating Bacula API capabilities
- Performance testing API clients
- BWeb Management Suite integration for full graphical interface

Key capabilities:

- 100% wire-compatible with the Bacula Enterprise REST API
- Full Enterprise Edition catalog schema v1027 with 40+ tables
- Generates realistic backup data across 30+ client profiles (mail servers, databases, VMs, cloud, IoT, and more)
- Supports both HTTP and HTTPS (TLS) connections
- OAuth2 and HTTP Basic Auth authentication
- Configurable error rates for testing error-handling logic
- CLI tool for quick operations
- Runs as a systemd service with auto-start and auto-restart

2. Getting Started

Installation

Run the interactive installer:

```
# As root (installs to /opt/bacula-mock)
sudo ./install.sh

# As regular user (installs to ~/bacula-mock)
./install.sh

# Custom directory
./install.sh -d /srv/bacula-mock
```

The installer guides you through the setup and asks:

1. Whether to create a systemd service (recommended) 2. Whether to enable TLS/HTTPS (optional)

For non-interactive (automated) installation:

```
sudo ./install.sh -y           # Accept all defaults
sudo ./install.sh -y --tls     # With TLS enabled
sudo ./install.sh -y -s       # Without systemd service
```

All installer flags:

Flag	Description
`-h`, `--help`	Show help
`-y`, `--yes`	Non-interactive mode
`-s`, `--skip-systemd`	Skip systemd service
`-p PORT`, `--port PORT`	Custom port (default: 9101)
`-d DIR`, `--dir DIR`	Custom install directory
`-t`, `--tls`	Enable TLS with self-signed certificate
`-u`, `--upgrade`	Upgrade mode (preserves config)
`-v`, `--verbose`	Verbose output

First Start

After installation with systemd:

```
# Start the service
sudo systemctl start bacula-mock

# Verify it is running
sudo systemctl status bacula-mock

# Check the health endpoint
curl http://localhost:9101/mock/health
# With TLS: curl -k https://localhost:9101/mock/health
```

Without systemd:

```
bacula-mock start
```

Generating Test Data

The server starts with an empty database. Generate test data with:

```
# Generate 100 clients with 25% error rate (default)
bacula-mock init

# Custom: 200 clients, 10% error rate
bacula-mock init 200 0.10

# Reset and start fresh
bacula-mock reset
bacula-mock init 50 0.30
```

After initialization, the server has realistic backup jobs, clients, pools, volumes, and log entries ready for querying.

3. TLS (HTTPS) Configuration

The Bacula Mock System supports encrypted HTTPS connections via TLS. This is useful for testing dashboard integrations over HTTPS or simulating production-like environments.

Enabling TLS During Installation

The easiest approach — the installer generates everything automatically:

```
# Interactive: installer asks about TLS
./install.sh

# Non-interactive: enable TLS directly
./install.sh -y --tls
```

When TLS is enabled during installation, the installer:

1. Checks that `openssl` is installed on your system
2. Generates a 4096-bit RSA private key
3. Creates a self-signed X.509 certificate (valid for 365 days)
4. Stores both files in `$INSTALL_DIR/certs/`
5. Configures `config.yaml` with TLS enabled

The generated certificate includes Subject Alternative Names (SANs) for the system hostname, `localhost`, and `127.0.0.1`.

Enabling TLS After Installation

If you installed without TLS and want to enable it later:

```
cd /opt/bacula-mock    # or your install directory

# Step 1: Generate a self-signed certificate
openssl req -x509 -newkey rsa:4096 \
  -keyout certs/server.key \
  -out certs/server.crt \
  -days 365 -nodes \
  -subj "/CN=$(hostname)" \
  -addext "subjectAltName=DNS:$(hostname),DNS:localhost,IP:127.0.0.1"

# Step 2: Secure the private key
chmod 600 certs/server.key

# Step 3: Enable TLS in config.yaml
#   Set server.tls.enabled to true
```

```
vim config.yaml

# Step 4: Set TLS environment variable for the CLI tool
#   Add BACULA MOCK_TLS=true to .env
echo "BACULA MOCK_TLS=true" >> .env

# Step 5: Restart the service
sudo systemctl restart bacula-mock
```

The relevant config.yaml section:

```
server:
  host: "0.0.0.0"
  port: 9101
  debug: false
  tls:
    enabled: true           # ← set to true
    cert_file: "certs/server.crt" # path to certificate
    key_file: "certs/server.key"  # path to private key
```

Using the API with TLS

When TLS is enabled, the server only accepts HTTPS connections.

