

День 5. Gitlab CI. Построение релизного pipeline (конвейера для развертывания новой версии приложения).

Теперь рассмотрим способы, которыми мы можем развернуть наше приложение на сервере. В случае, если мы имеем один или несколько серверов мы можем воспользоваться различными способами. Самый простой – выполнить последовательность команд на удаленном сервере, подключившись туда по протоколу [ssh](#). Осуществить это можно, написав простой скрипт на `bash`. Для этого нужно будет произвести предварительную настройку сервера, установить все зависимости. Другой, более продвинутый способ – использовать систему управления конфигурациями `ansible` (наверняка способов гораздо больше, но мы поговорим только об этих). Для использования `ansible` нам необходимо установить `python` на целевые хосты (так как он написан на `python`), а также положить публичные `ssh` ключи пользователя, под которым мы будем подключаться к ним. Конфигурация `ansible` – `yaml` файлы.

Структура стандартного `ansible` проекта состоит из

- окружений (`inventory`),
- плейбуков (`playbooks`)
- и ролей (`roles`).

Inventory содержит список хостов и параметров для них. **Плейбуки** объединяют роли и хосты, для которых они должны примениться. **Роли** – список заданий и шаблонов для выполнения конкретного действия (установки приложения, например). Структура директорий может быть различной, но данные абстракции сохраняются. На рис. 1 мы видим пример простого плейбука, в котором выполняется установка `apache` сервера и копирование файла `index.html` для раздачи на сервере. Даже человеку, который никогда не работал с данным инструментом, интуитивно понятны шаги установки. Каждый плейбук может состоять напрямую из заданий (`tasks`), как здесь, что не очень удобно, так как не позволяет их переиспользовать в других плейбуках, либо из ролей. За различные действия, такие как установка пакета, копирование файла, работа с докер контейнером в `ansible` отвечают различные модули. В нашем примере в 1-ой таске используется модуль `yum`, который отвечает за работу с пакетным менеджером `yum`. Документацию по модулям можно найти на [официальном сайте](#).

AN EXAMPLE APACHE.YML PLAYBOOK MIGHT LOOK LIKE:

```
---
- name: Apache server installed
  hosts: web
  become: true
  tasks:

  - name: latest Apache version installed
    yum:
      name: httpd
      state: latest

  - name: Apache enabled and running
    service:
      name: httpd
      enabled: true
      state: started

  - name: copy index.html
    copy:
      src: web.html
      dest: /var/www/html/index.html
```

Рисунок 1 Пример простого playbook

Таким образом, для наших целей нам понадобится модуль `docker_container`. Если приложению требуется какой-либо конфиг, будет крайне плохой практикой вшивать его в контейнер, так как для каждого изменения конфига придется пересобирать контейнер. Таким образом наилучшим способом будет воспользоваться модулями `file` либо `template`. Второй позволяет подставлять использовать `jinja` шаблоны и параметризировать, генерировать файлы для различных окружений.

Наиболее современным способом является использование [оркестраторов](#), в том числе Kubernetes.

Рассмотрим релизный пайплайн в большой команде разработки. При добавлении какого-либо функционала в приложение разработчик создаст новую ветку. (рис. 2)

