Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Кафедра Информатики

ПРОЕКТ

по курсу «Обеспечение качества программного обеспечения»

ВЕКТОРНЫЙ КАЛЬКУЛЯТОР

Студент:
гр. 758641
Ярош Г.И.

Проверил:
Неборский С.Н.

Минск, 2018

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Цель

Разработать веб-приложение предоставляющее функциональность калькулятора оперирующего векторами действительных чисел. Спроектировать модель качества разработанного приложения.

1.2 Задачи

1. Реализовать веб-приложение с использованием языка программирования Go, которое должно предоставлять следующие функции, реализованные в виде RESTful Web API:
   a. Сложение двух векторов;
   b. Вычитание двух векторов;
   c. Скалярное произведение двух векторов.
2. Произвести контейнеризацию приложения и развернуть приложение с помощью сервиса Amazon Container Service.
3. Сконфигурировать функцию автомаштобирования приложения при высокой нагрузке.
4. Спроектировать модель качества содержащей четыре характеристики качества.

2 МОДЕЛЬ КАЧЕСТВА

При построении модели качества и выборе характеристик качества мною использовался стандарт ISO/IEC 9126. В модель качества мною были включены следующие характеристики:

- Функциональность.
- Надежность.
- Эффективность.
- Мобильность.

2.1 Функиональность

Функциональность описывает соответствие реализованных функций ПО требуемой пользователем функциональности.

Подхарактеристики:

- *Пригодность для применения*. В качестве метрики для этой характеристики можно рассматривать следующее отношение N_i/N, где N_i – количество реализованных операций над двумя векторами, а N – количество возможных операций над двумя векторами.
- *Корректность*. В качестве метрики корректности можно взять отношение $R_{correct}/R$, где $R_{correct}$ – количество ответов приложения с корректным результатом операции над двумя векторами, а R – количество всех ответов приложения.

2.2 Надежность

Характеристика надежности описывает способность приложения сохранять свой уровень функциональности за некоторый промежуток времени при некоторых установленных условиях.

Подхарактеристики:

- *Отказоустойчивость.* Метрикой отказоустойчивости можно принять значение следующего выражения : $1 - R_{500}/R$, где R_{5xx} – количество ответов приложения со статус кодом 500 (количество ошибок сервера), а R – количество всех ответов приложения.
- *Доступность.* В качестве уровня доступности можно принять $1/t_{max}$, где t_{max} – максимальное время ответа на запрос за некоторый промежуток времени.

2.3 Эффективность

Эффективность показывает соотношение уровня качества функциональности ПО и используемых при этом уровне ресурсах.

Подхарактеристики:

- *Временная эффективность.* Метрикой временной эффективности можно принять t_{avg} – среднее время ответа на запрос за некоторый промежуток времени.
- *Использование ресурсов.* В качестве метрики использования ресурсов можно принять следующие отношения: CPU_{avg}/R_{avg} и MEM_{avg}/R_{avg}, где CPU_{avg} – средний уровень использования CPU, MEM_{avg} – средний уровень использования оперативной памяти, R_{avg} - среднее число запросов в секунду за некоторый промежуток времени.

2.4 Мобильность

Мобильность приложения определяется через его способность быть перенесенным из одного окружения в другое.

Подхарактеристики:

- *Простота установки.* В качестве метрики простоты установки можно принять время необходимое на полную настройку окружения, необходимого для развертывания приложения.
- *Адаптируемость.* Метрикой адаптируемости можно считать время Δt, необходимое для создания новой копии приложения при высокой нагрузке.

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Создание приложения

Приложение разрабатывалось с использованием языка программирования Go. Для реализации обработки HTTP запросов был использован модуль net/http. Приложение содержит один единственный эндпоинт, который на вход принимает два вектора и операцию, которую необходимо выполнить. В ответе на запрос приложение возвращает результирующий вектор или вектор, содержащий одно значение – результат скалярного произведения (рис. 1, 2).

