|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| **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 3

1.1 Цель 3

1.2 Задачи 3

2 МОДЕЛЬ КАЧЕСТВА 4

2.1 Функиональность 4

2.2 Надежность 4

2.3 Эффективность 4

2.4 Мобильность 5

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ 6

3.1 Создание приложения 6

3.2 Нагрузочное тестирование 9

4 ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД 11

4.1 Исходный код веб-приложения 11

4.2 Исходный код нагрузочных тестов 14

4.3 Исходный код Dockerfile 14

# ПОСТАНОВКА ЗАДАЧИ

## Цель

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

## Задачи

1. Реализовать веб-приложение с использованием языка программирования Go, которое должно предоставлять следующие функции, реализованные в виде RESTful Web API:
   1. Сложение двух векторов;
   2. Вычитание двух векторов;
   3. 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.

# 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.

## Функиональность

Функциональность описывает соответствие реализованных функций ПО требуемой пользователем функциональности.

Подхарактеристики:

* *Пригодность для применения*. В качестве метрики для этой характеристики можно рассматривать следующее отношение , где – количество реализованных операций над двумя векторами, а – количество возможных операций над двумя векторами.
* *Корректность*. В качестве метрики корректности можно взять отношение , где – количество ответов приложения с корректным результатом операции над двумя векторами, а – количество всех ответов приложения.

## 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 : , где – количество ответов приложения со статус кодом 500 (количество ошибок сервера), а – количество всех ответов приложения.
* *Доступность.* В качестве уровня доступности можно принять , где – максимальное время ответа на запрос за некоторый промежуток времени.

## Эффективность

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

Подхарактеристики:

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

## 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 необходимое для создания новой копии приложения при высокой нагрузке.

