Практика из РОАДМАПА

1. Git

Цель: довести базовые команды до автоматизма, чтобы к концу блока Linux пальцы сами набирали `git push`.

Шаг 1 — Создать репозиторий

Зарегистрируйтесь на GitHub или GitLab. Создайте пустой репозиторий — он станет вашим конспектом на весь путь

Шаг 2 — Освоить 5 базовых команд

Всё, что нужно на старте:

1. `git init` — инициализировать репозиторий
2. `git add .` — добавить изменения
3. `git commit -m "описание"` — зафиксировать
4. `git push` — запушить в репу
5. `git pull` — забрать изменения

Шаг 3 — Разобраться с ветками и тегами

- Создать ветку, переключиться между ветками
- Понять, зачем нужны теги

Шаг 4 — Научиться писать Markdown

Начните вести конспекты в `.md`. В этом формате пишутся все README, документация к проектам, и нейронки рендерят ответы тоже в Markdown.

Шаг 5 — Выработать привычку

С этого момента и до конца обучения: **каждый день** `пишете конспект по тому, что изучили → коммитите → пушите`

2. Linux

Цель: перестать бояться терминала и научиться жить в Linux.

Этап 1 — Базовое освоение

Шаг 1 — Поставить Linux

Установить Linux второй системой на компьютер (Ubuntu/Fedora — что угодно). Не в виртуалку, а именно второй системой — чтобы было больно и полезно.

Шаг 2 — Начать на нём жить

Переключиться на Linux для повседневных задач. У вас отвалится звук, слетят драйвера, разрешение будет кривое — это нормально. Пока будете чинить, перестанете бояться терминала.

Шаг 3 — Освоить базовые команды

- `cd` — переход между директориями
- `mkdir`, `touch` — создание папок и файлов
- `rm`, `rm -rf` — удаление
- Запуск скриптов
- Редактирование файлов в `nano` или `vim`

Шаг 4 — Пройти тренажёр по Vim

Этап 2 — Процессы

Шаг 5 — ps aux и kill

- Ввести `ps aux`, научиться находить нужный процесс и читать его PID
- Убить зависший процесс через `kill <PID>`

- Понять разницу между `kill`, `kill -9`, `kill -15`

Шаг 6 — SystemD

- Понять, что такое SystemD и чем отличается от InitV
- Написать простой unit-файл, чтобы ваш сервис запускался автоматически после перезагрузки

Шаг 7 — Разобрать теорию по процессам

Для собесов: зомби-процесс, процесс-сирота, OOM Killer, директория `/proc`.

Этап 3 — Мониторинг ресурсов

Шаг 8 — top / htop

- Запустить `top`, разобрать все параметры: SI, WA, Load Average
- Научиться сортировать процессы по CPU и памяти (колонка RES)
- Найти процесс, который жрёт ресурсы, и убить его

Этап 4 — Диски

Шаг 9 — df + du

1. `df -h` — посмотреть общую картину, найти раздел, где кончается место
2. `du -sh *` — зайти в этот раздел и найти, что именно всё съело

Шаг 10 — mount и fstab

- Примонтировать диск через `mount`
- Прописать в `/etc/fstab`, чтобы монтирование сохранилось после перезагрузки

Шаг 11 — Дополнительно по дискам

- Разобраться с inode (диск не полный, а система говорит «нет места»)
- Изучить типы файловых систем: ext4, xfs, tmpfs, overlay

- Почитать про RAID (1, 0, 10) и LVM

Этап 5 — Главная практика по Linux

Шаг 12 — Сгенерировать приложение

Попросить нейронку (ChatGPT, Claude, Gemini, etc..) сгенерировать код простейшего приложения: **Фронтенд + Бэкенд + База Данных**. Условие: никакого Docker, голое приложение.

