В этом уроке мы с вами разберемся с компиляцией и build-тегами.

Для начала давайте поймем, что такое компиляция. Компиляция – это процесс, при котором написанный нами код, а как правило, мы его пишем в текстовом виде, он компилируется в набор машинных кодов и инструкций, которые понятны нашей машине. То есть код, который у нас является неким текстом, переходит в набор команд, которые понятны нашему процессору. Получается, что, если мы находимся на системах с различными процессорами, то итоговый вариант может быть разный. Потому что для каждого процессора есть свои нюансы, при написании команд.

И как вы видите на слайде пример приведен для языка (НРЗ 00:53). Когда у нас есть сначала фрагмент кода hello_world, мы запускаем процесс компиляции. Компиляцией занимается инструмент, который называется – компилятор. И после того, как код скомпилировался, на выходе мы получаем что-то, что понятно нашему процессору.

Что же такое кросс-компиляция? Кросс-компиляция — это, на самом деле, тоже процесс компиляции. Только отличие состоит в том, что мы, находясь в одной системе, компилируем код для другой системы. То есть, мы, находясь в рамках одной архитектуры, в рамках одного какого-то процессора, сможем компилировать код и сказать, что этот код будет использован на машине с другим процессором, с другой архитектурой. И тогда наш компилятор поймет, что нужно компилировать наш код под определенный набор инструкций, который отличается от инструкций на той машине, на которой мы запускаем компиляцию.

И так как Go – это компилируемый язык, то все вышесказанное справедливо и для него тоже. В Go мы передаем наши исходные файлы в компилятор Go, и на выходе получаем бинарник, с помощью которого можем запускать свой сервис.

На практике мы посмотрим, какие возможности есть для компиляции, и для кросс-компиляции. Компилировать мы с вами будем код, в котором происходит вызов функции последовательности fibonacci. И вывода результата на экран. Для того, чтобы произвести компиляцию, нужно вызвать следующую команду – go build. И на самом деле, по default-у этого достаточно. Вы можете видеть, что появился файлик slurm. Если теперь его запустить, то произойдет запуск нашей программы, которая написана в main-е. Здесь мы посчитали в последовательности fibonacci для числа 10, и вывели последнее число в последовательности.

А теперь давайте займемся кросс-компиляцией. На данный момент я нахожусь в системе macOS. Но я хочу, например, скомпилировать свою программу для Windows. В Makefile-е я подготовил уже заранее команды. Для этого мне необходимо указать environment переменную GOOS и GOARCH. Для GOOS мы указываем значение Windows, а GOARCH – 386, что будет означать 32-х битную архитектуру.

Давайте вызовем эту команду. И посмотрим, что у нас появился файл exe, который будет запускаться в среде Windows. И то же самое мы можем сделать для компиляции нашего кода в 64-х битную архитектуру под Windows. На выходе мы также получаем файл с расширением exe, который тоже будет запускаться в среде Windows. Результатом этого запуска будет абсолютно то же самое, что и предыдущего нашего файла. Выполнится все, что у нас написано в нашей main функции.

Данная возможность очень помогает при доставке нашего кода на production. Когда мы в процессе build-а нашего кода можем, например, в pipeline-е запустить компиляцию под необходимую нам архитектуру, которая будет использоваться на production-е.

Помимо выбора операционной системы и архитектуры, во время компиляции, можно воспользоваться build-тегами. Для чего они нужны? Например, мы хотим собирать нашу программу в разных режимах. Допустим, у нас есть режим dev, в котором мы хотим засетить какие-то свои environment переменные. Например, если у нас в режиме dev стартует наше приложение, то мы здесь устанавливаем environment переменную APP_ENV, значение dev. Ну и просто выводим на экран сообщение о том, что приложение запущено в dev режиме. Для этого мы можем использовать build-теги. Что это значит?

Вот здесь вы видите такой интересный комментарий у меня вначале файла. Он говорит, что при build-е, если встречается тег dev, то этот файл войдет в компиляцию. Если тега dev не будет, то данный файл будет игнорироваться. То есть, если мы сделаем компиляцию нашего проекта таким образом, а потом его запустим, все отработает, как и раньше. Но, если мы сделаем go build? и передадим туда tags dev, то вот этот файл уже попадет в нашу компиляцию. И мы автоматически присвоим вот этой переменной значение dev. Давайте запустим, скомпилируем и запустим. Как вы видите, произошел вывод на экран, который мы и описывали в нашем файлике dev.go.

Либо, например, у нас есть еще один файл с другим тегом debug. В этом файле происходит практически то же самое. Только устанавливается environment переменная APP_DEBUG. И выводится на экран сообщение о том, что наше приложение работает в debug режиме. Для этого нам уже нужен другой тег debug, и запустим нашу программу. И здесь мы снова видим, что на экран вывелось то, что мы и ожидали.

Ну и, на самом деле, мы можем воспользоваться двумя этими тегами, указав это при компиляции. Нам нужно tags. И у нас dev, и debug. Теперь, когда мы вызовем

нашу программу, запустим нашу программу, мы в нее включим и файлик debug, и файлик dev.