

Конспект

Урок 1

Непрерывная интеграция (Continuous Integration, CI) и непрерывная поставка (Continuous Delivery, CD) представляют собой культуру, набор принципов и практик, которые позволяют разработчикам чаще и надежнее разворачивать изменения программного обеспечения. Есть много систем, которые помогают реализовывать данный подход, такие как Jenkins, Teamcity, Gitlab. Мы будем рассматривать данный подход на примере Gitlab. Этот инструмент поможет нам решить такие задачи, как

- Деплой приложений (развертывание на удаленном сервере/в кластере)
- Запуск скриптов автоматизации (бекапы, любые регулярные операции)
- Запуск тестовых стендов
- Запуск автотестов, линтеров
- Контроль качества кода (код-ревью)
- Отслеживание состояния продакшн-окружения
- Возвращение контура к предыдущему состоянию

Рассмотрим простой пайплайн для проекта на Django, который мы хотим запускать в Docker контейнере





Name	Last commit	Last update
 mysite	init	2 weeks ago
 .gitlab-ci.yml	fix	2 weeks ago
 Dockerfile	init	2 weeks ago
 README.md	Initial commit	2 weeks ago

Рисунок 1

Описание пайплайна находится в файле .gitlab-ci.yml. также нам нужен сам код проекта в папке mysite и Dockerfile. Файл readme.md может содержать краткую информацию о проекте.

Рассмотрим содержание файла .gitlab-ci.yml (рис. 2)

Начинается он с объявления переменной, которая будет помещена как переменная окружения во всех джобах. Порядок блоков файла на самом деле может быть любым, так как очередность джоб определяется блоком stages – стадии. Каждая джоба должна принадлежать к стадии.

Далее следует описание джоб. В джобе pylint мы указываем docker образ, в котором мы хотим запускать скрипт, а также переопределяем его entrypoint (точка входа, стартовая команда). Далее идет скрипт - bash команды.

Во второй джобе отличительной чертой будет блок rules, который говорит о том, что джоба должна запускаться только при наличии переменной CI_COMMIT_TAG. Хитрость состоит в том, что Gitlab автоматически добавляет данную переменную, если был создан тег. Подробное описание всех опций можно посмотреть в [документации](#).

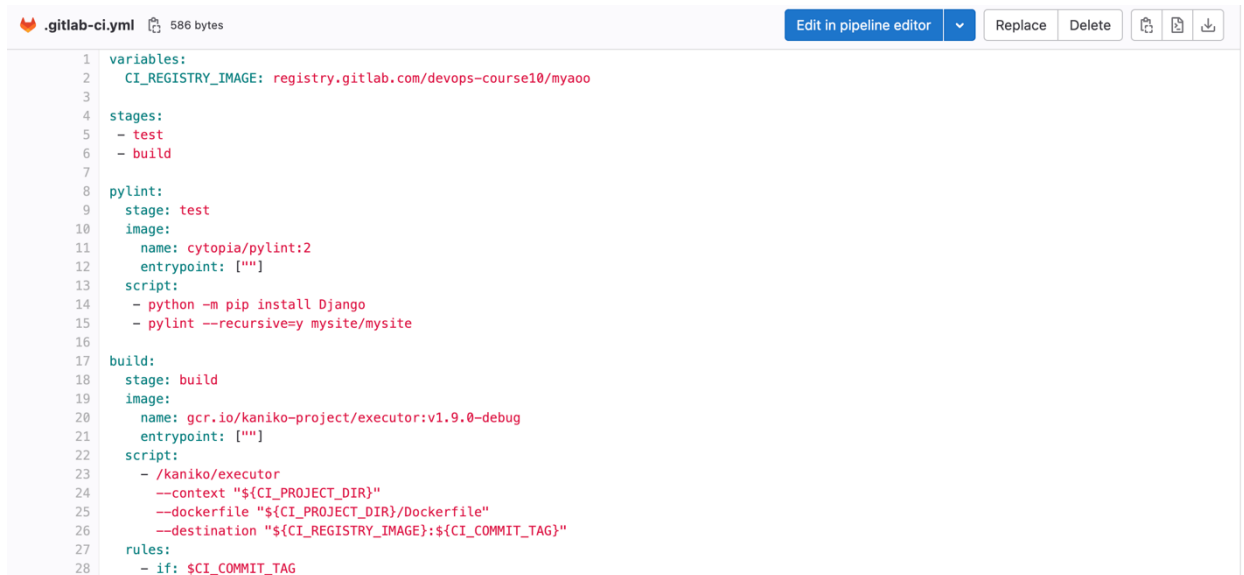


Рисунок 2

На вкладке CI/CD -> Pipelines можно увидеть все ваши пайплайны, которые запускались. По клику на них можно увидеть джобы, и посмотреть вывод докер контейнера.