With curl (self-signed certificate):

```
# The -k flag skips certificate verification (required for self-signed certs)
curl -k https://localhost:9101/mock/health
curl -k -u admin:bacula2026 https://localhost:9101/cat/Client
```

With the CLI tool:

The bacula-mock CLI reads the BACULA MOCK_TLS variable from .env or the environment:

```
# Option 1: Set in .env (persistent)
# Add this line to your .env file:
BACULA MOCK_TLS=true

# Option 2: Export for current session
export BACULA MOCK_TLS=true

# Then use the CLI normally – it handles self-signed certs automatically
bacula-mock health
bacula-mock clients
bacula-mock smoke-test
```

With Python (requests library):

```
import requests

BASE = 'https://localhost:9101'
AUTH = ('admin', 'bacula2026')

# verify=False for self-signed certificates
clients = requests.get(f'{BASE}/cat/Client', auth=AUTH, verify=False).json()
```

With JavaScript (Node.js):

```
// For self-signed certificates, set NODE_TLS_REJECT_UNAUTHORIZED=0
// or use a custom https agent
const https = require('https');
const agent = new https.Agent({ rejectUnauthorized: false });
```

```
fetch('https://localhost:9101/cat/Client', {
  headers: { 'Authorization': 'Basic ' + btoa('admin:bacula2026') },
  agent
});
```

Using a CA-Signed Certificate

For production-like environments, replace the self-signed certificate with one from a Certificate Authority:

```
# Replace the certificate files
cp /path/to/your/certificate.crt /opt/bacula-mock/certs/server.crt
cp /path/to/your/private.key /opt/bacula-mock/certs/server.key

# Secure the private key
chmod 600 /opt/bacula-mock/certs/server.key

# Restart the service
sudo systemctl restart bacula-mock
```

With a CA-signed certificate, the `-k` flag in `curl` is no longer needed, and `verify=False` in Python can be removed.

Disabling TLS

To switch back to plain HTTP:

```
# In config.yaml
server:
  tls:
    enabled: false
```

Also update `.env`:

```
BACULA MOCK_TLS=false
```

Then restart:

```
sudo systemctl restart bacula-mock
```

4. Authentication

The API supports two authentication methods. Both work identically over HTTP and HTTPS.

HTTP Basic Auth

The simplest method — include credentials with every request:

```
curl -u admin:bacula2026 http://localhost:9101/cat/Client

# With TLS:
curl -k -u admin:bacula2026 https://localhost:9101/cat/Client
```

OAuth2 Tokens

Request a short-lived token, then use it for subsequent calls:

```
# Step 1: Get a token
TOKEN=$(curl -s -X POST http://localhost:9101/oauth/token \
  -H "Content-Type: application/json" \
```

```
-d '{"username":"admin","password":"bacula2026"}' \
| python3 -c "import json,sys; print(json.load(sys.stdin)['access_token'])"
```

```
# Step 2: Use the token
curl -H "Authorization: Bearer $TOKEN" http://localhost:9101/cat/Client
```

Tokens expire after **12 seconds** (matching the Bacula Enterprise default). After expiry, request a new token.

Using the CLI:

```
bacula-mock token
```

Changing Credentials

Edit `config.yaml`:

```
authentication:
  username: "myadmin"
  password: "my-secure-password"
```

Then restart the service and update `.env` to match:

```
sudo systemctl restart bacula-mock
```

5. Using the CLI Tool

The `bacula-mock` command-line tool is installed to `/usr/local/bin` during setup. It reads connection settings from `.env` or environment variables.

Add `--raw` to any query command to get unformatted JSON output suitable for piping.

