Обучающие материалы для IT

#8 от 01 октября 2023

Подготовлено для студентов курсов

Содержание

0. Очень важное замечание

Перед началом выполнения заданий, необходимо зарегистрироваться на GitHub и создать новый репозиторий, в который вы будете загружать все свои выполненные задания.
Результатом выполнения задачи будет либо код (Dockerfile, Bash скрипт, JSON файлы и т.д.), либо скриншоты из вашего браузера или другого места, где видно, что у вас все работает как требует задание.

ВАЖНО: ни при каких обстоятельствах не публикуйте персональные или секретные данные (API Keys, SSH private keys, пароли) в репозиторий в открытом виде.

Git

Git и Системы Контроля Версий (СКВ) / **version control system** (**VCS**)

Git - это распределенная система контроля версий, разработанная Линусом Торвальдсом. Она предназначена для управления версиями и отслеживания изменений в коде и других файлах во время разработки программного обеспечения. Git является одной из наиболее популярных Систем Контроля Версий (VCS) и широко используется разработчиками по всему миру.

Цели и проблемы, которые решает VCS :

- **Отслеживание Изменений**: VCS позволяют разработчикам отслеживать изменения в файлах и директориях, включая добавление, удаление и изменение кода. Это позволяет разработчикам следить за историей проекта.
- **Коллаборация:** VCS облегчают совместную работу над проектами. Разработчики могут работать над одним и тем же проектом, не переписывая исходный код друг друга, а просто сливая (мерджа) изменения.
- **Управление Версиями:** VCS позволяют создавать версии проекта. Это значит, что вы можете вернуться к предыдущей версии проекта, если что-то сломалось, или переключиться на новую версию.
- **Резервное Копирование:** VCS позволяют создавать резервные копии кода и проектов. Это важно для предотвращения потери данных.
- **Отслеживание Авторства:** VCS сохраняют информацию об авторах каждого изменения, что полезно для атрибуции и обратной связи.
- **Управление Конфликтами**: При совместной работе могут возникать конфликты при изменении одного и того же файла. VCS предоставляют инструменты для разрешения этих конфликтов.

Git и другие VCS играют ключевую роль в разработке программного обеспечения и помогают управлять сложными проектами, повышая эффективность и сотрудничество между разработчиками.

Git: Основные Команды

1. **git init:** Инициализация нового репозитория. Создает новый Git-репозиторий в текущей директории.
2. **git clone <URL>:** Клонирование существующего репозитория. Создает локальную копию удаленного репозитория.
3. **git add <файл>:** Добавление файла в индекс (staging area) для последующего коммита. Используйте . для добавления всех измененных файлов.

4. **git commit -m "Сообщение коммита":** Создание нового коммита с сохранением всех изменений, добавленных в индекс.
5. **git status:** Просмотр состояния вашего рабочего каталога и индекса. Отображает измененные и добавленные файлы.
6. **git log:** Просмотр истории коммитов. Отображает список всех коммитов в репозитории.
7. **git branch:** Просмотр списка веток. Отображает список локальных веток и выделяет текущую ветку.
8. **git checkout <ветка>:** Переключение на другую ветку. Используйте **git checkout -b <новая_ветка>** для создания и переключения на новую ветку.
9. **git pull:** Получение последних изменений из удаленного репозитория и слияние их с текущей веткой.
10. **git push:** Отправка локальных изменений в удаленный репозиторий.
11. **git diff**: Просмотр изменений между текущим рабочим состоянием и последним коммитом.
12. **git reset <файл>**: Отмена добавления файла в индекс (unstage). Используйте --hard для сброса коммита и индекса.

Задача: Начало работы с Git

Цель: Создать новый репозиторий, выполнить коммиты и пуш в удаленный репозиторий.

1. Инициализируйте новый Git-репозиторий в пустой директории.
2. Создайте новый текстовый файл с именем hello.txt и добавьте в него любой текст.
3. Добавьте hello.txt в индекс и создайте коммит с сообщением "Добавлен файл hello.txt".
4. Создайте удаленный репозиторий на платформе, такой как GitHub.
5. Свяжите ваш локальный репозиторий с удаленным репозиторием (git remote add origin <URL>).
6. Отправьте коммиты в удаленный репозиторий с помощью git push origin master.

Работа с ветками

Цель: Создать и переключаться между ветками, выполнить слияние веток.