# 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).

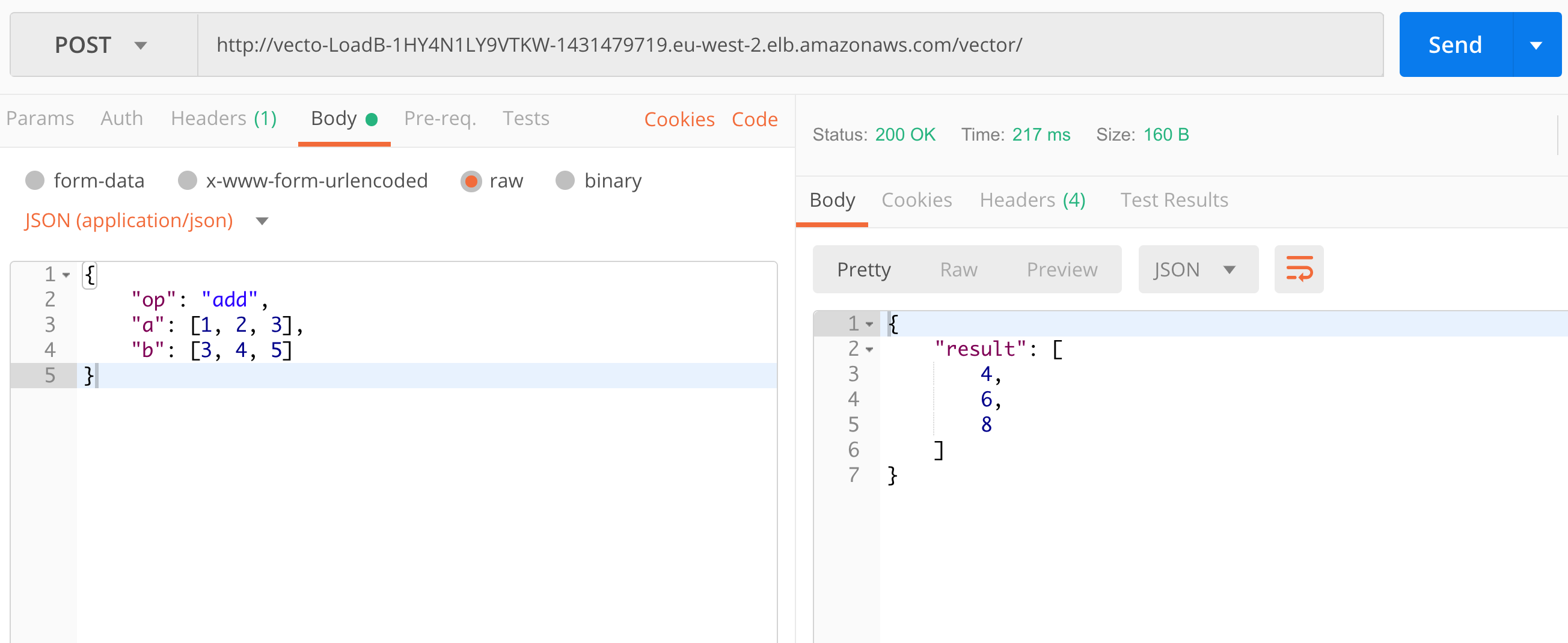


Рис. 1. Операция сложения двух векторов

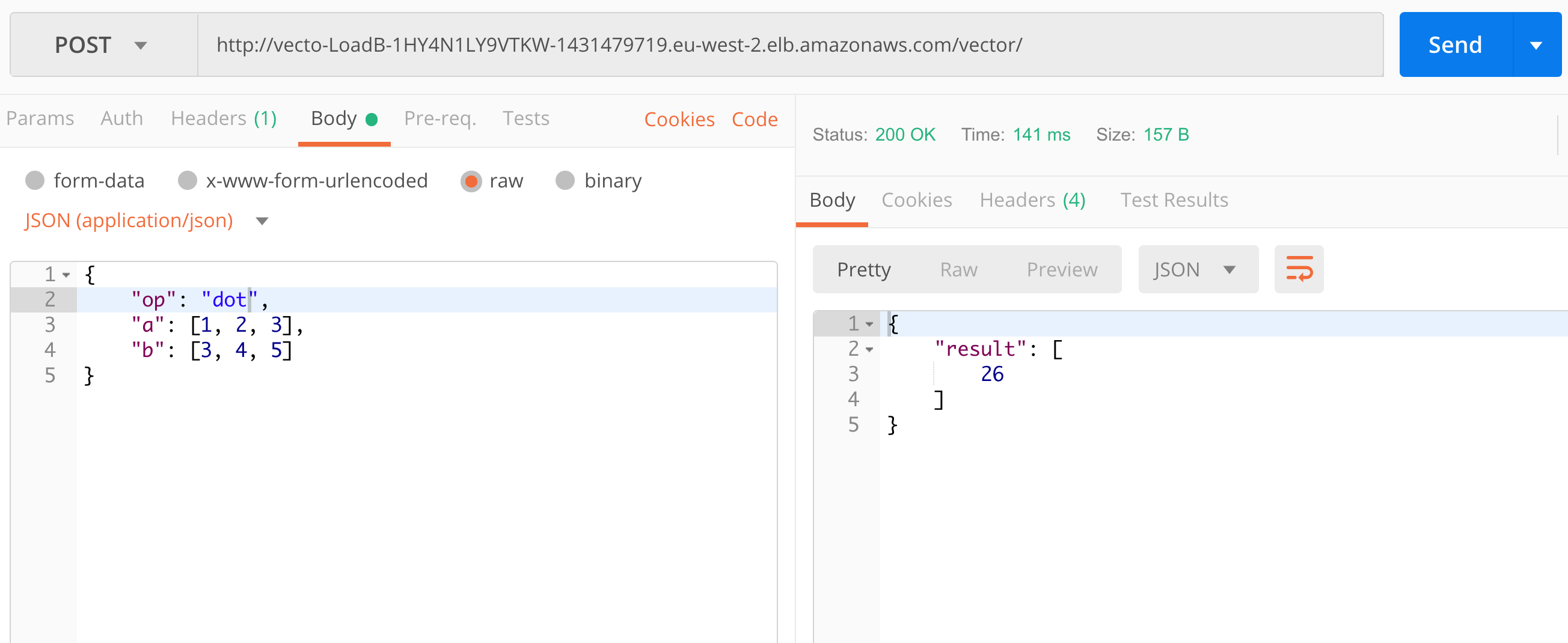


Рис. 2. Операция скалярного произведения двух векторов

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).

