Continuous Integration

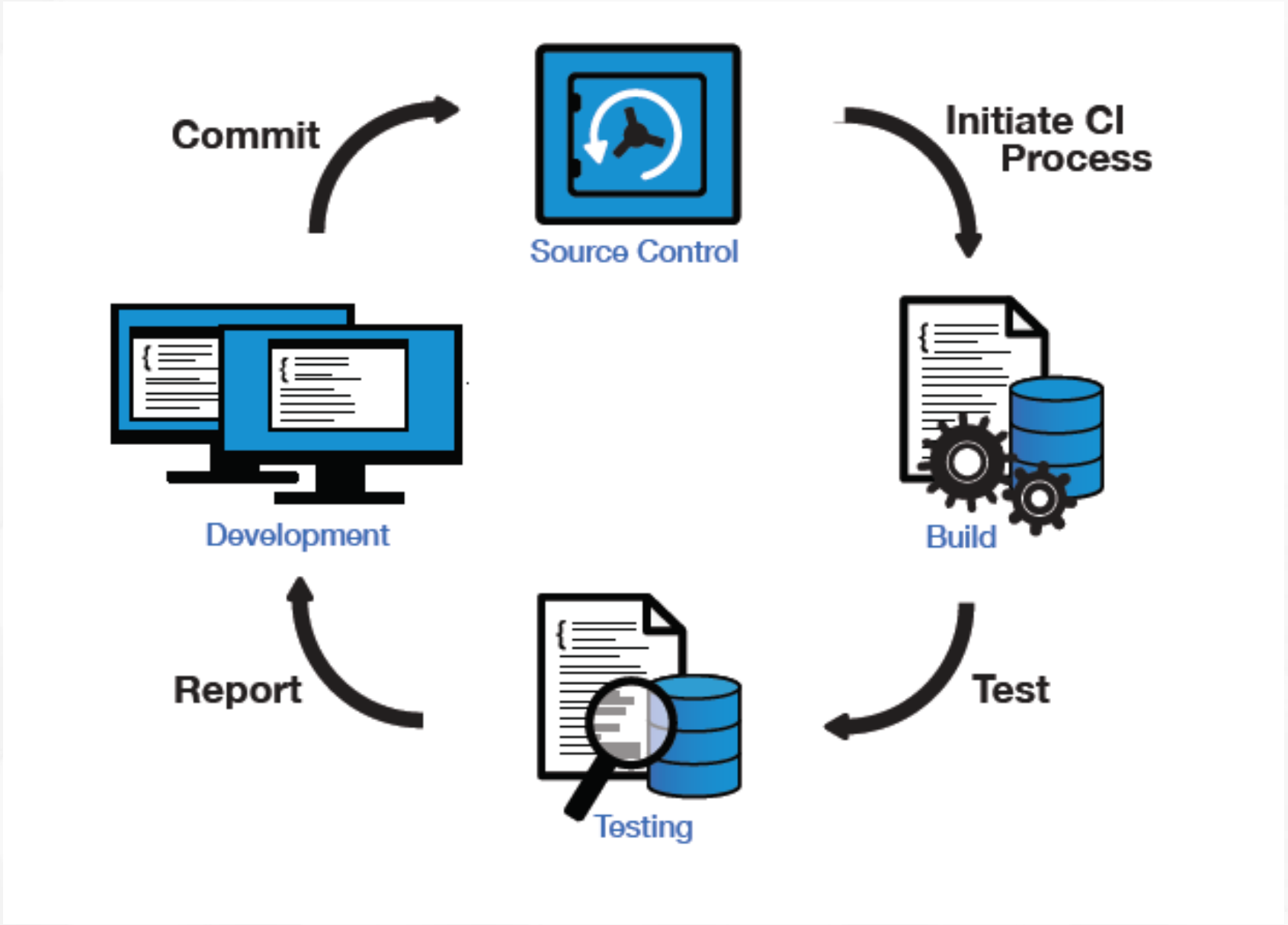
Continuous Integration

- это практика разработки программного обеспечения, в которой члены команды проводят интеграцию не реже чем раз в день. Результаты интеграции проверяются автоматически, используя автотесты и статический анализ кода.
- Использование CI позволяет вовремя отслеживать ошибки интеграции, сделать систему и процесс разработки более «прозрачными» для всех участников команды
- Фактически, CI позволяет избавиться от предположений, при процессе разработки ПО. Менеджер предполагает, что продукт готов и стабилен, программист — что в коде нет ошибок и т. д.