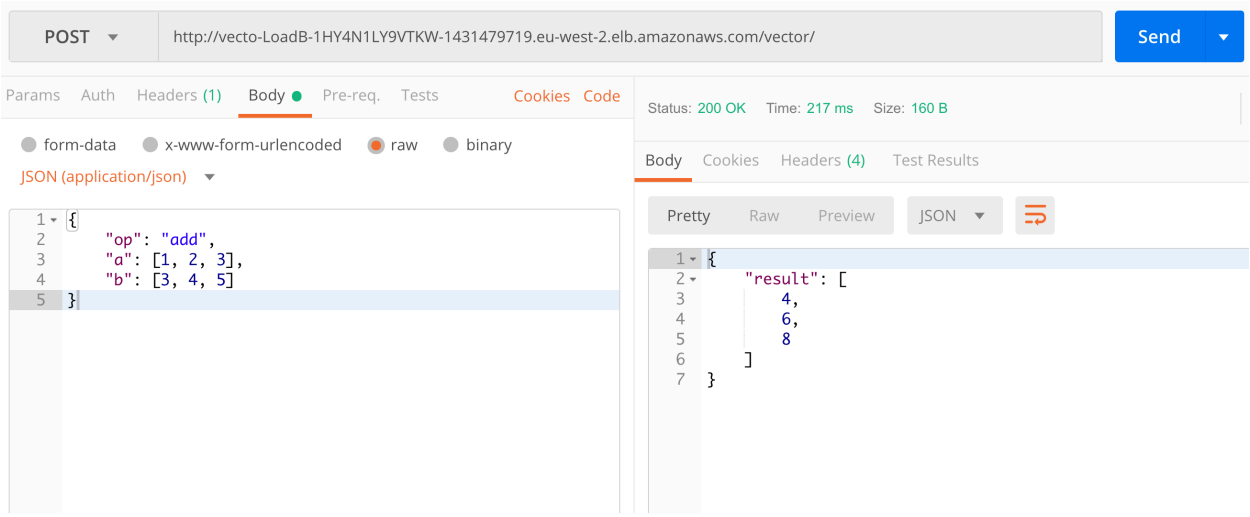

Рис. 1. Операция сложения двух векторов

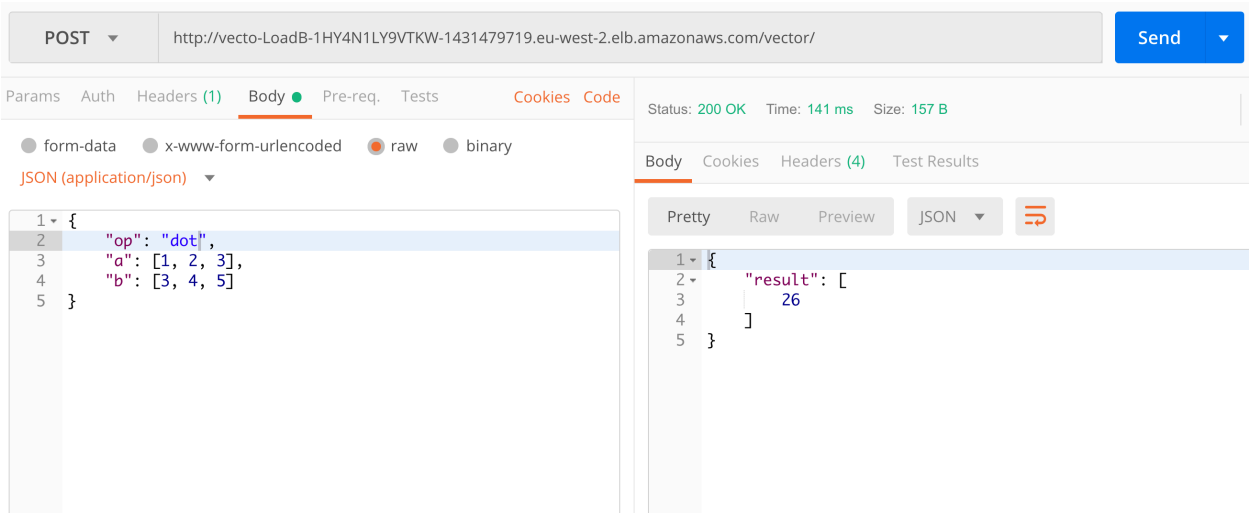

Рис. 2. Операция скалярного произведения двух векторов

Разработанное приложение было контейнеризированно с помощью технологии Docker. Для этого был написан Dockerfile, внутри которого описывались этапы сборки приложения. Сборка Docker-образов и последующая загрузка их в репозиторий реализована с помощью сервиса Travis CI (рис. 3).

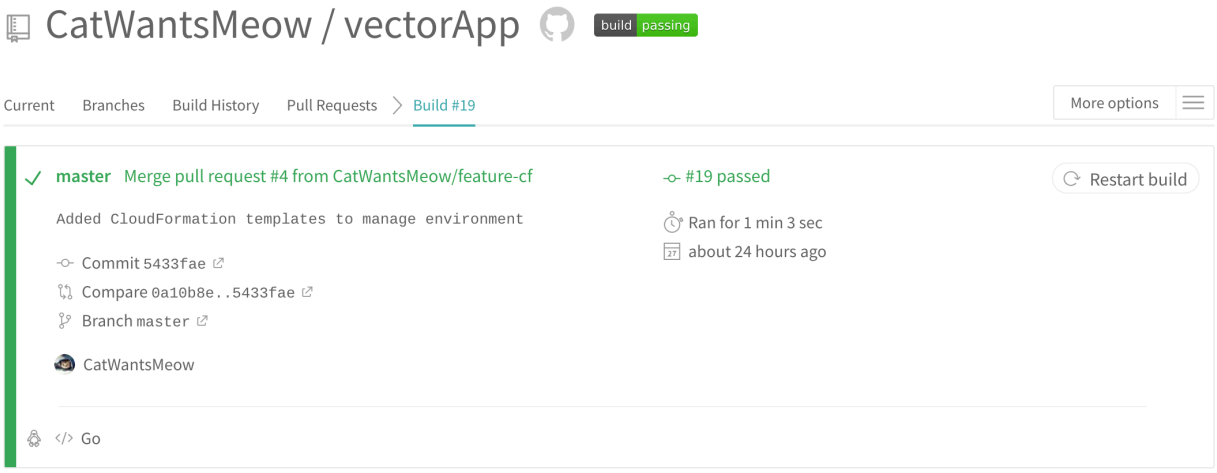

Рис. 3. Сборка Docker-образов на Travis CI

Для развертывания приложения использовался сервис Amazon Container Service (рис. 4, 5). В нем был создан TaskDefinition, описывающий параметры запуска контейнера с приложением, а также ECS кластер и ECS Service – сущности представляющие работающее приложение. Для балансировки нагрузки был создан Application Load Balancer, который был соответствующим образом сконфигурирован для работы с приложением.

В качестве способа запуска контейнеров был выбран FARGATE. Этот способ позволяет не запускать контейнеры на отдельных предварительно созданных инстансах EC2, а абстрагироваться от этого уровня и запускать каждый контейнер с выделенным количеством CPU и оперативной памяти (рис. 6).

