

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий  
Высшая школа искусственного интеллекта

КУРСОВАЯ РАБОТА

По дисциплине:

«Облачные вычисления для решения задач в машинном обучении»

Выполнил  
студент группы 3540201/00301

А. И. Хохлявин

Руководитель

А. А. Лукашин

Санкт-Петербург  
2022 г.

# Содержание

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Введение и постановка задачи</b>                       | <b>3</b> |
| <b>2</b> | <b>Описание решения</b>                                   | <b>3</b> |
| 2.1      | Подготовка дистрибутива приложения, публикация артефактов | 3        |
| 2.2      | Подготовка и настройка инфраструктуры . . . . .           | 5        |
| 2.3      | Деплоймент . . . . .                                      | 6        |
| 2.4      | Демонстрация работы . . . . .                             | 7        |
| <b>3</b> | <b>Заключение</b>   | <b>8</b> |

# 1 Введение и постановка задачи

В мире IT всё чаще компании прибегают к использованию готовых облачных решений для размещения своих приложений. Это дешевле и надёжнее, чем строить собственную сетевую инфраструктуру с нуля. Лучшие практики масштабирования, управления нагрузкой, безопасности и скорости доступа к данным уже реализованы крупными игроками, например, Amazon Web Services. К тому же, на рынке существует широкий инструментарий для удобной настройки инфраструктуры вокруг приложения любого типа на базе AWS (или любого другого достаточно популярного облачного сервиса, например, Yandex.Cloud).

В этой курсовой мы учимся работать с облачными сервисами. Нам предлагается разработать веб-приложение и разместить его в одном из облачных решений от компаний Amazon, Yandex, etc.

**Цель работы:** освоить работу с облачными сервисами на примере подготовки инфраструктуры для развертывания простого приложения.

В ходе работы предлагается выполнить следующие шаги:

1. Подготовка дистрибутива приложения и публикация артефактов
2. Подготовка и настройка инфраструктуры
3. Деплоймент (размещение артефактов на виртуальных машинах)
4. Демонстрация работы

## 2 Описание решения

### 2.1 Подготовка дистрибутива приложения, публикация артефактов

В ходе работы мы разрабатываем веб-сервис наподобие твиттера: одностраничное веб-приложение без авторизации, с единой лентой сообщений от пользователей, отсортированных в обратном хронологическом порядке (так что сообщения, опубликованные раньше, находятся наверху страницы).

В качестве языка программирования был выбран Python, в качестве веб-фреймворка – легковесный Flask, позволяющий работать с HTML-шаблонизатором Jinja и создавать простое API. Для наших целей этого более чем достаточно.

Кратко опишем бизнес-логику приложения через его возможности:

- Просмотр всех когда-либо опубликованных постов (по API и с помощью веб-интерфейса приложения) – получение всех существующих записей таблицы `PostsTable` и отображение на странице.

- Публикация нового поста (по API и с помощью веб-интерфейса приложения) – добавление новой записи в таблицу `PostsTable`.

Таблица `PostsTable` при этом – это таблица легко-масштабируемой NoSQL базы данных Amazon DynamoDB. Обращение к базе данных из Flask будем осуществлять с помощью библиотеки `boto3`. Оформление главной страницы с постами и формы отправки нового поста производим при помощи Twitter Bootstrap, для контейнеризации используем Docker.

Сборка осуществляется вызовом `docker build --tag <container_name> .` из директории `/app`, в которой находится `Dockerfile`, где `<container_name>` – это human-readable имя контейнера для последующей публикации в регистре контейнеров AWS (предварительно необходимо настроить AWS CLI (ключи и регион), вызвав `aws configure`.

Само создание контейнера в регистре контейнеров осуществляется консольной командой

```
aws ecr create-repository \  
--repository-name <container_name> \  
--image-scanning-configuration scanOnPush=true \  
--region us-east-1
```

После чего в консоли разработчика Amazon ECR отобразится новый контейнер с именем `<container_name>`.