Server Management

```
bacula-mock start      # Start via systemd
bacula-mock stop       # Stop via systemd
bacula-mock restart    # Restart via systemd
```

Data Management

```
bacula-mock init                # Initialize with defaults (100 clients, 25% errors)
bacula-mock init 200 0.10       # 200 clients, 10% error rate
bacula-mock reset               # Delete all mock data
```

Querying Data

```
bacula-mock clients             # List clients (default limit: 50)
bacula-mock clients 100         # List up to 100 clients
bacula-mock client mail-server-001 # Show a specific client
bacula-mock client-status mail-server-001 # FD status for a client
```

```
bacula-mock jobs                # List recent jobs
bacula-mock jobs 100            # List up to 100 jobs
bacula-mock jobs-failed         # List failed jobs
bacula-mock jobs-failed 50      # Last 50 failed jobs
bacula-mock jobs-for 1          # Jobs for client ID 1
bacula-mock jobs-for 1 50       # Up to 50 jobs for client ID 1
```

```
bacula-mock joblog 42           # Logs for job ID 42
bacula-mock jobtotals           # Aggregate job statistics
```



```
bacula-mock pools           # List pools
bacula-mock dir-status      # Director status (running/terminated jobs)
bacula-mock config-clients  # Client resource configurations
```

Exports and Testing

```
bacula-mock export-csv      # Export all jobs as CSV
bacula-mock export-errors   # Export failed jobs as JSON
bacula-mock health          # Quick health check
bacula-mock smoke-test      # Full API endpoint test
bacula-mock token           # Request an OAuth2 token
```

6. Using the REST API Directly

All examples below use HTTP. When TLS is enabled, replace `http://` with `https://` and add `-k` for self-signed certificates.

Catalog Endpoints

These endpoints query the simulated Bacula catalog database. All return the standard envelope format: `{ "error": 0, "limit": N, "data": [...], "offset": 0 }`.

```
# Clients
curl -u admin:bacula2026 "http://localhost:9101/cat/Client?limit=20"
curl -u admin:bacula2026 "http://localhost:9101/cat/Client?clientid=1"
curl -u admin:bacula2026 "http://localhost:9101/cat/Client?name=mail-server-001"

# Jobs
curl -u admin:bacula2026 "http://localhost:9101/cat/Job?limit=10"
curl -u admin:bacula2026 "http://localhost:9101/cat/Job?jobstatus=E&limit=50"
curl -u admin:bacula2026 "http://localhost:9101/cat/Job?level=F"
curl -u admin:bacula2026 "http://localhost:9101/cat/Job?clientid=1&limit=20"

# Job Logs
curl -u admin:bacula2026 "http://localhost:9101/cat/JobLog?jobid=42"

# Job Totals
curl -u admin:bacula2026 "http://localhost:9101/cat/JobTotals"

# Pools
curl -u admin:bacula2026 "http://localhost:9101/cat/Pool"

# Storage
curl -u admin:bacula2026 "http://localhost:9101/cat/Storage"

# Media (Volumes)
curl -u admin:bacula2026 "http://localhost:9101/cat/Media"

# Enterprise Edition Catalog Tables
curl -u admin:bacula2026 "http://localhost:9101/cat/FileSet"
curl -u admin:bacula2026 "http://localhost:9101/cat/Object"
curl -u admin:bacula2026 "http://localhost:9101/cat/Location"
curl -u admin:bacula2026 "http://localhost:9101/cat/Snapshot"
curl -u admin:bacula2026 "http://localhost:9101/cat/Device"
curl -u admin:bacula2026 "http://localhost:9101/cat/MediaType"
curl -u admin:bacula2026 "http://localhost:9101/cat/Events"
curl -u admin:bacula2026 "http://localhost:9101/cat/TagJob"
curl -u admin:bacula2026 "http://localhost:9101/cat/JobMedia"
```