1. Создайте новую ветку с именем "feature" (git branch feature) и переключитесь на нее (git checkout feature).
2. Внесите изменения в файл hello.txt, например, добавьте еще текст.
3. Добавьте и закоммитьте изменения в ветке "feature".
4. Переключитесь обратно на ветку "master".
5. Слейте ветку "feature" с веткой "master" (git merge feature) и решите возможные конфликты.

Работа с удаленным репозиторием и ветками

Цель: Работа с удаленными ветками и синхронизация с удаленным репозиторием.

1. Склонируйте удаленный репозиторий на вашу локальную машину.
2. Создайте новую ветку с именем "feature-remote" (git checkout -b feature-remote).
3. Внесите изменения в файл hello.txt и закоммитьте их.
4. Отправьте новую ветку "feature-remote" в удаленный репозиторий (git push origin feature-remote).
5. В GitHub создайте запрос на слияние (Pull Request) для ветки "feature-remote" в ветку "master".

Попросите другого студента (или преподавателя) просмотреть и принять ваш Pull Request.

1. Linux

Материалы и литература

- описание дистрибутивов:

  Какой linux выбрать?

- начальная установка

  Разметка жёсткого диска при установке linux

- работа с менеджером пакетов aptitude, dpkg

  Работа с пакетами при помощи dpkg

  Aptitude

  AptPreferences

  (справочник): https://debian-handbook.info/browse/ru-RU/stable/apt.html

- работа с менеджером пакетов yum

  Yum, шпаргалка

- GNU core utils

  https://ru.wikipedia.org/wiki/GNU_Coreutils

  https://github.com/uran1980/web-dev-blog/blob/master/Linux/linux-commands.md

  (справочник): https://www.gnu.org/software/coreutils/manual/coreutils.pdf (все читать не нужно, но можно активно обращаться как к справочнику в процессе разработки скриптов). Что нужно знать: cat, head, less, wc, sort, uniq, ls, cp, dd, mv, rm, mkdir, chown, chmod, touch, du, df, echo, tee, pwd, date, hostname

- bash

  Chapter 1. Bash and Bash scripts

  Chapter 2. Bash and Bash scripts

Chapter 3. Bash and Bash scripts

Chapter 4. Bash and Bash scripts

Chapter 5. Bash and Bash scripts

sed + awk - можно изучать из этих статей или на примерах в Интернете.

Искусство командной строки

- curl, httpie и wget

wget - руководство GNU Wget

Wget - насос для Интернета

HTTPie — современный HTTP-клиент, похожий на команды Curl и Wget

- network utilities

LINUX: КОМАНДЫ ДЛЯ НАСТРОЙКИ СЕТИ ИЗ КОНСОЛИ

- web servers

Документация по Apache

Документация по Nginx

Создание самоподписанного сертификата. Для Apache в Ubuntu

CertBot (Бесплатный центр сертификации Let's Encrypt)

Замечание к практическому заданию

При выполнении заданий из данного раздела, пожалуйста пользуйтесь следующими ресурсами:
https://explainshell.com/ - поможет вам понять что означает команда или набор команд в скрипте.

https://www.shellcheck.net/ - поможет проверить ваши скрипты на наличие багов и ошибок.

Практическое задание

1.1. Написать скрипт определяющий запущен ли скрипт от имени пользователя root.
1.2. Написать скрипт, который выводит текущую дату и время в формате:
`2023-07-20 12:00:00`
1.3. Написать скрипт, который создает новый каталог с именем my_new_dir и переходит в него.

1.4. Написать скрипт, который копирует файл my_file.txt из каталога ~/ в каталог /tmp

1.5. Написать скрипт, который удаляет файл my_file.txt из каталога ~/

1.6. Написать скрипт, который выводит список файлов в каталоге ~/

1.7. Написать скрипт, который запрашивает у пользователя имя файла и выводит его содержимое.

```
Введите имя файла: my_file.txt
Содержимое файла my_file.txt:
Hello, world!
```

1.8. Написать скрипт, который запрашивает у пользователя имя каталога и выводит список файлов в нем.

```
Введите имя каталога: /home/user
Файлы в каталоге /home/user:
my_file.txt
my_other_file.txt
```

1.9. Написать скрипт, который запрашивает у пользователя имя файла и выводит его содержимое. Если файла не существует, вывести сообщение об ошибке.