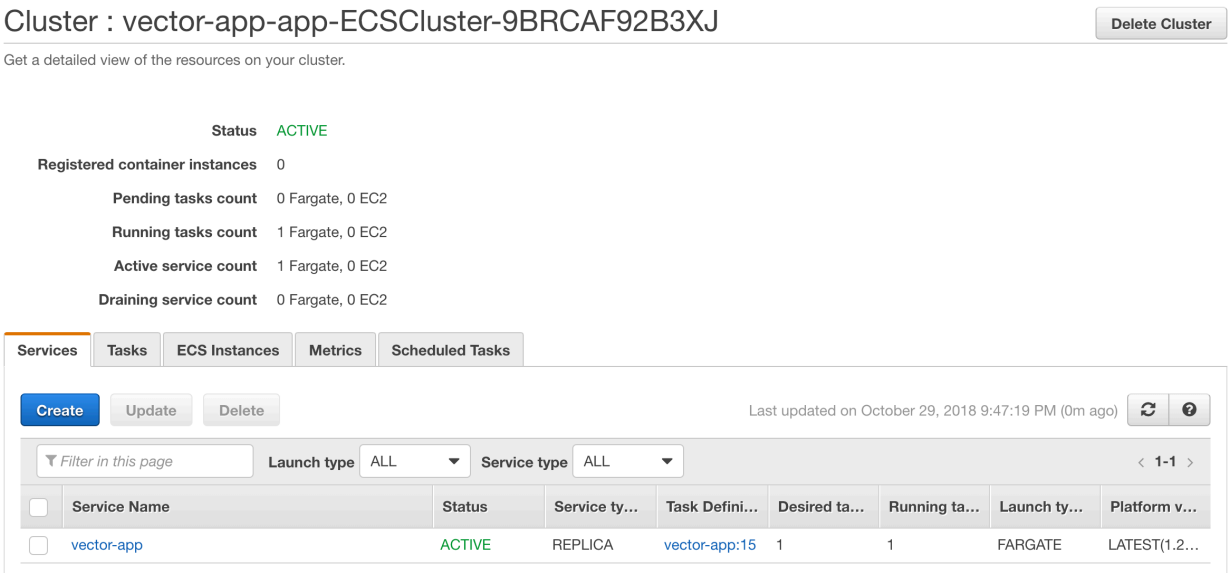

Рис. 4. Окно с описанием кластера ECS

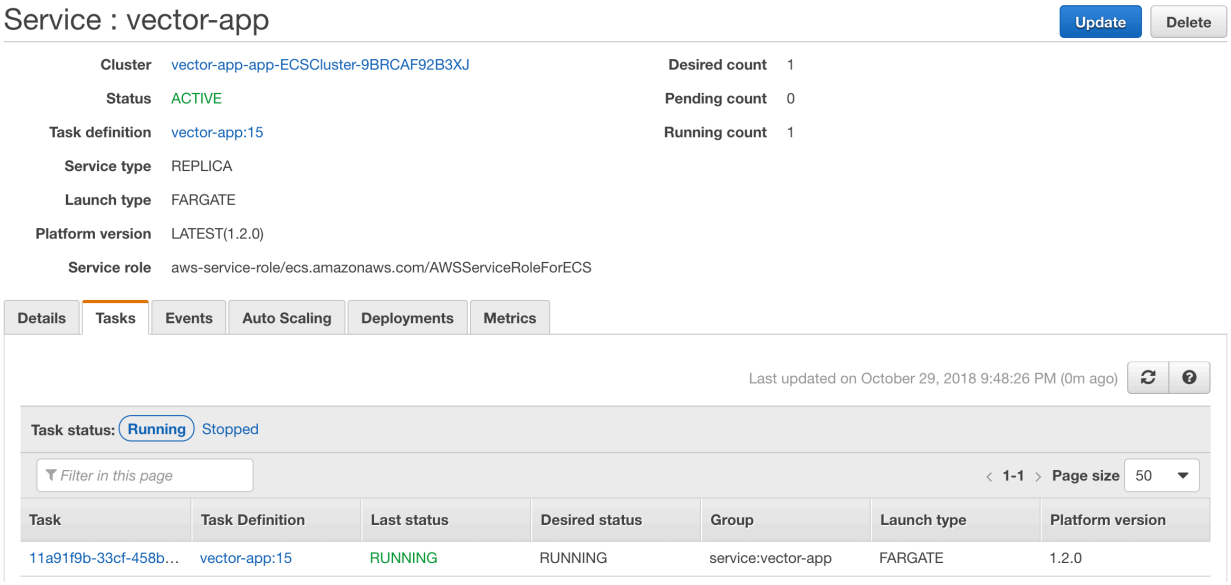

Рис. 5. Окно с описанием сервиса ECS и запущенными задачами ECS

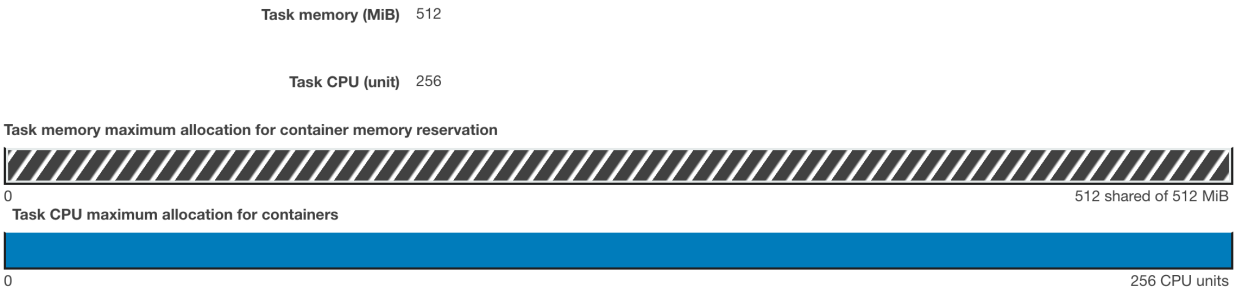

Рис. 6. Окно с описанием ресурсов контейнера

Также для приложения было настроено автомаштабирование (рис. 6). В качестве метрики, по которой определяется необходимость масштабирование, было выбрано среднее значение загрузки ЦПУ за минуту, а критическое значение, при котором происходит масштабирование, задано в 60 процентов.

