ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

РЕФЕРАТ

по дисциплине: «**Администрирование информационных систем**»

на тему «**Запуск Java Spring Boot приложений в промышленной среде: настройка systemd и проксирование**»

Выполнил: студент гр. ПРО-436Б
Загидуллин Т. Р.
Проверил:
доцент каф. ВМиК
Чернышев Е. С.

Уфа 2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Java Spring Boot стал стандартом для создания микросервисов и веб-приложений в корпоративной среде благодаря своей простоте и концепции «Convention over Configuration». Встроенный веб-сервер (Tomcat, Jetty или Undertow) позволяет запускать приложение одной командой, что идеально подходит для этапа разработки. Однако перенос приложения в промышленную среду (production) требует решения ряда инфраструктурных задач, которые встроенные средства фреймворка не закрывают полностью.

В промышленной эксплуатации критически важны надежность, безопасность и управляемость. Приложение должно автоматически перезапускаться после сбоев или перезагрузки сервера, работать с привилегированными портами (80, 443) без прав суперпользователя, корректно обрабатывать SSL-сертификаты и эффективно отдавать статический контент.

Для решения этих задач используется связка из менеджера системных служб (systemd) и обратного прокси-сервера (Reverse Proxy, чаще всего Nginx). Systemd берет на себя управление жизненным циклом процесса, логирование и автозапуск, в то время как Nginx обеспечивает маршрутизацию трафика, балансировку нагрузки и терминацию HTTPS.

Актуальность темы обусловлена тем, что неправильная конфигурация окружения является одной из самых частых причин уязвимостей и простоев сервисов. Понимание того, как превратить исполняемый JAR-файл в полноценный системный сервис, является базовой компетенцией для системных администраторов и DevOps-инженеров.

Цель данного реферата – рассмотреть процесс подготовки сервера для запуска Java Spring Boot приложений, детально изучить написание unit-файлов для systemd и конфигурацию Nginx для безопасного проксирования запросов.

1 Особенности запуска Java-приложений в Production

1.1 Формат поставки (Fat JAR)

Традиционно Java-приложения поставлялись в виде WAR-файлов, которые необходимо было разворачивать в отдельном контейнере сервлетов (например, Apache Tomcat). Spring Boot изменил эту парадигму, популяризировав формат «Fat JAR» (или «Uber JAR»).

Fat JAR – это самодостаточный исполняемый архив, который содержит не только скомпилированные классы приложения, но и все необходимые библиотеки (dependencies), а также встроенный веб-сервер. Это позволяет запускать приложение командой вида java -jar app.jar на любой машине, где установлен JRE (Java Runtime Environment).

Такой способ запуска имеет недостатки в промышленной среде:

1) Отсутствие фонового режима – при закрытии терминала приложение останавливается.

2) Сложность управления – нет встроенных механизмов рестарта при сбоях.

3) Привилегии – для привязки к портам ниже 1024 требуются права root, что небезопасно.

Поэтому «голый» запуск JAR-файла используется только при разработке, а на серверах приложение оборачивается в системную службу.

1.2 Требования к окружению

Перед настройкой сервисов необходимо подготовить операционную систему (обычно дистрибутивы Linux: Ubuntu, CentOS, Debian).

Ключевые требования:

1) Наличие JDK/JRE – версия Java должна соответствовать версии, использованной при сборке проекта. Рекомендуется использовать LTS-версии (17, 21) от надежных поставщиков (Eclipse Temurin, Amazon Corretto).

2) Выделенный пользователь – запуск приложения от имени root является грубым нарушением безопасности. Создается специальный системный пользователь (например, spring-app) с ограниченными правами, который имеет доступ только к папке с приложением и логам.

3) Структура директорий – принято размещать приложения в /opt или /var/www, отделяя исполняемые файлы от конфигурационных файлов (application.properties), чтобы обновлять код без потери настроек.

2 Управление процессом через systemd

Systemd – это система инициализации и диспетчер служб в Linux, который пришел на смену устаревшему SysVinit. Он позволяет гибко управлять процессами, зависимостями и ресурсами.