Шаг 13 — Запустить руками на Linux

- Python → разобраться с `pip`, зависимостями, конфликтами версий
- Node.js → разобраться с `npm`
- Java → разобраться с Maven

Шаг 14 — Поднять PostgreSQL

1. Установить PostgreSQL
2. Создать пользователя
3. Выдать права
4. Открыть порт
5. Создать базу
6. Подключить к бэкенду

> **Этот проект — вы можете развивать на всем пути обучения.** Дальше вы будете упаковывать его в Docker, прогонять через CI/CD, деплоить в Kubernetes.

3. Сети

Цель: уметь диагностировать сетевые проблемы и настроить базовую инфраструктуру.

Шаг 1 — Столкнуться с проблемами

Если делали практику из предыдущего блока — проблемы уже есть: backend не видит базу, сайт не открывается, порт закрыт. Начните с их решения.

Шаг 2 — Освоить команды диагностики

- `ip -a` — информация о своём IP
- `ping` — проверить, жив ли сервер
- `ss -tulpn` / `netstat -tulpn` — какие порты открыты и кто их занимает

Шаг 3 — Настроить Nginx как reverse proxy

1. Установить Nginx
2. Написать конфиг, чтобы запрос приходил на Nginx, а он перекидывал на приложение
3. Запомнить: применять изменения через `nginx -s reload`, а не `restart`

Шаг 4 — Настроить SSH

1. Сгенерировать ключ: `ssh-keygen`
2. Закинуть на сервер: `ssh-copy-id`
3. Отключить вход по паролю в конфиге
4. Отключить вход из-под root

Шаг 5 — Установить fail2ban

Поставить и настроить fail2ban для защиты от брутфорса

4. Docker

Цель: научиться упаковывать приложение в контейнер и работать с образами.

Шаг 1 — Освоить базовые команды

1. `docker build` — собрать образ из Dockerfile
2. `docker tag` — протегировать образ

3. `docker login` — подключиться к registry
4. `docker push` — отправить образ в registry

Шаг 2 — Оптимизировать Dockerfile

Взять Dockerfile (свой, сгенерированный или любой другой) и применить бест практисы:

- Использовать Alpine как базовый образ
- Чистить кэш
- Не запускать от root
- Оптимизировать слои для ускорения сборки

Шаг 3 — Разобраться с registry

Понять концепцию: собрали образ → запушили в registry → скачали на следующем стенде → выложили. Для практики хватит Docker Hub или GitLab Registry.

Шаг 4 — Практика с мониторингом

1. Взять готовый `docker-compose.yml` для **Prometheus + Grafana**
2. Запустить: `docker compose up -d`
3. Подключить **Node Exporter** для своей виртуалки
4. Настроить графики и построить дашборды в Grafana

> Эта практика убивает двух зайцев: и Docker осваиваете, и с мониторингом знакомитесь.

5. CI/CD

Цель: автоматизировать сборку, тестирование и деплой вашего проекта.

Шаг 1 — Поставить GitLab Runner

Установить GitLab Runner на свою виртуалку и подключить к GitLab — это тот агент, который будет выполнять пайплайн.

Шаг 2 — Написать pipeline из трёх стадий

Создать файл `.gitlab-ci.yml` в корне проекта:

1. **Build** — собрать Docker-образ вашего приложения
2. **Test** — подключить линтер (flake8 для Python, eslint для JS и т.д.)
3. **Deploy** — подключиться по SSH к серверу и обновить контейнер

Шаг 3 — Запустить и отладить

Запушить `.gitlab-ci.yml`, посмотреть, как пайп отрабатывает, починить ошибки.

Шаг 4 — Для общего развития

Разобрать пару пайпов на Groovy (Jenkins) — чтобы понимать, что это такое, когда встретите на работе.

6. Ansible

Шаг 1 — Создать inventory

Создать файл `inventory` с IP-адресами серверов. Разбить на группы: web, db, backend.