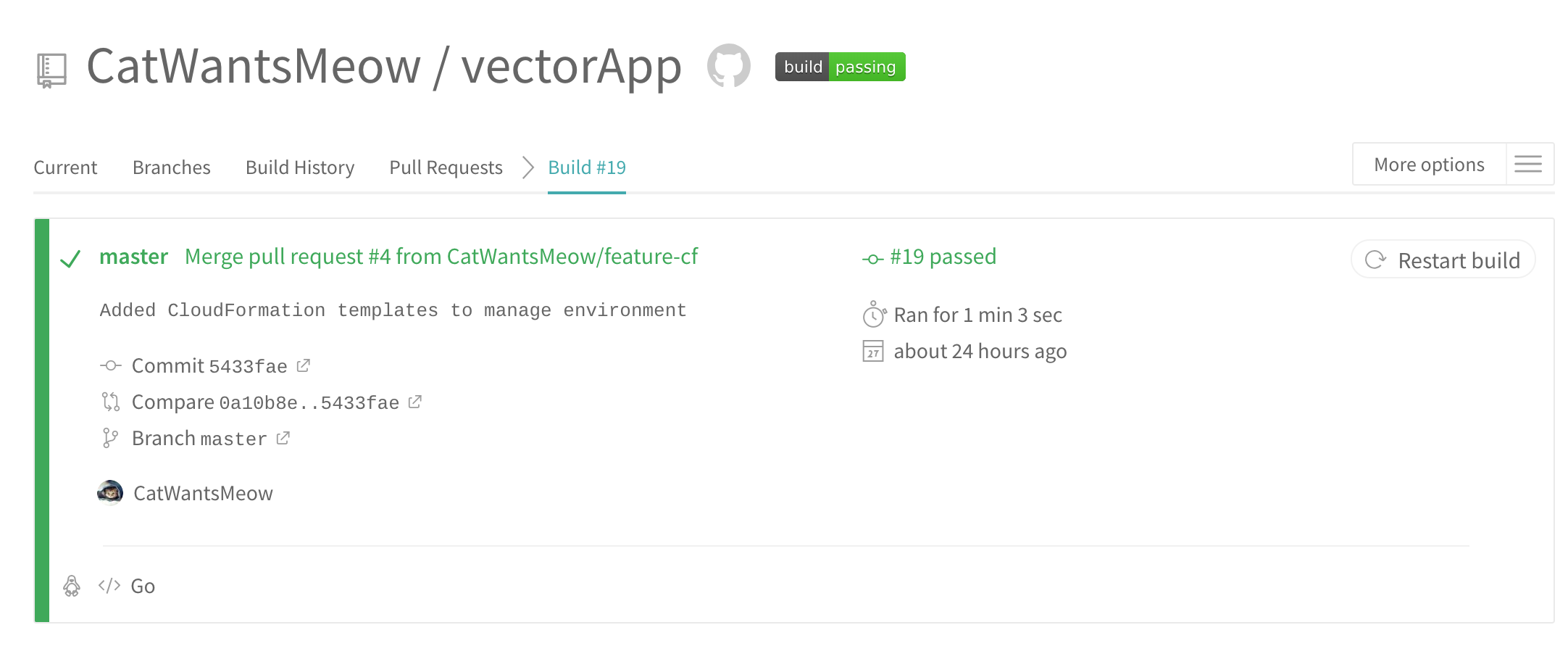
****

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 – cущности 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).

