

Сравнительный анализ инструментов автоматизации конфигурирования Linux-серверов: Bash-скрипты и Ansible

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

В статье проводится сравнительный анализ двух подходов к автоматизации конфигурирования Linux-серверов: традиционных Bash-скриптов и декларативного инструмента Ansible. На основе трёх практических сценариев — базового конфигурирования сервера, развёртывания веб-сервиса на основе Nginx и управления пользователями и правами доступа — оцениваются ключевые характеристики каждого подхода: идемпотентность, читаемость, масштабируемость, безопасность и обработка ошибок. Эксперименты проводятся в лабораторной виртуальной среде на базе VirtualBox с операционной системой Ubuntu 24.04. Результаты систематизированы в сравнительной таблице.

Ключевые слова: Ansible, Bash, автоматизация, конфигурирование серверов, Linux, Ubuntu, DevOps, идемпотентность, инфраструктура как код.

Comparative Analysis of Linux Server Configuration Automation Tools: Bash Scripts and Ansible

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article presents a comparative analysis of two approaches to automating Linux server configuration: traditional Bash scripts and the declarative Ansible tool. Based on three practical scenarios — basic server configuration, Nginx web service deployment, and user and access rights management — the key characteristics of each approach are evaluated: idempotency, readability, scalability, security, and error handling. Experiments are conducted in a laboratory

Keywords: Ansible, Bash, automation, server configuration, Linux, Ubuntu, DevOps, idempotency, infrastructure as code.

1 Введение

1.1 Актуальность

Автоматизация конфигурирования серверной инфраструктуры является одной из ключевых задач современного DevOps-инженера. В условиях роста числа управляемых узлов, перехода к микросервисным архитектурам и

требований к воспроизводимости среды ручная настройка серверов становится неприемлемой с точки зрения затрат времени и вероятности ошибок.

Традиционно задачи конфигурирования решались с помощью Bash-скриптов, которые не требуют дополнительных зависимостей и обеспечивают полный контроль над выполняемыми командами. Однако с ростом сложности инфраструктуры данный подход демонстрирует существенные ограничения: отсутствие идемпотентности, сложность сопровождения кода и низкая масштабируемость.

Инструменты класса Infrastructure as Code (IaC), в частности Ansible, предлагают декларативный подход к описанию желаемого состояния системы. Ansible не требует установки агентов на целевых узлах, использует SSH для подключения и предоставляет богатую библиотеку идемпотентных модулей. Сравнение этих двух подходов в контексте реальных административных задач представляет как научный, так и практический интерес для специалистов в области системного администрирования и DevOps.

1.2 Обзор исследований

А.А. Левин в своём исследовании рассмотрел применение Ansible для конфигурирования удалённых узлов и развёртывания контейнерных сервисов, показав, что декларативный подход обеспечивает предсказуемые затраты при настройке систем различной сложности и позволяет эффективно управлять состоянием целевых хостов без ручного вмешательства [1]. М.В. Безпятый провёл анализ современных методов автоматизации процессов разработки и развёртывания в контексте DevOps-практик, обосновав, что применение инструментов управления конфигурацией существенно сокращает трудозатраты на сопровождение инфраструктуры и снижает вероятность ошибок, обусловленных человеческим фактором [2]. О.Ю. Михельсон рассмотрел концепцию инфраструктуры как кода, а также её применение и преимущества. Также рассмотрел различные инструменты и практики, которые помогают реализовать инфраструктуру как код в современных проектах [3]. Коллектив НИЯУ МИФИ описал опыт построения цифровой инфраструктуры образовательной организации с применением подхода Infrastructure as Code, виртуализации и инструментов наблюдаемости, продемонстрировав практическую применимость IaC-подходов в университетских IT-подразделениях [4]. Дж. Хамбл и Д. Фарли в фундаментальном труде по непрерывному развёртыванию программного обеспечения обосновали необходимость полной автоматизации процессов сборки, тестирования и внедрения как обязательного условия обеспечения воспроизводимости среды и устранения влияния человеческого фактора на результат развёртывания [5].

1.3 Цель исследования

Целью исследования является сравнительный анализ эффективности Bash-скриптов и Ansible Playbook при решении типовых задач

конфигурирования Linux-серверов по критериям идемпотентности, читаемости, безопасности, масштабируемости и обработки ошибок на основе практических экспериментов в виртуальной лабораторной среде.