2.1 Понятие systemd и Unit-файлов

В systemd каждая управляемая сущность описывается в виде «юнита» (unit). Для запуска приложений используются файлы с расширением .service. Эти файлы обычно располагаются в директории /etc/systemd/system/.

Unit-файл имеет декларативный синтаксис и состоит из секций:

1) [Unit] – описание сервиса, порядок запуска (например, после загрузки сети).

2) [Service] – параметры запуска процесса, пользователь, переменные окружения, политики перезапуска.

3) [Install] – определяет, на каком уровне запуска (target) сервис должен быть активирован.

2.2 Создание конфигурации службы

Ключевые директивы типичного файла конфигурации myapp.service:

1) User – запускает процесс от имени непривилегированного пользователя.

2) ExecStart – команда запуска. Важно указывать полные пути к исполняемым файлам.

3) SuccessExitStatus=143 – указывает systemd, что код выхода 143 (стандартный SIGTERM для Java) является нормальным завершением, чтобы система не считала это сбоем.

4) Restart=always – гарантирует, что systemd попытается перезапустить приложение, если оно упадет с ошибкой.

2.3 Управление жизненным циклом приложения

После создания файла сервиса необходимо выполнить команду systemctl daemon-reload, чтобы systemd перечитал конфигурацию.

Основные команды управления:

1) systemctl start myapp – запуск приложения.

2) systemctl enable myapp – добавление в автозагрузку (старт при включении сервера).

3) systemctl status myapp – просмотр текущего состояния, последних логов и PID процесса.

4) systemctl stop myapp – корректная остановка (отправка SIGTERM).

3 Проксирование запросов (Nginx)

3.1 Зачем нужен Reverse Proxy

Даже если Spring Boot запущен как сервис, он обычно слушает порт 8080. Выставлять этот порт напрямую в интернет не рекомендуется. Стандартной практикой является использование Nginx в качестве Reverse Proxy.

Использование промежуточного веб-сервера перед Java-приложением решает несколько задач:

1) Безопасность – Nginx работает как щит, скрывая детали реализации бэкенда и защищая от некоторых видов атак (например, медленные соединения).

2) Работа на стандартных портах – Nginx запускается от root и слушает порты 80 (HTTP) и 443 (HTTPS), перенаправляя трафик на порт 8080.

3) SSL/TLS Терминация – шифрование ресурсоемко. Nginx берет на себя работу с сертификатами, разгружая Java-приложение, которое работает по обычному HTTP внутри локального контура.

4) Раздача статики – Nginx отдает статические файлы (изображения, CSS, JS) в разы быстрее, чем Tomcat, встроенный в Spring Boot.

3.2 Настройка Nginx для Spring Boot

Конфигурация проксирования описывается в блоке server. Пример базовой настройки: server server_name example.com; proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr; proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header X-Forwarded-Proto $scheme;

Директива proxy_pass указывает адрес локального Java-приложения. Остальные директивы необходимы для корректной передачи информации о запросе.

3.3 Обработка заголовков и SSL

При проксировании теряется информация об исходном IP-адресе клиента, так как для Java-приложения запрос приходит от localhost. Заголовки X-Forwarded-* критически важны.

В файле настроек Spring Boot (application.properties) необходимо добавить строку: server.forward-headers-strategy=native

Это заставит Tomcat воспринимать заголовки от Nginx и корректно определять реальный IP пользователя и протокол (http или https), что важно для генерации правильных ссылок внутри приложения (HATEOAS) и редиректов.

Для настройки HTTPS часто используется утилита **Certbot**, которая автоматически получает сертификаты Let’s Encrypt и модифицирует конфиг Nginx, добавляя пути к ключам и настройки перенаправления с HTTP на HTTPS.

4 Безопасность и мониторинг

4.1 Управление пользователями и правами

При запуске в промышленной среде необходимо следовать принципу наименьших привилегий. Файл .jar должен принадлежать пользователю root с правами на чтение для пользователя сервиса (chmod 500), чтобы скомпрометированное приложение не могло перезаписать само себя.