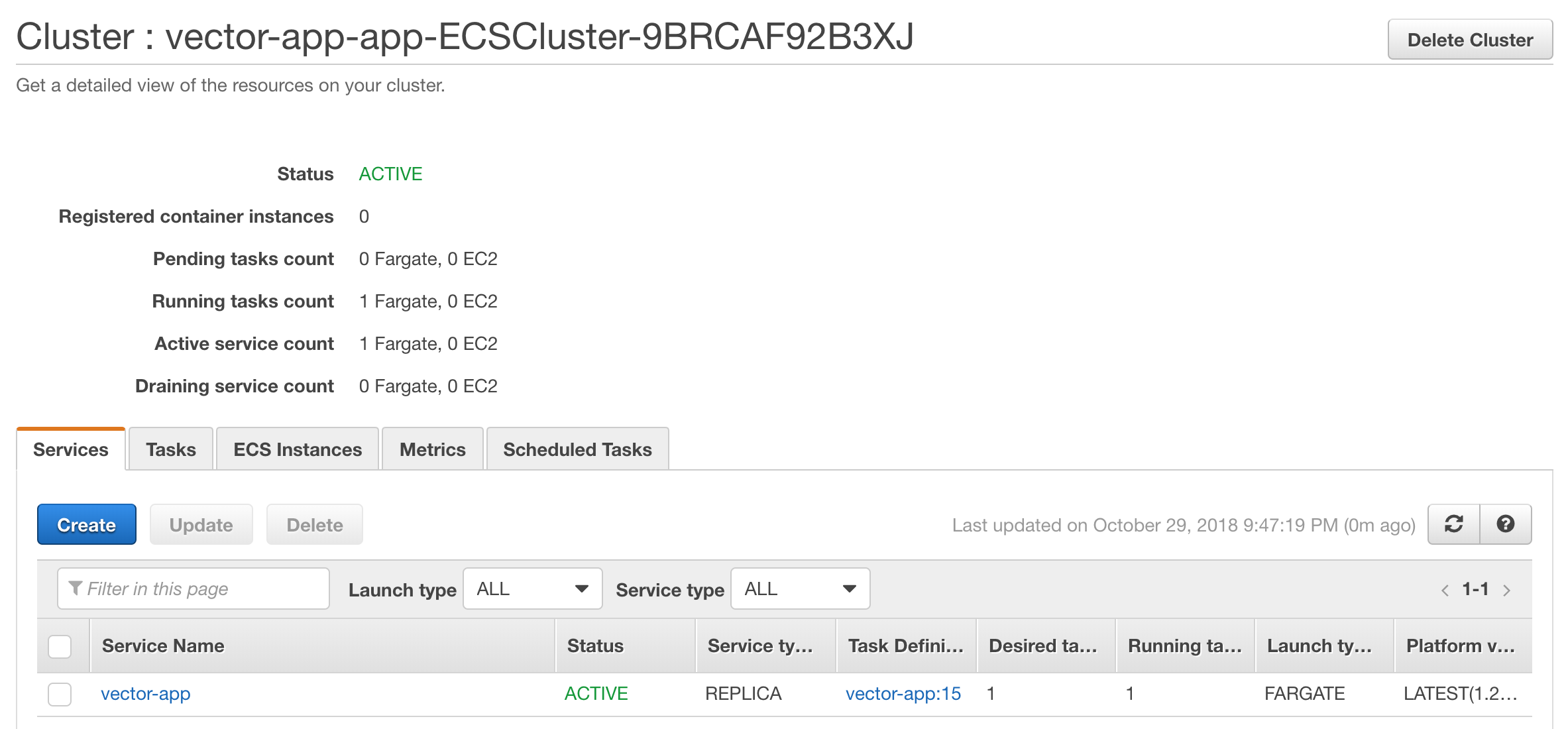


Рис. 4. Окно с описанием кластера ECS

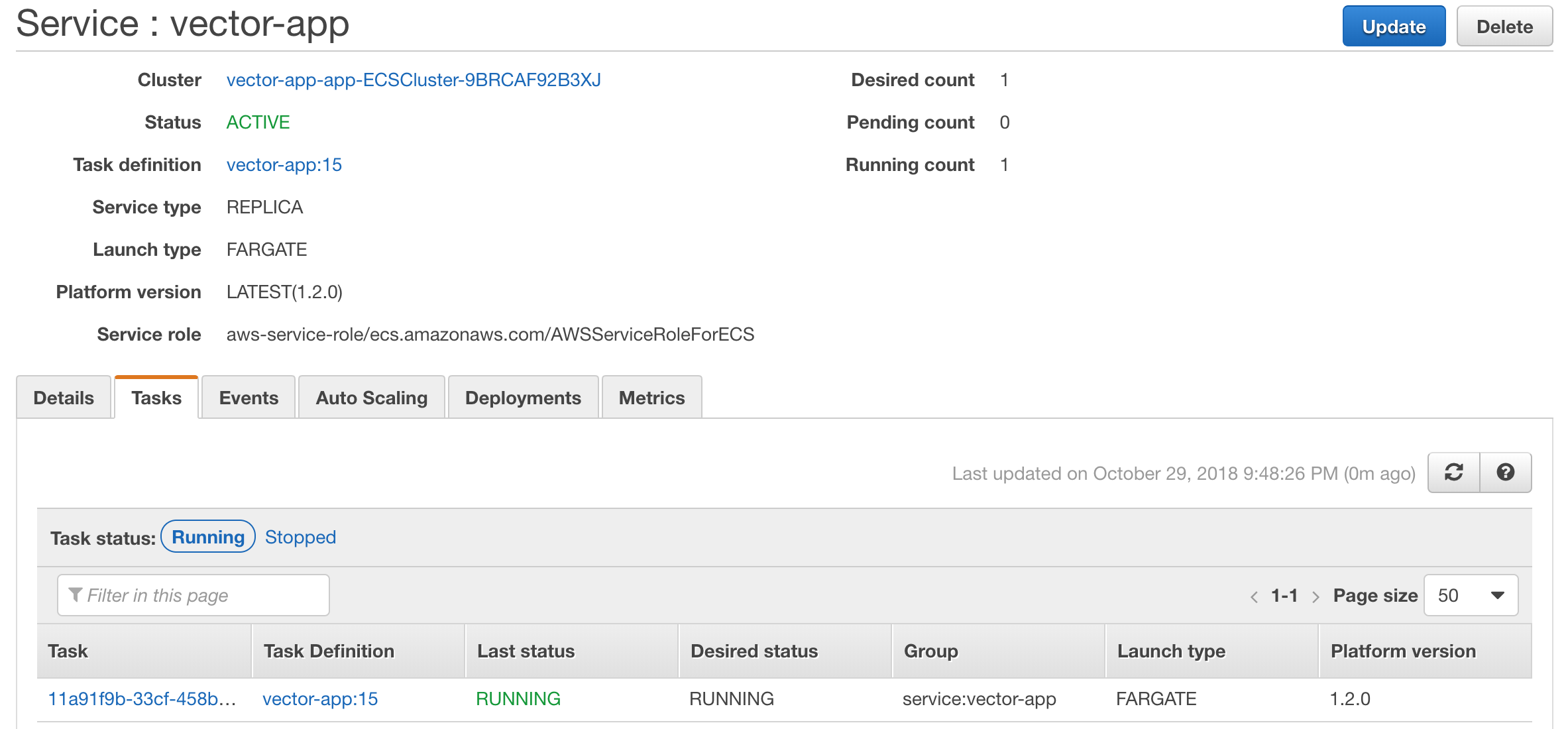


Рис. 5. Окно с описанием сервиса ECS и запущенными задачами ECS