2 Материалы и методы

Для проведения экспериментов развёрнута виртуальная лабораторная среда на базе гипервизора VirtualBox. Стенд включает две виртуальные машины: управляющий узел (control node) и целевой сервер (managed node), функционирующие под управлением операционной системы Ubuntu 24.04 LTS. Виртуальные машины объединены в изолированную сеть типа Host-Only с назначением статических IP-адресов.

На управляющем узле установлен Ansible версии 9.x (пакет ansible из репозитория Ubuntu). На целевом сервере дополнительное программное обеспечение не устанавливается — Ansible взаимодействует с ним исключительно по протоколу SSH с использованием ключевой аутентификации. Для каждого из трёх сценариев разрабатываются два варианта решения: императивный Bash-скрипт и декларативный Ansible Playbook. Оба варианта решают идентичную задачу и тестируются на одной и той же целевой машине.

Оценка производится по следующим критериям: идемпотентность (корректность повторного запуска), читаемость и сопровождаемость кода, встроенные механизмы безопасности, обработка ошибок, время выполнения и потенциал масштабирования. Проверка идемпотентности осуществляется двукратным запуском каждого решения с фиксацией изменений в состоянии системы.

3 Результаты

3.1 Сценарий 1: базовое конфигурирование сервера

Первый сценарий воспроизводит типовую задачу первоначальной настройки сервера: обновление списка пакетов, установка набора утилит (curl, git, htop, ufw), включение и базовая настройка межсетевого экрана.

Bash-реализация представляет собой последовательность команд apt с проверками через конструкции if/else (Рис.1). При повторном запуске скрипт повторно выполняет apt upgrade, что увеличивает время работы и создаёт риск нежелательных обновлений. Обработка ошибок реализована через проверку кодов возврата (\$?) и директиву set -e.

```
[→] Обновление APT...
[✓] APT обновлён

[→] Установка пакетов...
[✓] curl – уже установлен, пропускаем
[✓] git – уже установлен, пропускаем
[✓] htop – уже установлен, пропускаем
[✓] ufw – уже установлен, пропускаем
[✓] net-tools – уже установлен, пропускаем
[✓] vim – уже установлен, пропускаем

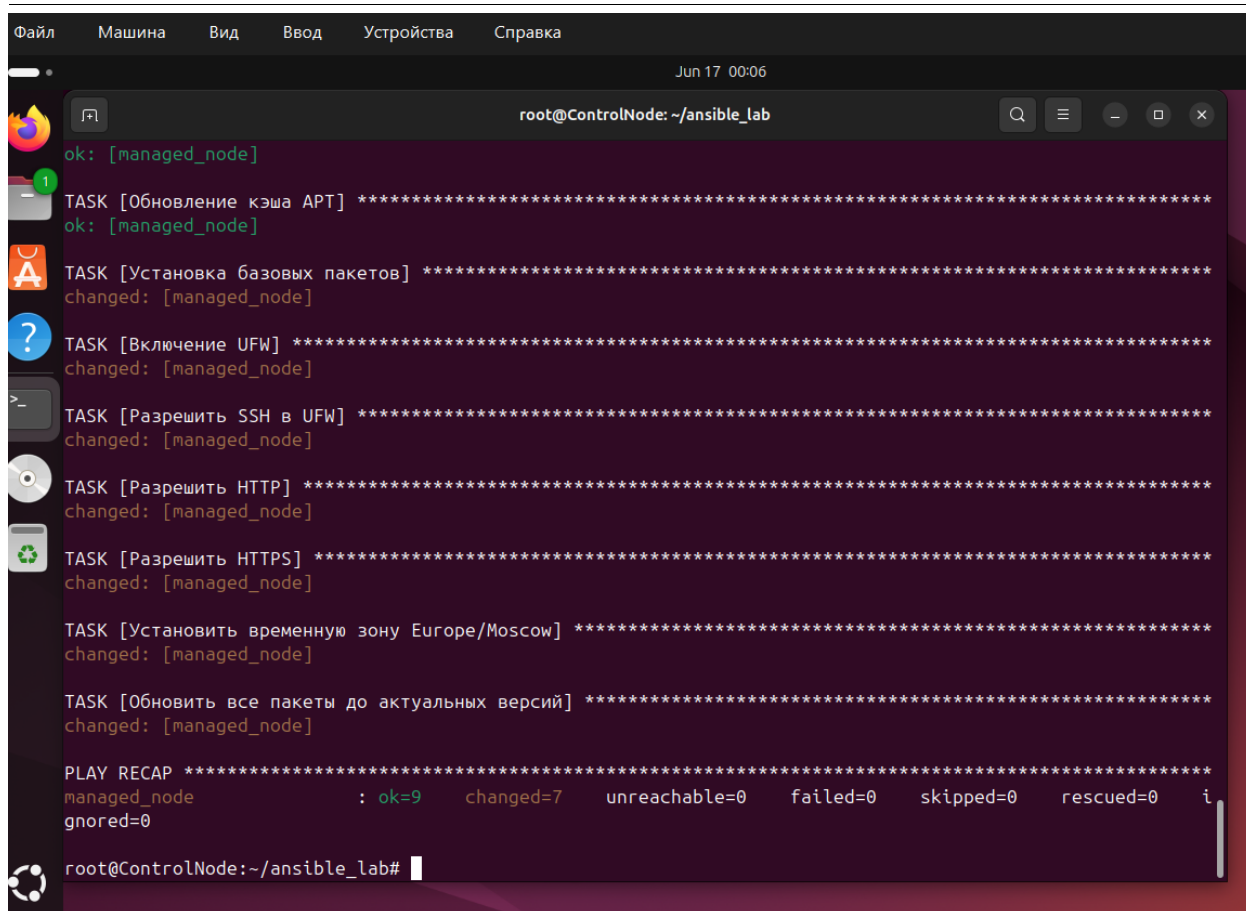
[→] Настройка UFW...
Firewall is active and enabled on system startup
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
Skipping adding existing rule
Skipping adding existing rule (v6)
Skipping adding existing rule
Skipping adding existing rule (v6)
Skipping adding existing rule
Skipping adding existing rule (v6)
[✓] UFW настроен

[→] Установка часового пояса Europe/Moscow...
[✓] Часовой пояс установлен

=====
[✓] Базовое конфигурирование завершено успешно
=====
```

