Итак, формат, который уже, в принципе, довольно давно с нами. Уже завоевал свою популярность в определенной нише - о ней чуть попозже - YAML. Yet Another Markup Language.

Соответственно, он тоже текстовый, также как и JSON, также как и XML. Он является format based. Это значит, что данные у вас представляются, путём форматирования текста - питонщики поймут, о чём я. Если вы Python никогда не видели, то там как бы вложение блоков определяется количеством пробелов в этом блоке. То есть, если у вас вот на одной линии там стоят какие-то конструкции, да, потом у вас цикл for, то в цикле for у вас не фигурные скобочки ставятся, как в других языках программирования, а просто делается отступ.

То же самое в YAML. Если у вас есть какой-то объект, туда вложены какие-то объекты, то они будут иметь отступ. И это нативная фича YAML. Вследствие чего, он легко читается людьми. Не надо его засовывать ни в какие парсеры, не надо расставлять отступы. Это часть формата. Вы его просто открываете в любом текстовом редакторе Cad, Wim, вообще, что угодно, читайте и всё понимаете.

Соответственно, меньше используется для передачи данных по сети по двум причинам. Во-первых, он не встроен в браузеры, то есть JavaScript его не поддерживает. А во-вторых, YAML часто используют для написания каких-то комментариев. И, ну, как-то так получилось, да, что вот JSON используют, как вот именно хардкорный формат передачи связки frontend-а с back-end-ом, обмена данными между сервисами, чтобы далеко не ходить, да.

А YAML, соответственно, ну, вот как-то так прижился в качестве инструмента в именно конфигурации. Вот сейчас очень много конфигураций именно пишется в YAML, потому что удобно. Вы его открывайте, вы понимаете, уже сразу видите структуру конфигурации.

Есть такая штука в YAML, как анкоры, то есть, ссылки на другие части этого файла, по ним удобно следовать. Соответственно, есть комментарии, всё очень удобно, всё очень круто. Идеально вообще сейчас, на данный момент, существующий формат для того, чтобы положить какую-то конфигурацию файла и её считать внутри своего языка.

Ну, и, соответственно, вследствие того же самого, используется часто в инструментах именно описания инфраструктурных задач, инфраструктуры для того, чтобы писать архитектуру вашей системы, что, в принципе, тоже является конфигурацией. Да, то есть, вот этот ультимативный формат именно для конфигурации.

Давайте посмотрим, как, соответственно, данные маршалятся в YAML, как они отмаршиваются, отмаршливаются, да, хорошее слово, анмаршлятся из YAMLa. И немножко посмотрим на сам формат.

Итак, формат YAML. Где его можно встретить?

Ну, в принципе, в инструментах DevOps-а. В нашем любимом Ansible есть YAML, соответственно, все действия описываются YAML-ом. Я вам я покажу, как пример (НРЗ 03:04). То есть конфигурация, конфигурация значения для to the home, которая нужна для сборки, соответственно, пакетов и управления (НРЗ 03:12) в Kubernetes.

Собственно, выглядит YAML вот так. У нас здесь три опциональные, три, значит, дефиса, есть какое-то значение, через двоеточие у него, соответственно, есть какое-то, вернее, поле через двоеточие у него, значение. И вот мы видим, что Autoscaling, у него нет никаких закрывающих тегов. Но мы понимаем интуитивно, что Autoscaling в себя включает вот это вот всё, да, то есть, отступы нам позволяют просто понять, что значат эти поля включены в Autoscaling.

Отступы — это проблема в YAML, потому что, если вы потеряли хотя бы одно, один пробел, парсер уже не понимает, что это за отступы такие. Вот. Но для того, для валидного YAML, который уже кем-то был написан, читать, как видите, весьма удобно.

Тире, соответственно, обозначают массивы. Вот такие вот, значит, фигурные скобочки тоже могут обозначать объекты, как в JSON. И вообще, в принципе, YAML считается таким как-бы подтипом JSON. Естественно, JSON-ским интерпретатором он разобран быть не может, по причине отсутствия запятых. Но в целом, да, вы можете это воспринимать, как такой некий подтип JSON.

Давайте посмотрим, как с ним работать.

Давайте, соответственно, посмотрим на простой, простейший пример YAML. Вот он, то есть, две переменные, обе типа, целочисленного типа. Строки в YAML можно делать вот так. То есть, нет никакой проблемы в том, что вы их не экранируете, кавычки, как раз по причине того, что в YAML всё делается за счёт переноса. Если вам нужно, соответственно, строку перенести, то делается вот такой вот (НРЗ 04:55), и, соответственно, здесь уже пишется, какая-то строка, и будет считаться до следующего, значит, отдельного enter, что это строка относится к YAML.

Но для того, чтобы включать какие-то структуры данных, содержащие в себе ключевые слова, вы можете также эту строку экранировать. Как я уже говорил, да, это будет более расширенная, так сказать, версия JSON в этом случае.

Соответственно, этот пример должен у нас мапиться, уже можно догадаться по примеру предыдущих XML и JSON, маршеров, анмаршеров, вернее, что он ужимается в структуру, содержащую в себе hits и какой-то time. Давайте time сделаем каким-нибудь там timestamp-ом реальным, чтобы было похоже.