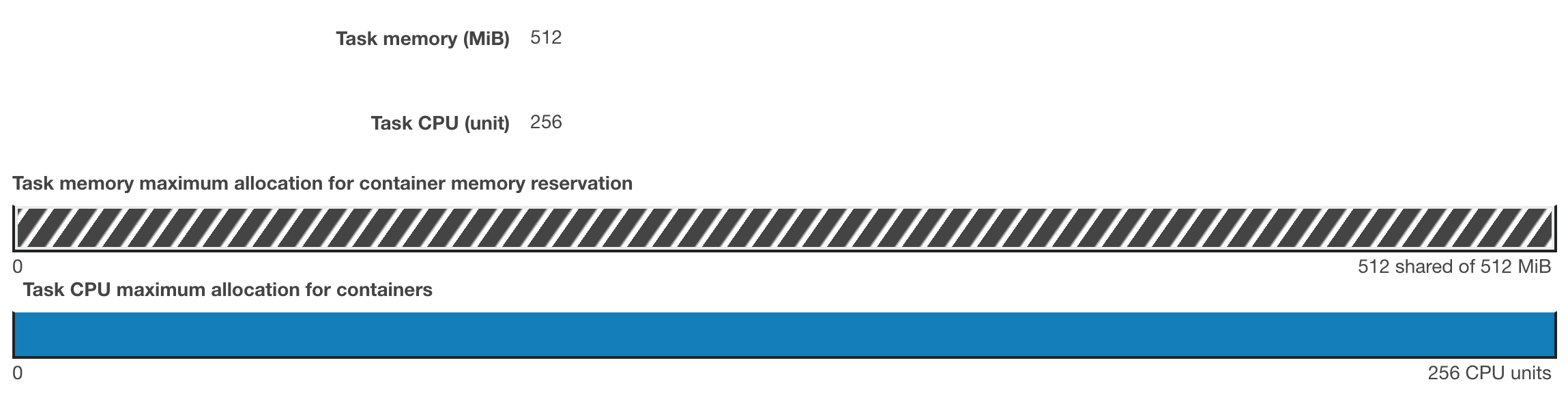


Рис. 6. Окно с описанием ресурсов контейнера

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.