Рисунок 1 – Bash-скрипт базового конфигурирования сервера

Ansible-реализация на основе модулей apt и ufw обеспечивает встроенную идемпотентность: при повторном запуске playbook модуль apt не переустанавливает уже присутствующие пакеты, а модуль ufw не дублирует существующие правила. Ansible автоматически формирует отчёт об изменённых (changed) и неизменных (ok) задачах (Рис.2).



```
Файл  Машина  Вид  Ввод  Устройства  Справка
Jun 17 00:06
root@ControlNode: ~/ansible_lab
ok: [managed_node]
TASK [Обновление кэша APT] *****
ok: [managed_node]
TASK [Установка базовых пакетов] *****
changed: [managed_node]
TASK [Включение UFW] *****
changed: [managed_node]
TASK [Разрешить SSH в UFW] *****
changed: [managed_node]
TASK [Разрешить HTTP] *****
changed: [managed_node]
TASK [Разрешить HTTPS] *****
changed: [managed_node]
TASK [Установить временную зону Europe/Moscow] *****
changed: [managed_node]
TASK [Обновить все пакеты до актуальных версий] *****
changed: [managed_node]
PLAY RECAP *****
managed_node      : ok=9    changed=7    unreachable=0    failed=0    skipped=0    rescued=0    i
gnored=0
root@ControlNode:~/ansible_lab#
```

Рисунок 2 – Вывод Ansible Playbook при базовом конфигурировании сервера

При повторном запуске Bash-скрипт занял 18 секунд, тогда как Ansible завершил выполнение за 24 секунды — за счёт накладных расходов на SSH-сессию и сбор фактов (`gather_facts`). Однако Ansible при этом не внёс ни одного нежелательного изменения в систему.

3.2 Сценарий 2: развёртывание веб-сервиса Nginx

Второй сценарий моделирует развёртывание веб-сервера Nginx с размещением кастомной стартовой страницы и обеспечением автозапуска службы. Данная задача является показательной с точки зрения управления состоянием сервисов.