Command Endpoints

These simulate Bacula director commands:

```
# Run a backup job
curl -u admin:bacula2026 -X POST http://localhost:9101/cmd/run \
  -H "Content-Type: application/json" \
  -d '{"job": "backup-mailserver-001-full", "level": "F", "client": "mailserver-001"}'

# Cancel a job
curl -u admin:bacula2026 -X POST http://localhost:9101/cmd/cancel \
  -H "Content-Type: application/json" -d '{"jobid": 99}'

# Restore
curl -u admin:bacula2026 -X POST http://localhost:9101/cmd/restore \
  -H "Content-Type: application/json" -d '{"jobid": 42, "where": "/tmp/restore"}'

# Estimate job size
curl -u admin:bacula2026 -X POST http://localhost:9101/cmd/estimate \
  -H "Content-Type: application/json" -d '{"client": "mailserver-001", "level": "F"}'

# Label a volume
curl -u admin:bacula2026 -X POST http://localhost:9101/cmd/label \
  -H "Content-Type: application/json" -d '{"volume": "Vol-001", "pool": "Default"}'

# List queries
curl -u admin:bacula2026 "http://localhost:9101/cmd/list?jobs&limit=10"
curl -u admin:bacula2026 "http://localhost:9101/cmd/list?clients&limit=10"
```

Status and Resource Endpoints

```
# Director status
curl -u admin:bacula2026 http://localhost:9101/status/director

# Client FD status
curl -u admin:bacula2026 -X POST http://localhost:9101/status/client \
  -H "Content-Type: application/json" -d '{"name": "mail-server-001"}'

# Storage status
curl -u admin:bacula2026 -X POST http://localhost:9101/status/storage \
  -H "Content-Type: application/json" -d '{"name": "File-Storage"}'

# Director resources (config)
curl -u admin:bacula2026 http://localhost:9101/res/director
curl -u admin:bacula2026 "http://localhost:9101/res/director?resource=Client"
```

Mock Utilities

These endpoints are specific to the mock server and are not part of the Bacula Enterprise API:

```
# Health check (no auth required)
curl http://localhost:9101/mock/health

# Initialize with test data
curl -u admin:bacula2026 -X POST http://localhost:9101/mock/initialize \
  -H "Content-Type: application/json" \
  -d '{"clients": 100, "error_rate": 0.25}'

# Reset all data
curl -u admin:bacula2026 -X POST http://localhost:9101/mock/reset

# API documentation page
```

```
# Open in browser: http://localhost:9101/docs
```

7. Configuration

All configuration is in `config.yaml` in the installation directory.

Server Settings

```
server:
  host: "0.0.0.0"          # 0.0.0.0 = all interfaces, 127.0.0.1 = local only
  port: 9101              # TCP port
  debug: false            # Extra debug logging
  tls:
    enabled: false        # true = HTTPS, false = HTTP
    cert_file: "certs/server.crt" # TLS certificate path
    key_file: "certs/server.key"  # TLS private key path
```

Simulation Settings

```
simulation:
  default_clients: 100    # Number of simulated clients
  error_rate: 0.25       # Fraction of jobs that fail (0.0-1.0)
  update_interval: 30    # Seconds between background updates
```

Client Profiles

Client profiles define the types of systems being simulated. Each profile generates clients with realistic hostnames, backup sizes, and error messages:

```
client_profiles:
  mailserver:
    count: 20              # Number of mail server clients
    avg_size_gb: 80        # Average backup size
    avg_files: 15000       # Average file count
    error_types:           # Realistic error messages
      - "Mailbox locked by user"
      - "IMAP connection timeout"

  database:
    count: 15
    avg_size_gb: 50
    avg_files: 5000
    error_types:
      - "Database connection timeout"
      - "Transaction log unavailable"
```

The default configuration includes 30+ profiles covering mail servers, databases, NFS, Linux/Windows systems, web servers, SAP HANA, Kubernetes, VMware, cloud platforms (AWS, Azure, GCP), containers, network appliances, and more.

Applying Changes

After editing `config.yaml`:

```
# Restart the service
sudo systemctl restart bacula-mock

# Re-initialize test data to reflect new profiles
bacula-mock init
```

8. Common Workflows

Building a Monitoring Dashboard

Typical data flow for a dashboard:

```
# 1. Get all clients
bacula-mock clients 200 --raw

# 2. Get job totals for the overview panel
bacula-mock jobtotals --raw

# 3. Get failed jobs for the alert panel
bacula-mock jobs-failed 50 --raw

# 4. Get director status for running jobs
bacula-mock dir-status --raw

# 5. Drill into a specific client
bacula-mock client mail-server-001 --raw
bacula-mock jobs-for 1 50 --raw
bacula-mock joblog 42 --raw
```