Пароли к базам данных и API-ключам не должны храниться в коде. Systemd позволяет передавать их через переменные окружения (EnvironmentFile в unit-файле) или использовать секреты, если приложение разворачивается в облаке.

4.2 Логирование (Journald)

По умолчанию Spring Boot выводит логи в консоль (stdout). Systemd автоматически перехватывает этот поток и сохраняет его в системный журнал (journald).

Для просмотра логов используется команда journalctl:

1) journalctl -u myapp -f – просмотр логов в реальном времени (аналог tail -f).

2) journalctl -u myapp --since "1 hour ago" – логи за последний час.

Это избавляет от необходимости настраивать сложные appender-ы в logback.xml для ротации файлов, так как journald управляет размером логов самостоятельно.

4.3 Тюнинг JVM

В unit-файле в строке ExecStart важно передавать параметры JVM для ограничения потребления памяти, особенно в контейнерных средах или на VPS.

Ключевые флаги:

1) -Xms и -Xmx – начальный и максимальный размер кучи (Heap).

2) -XX:+UseG1GC – использование сборщика мусора G1, который оптимизирован для серверных приложений.

Пример: /usr/bin/java -Xmx512m -jar app.jar. Без этих ограничений Java может потребить всю доступную память сервера, что приведет к срабатыванию OOM Killer (Out Of Memory Killer) на уровне ОС и принудительному завершению процесса.

ЗАКЛЮЧЕНИЕ

В ходе написания реферата был детально рассмотрен процесс развёртывания Java Spring Boot приложений в промышленной среде Linux. Показано, что простой запуск через java -jar недостаточен для обеспечения требований надежности и безопасности, предъявляемых к современным информационным системам.

Использование systemd в качестве менеджера процессов позволяет автоматизировать запуск приложения, управлять его жизненным циклом и обеспечивать автоматический перезапуск при сбоях. Systemd тесно интегрируется с системой логирования ОС, предоставляя удобные инструменты для диагностики.

Настройка Nginx в качестве обратного прокси-сервера (Reverse Proxy) решает задачи безопасности, позволяя скрыть приложение за надежным веб-сервером, обеспечить терминацию SSL/TLS шифрования и эффективную раздачу статического контента. Была подчеркнута важность корректной передачи HTTP-заголовков (X-Forwarded-For) для правильной работы приложения за прокси.

Также были затронуты аспекты безопасности, такие как запуск от непривилегированного пользователя и ограничение ресурсов JVM. Комплексное применение рассмотренных технологий (Spring Boot + Systemd + Nginx) образует надежный, производительный и безопасный стек для эксплуатации Java-приложений, являющийся отраслевым стандартом вне контейнерных оркестраторов.

СПИСОК ЛИТЕРАТУРЫ

1. Walls, C. Spring Boot in Action / C. Walls. – Manning Publications, 2016. – 240 p. ISBN 978-1617292545.
2. Spring.io. Spring Boot Reference Documentation [Электронный ресурс]. – URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата обращения: 17.12.2025).
3. Freedesktop.org. systemd System and Service Manager [Электронный ресурс]. – URL: https://www.freedesktop.org/wiki/Software/systemd/ (дата обращения: 18.12.2025).
4. Nginx.org. NGINX Reverse Proxy [Электронный ресурс]. – URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (дата обращения: 18.12.2025).
5. Baeldung. Deploying Spring Boot Applications [Электронный ресурс]. – URL: https://www.baeldung.com/spring-boot-deployment (дата обращения: 18.12.2025).
6. DigitalOcean. How To Run Spring Boot as a Systemd Service [Электронный ресурс]. – URL: https://www.digitalocean.com/community/tutorials/how-to-deploy-a-spring-boot-application-on-ubuntu (дата обращения: 18.12.2025).
7. Oracle. Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide [Электронный ресурс]. – URL: https://docs.oracle.com/en/java/javase/17/gctuning/ (дата обращения: 18.12.2025).

Ссылка на Github: https://github.com/AIS-436/Zagidullin_Timur_Radicovich_9