Соответственно, размапливается этот тег следующим образом: определяется структура, определяются её поля. То есть, здесь у меня в данном случае определены поля типа int и определена… точно такие же теги.

Соответственно, переменная GetConf прочитает файлик. Файлик мы читаем с помощью ioutil.ReadFile в этот раз, не с помощью os.ReadFile. Но это, в принципе, можно понять, что это одно и то же, это просто обертка такая в библиотеке ioutil, на тот случай, если у вас там, библиотека у вас недоступна, или на тот случай, если вы хотите унифицировать всё под одной библиотекой.

Соответственно, после чтения YAML, после чтения файла, мы можем этот файл размапить в нашу переменную config.

Ну, здесь, то есть, можно увидеть, да, что здесь ну, ничего, ничего реально не отличается от, от других, других маперов. То есть, все вот эти вот текстовые форматы мапятся плюс-минус одинаково. У вас меняются только какие-то там атрибуты, да, у структуры и, соответственно, меняется здесь заголовок в библиотеке, которую вы используете.

Давайте посмотрим, как это работает, что у нас получается в конфиге. Можно посмотреть в отладчике, соответственно, в конфиге у нас хиты и время сохранились, да, в отладчике, то есть, мы видим, да, что YAML у нас представлен просто в виде конкретной строки.

Ну, в принципе, вот базовое чтение YAML, ничем не отличается от базового чтения JSON и базового чтения XML. Что у нас с созданием YAML, да, всё то же самое, если нам нужно создать YAML, мы точно также определяем структуру. И маршелим её в наш out. Тут я даже ничего пояснять дополнительно не буду, здесь всё, собственно говоря, понятно, да.

Теперь давайте посмотрим на некоторые расширения YAML, присущие уже самому YAML, и как с ними работает, соответственно, сам Go.

Итак, YAML в себя включают классную штуку, которая обусловила его интенсивное использование в различных системах конфигурации. Называется она анкоры, и встроена в спецификацию формата, соответственно, любой маршалер и анмаршалер, вернее любой анмаршалер, который работает с YAML должен её поддерживать просто по спецификации.

Смотрите, вот здесь у нас есть поле definitions. У него есть поле steps, и step определён именем, вот такой вот ссылкой. А это значит, что этот step можно засунуть куда-то ещё, и как бы он встроится непосредственно в место, куда его запихнули. То есть, в данном случае, мы step вот тут вот всё определили и, значит, пометили, как бьёт step, и в дальнейшем в определении Pipeline-ов мы можем использовать ссылку на этот step. И маршалером это должно развернуться, непосредственно, вот в это значение.

Давайте посмотрим, как это работает на примере, на примере моего метода, называемого ReadBigConf.

Соответственно, здесь я создал структуру BuildConf, которая всё в себе содержит. Ну, ленивое, да, ленивое чтение, которое в себе содержит мапы с интерфейсом в Definitions и в Pipelines. Соответственно, здесь всё то же самое. Я анмаршелю. Давайте посмотрим на вывод этой самой, этой самой функции. И что мы, соответственно, здесь видим? Вот у нас есть definitions, у них есть, соответственно, значение step, в нём есть значение там name, script, mvn package. Ну, то же самое, что и здесь, да. И посмотрим в pipelines. У нас есть branchи, есть develop. Давайте заглянем в его step. Branches. Здесь develop. И давайте заглянем в его, соответственно, step. А в step что у нас лежит? А в step у нас лежит Build and

test, script и артефакты. У нас вот получилось вот, вот это. Ну, вернее, на самом деле не так, у нас получилось вот так. То есть, как можно увидеть, это произошло автоматически с помощью маршалера, и нам уже не нужно над этим думать. Вот такая вот прикольная фича есть именно у YAML и у его, соответственно, анмаршалера.

Возвращаясь, соответственно, к нашему примеру с тегами и с его запятыми. В YAML, естественно, тоже можно определить кастомный анмаршалер.

Определяется он следующим образом. Соответственно, здесь мы видим опять нашу структуру. Виден Tags и здесь определенная структура SlicesTags, в которой определенный метод анмаршал YAML, в который передается не массив байт, по сравнению с, например, нашим JSON, да, а передаётся уже YAML Node. То есть, YAML сам всё разбивает на Node-ы. И для каких-то Node, которых существует вот этот метод, он передаёт туда Node-у и ждёт, что метод вернёт какой-то уже тип.

Соответственно, у Node-а есть много чего. У Node-а есть тип, того, что из себя эта Node-а представляет. У Node-ы есть там Style. Style нам показываем, к какому типу отформатирована Node-а.

Но что нас интересует — это строковое значение этой самой Node-ы. Ну, в данном случае, да, это такое, опять же, ленивое программирование. Здесь просто присваиваем Split по этому строковому значению. И таким образом, здесь наши теги превратятся в опять же в массив.

Если мы запустим наш, нашу приложеньку, то в config-е у нас в Messages будут теги, которые являются slice-ом.

Ну, то есть, так можно делать и с JSON, и с XML, и, соответственно, с YAML-ом.