Error Analysis

Find and investigate failed jobs:

```
# List failed jobs
bacula-mock jobs-failed 100

# Check logs for a failed job
bacula-mock joblog 42

# Export all errors as JSON for analysis
bacula-mock export-errors > failed_jobs.json

# Export all jobs as CSV for spreadsheet analysis
bacula-mock export-csv > all_jobs.csv
```

Load Testing

Test your dashboard against high request volumes:

```
# Apache Bench: 1000 requests, 10 concurrent
ab -n 1000 -c 10 \
  -H "Authorization: Basic $(echo -n admin:bacula2026 | base64)" \
  "http://localhost:9101/cat/Job?limit=10"
```

Test Scenarios

Generate different data sets for different testing needs:

```
# Clean data (no errors)
bacula-mock reset && bacula-mock init 10 0.0

# Disaster simulation (90% failure rate)
bacula-mock reset && bacula-mock init 100 0.90

# High load (many clients)
bacula-mock reset && bacula-mock init 500 0.25
```

```
# Minimal dataset
bacula-mock reset && bacula-mock init 5 0.15
```

9. External Test Client (Binary)

Overview

The Bacula Mock System includes a standalone compiled test client that can validate any running server instance from any machine on the network. It is a single binary with **zero dependencies** — no Python, no libraries, no runtime needed.

- **Language:** Pure C11 (POSIX sockets, built-in JSON parser, built-in Base64 encoder)
- **Platforms:** Linux (x86-64), macOS (arm64/x86-64)
- **Tests:** 321 tests across 12 groups covering all API endpoints
- **Location:** tests/external_client/

Building from Source

The binary must be compiled for your target platform. A pre-compiled Linux binary is included.

Linux:

```
cd tests/external_client
make
```

macOS:

```
cd tests/external_client
cc -O2 -Wall -std=c11 -o bacula_mock_test bacula_mock_test.c
```

Static build (fully self-contained, no shared libraries):

```
make static
```

NOTE: Tip: The only requirement is a C compiler (`gcc` on Linux, `cc/clang` on macOS). No additional libraries are needed.

Running Tests

Test a local server:

```
./bacula_mock_test
```

Test a remote server:

```
./bacula_mock_test -h 192.168.10.67 -p 9101 -v
```

All options:

Option	Description	Default
<code>-h HOST</code>	Server hostname or IP	<code>`127.0.0.1`</code>
<code>-p PORT</code>	Server port	<code>`9101`</code>
<code>-u USER</code>	Username	<code>`admin`</code>

`-P PASS`	Password	`bacula2026`
`-v`	Verbose (details + timing)	
`-q`	Quiet (summary only)	
`-j`	JSON output	
`--help`	Show help	

Exit codes:

Code	Meaning
`0`	All tests passed
`1`	One or more failures
`2`	Server unreachable

NOTE: Note: The test client initializes its own test data via `/mock/reset` and `/mock/initialize`. Any existing data on the server will be reset during the test run.

Output Modes

Default output — color-coded pass/fail with compatibility matrix:

```
■■■ 4. Catalog Endpoints (/cat/) ■■■
✓ PASS /cat/Client envelope
✓ PASS /cat/Client fields (6 Bacula fields)
✓ PASS /cat/Job full field coverage (28/28 Bacula fields)
```

Verbose (-v) — adds timing and data details:

```
✓ PASS /cat/Job?jobstatus=T filter correct (20 jobs, all status=T) [1.8ms]
✓ PASS POST /cmd/run (start backup) [1.1ms]
    JobId=353
```

JSON (-j) — machine-readable for CI/CD pipelines:

```
./bacula_mock_test -h server -j > results.json

{
  "version": "3.0.0",
  "server": "server:9101",
  "total": 321,
  "passed": 321,
  "failed": 0,
  "success_rate": 100.0,
  "tests": [
    { "group": "1. Connectivity & Health", "name": "/mock/health (no auth)", "passed": true, "ms": 1.2 },
    ...
  ]
}
```

Test Groups

The 321 tests cover all API functionality:

#	Group	Tests	Coverage
---	-------	-------	----------

1	Connectivity & Health	5	Server availability, health endpoint
2	Authentication	22	OAuth2, Basic Auth, invalid credentials
3	Data Initialization & Reset	13	Initialize, reset, error rate validation
4	Catalog — Wire Compatibility	57	Field counts, response format, data types
5	Filtering, Pagination & Sorting	23	Limit, offset, status/level filters
6	Command Endpoints	34	run, cancel, restore, label, prune, purge
7	Status Endpoints	23	Director, client, storage status
8	Resources, Configuration & BVFS	49	Config read/write, file browsing
9	Data Integrity & Consistency	11	Cross-table consistency, time ordering
10	Security Audit	61	Auth enforcement, error handling
11	Edge Cases & Robustness	15	Invalid params, empty results, boundaries
12	Performance & Concurrency	8	Concurrent requests, response times

CI/CD Integration

Use the JSON output and exit code for automated pipelines:

```
# GitLab CI / GitHub Actions example
./bacula_mock_test -h $SERVER_HOST -j > test-results.json
if [ $? -ne 0 ]; then
    echo "Integration tests failed!"
    cat test-results.json | jq '.tests[] | select(.passed == false)'
    exit 1
fi

# Quick smoke test in a deployment script
./bacula_mock_test -h production-server -q || echo "WARNING: Server validation failed"
```

9b. BWeb Integration

The Bacula Mock Server can be connected to the BWeb Management Suite for a full graphical interface. See the separate **BWeb + Bacula Mock Server Integration Guide** for complete instructions.

Quick overview:

- BWeb connects to a SQLite catalog synced from the mock server
- The `sync_mock_to_sqlite.py` script mirrors in-memory data to SQLite every 60 seconds
- `lighttpd` reverse-proxies API requests from BWeb (port 9180) to the mock server (port 9101)
- Install with: `sudo ./install-bweb-mock.sh` (requires Bacula Enterprise subscription for BWeb package)

Available BWeb pages with mock data: Jobs, Clients, Media/Volumes, Pools, Statistics, Job Logs, Client Groups.

11. Service Management

Systemd Commands

```
sudo systemctl start bacula-mock      # Start
sudo systemctl stop bacula-mock       # Stop
sudo systemctl restart bacula-mock    # Restart
sudo systemctl status bacula-mock     # Check status
sudo systemctl enable bacula-mock     # Enable auto-start on boot
sudo systemctl disable bacula-mock    # Disable auto-start
```

Log Files

```
# Systemd journal (live)
sudo journalctl -u bacula-mock -f

# Last 100 log lines
sudo journalctl -u bacula-mock -n 100

# Application log file
tail -f /opt/bacula-mock/logs/bacula_mock.log

# Installation log
cat /opt/bacula-mock/installation.log
```

Firewall Configuration

If accessing the API from another machine, open the port:

```
# UFW (Ubuntu/Debian)
sudo ufw allow 9101/tcp

# firewalld (RHEL/CentOS)
sudo firewall-cmd --permanent --add-port=9101/tcp
sudo firewall-cmd --reload
```

12. Upgrading

Upgrade to a new version while preserving your configuration, logs, data, and TLS certificates:

```
cd /path/to/new-release
./install.sh --upgrade
sudo systemctl restart bacula-mock
```

The upgrade process:

- Creates a timestamped backup in `backups/`
- Preserves `config.yaml`, `logs/`, `data/`, and `certs/`
- Updates source code, scripts, and documentation
- Writes new defaults to `config.yaml.dist` for comparison

To enable TLS during an upgrade:

```
./install.sh --upgrade --tls
```

For the full upgrade guide, see `UPGRADE.md`.

13. Troubleshooting

Server won't start

```
# Check the status and recent logs
sudo systemctl status bacula-mock
sudo journalctl -u bacula-mock -n 50 --no-pager
```

Common causes: port already in use, configuration syntax error, missing certificate files (if TLS is enabled).

