Всем привет. С вами снова я – Тигран. И мы продолжаем с вами говорить об инструментах языка Go. На этом уроке мы рассмотрим такую немаловажную тему, как «Проверка качества кода».

Конечно же, в первую очередь, на ум приходят линтеры различные, но помимо них мы ещё посмотрим, как можем оптимизировать наш код, как мы можем его профилировать и взглянем немножко на обнаружение Race condition-ов в языке Go.

Что же такое «качество кода»? Как нам понять – наш код является качественным или нет? На самом деле, в Go есть набор правил и соглашений, которым принято придерживаться. И о качестве кода можно говорить, например, путем проверки нашего кода на соответствие этим правилам. И есть такое понятие, как «линтер». То есть линтер – это утилита, которая как раз позволяет нам выполнять те или иные проверки. Линтеры могут проверить наш код на простые ошибки, какие-то неточности оформления и как раз-таки расхождения с нашими соглашениями.

Линтеров, на самом деле, достаточно много, но мы рассмотрим некоторые из них. Например, gofmt – это один из встроенных в стандартный пакет Go линтеров, который может осуществить проверки оформления нашего кода. И после проверки на соответствия стилю он также может это исправлять.

Например, ещё go vet – это также часть стандартного пакета, и эта утилита производит проверки каких-то типичных ошибок, например, мы осуществили какое-то присваивание и не воспользовались этой переменной. Также там

присутствуют ошибки при работе с многопоточностью, синхронизациями, build тегами и так далее.

В свою очередь staticcheck, например, — это более апгрейженный вариант go vet-a. Вдобавок ко всем проверкам из go vet-a, статик ещё проверяет на лишний код ошибки с различной многопоточностью, использование каких-то нестандартным конструкций. Всё это проверяется в данном линтере и сообщается программисту, что есть несоответствия соглашения.

Следующий достаточно известный линтер – это golint. Он разработан командой Go и, на самом деле, производит множество проверок на основе таких документов, как Code Review Comments. Этот документ опубликован в сети Интернет, и в дополнительных материалах к уроку будет представлен ссылка. Кому интересно, могут пройти по ссылке и почитать. Единственное – он на английском языке.

Линтер errcheck проверяет нашу работу с ошибками на соответствия определенным требованиям. Эти требования описаны в документе Effective Go. Ссылка на этот документ тоже есть в дополнительных материалах к уроку. Например, линтер может проверить, что мы игнорируем возвращаемые функции ошибки и их не проверяем, что не очень хорошо делать.

И, например, есть линтер goconst, который может увидеть, что вы в коде очень много используете литералов и чисел, и предложит вам вынести эти данные, например, в константу.

Следующим довольно интересным линтером является линтер fieldalignment. Он помогает нам находить структуры, которые мы могли бы оптимизировать по потреблению памяти. Оптимизация эта происходит с помощью сортировки полей в данной структуре. Из-за особенностей выделения памяти в языке Go как вы можете видеть на слайде – в первом варианте структура с полями abc будет занимать 32 бита. Происходит это так, потому что необходимо заполнять пустые пространства между полями дополнительными битами. Но если сортировать по-другому и выстроить эту структуру с полями acb, то пустых пространств становится меньше, поэтому во 2-м варианте структура будет занимать 24 бита. Тем самым, довольно простой операцией сортировки полей мы значительно сэкономили по потреблению памяти. И как раз-таки линтер fieldalignment подсвечивает нам эту проблему и предлагает её решение.

Все эти линтеры безусловно можно использовать для проверки качества нашего кода. И при нахождении некоторых изъянов и проблем мы можем код улучшать и оптимизировать.

Само понятие «оптимизации кода» скрывает под собой процесс нахождения некоторых блоков и участков, которые как раз-таки влияют на нашу производительность. Но и само собой, нам необходимо изменить эти участки кода с целью увеличить скорость работы или, например, уменьшить потребление ресурсов.

Но было бы достаточно неудобно разработчику каждый раз вызывать руками этот весь набор линтеров, который мы рассмотрели. На самом деле, их достаточно много, мы рассмотрели всего несколько из набора. То есть, допустим, если у нас

есть десяток таких линтеров – нам пришлось бы руками 10 раз вызывать разные линтеры и проверять, что у нас не так происходит в коде. Для того, чтобы этот процесс сделать более удобным, в языке Go есть пакеты, которые уже за нас собирают наборы различных линтеров и запускают их над нашим кодом.