1.10. Написать скрипт, который запрашивает у пользователя имя каталога и выводит список файлов в нем. Если каталог не существует, вывести сообщение об ошибке

1.11. Написать скрипт, который запрашивает у пользователя имя файла и заменяет все вхождения строки error строкой warning. Если файла не существует, вывести сообщение об ошибке

1.12. Написать скрипт, который ищет файлы по заданному шаблону: все файлы содержащие текст error в каталоге с логами /var/log. Если файлы не найдены, вывести сообщение об ошибке

1.13. * Вам нужно разработать Bash-скрипт для создания менеджера свободного места на диске, который будет следить за использованием дискового пространства на вашем компьютере и предоставлять информацию о статистике использования свободного места. Скрипт должен включать следующие функциональности:

- ☐ Отображение текущей статистики: Скрипт должен выводить информацию о текущем использовании свободного места на всех монтируемых дисках.
- ☐ Предупреждения о нехватке места: Если свободное место на каком-либо диске опускается ниже заданного порога, скрипт должен отправлять предупреждение, например, по электронной почте или в лог-файл.
- ☐ Очистка устаревших файлов: Скрипт должен предоставлять опцию для автоматической очистки устаревших или временных файлов на выбранном диске, чтобы освободить дополнительное место.
- ☐ Логирование: Скрипт должен вести лог, в котором записываются события, связанные с управлением свободным местом, включая предупреждения и операции по очистке.
- ☐ Использование функций и циклов: Скрипт должен быть разбит на функции для выполнения различных задач и использовать циклы для обхода дисков и анализа статистики.
- ☐ Обработчик case для аргументов: Используйте оператор case для обработки аргументов командной строки и выполнения соответствующих действий.

2. Контейнеризация и виртуализация (Vagrant,Docker,Packer,Kubernetes)

Материалы и литература

- Vagrant:

Оригинал:

Vagrant Documentation

- цикл статей на русском:

Системы виртуализации

Vagrant. Установка и первый запуск

Vagrant. Создание собственного box-a

- блог на русском

Начало работы с Vagrant

- Еще небольшая статья по старту

Vagrant. Начало работы. Часть 1 из 2

Vagrant. Начало работы. Часть 2 из 2

- Docker:

Часть 1: основы

Часть 2: термины и концепции

Часть 3: файлы Dockerfile

Часть 4: уменьшение размеров образов и ускорение их сборки

Часть 5: команды

Часть 6: работа с данными

- Packer:

Introduction to Packer

Быстрая сборка образов ОС с помощью Packer

How to use Packer to create Ubuntu 18.04 Vagrant boxes

Vagrant Builder

- Kubernetes:

Документация по Kubernetes

Шпаргалка по kubectl

Основы Kubernetes

Установка Kubernetes с помощью Minikube

Установка Kubernetes с помощью Kubespray

- видео:

Vagrant Beginner (Part 1)

Vagrant and Packer (part 1)

Vagrant Crash Course: Vagrant for beginners

Канал по Kubernetes и не только

Замечание к практическому заданию

Перед выполнением заданий необходимо ознакомиться с документацией. Практически все задания можно выполнить более чем одним способом, любой рабочий вариант будет верным.

Практические задания

Задача 2.1: Управление виртуальной машиной с использованием Vagrant

Вы учитеся администрированию и хотите научиться создавать и управлять виртуальными машинами с помощью инструментов виртуализации. Для начала, вам нужно научиться использовать Vagrant и VirtualBox для создания и управления виртуальной машиной.

Установите Vagrant и VirtualBox: Если у вас еще не установлены Vagrant и VirtualBox, установите их на вашем компьютере.

Инициализация проекта: Создайте новую директорию для вашего проекта. В этой директории выполните команду vagrant init, чтобы создать файл Vagrantfile.
Настройка виртуальной машины: Откройте Vagrantfile и настройте виртуальную машину следующим образом:

1. Используйте базовый образ, например, Ubuntu 20.04.
2. Установите количество CPU и объем оперативной памяти, которые вы считаете подходящими.
3. Пропишите проксирование портов, чтобы вы могли получить доступ к виртуальной машине извне.
4. Запуск виртуальной машины: Запустите виртуальную машину с помощью команды vagrant up.