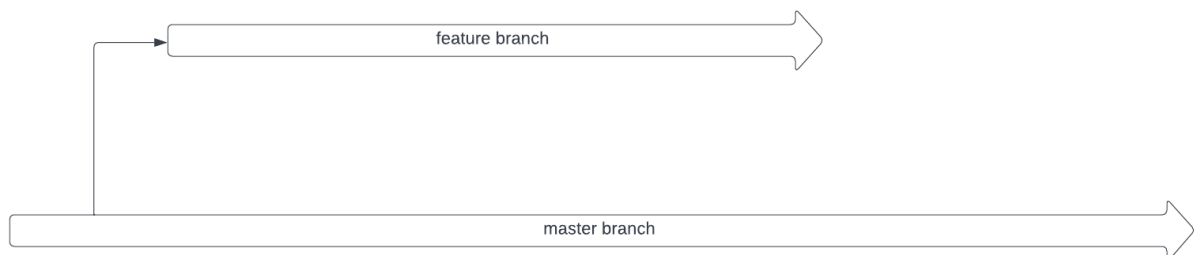


Рисунок 2

Однако ему будет необходимо прогнать тесты, написанные тестировщиками, чтобы покрыть основные кейсы, посмотреть что-то самому, проверить на схожей с production инфраструктуре. Для этого ему понадобится динамический стенд – то есть имеющий очень короткий срок жизни. Он не оставляет после себя артефактов и автоматически удаляется (рис. 3)

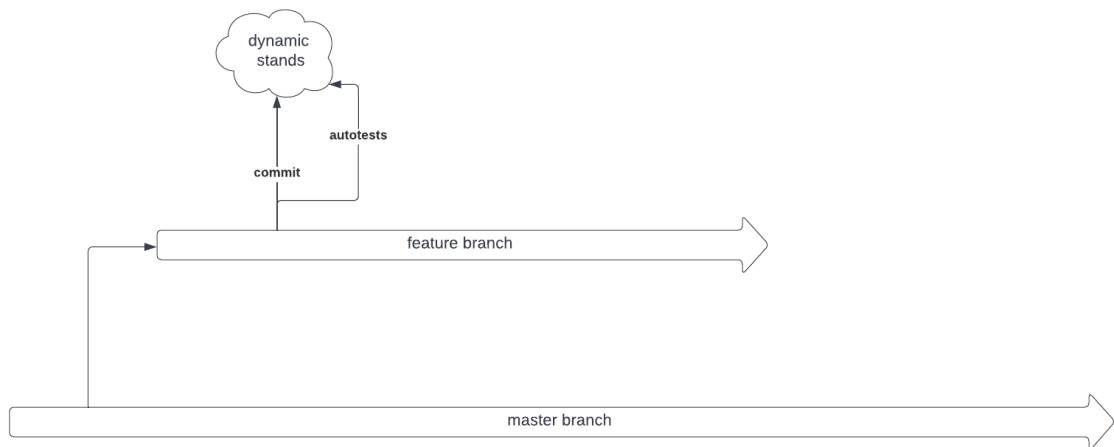


Рисунок 3

Однако на таком стенде невозможно проверить интеграцию с другими приложениями. Поэтому приложение, пришедшее первичную проверку, отправляется на статические стенды, где более тщательно проверяется с персистентными данными, а также проверяются интеграции с другими приложениями (рис. 4).

В конце концов, после всего тестирования, код ревью (code review) на merge request и merge в мастер, код попадает на продакшен контур с помощью автоматизации (рис. 5).