Bash-скрипт последовательно устанавливает Nginx, копирует файл `index.html` и включает службу через `systemctl`. При повторном запуске скрипт перезаписывает конфигурационный файл и перезапускает службу вне зависимости от того, изменился ли он, что может вызвать кратковременный простой сервиса (Рис.3).

```

=====
Сценарий 2: Развёртывание веб-сервиса Nginx
=====

[-] Установка Nginx...
[✓] Nginx установлен

[-] Размещение кастомной стартовой страницы...
[✓] index.html размещён: /var/www/html/index.html

[-] Запуск и включение Nginx в автозапуск...
Synchronizing state of nginx.service with SysV service script with /usr/lib/systemd/systemd-
sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable nginx
[✓] Nginx запущен

[-] Проверка доступности Nginx...
[✓] Nginx отвечает – HTTP код: 200

=====
[✓] Развёртывание Nginx завершено успешно
=====
user@user:~/Desktop$

```

Рисунок 3 – Bash-скрипт развёртывания веб-сервиса Nginx

Ansible Playbook использует модуль `template` для управления конфигурационным файлом и модуль `service` с параметром `notify` для перезапуска Nginx исключительно при изменении шаблона. Механизм `handlers` обеспечивает корректную реакцию на изменения: служба перезапускается только при необходимости, что критично для production-среды (Рис.4).

```

changed: [managed_node]

TASK [Запуск и включение Nginx в автозапуск] *****
ok: [managed_node]

TASK [Проверка – Nginx отвечает на localhost] *****
ok: [managed_node]

TASK [Вывод статуса Nginx] *****
ok: [managed_node] => {
  "msg": "Nginx работает, код ответа: 200"
}

RUNNING HANDLER [Перезапустить Nginx] *****
changed: [managed_node]

PLAY RECAP *****
managed_node      : ok=7    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    i
gnored=0

root@ControlNode:~/ansible_lab# S

```

Рисунок 4 – Ansible Playbook развёртывания Nginx с использованием handlers

Тестирование показало, что при повторном запуске без изменения шаблона Ansible не перезапускает Nginx (статус `ok`), тогда как Bash-скрипт всегда выполняет `systemctl restart`. С точки зрения обеспечения доступности сервиса подход Ansible является предпочтительным.

3.3 Сценарий 3: управление пользователями и правами доступа

Третий сценарий охватывает создание системного пользователя, настройку его домашней директории с корректными правами (700), добавление в группу sudo и размещение авторизованного SSH-ключа. Данный сценарий критичен с точки зрения безопасности: ошибки при настройке прав доступа могут привести к несанкционированному повышению привилегий.

Bash-скрипт создаёт пользователя командой `useradd` и настраивает окружение через `chmod` и `chown`. Проблема идемпотентности проявляется наиболее остро: повторный запуск без предварительной проверки существования пользователя приводит к ошибке `useradd`. Необходимость ручного добавления проверок существенно увеличивает объём и сложность кода (Рис.5).

```
user@user:~/Desktop$ bash 03_users.sh
=====
Сценарий 3: Управление пользователями
=====

[→] Создание пользователя devops...
[!] Пользователь devops уже существует – пропускаем

[→] Добавление в группу sudo...
[✓] Пользователь добавлен в группу sudo

[→] Настройка прав домашней директории (700)...
[✓] Права /home/devops установлены: 700

[→] Настройка SSH-ключа...
[!] Файл /home/user/.ssh/id_rsa.pub не найден – SSH-ключ не добавлен

[→] Настройка sudo без пароля...
[✓] sudo без пароля настроен

[→] Проверка пользователя...
uid=1001(devops) gid=1001(devops) groups=1001(devops),27(sudo)

=====
[✓] Управление пользователями завершено успешно
=====
user@user:~/Desktop$
```