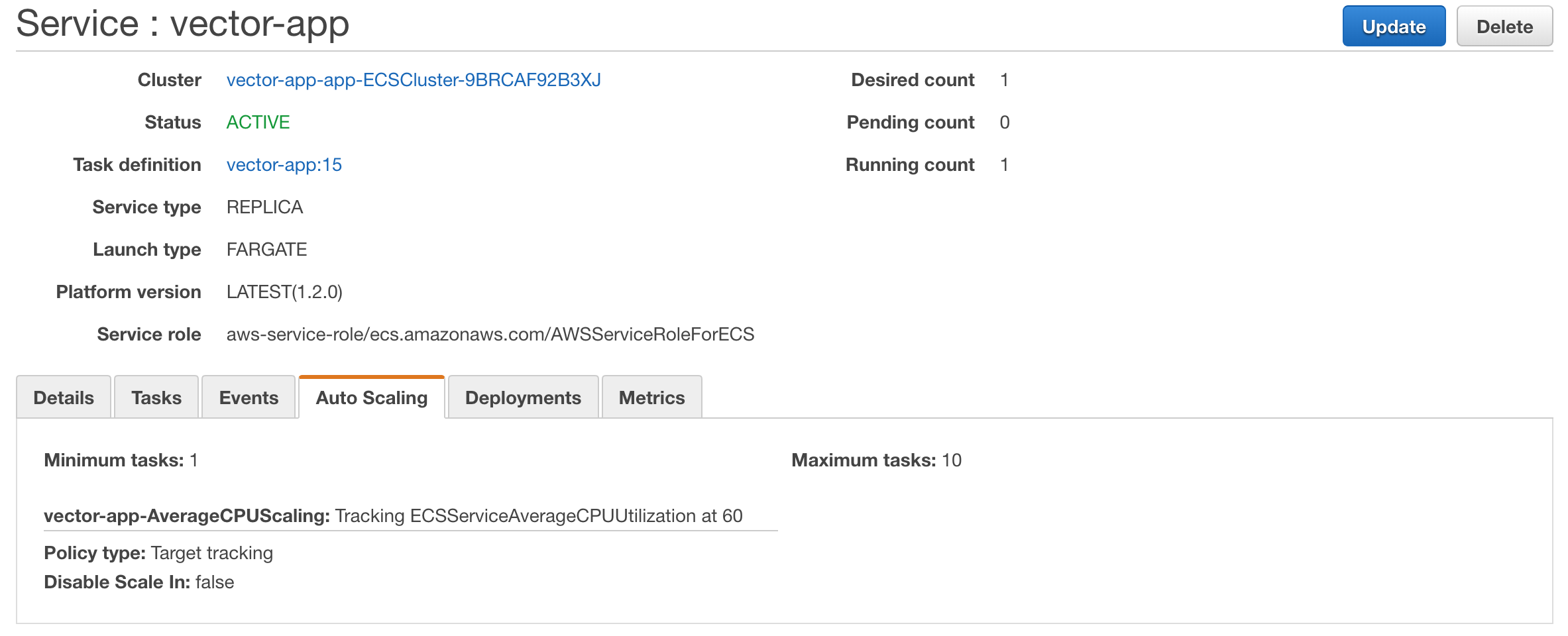


Рис. 6. Окно с параметрами автомасштабирования

## 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

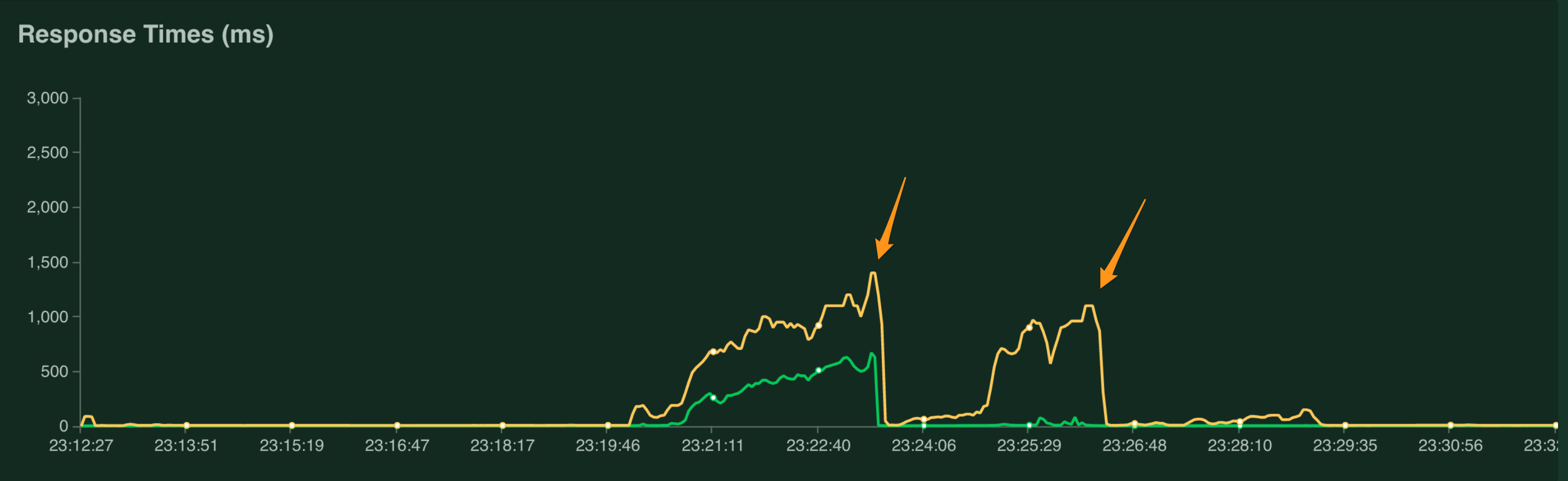


Рис. 8. График медианы (зеленый) и 95% (оранжевый) перцентили времени ответа

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).