Далее необходимо залогиниться в ECR, выполнив команду, приведённую ниже. Вместо `<account_id>` нужно подставить идентификатор своего аккаунта на Amazon, а в качестве имени контейнера то имя, которое было ему дано при создании.

```
aws ecr get-login-password --region us-east-1 \  
| docker login --username AWS --password-stdin \  
<account_id>.dkr.ecr.us-east-1.amazonaws.com/<container_name>
```

Далее необходимо связать локальный докер-образ с удалённым регистром контейнеров AWS ECR, для чего нужно выполнить в консоли:

```
docker tag <container_name>:latest \  
<account_id>.dkr.ecr.us-east-1.amazonaws.com/<container_name>:latest
```

После чего остаётся только запустить изменения в удалённый регистр контейнеров:

```
docker push \  
<account_id>.dkr.ecr.us-east-1.amazonaws.com/<container_name>:latest
```

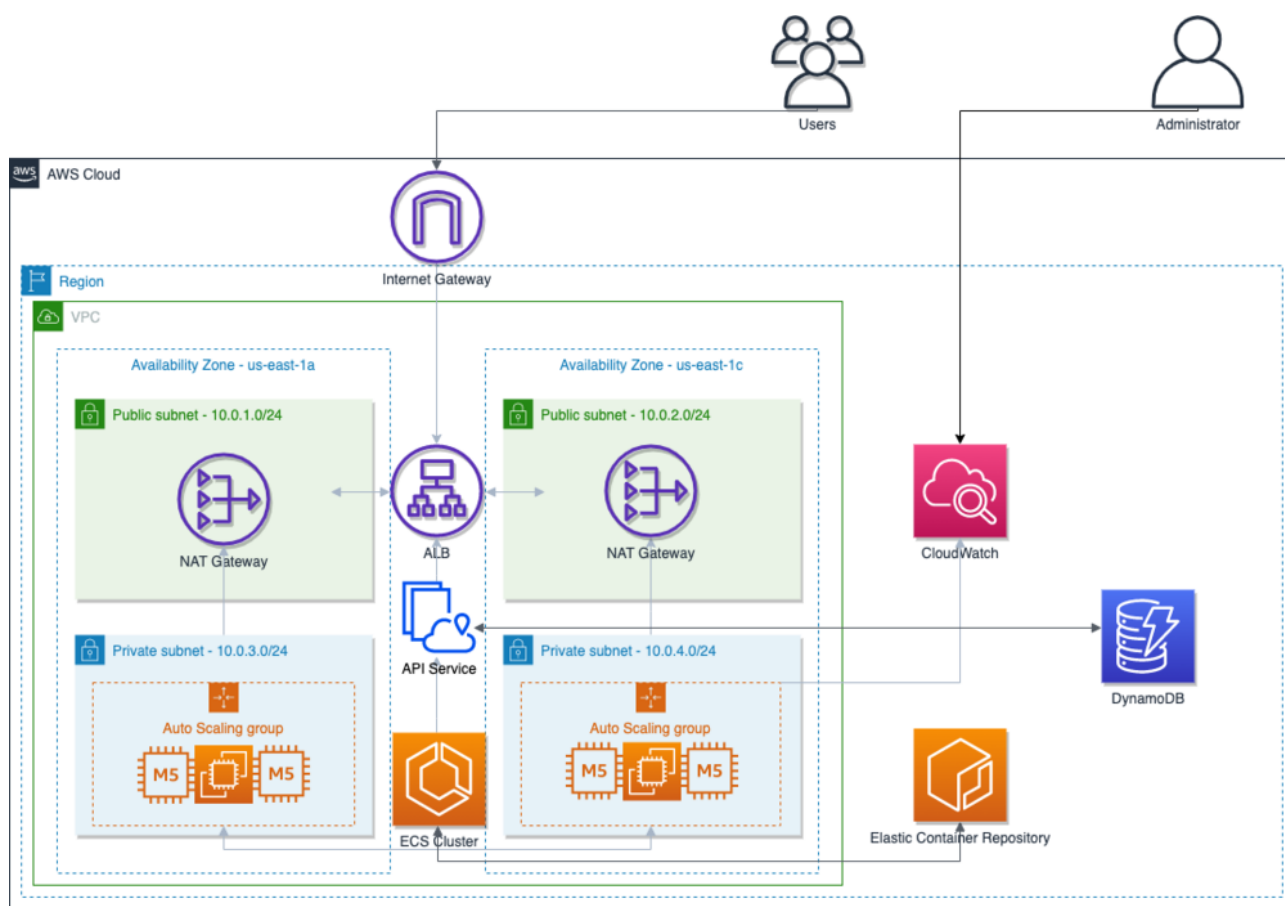


Рис. 2.1: Схема инфраструктуры для приложения

## 2.2 Подготовка и настройка инфраструктуры

Для начала высокочувствительно взглянем на предлагаемую инфраструктуру. На рисунке 2.1 приведена схема организации архитектуры в данной работе.