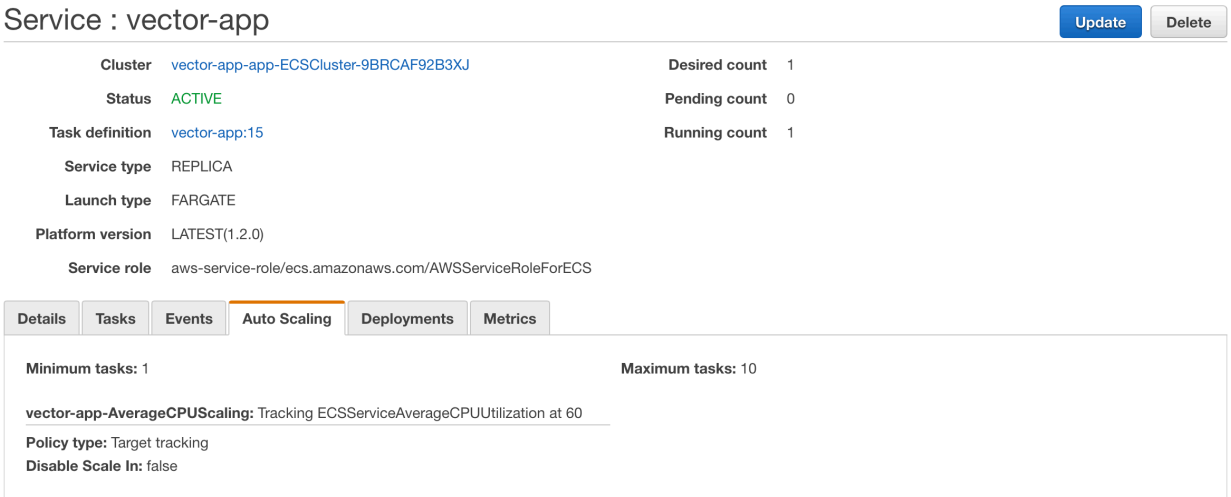

Рис. 6. Окно с параметрами автомасштабирования

3.2 Нагрузочное тестирование

Для демонстрации процесса автомасштабирования, с помощью библиотеки Locust были разработаны нагрузочные тесты и проведено нагрузочное тестирование. Нагрузка на приложение создавалась постепенно: каждую секунду добавлялся пользователь. График количества запросов в секунду приведен на рисунке 7. График медианы и 95% перцентили времени ответа приведен на рисунке 8. Оранжевыми стрелками обозначены моменты запуска дополнительных контейнеров.

