**Linux Systemd Service Management for Java Applications**

This guide demonstrates how to create and manage a systemd service for a Java JAR application, enabling automatic startup, proper service management, and system integration.

**Initial Process Check**

**ps aux | grep "jar"**

**Purpose**: Check if any JAR processes are currently running **Explanation**:

* ps aux lists all running processes
* grep "jar" filters for processes containing "jar"
* Shows only the grep command itself (PID 2780), indicating no JAR applications are running
* Useful for verifying clean system state before deployment

**Directory Navigation**

**cd ~/Documents/demo\_linux**

**Purpose**: Navigate to the project directory **Explanation**:

* Changes to the directory containing the Spring Boot project
* Prepares for copying the JAR file to its deployment location

**Application Deployment**

**sudo mkdir -p /opt/myapp**

**Purpose**: Create deployment directory with proper structure **Explanation**:

* mkdir -p creates directory and any necessary parent directories
* /opt/myapp follows Linux Filesystem Hierarchy Standard for optional applications
* sudo required as /opt is a system directory requiring root privileges
* Creates a dedicated space for the application outside user directories

**sudo cp ~/Documents/demo\_linux/build/libs/demo\_linux-0.0.1-SNAPSHOT.jar /opt/myapp/app.jar**

**Purpose**: Copy JAR file to deployment location with simplified name **Explanation**:

* Copies the built JAR from development location to production location
* Renames from demo\_linux-0.0.1-SNAPSHOT.jar to app.jar for simplicity
* sudo required to write to system directory
* Centralizes application in standard system location

**Systemd Service Creation**

**sudo nano /etc/systemd/system/myapp.service**

**Purpose**: Create systemd service configuration file **Explanation**:

* /etc/systemd/system/ is the directory for system-wide service files
* myapp.service defines the service configuration
* nano is used to edit the file (could also use vim or other editors)
* sudo required to create files in system directories

**Typical service file content would include:**

[Unit]

Description=My Spring Boot App

After=network.target

[Service]

Type=exec

User=kali

ExecStart=/usr/bin/java -jar /opt/myapp/app.jar

StandardOutput=file:/opt/myapp/app.log

StandardError=file:/opt/myapp/app.log

Restart=always

RestartSec=10

[Install]

WantedBy=multi-user.target

**Service Configuration and Activation**

**sudo systemctl daemon-reload**

**Purpose**: Reload systemd configuration to recognize new service **Explanation**:

* systemctl is the command for controlling systemd services
* daemon-reload tells systemd to re-read all service files
* Required after creating or modifying service files
* Updates systemd's internal service registry

**sudo systemctl enable myapp.service**

**Purpose**: Enable service to start automatically at boot **Explanation**:

* enable creates symlinks to start the service at appropriate run levels
* Creates symlink: /etc/systemd/system/multi-user.target.wants/myapp.service
* Service will now start automatically when system boots
* Does not start the service immediately, only enables auto-start

**Service Startup**

**sudo systemctl start myapp.service**

**Purpose**: Start the service immediately **Explanation**:

* start command begins running the service
* Executes the command defined in ExecStart in the service file
* Service runs in background managed by systemd
* No output shown if successful

**Log Monitoring**

**tail -f /opt/myapp/app.log**

**Purpose**: Monitor application logs in real-time **Explanation**:

* tail -f follows the log file as new entries are added
* Shows Spring Boot startup sequence with timestamps
* Displays Tomcat initialization and startup on port 8080
* Application started successfully in 9.326 seconds with PID 5214
* Ctrl+C exits the tail command

**Service Status Verification**

**sudo systemctl status myapp.service**

**Purpose**: Check service status and detailed information **Explanation**:

* Shows service is active (running) since 00:42:17 EDT
* Displays main process ID (5214) and resource usage
* Memory usage: 192.1M (peak 194.1M)
* CPU time consumed: 22.157s
* Shows 35 active tasks
* Includes recent log entries from systemd journal

**Additional Process Verification**

**tail -1000 nohup.out**

**Purpose**: Check previous nohup output (from earlier session) **Explanation**:

* Shows Gradle build output from previous session
* Indicates successful Gradle build
* Demonstrates the difference between manual nohup execution and systemd service

**Port Verification**

**sudo lsof -i :80**

**Purpose**: Check if anything is listening on port 80 **Explanation**:

* lsof -i :80 lists processes using port 80
* Returns empty result, indicating port 80 is not in use
* Useful for checking standard HTTP port availability

**sudo lsof -i :8080**

**Purpose**: Verify application is listening on correct port **Explanation**:

* Shows Java process (PID 5214) listening on port 8080
* \*:http-alt (LISTEN) indicates listening on all interfaces
* Confirms service is properly running and accepting connections
* User 'kali' owns the process as configured in service file

**Service Management**

**sudo systemctl stop myapp.service**

**Purpose**: Stop the running service **Explanation**:

* stop command gracefully shuts down the service
* Sends termination signal to the Java process
* Service stops but remains enabled for future auto-start
* No output indicates successful operation

**sudo systemctl status myapp.service (after stop)**

**Purpose**: Verify service has stopped **Explanation**:

* Status shows inactive (dead) since 00:48:17 EDT
* Service ran for 5min 59.898s total duration
* Process exited with status 143 (SIGTERM - graceful shutdown)
* Memory peak was 194.4M, consumed 23.727s CPU time
* Shows complete lifecycle information

**sudo lsof -i :8080 (after stop)**

**Purpose**: Confirm port is no longer in use **Explanation**:

* Returns empty result, confirming Java process has released port 8080
* Verifies clean shutdown of the application
* Port is now available for other applications

**Key Benefits of Systemd Service Management**

1. **Automatic Startup**: Service starts automatically at boot time
2. **Process Management**: Systemd monitors and can restart failed processes
3. **Logging Integration**: Automatic log management and rotation
4. **Resource Control**: Can set memory, CPU, and other resource limits
5. **Security**: Runs with specified user permissions, not root
6. **Standard Interface**: Uses standard systemctl commands for management
7. **Service Dependencies**: Can specify startup order and dependencies

**Common Service Management Commands**

# Start service

sudo systemctl start myapp.service

# Stop service

sudo systemctl stop myapp.service

# Restart service

sudo systemctl restart myapp.service

# Check status

sudo systemctl status myapp.service

# Enable auto-start

sudo systemctl enable myapp.service

# Disable auto-start

sudo systemctl disable myapp.service

# View logs

sudo journalctl -u myapp.service -f

This approach provides enterprise-grade service management for Java applications on Linux systems, replacing manual process management with robust, automated service control.