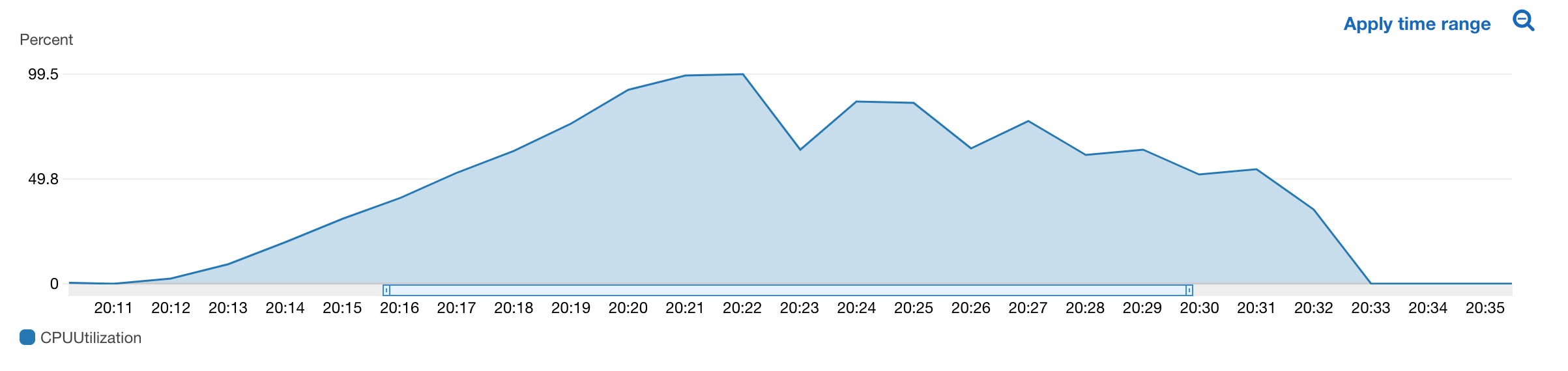


Рис. 9. График загрузки ЦПУ

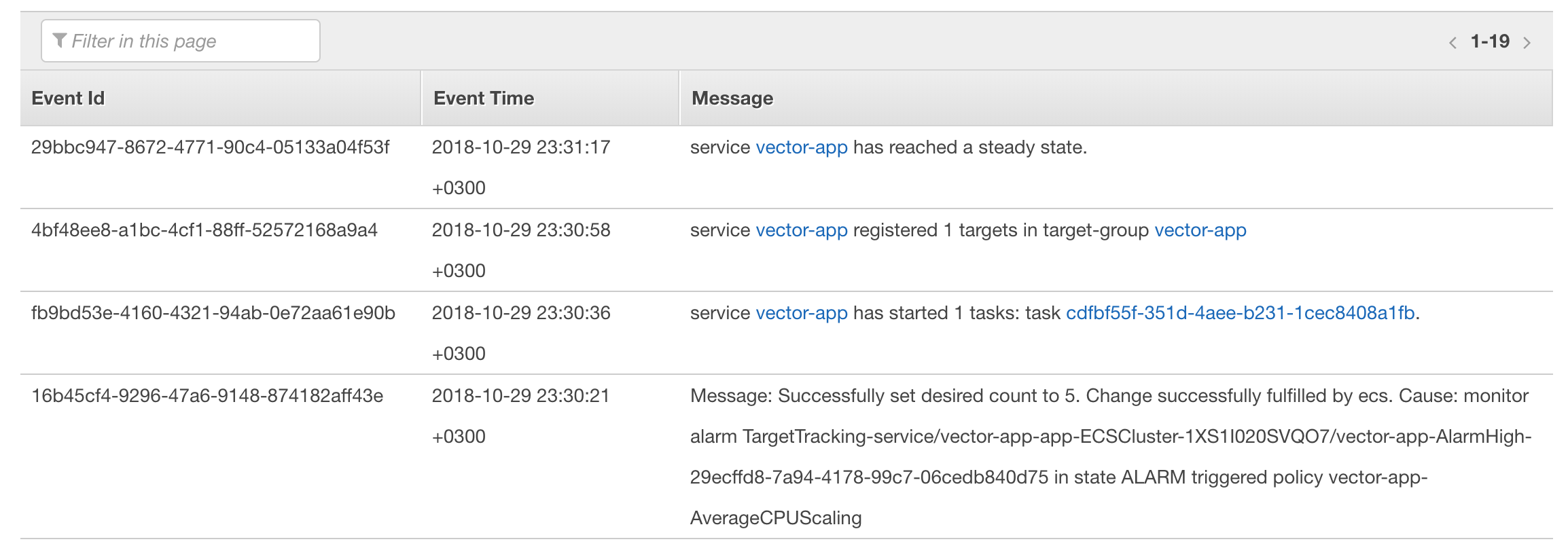


Рис. 10. Лог запуска новой задачи

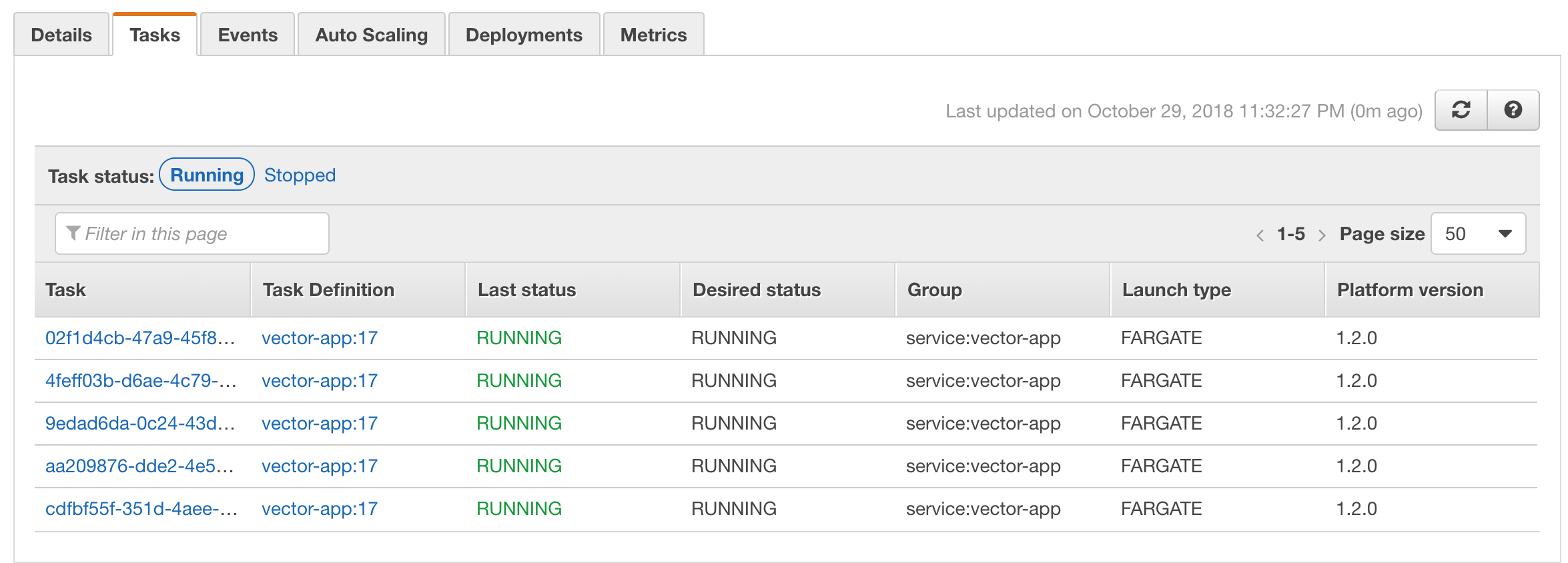


Рис. 11. Список запущенных задач в конце нагрузочного тестирования

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

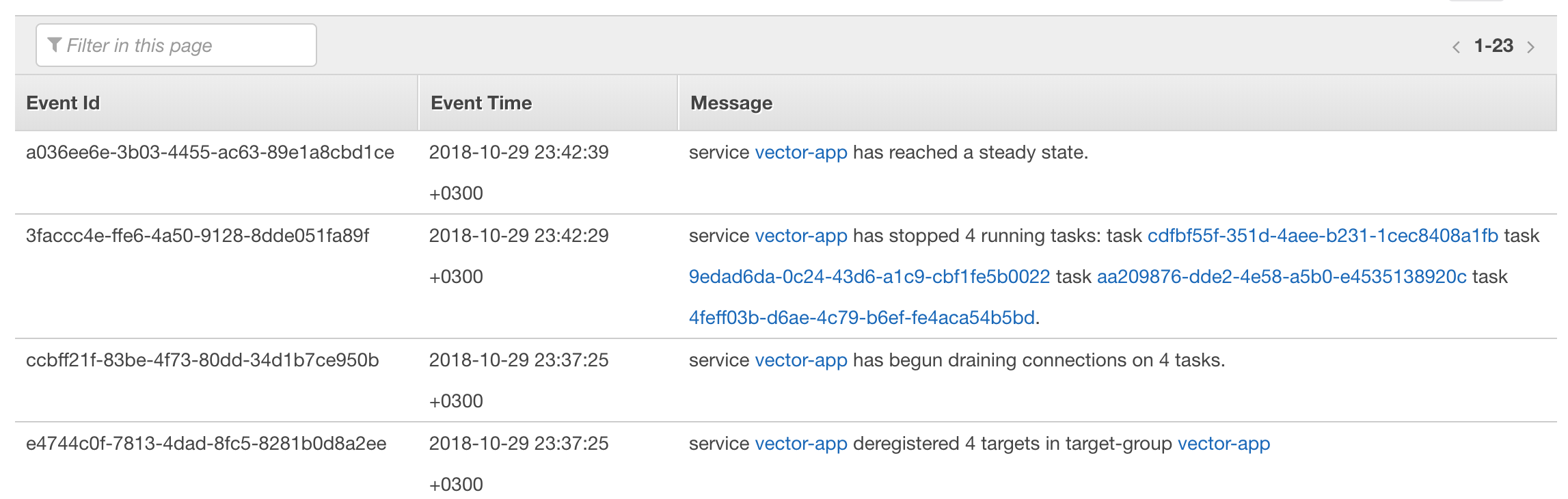


Рис. 12. Лог удаления дополнительных задач

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД

Полный исходный код доступен по ссылке:

* <https://github.com/CatWantsMeow/vectorApp>

## 4.1 Исходный код веб-приложения

*// 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  
}  
  
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"]