Рис. 7. График количества запросов в секунду

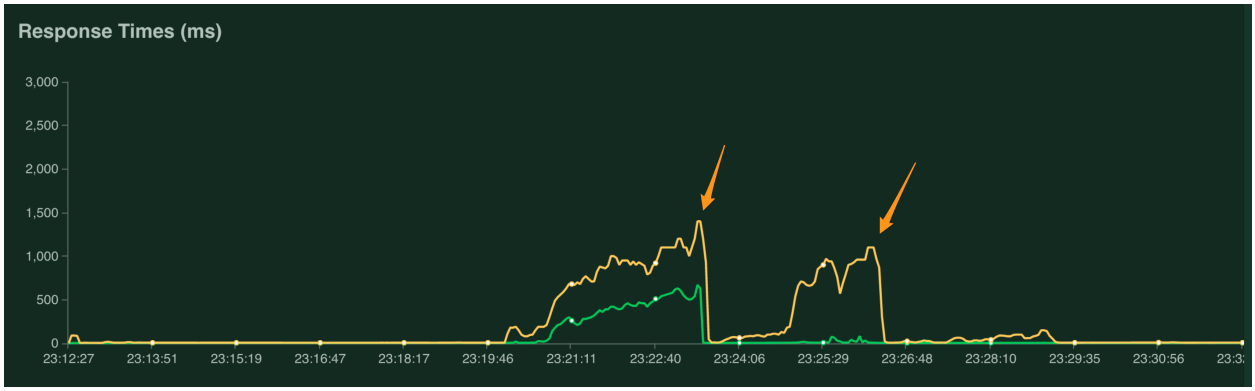

Рис. 8. График медианы (зеленый) и 95% (оранжевый) перцентили времени ответа

При повышении нагрузки, ECS отслеживал среднюю загрузку ЦПУ и при превышении критического значения в 60% принимал решение запустить дополнительную задачу ECS (рис. 9, 10, 11).

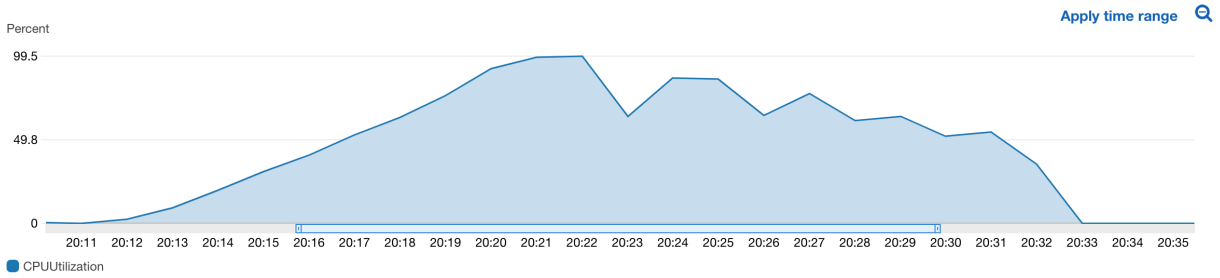

Рис. 9. График загрузки ЦПУ

Filter in this page

1-19

Event Id	Event Time	Message
29bbc947-8672-4771-90c4-05133a04f53f	2018-10-29 23:31:17 +0300	service vector-app has reached a steady state.
4bf48ee8-a1bc-4cf1-88ff-52572168a9a4	2018-10-29 23:30:58 +0300	service vector-app registered 1 targets in target-group vector-app
fb9bd53e-4160-4321-94ab-0e72aa61e90b	2018-10-29 23:30:36 +0300	service vector-app has started 1 tasks: task cdfbf55f-351d-4aee-b231-1cec8408a1fb.
16b45cf4-9296-47a6-9148-874182aff43e	2018-10-29 23:30:21 +0300	Message: Successfully set desired count to 5. Change successfully fulfilled by ecs. Cause: monitor alarm TargetTracking-service/vector-app-app-ECSCluster-1XS1I020SVQO7/vector-app-AlarmHigh-29ecffd8-7a94-4178-99c7-06cedb840d75 in state ALARM triggered policy vector-app-AverageCPUScaling