И одним из таких пакетов является golangci-lint. Давайте посмотрим на GitHub-е на страничку этого линтера. Таким образом она выглядит и здесь в документации в readme-файле можно видеть, что этот пакет является runner-ом linter-ов. То есть это не сам по себе линтер, а некий инструмент, который агрегирует в себе множество линтеров и запускает их за вас, чтобы вы не тратили время и не запускали по одиночке каждый линтер.

Давайте перейдем в документацию и посмотрим на список линтеров. Как вы видите, в данном пакете запускается множество различных линтеров, проверяющих наш код по тем или иным спецификациям. Для того, чтобы запускать эти линтеры вручную, нам бы потребовалось очень много времени. Да и это совсем уже неудобно. Как пользоваться этим пакетом – как раз мы рассмотрим на практике.

Итак, для начала давайте посмотрим, как мы можем установить данную утилитку. Перейдем в раздел Install в документации и здесь мы видим, что в первой части объясняется, как установить golangci-lint для CI, но нам нужно для локального использования. Мы будем использовать у себя на компьютере.

И мы здесь видим, что оказывается, что golangci-lint поставляется как отдельных бинарный файл. То есть это отдельная утилита, которая в бинарном виде лежит у

нас на компьютере, и мы её вызываем. И, конечно же, она скачивается посредством пакетного менеджера, например, для macOS-а как может быть brew. Для других систем оно может устанавливаться по-разному. И на самом деле, добрые люди уже написали небольшой shell script таким образом, который устанавливает нам всё, что нужно, проверяя наши платформы и так далее. Поэтому для установки golangci-linter-а я чаще всего использую как раз-таки данный shell script. Ссылку на этот script я также оставлю в дополнительных материалах.

Давайте перейдем обратно в документацию и посмотрим на раздел Quick Start. Здесь мы видим, что golangci-lint устанавливается в систему как команда, и у него есть различные параметры. Само собой, run будет запускать наши линтеры. Мы можем здесь также указывать такие директории. С файлами нам нужно проверить либо набор каких-то директорий. По стандарту есть раздел help, где мы получим описание того, что нас интересует.

Но самое интересное здесь – в настройке. Смотрите, мы можем запустить линтер и сказать ему, что: «Выключи все линтеры, которые у тебя есть». То есть в данном случае мы выключим весь набор линтеров и оставим только, например, errXcheck. И наш golangci-lint запустится и проверит наш код только посредством этого линтера errcheck. Но как мы видели в разделе списка линтеров – их у нас очень много и управлять ими таким способом не очень удобно. Каждый раз писать это в строке будет слишком усложнять читаемость наших команд. Поэтому golangci-linter может конфигурироваться.

Для этого у него есть возможность объявлять какие-то параметры в yaml-файлике. Мы можем создать такой. golangci.yml-файл или, допустим, json и toml, но мне удобнее всего yml, я использую именно его. И в yaml файле, на самом деле, достаточно много различных настроек, которые мы можем производить. Кому интересно, ссылку на документацию я оставлю в дополнительных материалах, и вы можете посмотреть подробнее по каждому параметру.

Но мы остановимся на небольшом участке, который будем смотреть на практике. Как вы видите, здесь можно параллельность настраивать, различные timeout-ы. Помимо всего этого есть формат вывода, есть раздел настроек по каждому линтеру отдельно, то есть каждый отдельный линтер мы можем как-то custom-изировать под себя.

Но, кроме этого, всего нас интересует такой раздел как Linters, где мы можем управлять набором наших линтеров. Как вы видите, здесь приведено достаточно много настроек по каждому линтеру, который мы можем делать, но, в конце концов, мы доходим до раздела Linters. Как вы видите, вот он здесь. В этом разделе мы можем также отключить все и включить только тот набор, который нам нужен. Либо также доступно включение всех и отключение каких-то отдельных линтеров.

Давайте посмотрим в коде теперь, как мы можем это применить к нашему коду? Итак, здесь я создал Makefile, потому что так будет намного удобнее. И в Makefile-е объявил несколько команд. Первая команда: install как раз-таки установит нам golangci-linter. Как мы это делаем? – Всё достаточно просто: мы просто curl-ом вызываем как раз-таки тот shell-файлик, о котором я говорил, и он

устанавливает у нас в нужную директорию наш бинарник с golangci-linter-ом. Давайте вызовем эту команду и посмотрим.

