1

Давайте соединим наши сервисы которые мы делали в 6 и 7 части. Для этого нам необходимо будет вставить checker из 7 части в структуру из 6 части. Для этого:

- Добавим в наши классы для связи с БД новый класс для health probes, и новую таблицу в БД содержащую в себе - домен, дату проверки, результат проверки.
- Табличку с метриками расширим именем опрашиваемого домена.
- Расширим реализацию интерфейса measurable дополнительными полями, для хранения соответствующих репозиториев(репозитория для хранения healthcheck и metrics)
- Реализуем методы GetMetrics, взяв код для сохранения метрик Prometheus и перенеся его в этот метод.
- Добавим метод Save для вызова GetHealth и GetMetrics и сохранения их данных в репозитории.
- Из Checker будем вызывать метод Save для всех добавленных items.
- для экономии времени вы можете в текущем репозитории сделать две папки в директории cmd
- ./cmd/checker/main.go - сервис опрашивающий по расписанию указанные цели (метрик и доменов)
- /cmd/api/main.go - сервис, предоставляющий rest api доступа к истории метрик и опросов доменов
- Добавим еще один endpoint для вызова списка хелсчеков по имени домена.

2

Соберем докер контейнеры для обоих mainов приложения по уже известному нам шаблону. Для этого нам необходимо создать два dockerfile положив их в папку docker в разные подпапки в корне приложения. Также не забываем что докер берет пути от места где вы его запускаете, а также что билды запускаются с помощью команды docker build -f с указанием пути к файлу и точкой для определения места сборки.

3

Давайте задеплоим наше приложение, а также сервис gometr в kubernetes (тут про деплоймент и два приложения в кубер + база данных).

Наши приложения(http сервер, scheduler и gometr) должны быть также загружены в kubernetes как deployments и к каждому из них должен быть привязан Service для обеспечения сетевого взаимодействия. Рассмотрим Service ниже:

```
apiVersion: v1
kind: Service
metadata:
 name: checkoutservice #это имя сервиса для  сетевого взаимодействия в формате например http://checkoutservice/. Kubernetes сам подставит сетевой адрес когда его получит.
spec:
 type: ClusterIP #присваиваем сервису внутренний IP для внутрикластерного взаимодействия
 selector: # для чего будет этот сервис?
   app: myapp #ставим наш лейбл
 ports:
   - name: http
     port: 8080 #порт сервиса
     targetPort: 8080 #порт пода куда ссылается сервис
```

Таким образом наш деплоймент будет состоять из четырех деплойментов - checker http, checker daemon, gometr и database и трех сервисов, поскольку checker daemon не поддерживает обращения к себе.