Рис. 10. Лог запуска новой задачи

Details | Tasks | Events | Auto Scaling | Deployments | Metrics

Last updated on October 29, 2018 11:32:27 PM (0m ago)

Task status: Running Stopped

Filter in this page

1-5 Page size 50

Task	Task Definition	Last status	Desired status	Group	Launch type	Platform version
02f1d4cb-47a9-45f8…	vector-app:17	RUNNING	RUNNING	service:vector-app	FARGATE	1.2.0
4feff03b-d6ae-4c79-…	vector-app:17	RUNNING	RUNNING	service:vector-app	FARGATE	1.2.0
9edad6da-0c24-43d…	vector-app:17	RUNNING	RUNNING	service:vector-app	FARGATE	1.2.0
aa209876-dde2-4e5…	vector-app:17	RUNNING	RUNNING	service:vector-app	FARGATE	1.2.0
cdfbf55f-351d-4aee-…	vector-app:17	RUNNING	RUNNING	service:vector-app	FARGATE	1.2.0

Рис. 11. Список запущенных задач в конце нагрузочного тестирования

После завершения нагрузочного тестирования, загрузка ЦПУ снизилась и, следовательно, ECS запустил процедуру удаления дополнительных задач (рис. 12).

Filter in this page

1-23

Event Id	Event Time	Message
a036ee6e-3b03-4455-ac63-89e1a8cbd1ce	2018-10-29 23:42:39 +0300	service vector-app has reached a steady state.
3faccc4e-ffe6-4a50-9128-8dde051fa89f	2018-10-29 23:42:29 +0300	service vector-app has stopped 4 running tasks: task cdfbf55f-351d-4aee-b231-1cec8408a1fb task 9edad6da-0c24-43d6-a1c9-cbf1fe5b0022 task aa209876-dde2-4e58-a5b0-e4535138920c task 4feff03b-d6ae-4c79-b6ef-fe4aca54b5bd.
ccbff21f-83be-4f73-80dd-34d1b7ce950b	2018-10-29 23:37:25 +0300	service vector-app has begun draining connections on 4 tasks.
e4744c0f-7813-4dad-8fc5-8281b0d8a2ee	2018-10-29 23:37:25 +0300	service vector-app deregistered 4 targets in target-group vector-app

Рис. 12. Лог удаления дополнительных задач

4 ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД

Полный исходный код доступен по ссылке:

- https://github.com/CatWantsMeow/vectorApp

4.1 Исходный код веб-приложения

```go
// file: api/handlers.go
package api

import (
   "encoding/json"
   "errors"
   "github.com/CatWantsMeow/vectorApp/app/vector"
   "net/http"
)

type RequestPayload struct {
   A  vector.Vector `json:"a"`
   B  vector.Vector `json:"b"`
   Op string        `json:"op"`
}

type ResponsePayload struct {
   Result vector.Vector `json:"result,omitempty"`
   Error  string        `json:"error,omitempty"`
}

func writeError(err error, w http.ResponseWriter) {
   w.WriteHeader(400)
   rsp := ResponsePayload{Error: err.Error()}
   json.NewEncoder(w).Encode(&rsp)
}

func validatePayload(payload RequestPayload) error {
   if payload.Op == "" {
      return errors.New("'op' parameter is required")
   }
   if payload.A == nil || len(payload.A) == 0 {
      return errors.New("'a' parameter is required")
   }
   if payload.B == nil || len(payload.B) == 0 {
      return errors.New("'b' parameter is required")
   }
   return nil
}

func CalculateHandler(w http.ResponseWriter, r *http.Request) {
   req := RequestPayload{}

   if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
      writeError(errors.New("failed to decode request JSON"), w)
      return
   }

   if err := validatePayload(req); err != nil {
      writeError(err, w)
      return
   }

   result, err := vector.Perform(req.Op, req.A, req.B)
   if err != nil {
      writeError(err, w)
      return
```

```
    }

    rsp := ResponsePayload{Result: result}
    json.NewEncoder(w).Encode(&rsp)
}

func HealthCheckHandler(w http.ResponseWriter, r *http.Request) {}


// file: vector/vector.go
package vector

import (
    "errors"
)

type (
    Vector    []float64
    Operation func(Vector, Vector) (Vector, error)
)

var Operations = map[string]Operation{
    "add": Add,
    "sub": Sub,
    "dot": Dot,
}

func checkLenghts(a Vector, b Vector) error {
    if len(a) != len(b) {
        return errors.New("vectors are different length")
    }
    return nil
}

func Add(a Vector, b Vector) (Vector, error) {
    if err := checkLenghts(a, b); err != nil {
        return nil, err
    }

    c := make(Vector, len(a), len(a))
    for i := 0; i < len(a); i++ {
        c[i] = a[i] + b[i]
    }
    return c, nil
}

func Sub(a Vector, b Vector) (Vector, error) {
    if err := checkLenghts(a, b); err != nil {
        return nil, err
    }

    c := make(Vector, len(a), len(a))
    for i := 0; i < len(a); i++ {
        c[i] = a[i] - b[i]
    }
    return c, nil
}

func Dot(a Vector, b Vector) (Vector, error) {
    if err := checkLenghts(a, b); err != nil {
        return nil, err
    }

    c := make(Vector, 1, 1)
    for i := 0; i < len(a); i++ {
        c[0] += a[i] * b[i]
    }
    return c, nil
}
```

```go
func Perform(op string, a Vector, b Vector) (Vector, error) {
   opHandler, exists := Operations[op]
   if !exists {
      return nil, errors.New("operation is not supported")
   }
   return opHandler(a, b)
}


// file: app.go
package main

import (
   "github.com/CatWantsMeow/vectorApp/app/api"
   "log"
   "net/http"
)

func main() {
   http.HandleFunc("/vector/", api.CalculateHandler)
   http.HandleFunc("/", api.HealthCheckHandler)
   log.Fatal(http.ListenAndServe(":8080", nil))
}
```

4.2 Исходный код нагрузочных тестов

```
# file: locustfile
import locust
import random
import json


class VectorAppUserBehaviour(locust.TaskSet):

    def on_start(self):
        b = [random.random() for _ in range(1000)]
        a = [random.random() for _ in range(1000)]
        self.payloads = {
            "small": json.dumps({
                "a": a[:10],
                "b": b[:10],
                "op": "dot"
            }),
            "medium": json.dumps({
                "a": a[:100],
                "b": b[:100],
                "op": "sub"
            }),
            "large": json.dumps({
                "a": a[:1000],
                "b": b[:1000],
                "op": "add"
            })
        }

    @locust.task(1)
    def send_req_with_small_payload(self):
        self.client.post('/vector/', data=self.payloads['small'])

    @locust.task(1)
    def send_req_with_medium_payload(self):
        self.client.post('/vector/', data=self.payloads['medium'])

    @locust.task(1)
    def send_req_with_large_payload(self):
        self.client.post('/vector/', data=self.payloads['large'])


class VectorAppUser(locust.HttpLocust):

    task_set = VectorAppUserBehaviour
    min_wait = 1000
    max_wait = 2000
```

4.3 Исходный код Dockerfile

```
FROM golang

COPY ./app /go/src/github.com/CatWantsMeow/vectorApp/app/
WORKDIR /go/src/github.com/CatWantsMeow/vectorApp/app/
RUN CGO_ENABLED=0 GOOS=linux go build . && \
    mkdir -p /go/bin && \
    mv -v app /go/bin/

FROM alpine
COPY --from=0 /go/bin/app /app
CMD ["/app"]
```