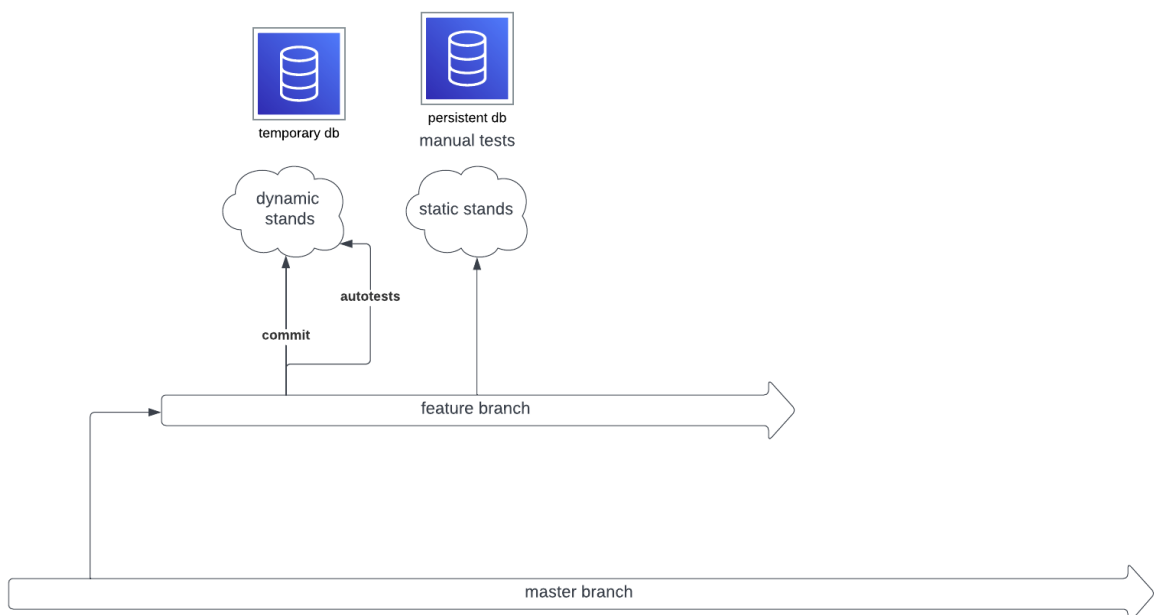


Рисунок 4

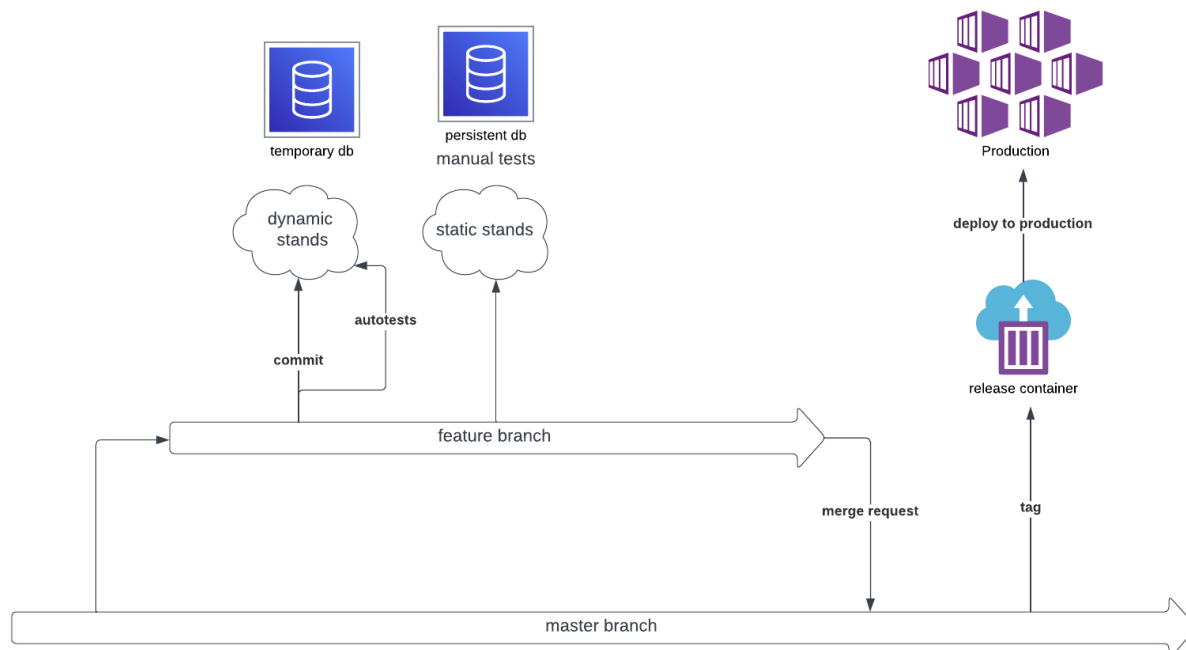


Рисунок 5

Полезные ссылки

1. <https://docs.gitlab.com/ee/ci/>
2. <https://www.ssh.com/academy/ssh/protocol#typical-uses-of-the-ssh-protocol>
3. <https://docs.ansible.com/>
4. <https://habr.com/ru/company/kts/blog/591355/#2>
5. <https://docs.gitlab.com/ee/api/>
6. <https://about.gitlab.com/topics/gitops/>