Вот как вы видите, идёт процесс установки. Мы определили платформу, определили версию, и прошло скачивание. В итоге мы получили репозиторий, куда скачали golangci-lint. Давайте посмотрим, что оно нам возвращает?

И мы видим, что golangci-lint показывает список наших линтеров, но как мы видим – он ещё говорит, что есть линтеры, которые отключены. Происходит это, потому что вы, наверное, заметили – у меня есть .golangci.yaml, в котором я немножко подконфигурировал свой линтер, но об этом я чуть-чуть позже расскажу.

Давайте вернемся к Makefile-у и посмотрим на следующую команду – это lint, которая как раз-таки выполняет golangci-lint run и запускает проверку всех файлов, которые находятся в нашей директории. Файлов у нас тут немного. Мы вернемся к файлу client, который рассматривали на одном из наших уроков, где мы объявляли структуру наших client-ов. Как вы видите, особо здесь ничего не поменялось. Единственное – могли заметить, что добавились комментарии.

Давайте теперь запустим наш линтер и посмотрим, что он скажем по поводу качества кода нашего файла client? Выполнились проверки и на самом деле, здесь есть 1 замечание. Как мы видим, формат такой, что отображается файл, где есть ошибка; отображается сообщение, что нужно поправить и название линтера, которое задетектило эту ошибку. Здесь мы видим, что golint обнаружил то, что мы обязательно должны оставить комментарий для нашей структуры, так как она объявлена как экспортированная. Здесь линтер заботится о тех людях, которые

пользуются вашим кодом и обязательно требуют, чтобы мы описали нашу структуру.

Давайте я сделаю таким образом, но даже если я опишу это так: следующий линтер godot начнёт ругаться, что в комментарии у меня не хватает точки. То есть у Go есть требование, что для комментариев, где мы описываем нашу структуру или метод – в конце необходимо ставить точку. В таком виде при запуске нашего линтера всё проходит замечательно – никаких ошибок нет.

Но на самом деле, нет ошибок, потому что в данном файлике, golangci.yaml я немножко поднастроил наш линтер. Как вы видите, в разделе Linters я сказал, что я хочу включить весь набор доступных линтеров, но при этом выборочно отключил некоторые линтеры.

Давайте посмотрим, к чему это приведет, если я, например, включу обратно gofmt? Как мы говорили – gofmt отвечает за стилизацию нашего кода. Я закомментирую эту строку, и это будет означать, что gofmt больше не будет в отключенном состоянии. И при следующем запуске golangci-lint-а также произойдёт проверка на основе gofmt.

Запускаем снова линтер, на самом деле и здесь gofmt начинает на нас ругаться, что нам необходимо оформить наш код по требованиям. Таким образом, это происходит – я воспользовался hotkey-ем для приведения нашего кода в формат gofmt. Снова запустим наш линтер и теперь всё хорошо. Можем пойти дальше и отключить ещё что-нибудь интересное.

Например, данный линтер всегда требует, чтобы у нас операция return производилась с новой строки. Давайте включим его и снова посмотрим, что нам скажет линтер. Как вы видите, он нам говорит ровно это, что не хватает пустой строки перед return. Добавляем пустую строку, запускаем снова, и всё проходит.

Ещё один интересный линтер – это линтер gosimple. Давайте попробуем его включить и запустим линтер с его проверкой. И смотрите, какой хороший совет он нам даёт. Что он говорит? – Что мы можем воспользоваться более компактной записью. Давайте перейдем в код и посмотрим, что да, действительно, у нас здесь if в таком виде, и функция возвращает bool. Поэтому мы, на самом деле, можем сделать следующее – вернуть результат логической операции и удалить лишние строчки. Таким образом, наш код будет более читаемый и займет меньшее количество строк.

Таким образом, линтеры позволяют нам всегда придерживаться каким-то соглашениям в коде и писать понятный, краткий и оптимизированный код. Не забывайте всегда проверять с помощью линтеров ваш код, держите настройку линтера в актуальном состоянии. И, на самом деле, могу показать на основе моих собственных проектов, что линтер всегда должен присутствовать ещё и в CI/CD вашего проекта, чтобы каждый раз при deploy-е вы сначала проверяли качество вашего кода, а потом уже доставляли его до production-а.