Идеологически CI базируется на следующих соглашениях

- часто (не менее 1 раза в день) «заливать» свой код в репозиторий
- писать автоматические тесты
- запускать private builds (процесс сборки, который выполняется с использованием исходного кода, в данный момент находящегося в локальном репозитории разработчика)
- не «заливать» неработающий код
- чинить сломанный build немедленно
- следить за тем, чтобы все тесты и проверки проходили
- не выкачивать из репозитория сломанный код

Continuous Integration



Continuous Integration

Базовый процесс интеграции выглядит следующим образом:

- Триггер. Событие, при котором запускается сборка продукта. Таким событием может быть: изменения в коде (push), определенное время, нажатие на кнопку.
- После срабатывания триггера стартует сборка проекта из исходников.
- Развертывание базы данных.
- Развертывание приложения.
- Тесты. Авто-тесты не являются обязательными, но их выполнение крайне желательно. Это один из важных пунктов хороших практик CI.
- Статус, отчеты, уведомления по результатам сборки. После прогона тестов получаем результат сборки, детальные отчеты по каждому из этапов интеграции

Build script

Скрипт сборки — это набор команд, которые будут выполнены при запуске процесса интеграции. Чаще всего он выглядит как следующий набор шагов:

- Очистка от результатов предыдущего запуска
- Компиляция (или статический анализ кода для интерпретируемых языков)
- Интеграция с базой данных
- Запуск автоматических тестов
- Запуск других проверок (соответствие код стандартам, проверка цикломатической сложности и т. д.)
- Разворачивание программного обеспечения

Автоматические тесты в CI

В CI используются тесты всех уровней за исключением исследовательских:

- модульные (unit) тесты
- компонентные тесты
- функциональные тесты
- системные тесты

По написанию и запуску тестов также принят ряд соглашений:

- более быстрые тесты должны запускаться первыми
- тесты должны быть разделены по категориям
- на все баги должны писаться тесты
- тест кейсы стоит ограничивать одной проверкой
- тесты должны выполняться без ошибок при повторном запуске

Преимущества CI

- Прежде всего, это регулярная интеграция всего приложения.
- Все делается автоматически, люди избавлены от рутины.
- Экономия времени.
- Работа над проектом прозрачна для всех участников команды. Становится проще ответить на вопросы что? где? когда?
- Уменьшаются риски получить «гранату». Дефекты находятся раньше. Это достигается путем запуска тестов и ранней отдачи нового/измененного функционала на тестирование.
- У нас есть всегда развернутое окружение для тестирования и демонстрации работы заказчику и прочим заинтересованным. Если ваша команда большая, и вы работаете одновременно в разных ветках репозитория. То теперь буквально в несколько движений вы можете настроить ветку кода на нужное окружение или собрать новое с нужной веткой.
- Можно безболезненно эмитировать процесс деплоя на тестовых серверах.
- Хорошая CI система позволяет поддерживать ряд инженерных практик (анализ кода, покрытие кода, юнит-тесты).

Узкие места CI

В любом инструменте существует ряд нюансов. Чтобы эффект от использования непрерывной интеграции был наибольшим рекомендуется использовать ряд общепринятых практик.

- Частая синхронизация. В этом заключается основной смысл CI. Нельзя позволять разработчикам длительное время не интегрировать изменения. Хорошей практикой является создание отдельных веток кода и окружений для длительных фич и рефакторинга.
- Решать все проблемы со сборкой максимально быстро. Это позволит избежать простоев. Настройте автоматические нотификации в случае упавшего билда. Это позволит разработчикам своевременно знать о проблемах. Не экономьте на железе. Сервер должен быть мощный.
- Должным образом настройте инфраструктуру. Процесс сборки должен быть надежным.

Узкие места CI(продолжение)

- Процесс сборки должен быть быстрым, не более 10 минут. Поэтому имеет смысл оптимизировать все шаги сборки. Во время сборки выполняйте только необходимые действия, ничего лишнего. Приемочные тесты не должны занимать много времени, т.к. любые простои не желательны. Пишите и прикручивайте к сборке авто-тесты. Тогда интеграция будет наиболее безопасной.
- CI будет эффективен только тогда, когда вся команды его принимает.
- Расширяйте возможности (анализ покрытия, статические анализаторы, статистика тестов).
- Максимально адаптируйте окружение под production-версию.

GitHub Actions
as **CI/CD** platform

Continuous Integration/Continuous Deployment

Термины

- Workflow
- Job
- Step
- Action
- Event

Continuous Integration/Continuous Deployment

Workflow – это высокоуровневый набор правил для того, чтобы делать какие-то действия при определенных условиях.