Кратко рассмотрим и опишем всё, что здесь происходит.

Итак, мы деплоим Amazon VPC (Virtual Private Cloud) с двумя приватными и двумя публичными подсетями в зонах доступности `us-east-1a` и `us-east-1c`. Мы прикрепили к VPC интернет-шлюз, чтобы сделать доступным ресурс для внешних пользователей за пределом частной сети. Ресурсы в частных подсетях получают доступ к Интернету через шлюзы NAT (Network Address Translation) в общедоступных подсетях.

Приложение будет размещено в кластере Amazon ECS, работающем на инстансах Amazon Elastic Compute Cloud (Amazon EC2), которые будут расположены в частных подсетях.

В качестве базы данных используется Amazon DynamoDB, а CloudWatch будет мониторить отзывчивость приложения, сигнализировать об изменениях в перформансе и в целом предоставлять информацию о здоровье веб-приложения. Наконец, как было отмечено выше, Amazon ECR используется для хранения докер-образа веб-приложения.

Для создания инфраструктуры мы используем подход *infrastructure as a code*, сильным представителем которого является разработка компании Hashicorp – Terraform.

В соответствие со сложившимися хорошими практиками разработки архитектуры, в папке с описанием инфраструктуры (**terraform**) соблюдается следующая файловая структура:

- **config.tf**: информация о провайдере Terraform и версиях AWS модулей
- **data.tf**: дата-хендлер, с помощью которого происходит шаринг данных между файлами
- **main.tf**: основной файл с описанием инфраструктуры.
- **outputs.tf**: выходные данные, передаваемые в консоль в процессе выполнения команд Terraform
- **variables.tf**: переменные Terraform

Этот набор файлов Terraform определяет политики безопасности сети и конфигурации кластеров. Основные ресурсы, которые регулируются: IAM – политики доступа к профилям AWS, VPC – конфигурация публичных и частных подсетей, маршрутизация и NAT-шлюз, EC2 – реализация автоскейлинга, ECS – конфигурация кластера, ALB – автобалансировщик нагрузки, DynamoDB – конфигурация таблицы с постами пользователей, CloudWatch – мониторинг приложения.

## 2.3 Деплоймент

Для разещения приложения в описанной на предыдущих шагах архитектуре необходимо выполнить несколько команд **terraform** в папке с конфигурационными файлами для инфраструктуры (**terraform/\*.tf**). Для проверки правильности описания инфраструктуры перед деплоем:

```
terraform fmt -recursive
terraform validate
```

Для просмотра изменений по сравнению с текущей архитектурой:

```
terraform plan
```



И наконец для применения изменений:

```
terraform apply
```

# MAT is for Micro Anonymous Twitter

Reimagined way of communication.

## Latest posts

-  [helen.pancake](#) @ 09.01.2022 21:50  
Here we go! This is another post. All this stuff made by Alex Khokhlyavin in the very beginning of 2022 year as SPbSTU course work on cloud technologies subject. Wish you all the best in New Year!
-  [xoxai](#) @ 09.01.2022 17:05  
The first post on this service ever. Ground Control to Major Tom.

Username

Post content

Share a story

Рис. 2.2: Внешний вид веб-приложения по публикации постов

## 2.4 Демонстрация работы

В результате успешного применения всех конфигураций инфраструктуры, в выводе консоли увидим публичный адрес автоматического балансировщика нагрузки, который и будет являться точкой входа в приложение для внешнего пользователя.

Так, опубликован веб-сервис для публичных постов. Внешний вид представлен на рисунке 2.2, а полностью работающее демо можно протестировать по ссылке: <http://ecsalb-1729053732.us-east-1.elb.amazonaws.com/>

Наконец, весь исходный код и материалы этой работы доступны на GitHub: <https://github.com/xoxai/mat>

## 3 Заключение

По результатам работы можно сказать, что мы научились работать с облачным сервисом Amazon, с помощью Terraform настроили архитектуру веб-приложения для анонимной публикации постов, разместили артефакты приложения в AWS ECR.