Итак, переменное окружение. Они же environment variables, они же envs. Вы уже наверняка с ними сталкивались. Давайте кратко вспомним, зачем нужны переменные окружения. Соответственно, мы можем естановить специфичные установки для приложения, какие-то пароли, флаги, значит, выполнение какие-нибудь. Например, списки исключений для Firewall-а. Или список, значит, пользователя для какого-то авторизационного сервиса. Ну то есть что-то можем, какие-то данные туда передать. Управлять этим самым приложением извне, потому что приложение разработчики пишут так, чтобы вы могли их использовать в разных условиях по-разному, слегка настраивая. Например, с разными базами данных, либо в приложении какая-то уже включена гибкость. Вы можете, я там не знаю, в приложении авторизовать пользователей, просто выдавая им ключи, а можете в приложении авторизовать пользователей, например, покидывая их на GitHub. И это все становиться специальной переменной окружения.

И они, естественно, принадлежат пользователям. То есть в рамках системы Linux env-ы у разных пользователей будут разные. То есть, когда вы запускаете приложение, оно окажется уже в какой-то среде environment-ов.

Давайте посмотрим, как эти переменные окружения, собственно говоря, реализуются на практике. Здесь у нас все просто. Я сейчас нахожусь в домене Window. И собственно говоря, переменная os, env работают в Window приблизительно также, как в Linux. То есть авторы библиотеки сделали все возможное, чтобы мы могли работать с переменными окружения одинаково в обоих системах. Соответственно, переменную окружения из приложения можно устанавливать. Но чаще всего вы будете их читать. Переменная getenv вернет вам значение, лежащее внутри, соответственно, переменной окружения. Либо вернет вам default-ное значение для данной переменной.

То есть в данном случае для, значит, строки, вернет пустую строку. Если вам нужно, если вам важно, тут даже написано, что, если вам важно различать между переменной, которая, значит, пустая, и переменная, которая вообще не была установлена, вам нужно LookupEnv. Потому что LookupEnv у нас отвечает за, значит, обработку того, существовала ли вообще переменная или нет. Для этого в LookupEnv возвращается вторая переменная bool. Она будет, соответственно, либо true, либо false. В зависимости от того, существовала переменная или нет. То есть, если вы хотите поставить, если вам важно отличать, что пустая строчка там была именно установлена, а не просто там есть, потому что вы так обработали переменную, то, соответственно, используйте его (HP3 02:52).

Ну и второе, что связано с переменными, это, соответственно, их можно перебирать. То есть вот os environ, соответственно, будет вам перебирать все переменные, то есть здесь у нас стандартная Go-шная тема, индексы и, соответственно, элементы. Если мы их просто выведем, там будут, значит, у нас строки. И они в Linux-е и Window-е будут выглядеть абсолютно одинаково. В смысле, я имею в виду переменные сами будут разные, но это одно и тоже, у нас есть имя переменной и, соответственно, знак равенства. Удобно, если вы не знаете, какие у вас переменные. Либо, если вам нужно в системе перебрать переменные и найти какие-то вам, соответственно, нужные. В Linux-е можно увидеть, как это выглядит. В принципе, ну практически также. Не практически, а также, то есть тоже имя переменной и, соответственно, ее значение.

Мы же обсудили, что, значит, существуют всякие различные переменные. Мы с ними можем работать. Это, в принципе, круто. Опять же, этот os.Setenv очень помогает, если запускать какие-то DevOps-овские тузы в своем окружении или в pipeline-ах CI/CD, потому что ну вы можете поставить какие-то переменные окружения, сразу что-то запустить, что-то проверить, да, (HP3 04:07). На go написано, все сама может сделать. Это круто.

Но в реальных приложениях, как правило, переменная окружения используется для именно конфигурации самого приложения. И удобно, когда у вас переменная окружения загружается внутрь какой-то структуры, и вы уже с ними работаете, как с полями этой структуры. То есть допустим, host, port, не знаю, протокол, там список пользователей. Да, что угодно. И поэтому существует куча библиотек, которые занимаются именно этим. То есть занимаются remapping-ом переменных окружения в, соответственно, структуры.

Одну из их я рассмотрю здесь. Они одинаковые практически все. Есть библиотека, значит, от наших китайских товарищей. Есть библиотека, соответственно, от (НРЗ 04:59). В общем, берите какую хотите, они все делают плюс-минус одно и тоже. Что они делают, они берут структуру какую-то, и в нее вгружают переменную окружения. То есть вот здесь в данном примере мы ставим переменную окружения, создаем, значит, config структуры, с помощью этой самой библиотечки. Мы загружаем в config, значит, наши переменные окружения, и просто можем с ними работать.

Если, опять же, по умолчанию, наши переменные окружения имеют префикс, потому что переменных окружения в системе много. Если мы просто назовет host и port, есть высокая вероятность, что переменные окружения будут перезаписаны или уже такие будут существовать. И вы то-то затрете важное в системе. Или ваше приложение, наоборот, получит какую-то фигню, вместо нормального ввода. Поэтому у каждой переменной окружения есть префикс, у этой данной библиотечки по умолчанию это префикс APP. Он настраивается, то есть здесь, например, префикс API. И вот, когда вы создаете переменную окружения, вы создаете loader, в него вы передаете API, как префикс. И при загрузке уже будет здесь в полях определяться API ES_HOST, API ES_PORT. И так далее. И соответственно, будет вот это все выводиться.

Можно посмотреть, как работает вот этот вот самый первый пример, который я вам выбрал себе в качестве учебного. Вот мы видим, что здесь ввелось 127.0.0.1. И ноль, потому что… Простите, вот. Вывелось 127.0.0.1. И вывелся ноль, потому что, соответственно, у нас не хватает PORT-а здесь. Я добавляю PORT в текст в строковом виде, выводиться PORT. Собственно говоря, здесь больше разбирать особо нечего. Так работают с переменными окружения. В промышленной разработке существуют какие-то, структуры и существуют переменные окружения. Существуют библиотеки в окружающей переменной окружения в эти структуры.