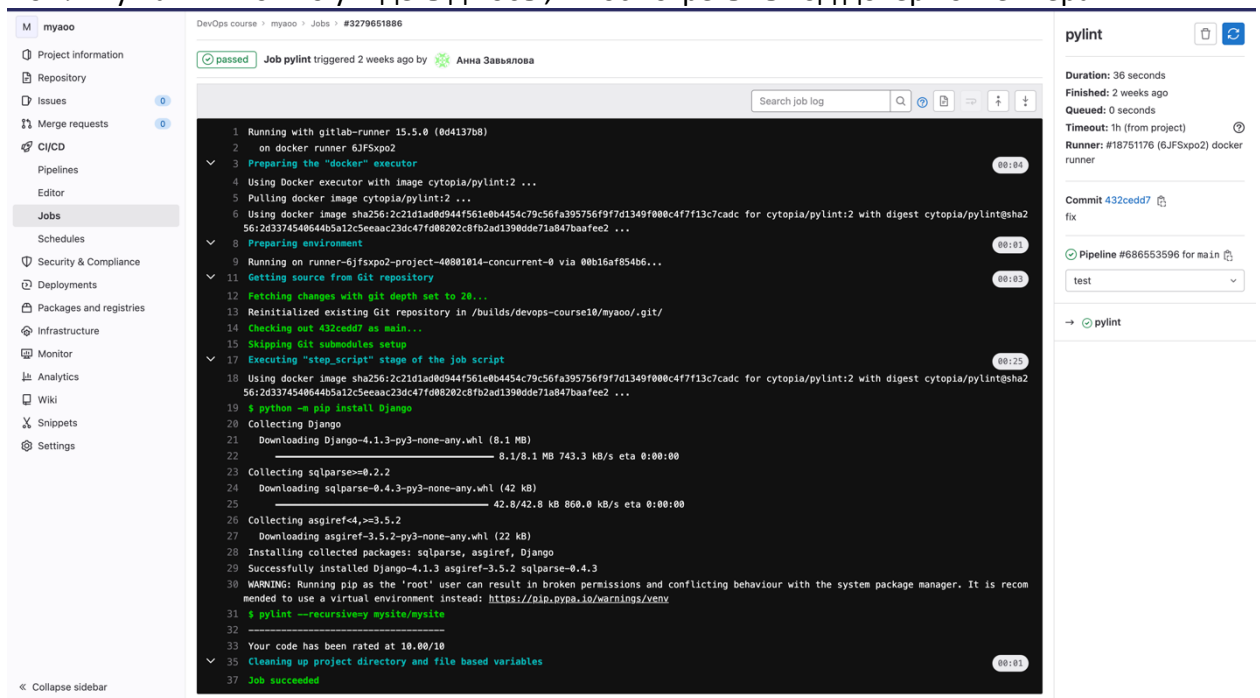


Рисунок 3

Таким образом разработчики без участия инженеров могут выполнять безопасные частые операции с контурами, заниматься тестированием, проводить релизы и тд. Однако для этого стоит обезопасить себя от человеческого фактора средствами Gitlab:

1. Деплоить только из master ветки.
Эта ветка будет отражать состояние production контура, и все основные средства защиты должны охранять production контур от случайных ошибок. Таким образом всегда будет понятно какая версия приложения/конфигурации находится на продакшене и успешен ли был последний деплой.
2. Управлять ролями участников проекта.

Возможность менять переменные окружения, а иногда и запускать пайплайны должна быть не у всех, так как это может привести к нарушению работы команды либо серьезным проблемам на продакшене, если будет выполнено без понимания процесса.

3. Запретить пуш в мастер ветку вне MR.
Коммиты к мастер ветке добавляются только после прохождения всех тестов и линтеров, а может и ревью от коллег.
4. Использовать защищенные ветки.
Они имеют множество преимуществ, например их невозможно удалить, а также можно использовать защищенные переменные, которые нельзя будет напечатать с помощью 'cat'.
5. Запретить пушить секреты
SSL сертификаты, SSH ключи и другие секреты не должны находиться в публичном репозитории – и даже в истории коммитов. В Gitlab есть механизм обнаружения таких файлов.

Также есть несколько хороших практик в построении ваших пайплайнов, которые помогут сберечь нервы.

1. Меньше дублирования – 1 репозиторий с CI на все проекты.
Копи-паст это всегда зло, так как вносит неуловимые ошибки. Также как и бесконечная вариативность увеличит временные затраты на поддержку до бесконечности. Проще иметь стандартизированный процесс.
2. Ручной запуск «опасных» джоб.
Если нам нужно время, чтобы понять, что приложение запустилось успешно, либо если планируются изменения на инфраструктуре, которые могут повлиять на других, лучше сделать такие джобы ручными и запускать по готовности.
3. Экономия ресурсов ранеров.
Выбирайте более легкие docker-образы, а также в скрипте старайтесь не проводить тяжелых по сри или памяти операций.
4. Не использовать `ter latest`!
Используйте уникальные теги, чтобы всегда знать, какая версия приложения находится на проде
5. Отделять develop-образы от релизных
Для тестирования собирается много образов, которые должны очищаться.

Также одним из современных подходов является [GitOps](#), который идет под руку с IAC (Infrastructure as Code / Инфраструктура как код). GitOps предполагает управление инфраструктурой средствами git, что возможно только если инфраструктура описана в виде кода или набора конфигураций. С этим нам помогут такие инструменты как Ansible, Terraform, Kubernetes. Terraform используется для описания инфраструктуры дата центров, тогда как с помощью Ansible мы можем развернуть приложения/инфраструктурные сервисы на классических серверах/виртуальных машинах. Преимущества данных систем:

- Скорость – нам не нужно выполнять множество операций вручную
- Безопасность – уменьшается риск человеческого фактора, ограничивается доступ к серверам
- Масштабируемость – добавление новой машины в кластер решается 1 строчкой кода
- Стандарт – описание может понять любой
- Восстановление после аварий – достаточно запустить пайплайн