Шаг 2 — Написать первый плейбук

Плейбук для новой виртуалки, который:

1. Обновит систему
2. Поставит Docker
3. Создаст пользователя
4. Настроит SSH
5. Установит нужные пакеты

Шаг 3 — Поработать с модулями

- `apt` / `yum` — установка пакетов

- `copy` — копирование файлов на сервер
- `template` — шаблоны конфигов с переменными

Шаг 4 — Написать роли

Когда плейбук разрастётся — разбить на роли:

- Роль для установки Nginx
- Роль для установки базы данных
- Роль для базовой настройки сервера

Шаг 5 — Масштабировать

Поднять несколько виртуалок (или арендовать в облаке) и прогнать плейбук на группу серверов — посмотреть, как Ansible работает с несколькими машинами.

7. Kubernetes

Цель: научиться деплоить приложения в кубер и управлять ими.

Этап 1 — Базовая практика

Шаг 1 — Поднять локальный кластер

Поставить **Minikube** или **k3d** (Kubernetes в Docker)

Шаг 2 — Написать все манифесты руками

Каждый манифест — хотя бы один раз вручную, чтобы прочувствовать параметры:

1. Pod
2. Deployment (+ rolling update)
3. StatefulSet (+ понять отличие от Deployment)
4. DaemonSet, Job, CronJob
5. ConfigMap, Secret

6. Service (ClusterIP, NodePort, LoadBalancer)
7. Ingress
8. PV, PVC, StorageClass

Шаг 3 — Настроить requests, limits и probes

Для каждого деплоймента прописать ресурсы и пробы.

Этап 2 — Проектная практика

Шаг 4 — Задеплоить проект в кубер

Взять свои frontend и backend и задеплоить в Kubernetes. Базу данных можно оставить на сервере — главное настроить подключение.

Шаг 5 — Написать Helm chart

1. Создать Helm chart для frontend
2. Создать Helm chart для backend
3. Задеплоить через Helm, как делают на реальных проектах

Этап 3 — Продвинутое

Шаг 6 — Попробовать managed Kubernetes в облаке

Арендовать кластер (Yandex Cloud и другие дают бонусы новым пользователям).

Шаг 7 — Изучить дополнительные темы

- Сетевые политики (Network Policies)
- RBAC (модели доступа)
- Istio (Service Mesh)
- Операторы для баз данных
- Способы развёртывания: bare metal и облака

8. Мониторинг и логирование

Цель: уметь понимать, что происходит с системой, и вовремя реагировать.

Шаг 1 — PromQL

Изучить язык запросов Prometheus. Важнее понять логику, чем выучить синтаксис.

Шаг 2 — Экспортеры

Поработать с экспортерами: node-exporter (серверы), postgres-exporter (база данных).

Шаг 3 — Grafana

- Разобраться с дашбордами и панелями
- Посмотреть публичную библиотеку готовых дашбордов
- Построить свои графики

Шаг 4 — Алертинг

Настроить алерты — чтобы приходили уведомления, когда что-то идёт не так.

Шаг 5 — Логирование

Познакомиться с ELK (Elasticsearch + Logstash + Kibana) / OpenSearch или Loki. Главное — научиться читать логи и понимать, куда копать.

9. Terraform и облака

Цель: уметь описывать инфраструктуру как код.

Шаг 1 — Поработать с облаками

В процессе изучения всех предыдущих инструментов арендовать машинки в облаках: Yandex Cloud, Timeweb, Selectel, Cloud.ru.

Шаг 2 — Разобраться с tfstate

Понять, что `.tfstate` — файл состояния инфраструктуры. Хранить его в S3, чтобы не потерять.

Шаг 3 — Модули

Научиться структурировать код с модулями для переиспользования.

Шаг 4 — Окружения

Понять, как работать с параметрами для разных окружений (dev, staging, prod) и как их изолировать.

Шаг 5 — Почитать про S3

Разобраться с объектным хранилищем — пригодится и для tfstate, и в целом по работе.