Job – это сущность, которая запускает какую-то задачу в рамках вашего workflow.

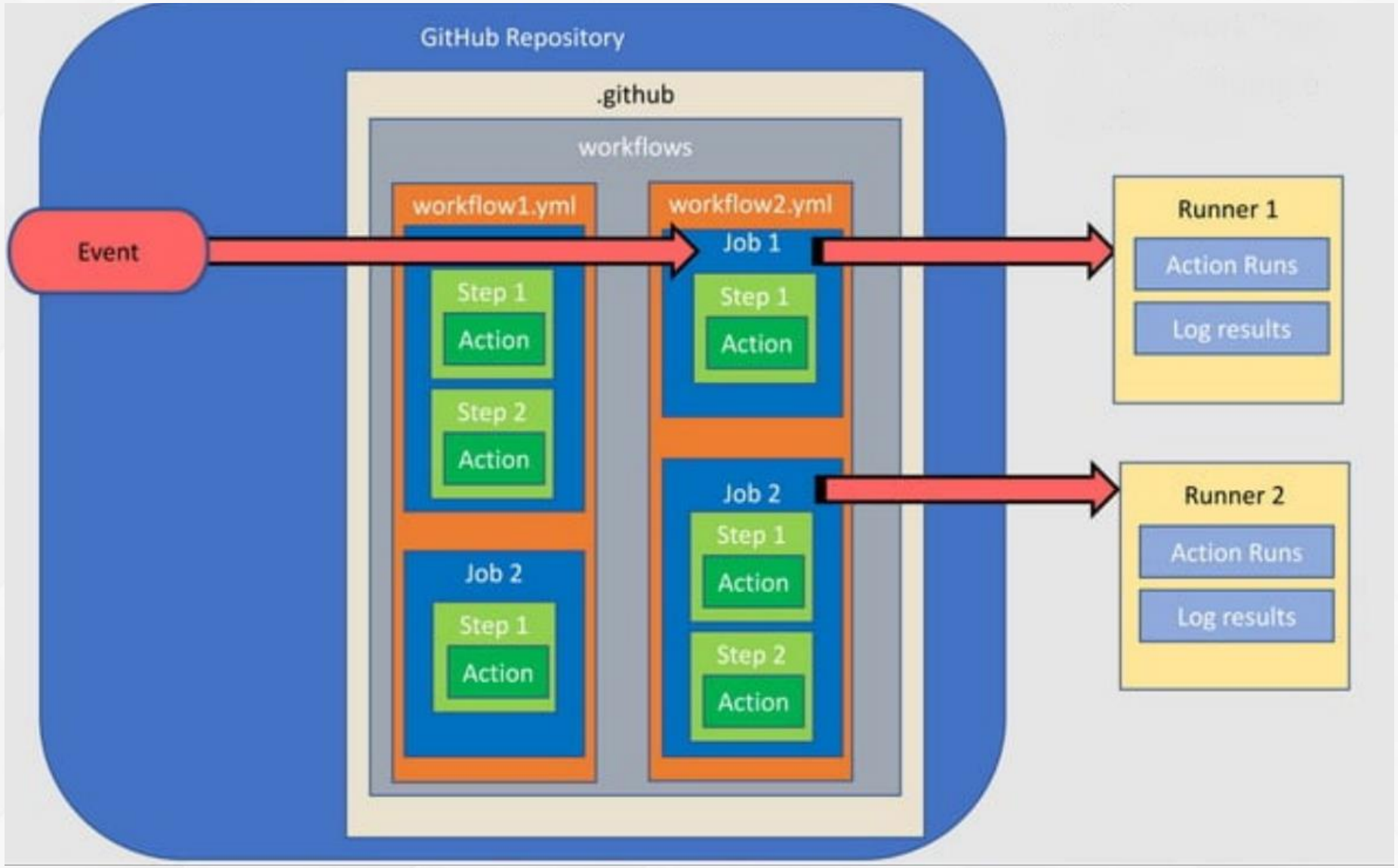
Step – это шаг. Т. е. один job может состоять из нескольких шагов.

Action – это тот переиспользуемый unit, который попал в название GitHub Actions, т. е. это самая минимальная часть, которую можно переиспользовать в нашем CI.

Event – это термин, который обозначает, что случилось какое-то действие. А если действие случилось, то мы запускаем какой-то конкретный workflow.

Continuous Integration/Continuous Deployment

Как это работает



Спасибо
за внимание!

Ваши вопросы...