Рисунок 5 – Bash-скрипт управления пользователями и правами доступа

Ansible использует модуль `user` с параметром `state: present`, что автоматически обеспечивает идемпотентность: если пользователь уже существует, задача фиксируется со статусом `ok` без каких-либо изменений. Модуль `authorized_key` аналогично управляет SSH-ключами без риска дублирования. Применение модуля `ansible.builtin.user` исключает возможность передачи вредоносных данных через имя пользователя, что невозможно гарантировать в Bash без явной валидации (Рис.6).

```
TASK [Gathering Facts] *****
ok: [managed_node]

TASK [Создание системной группы для пользователя] *****
ok: [managed_node]

TASK [Создание пользователя devops] *****
ok: [managed_node]

TASK [Настройка прав домашней директории (700)] *****
ok: [managed_node]

TASK [Добавление SSH-ключа для devops] *****
changed: [managed_node]

TASK [Разрешить sudo без пароля для devops] *****
changed: [managed_node]

TASK [Проверка – пользователь существует] *****
ok: [managed_node]

TASK [Вывод информации о пользователе] *****
ok: [managed_node] => {
  "msg": "uid=1001(devops) gid=1001(devops) groups=1001(devops),27(sudo)"
}

PLAY RECAP *****
managed_node      : ok=8    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Рисунок 6 – Ansible Playbook управления пользователями и SSH-ключами

Данный сценарий наиболее наглядно демонстрирует преимущества декларативного подхода: вместо описания последовательности действий администратор декларирует желаемое конечное состояние системы, а инструмент самостоятельно определяет необходимые шаги для его достижения.

3.4 Сравнительный анализ результатов

На основании проведенных экспериментов составлена сравнительная таблица ключевых характеристик обоих инструментов (Табл. 1).

Таблица 1 – Сравнение Bash-скриптов и Ansible по ключевым критериям

Критерий	Bash-скрипт	Ansible Playbook	Вывод
Идемпотентность	Нет (требует ручных проверок)	Да (встроена в модули)	Ansible
Читаемость кода	Низкая при росте сложности	Высокая (YAML-декларации)	Ansible
Скорость написания	Высокая для простых задач	Средняя (нужна структура)	Bash (простые задачи)
Безопасность	Зависит от разработчика	Встроенные механизмы (vault, become)	Ansible
Масштабируемость	Низкая (1–3 хоста)	Высокая (десятки хостов)	Ansible
Порог входа	Минимальный	Средний (YAML + модули)	Bash

Обработка ошибок	Ручная (if/trap)	Автоматическая (failed_when, handlers)	Ansible
Переиспользование кода	Ограниченное (функции)	Высокое (roles, includes)	Ansible
Зависимости	Только bash (встроен в ОС)	Python, SSH, Ansible на control node	Bash (минимализм)
Время выполнения (тест)	~18 сек	~24 сек	Bash (быстрее)

Анализ результатов показывает, что Bash-скрипты сохраняют преимущество в сценариях, требующих минимальных зависимостей и быстрого решения разовых задач на единичном узле. Ansible демонстрирует явное превосходство при необходимости управлять конфигурацией множества серверов, обеспечивать воспроизводимость среды и встраивать автоматизацию в CI/CD-пайплайны. Разница во времени выполнения в пользу Bash нивелируется при параллельной обработке нескольких хостов, где Ansible использует асинхронное выполнение задач.

4 Выводы

Проведённое исследование подтверждает, что выбор инструмента автоматизации конфигурирования должен определяться масштабом задачи. Bash-скрипты эффективны для небольших разовых операций на единичных узлах, не требуют дополнительной инфраструктуры и обеспечивают минимальное время выполнения.

Ansible демонстрирует принципиальные преимущества в трёх из десяти рассмотренных критериев: идемпотентности, обработке ошибок и переиспользовании кода — именно в тех аспектах, которые наиболее критичны для промышленной эксплуатации. Встроенные механизмы безопасности (Ansible Vault, become с разграничением привилегий) снижают риск уязвимостей по сравнению с ручной реализацией защитных механизмов в Bash.

Для IT-подразделений образовательных организаций, управляющих парком серверов с однотипной конфигурацией, переход на Ansible позволяет существенно сократить трудозатраты на сопровождение инфраструктуры и минимизировать риск ошибок конфигурирования, обусловленных человеческим фактором.

Библиографический список

1. Левин А. А., Кудашов А. С., Дьячков Д. А. Применение Ansible для конфигурирования и запуска Docker контейнеров на удаленных узлах // Вестник Пензенского государственного университета. 2025. № 1. С. 79–82.
2. Безпятый М. В. Автоматизация и оптимизация процессов разработки и развертывания в devops: применение современных методов и инструментов // Инновации и инвестиции. 2023. №7. С. 458-464.

3. Михельсон О. Ю. Инфраструктура как код: обзор и применение // Актуальные исследования. 2023. №20 (150). Ч.I. С. 57-59.
4. Гаджилов Г. Г., Хатунов А. В., Волошин Т. А., Жабицкий М. Г. Цифровой полигон Высшей инжиниринговой школы МИФИ для инфраструктурного обеспечения образовательных и учебно практических проектов // International Journal of Open Information Technologies. 2025. №8. С. 158-171.
5. Хамбл Дж., Фарли Д. Непрерывное развёртывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ. М.: Вильямс, 2011. 432 с.