Port already in use

```
sudo lsof -i :9101
# Kill the conflicting process or change the port in config.yaml
```

TLS certificate errors

If TLS is enabled but the certificate files are missing or expired:

```
# Check certificate files exist
ls -la /opt/bacula-mock/certs/

# Check certificate expiry
openssl x509 -in /opt/bacula-mock/certs/server.crt -noout -dates

# Regenerate self-signed certificate
openssl req -x509 -newkey rsa:4096 \
  -keyout /opt/bacula-mock/certs/server.key \
  -out /opt/bacula-mock/certs/server.crt \
  -days 365 -nodes -subj "/CN=$(hostname)"
chmod 600 /opt/bacula-mock/certs/server.key

sudo systemctl restart bacula-mock
```

Cannot connect to API

```
# Check the service is running
sudo systemctl status bacula-mock

# Test locally
curl http://localhost:9101/mock/health
# Or with TLS:
curl -k https://localhost:9101/mock/health

# Check firewall
```

```
sudo ufw status
```

CLI tool not found

```
# Verify the symlink
ls -la /usr/local/bin/bacula-mock

# If missing, recreate it
sudo ln -sf /opt/bacula-mock/scripts/bacula-mock /usr/local/bin/bacula-mock

# Or add to PATH
export PATH="/opt/bacula-mock/scripts:$PATH"
```

Virtual environment issues

```
cd /opt/bacula-mock
rm -rf venv
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
sudo systemctl restart bacula-mock
```

14. Uninstallation

```
# Stop and disable the service
sudo systemctl stop bacula-mock
sudo systemctl disable bacula-mock

# Remove the service file
sudo rm /etc/systemd/system/bacula-mock.service
sudo systemctl daemon-reload

# Remove the installation directory
sudo rm -rf /opt/bacula-mock

# Remove the CLI symlink
sudo rm -f /usr/local/bin/bacula-mock
```

15. Quick Reference

Commands

Task	Command
Install	<code>`sudo ./install.sh`</code>
Install with TLS	<code>`sudo ./install.sh --tls`</code>
Start	<code>`sudo systemctl start bacula-mock`</code>
Stop	<code>`sudo systemctl stop bacula-mock`</code>
Restart	<code>`sudo systemctl restart bacula-mock`</code>
Status	<code>`sudo systemctl status bacula-mock`</code>

Logs	<code>`sudo journalctl -u bacula-mock -f`</code>
Health check	<code>`bacula-mock health`</code>
Init data	<code>`bacula-mock init 100 0.25`</code>
Reset data	<code>`bacula-mock reset`</code>
Smoke test	<code>`bacula-mock smoke-test`</code>
Upgrade	<code>`./install.sh --upgrade`</code>

API Quick Test

```
# HTTP
curl -u admin:bacula2026 http://localhost:9101/cat/Client

# HTTPS (self-signed)
curl -k -u admin:bacula2026 https://localhost:9101/cat/Client
```

Default Credentials

Setting	Value
Username	<code>`admin`</code>
Password	<code>`bacula2026`</code>
Port	<code>`9101`</code>
Config file	<code>`\$INSTALL_DIR/config.yaml`</code>

Important Files

File	Purpose
<code>`config.yaml`</code>	Main configuration
<code>`.env`</code>	Environment variables for CLI and scripts
<code>`certs/server.crt`</code>	TLS certificate (if TLS enabled)
<code>`certs/server.key`</code>	TLS private key (if TLS enabled)
<code>`logs/bacula_mock.log`</code>	Application log
<code>`installation.log`</code>	Installation log

Environment Variables

Variable	Default	Description
<code>`BACULA MOCK_HOST`</code>	<code>`0.0.0.0`</code>	Server bind address
<code>`BACULA MOCK_PORT`</code>	<code>`9101`</code>	Server port

<code>`BACULA MOCK_USERNAME`</code>	<code>`admin`</code>	API username
<code>`BACULA MOCK_PASSWORD`</code>	<code>`bacula2026`</code>	API password
<code>`BACULA MOCK_TLS`</code>	<code>`false`</code>	Set to <code>`true`</code> for HTTPS in CLI

Further Documentation

- **README.md** — Project overview
- **INSTALLATION.md** — Detailed installation guide
- **CONFIG.md** — Configuration reference
- **API_REFERENCE.md** — Complete API endpoint reference
- **USAGE.md** — Advanced usage examples and scripting
- **UPGRADE.md** — Upgrade and rollback procedures

Trademarks

_"Bacula" is a registered trademark of Bacula Systems SA. _This software is not affiliated with or endorsed by Bacula Systems SA._

"Bacula" is a registered trademark of Bacula Systems SA. This software is not affiliated with or endorsed by Bacula Systems SA.