5. SSH-подключение: Подключитесь к виртуальной машине через SSH с помощью команды vagrant ssh. Убедитесь, что вы можете успешно войти в виртуальную машину.
6. Выключение и уничтожение: Выполните команды vagrant halt для выключения виртуальной машины и vagrant destroy для удаления ее.
7. Логирование: Ведите лог своих действий в файле, используя команды и результаты выполнения. Это поможет вам отслеживать, как вы выполняли каждый этап задачи.

Отчет: Подготовьте краткий отчет, в котором описаны все действия, выполненные вами при создании и управлении виртуальной машиной. Укажите, какие команды использовались и какие результаты были получены. Также укажите, если возникли какие-либо проблемы и как вы их решали.

Задача 2.2: Создание многокомпонентного окружения с использованием Vagrant и VirtualBox

После успешного овладения основами Vagrant и VirtualBox, вашей задачей будет создать многокомпонентное виртуальное окружение для тестирования и разработки. Это окружение будет включать в себя несколько виртуальных машин с разными конфигурациями.

Инициализация проекта: Создайте новую директорию для вашего многокомпонентного окружения. В этой директории выполните команду vagrant init, чтобы создать файл Vagrantfile.
Создание нескольких виртуальных машин: В файле Vagrantfile определите несколько виртуальных машин с разными характеристиками (разное количество CPU, объем оперативной памяти и др.). Каждая из них должна использовать базовый образ вашей операционной системы по выбору.
Конфигурация сети: Настройте сеть так, чтобы ваши виртуальные машины могли взаимодействовать друг с другом внутри виртуальной сети, а также имели доступ к интернету через NAT.
Скрипты установки: Создайте скрипты установки (Bash-скрипты) для каждой виртуальной машины, которые будут выполняться при их создании. Скрипты должны устанавливать и настраивать необходимое программное обеспечение или компоненты.
Запуск и настройка: Запустите все виртуальные машины с помощью команды vagrant up. Убедитесь, что они корректно настроены и могут взаимодействовать друг с другом.
Взаимодействие и тестирование: Войдите в каждую виртуальную машину через SSH с помощью vagrant ssh и проверьте их работоспособность. Попробуйте взаимодействовать между машинами, например, пинговать друг друга.
Логирование и документация: Ведите лог своих действий и подготовьте документацию, описывающую создание и настройку каждой виртуальной машины. Укажите, какие проблемы возникли, и как вы их решали.

Отчет: Подготовьте подробный отчет о создании многокомпонентного окружения. В отчете укажите все действия, которые вы выполнили, и описания результатов. Также предоставьте документацию и логи.

Задача 2.3: Автоматизация конфигурации и администрирование виртуальной машины с помощью Ansible

В этой задаче вам предстоит создать виртуальную машину с использованием Vagrant и VirtualBox, а затем автоматизировать её конфигурацию и выполнить несколько задач по администрированию Linux с помощью Ansible.

Инициализация проекта: Создайте новую директорию для вашего проекта. В этой директории выполните команду vagrant init, чтобы создать файл Vagrantfile.
Создание виртуальной машины: В файле Vagrantfile определите виртуальную машину с базовым образом Ubuntu 20.04 (или другой по вашему выбору). Укажите параметры CPU, RAM и другие настройки виртуальной машины.
Запуск виртуальной машины: Запустите виртуальную машину с помощью команды vagrant up.
Настройка Ansible: Установите Ansible на вашем локальном компьютере, если он ещё не установлен. Создайте файл ansible.cfg и настройте его для использования виртуальной машины.
Написание Ansible-плейбука: Создайте Ansible-плейбук для конфигурации вашей виртуальной машины. Плейбук должен включать в себя следующие задачи:

- Обновление пакетов на виртуальной машине.
- Установка и настройка веб-сервера (например, Apache или Nginx).
- Создание пользовательского аккаунта и установка SSH-ключа для аутентификации.
- Применение плейбука: Запустите Ansible-плейбук для конфигурации виртуальной машины. Убедитесь, что все задачи выполнились успешно.

Задачи по администрированию Linux: Добавьте следующие задачи в ваш Ansible-плейбук:

- Создание текстового файла с временем его создания.
- Установка дополнительного программного обеспечения (например nginx, apache2, etc).
- Создание расписания с использованием cron для выполнения задачи (например, регулярной очистки временных файлов).
- Логирование: Ведите лог своих действий и операций, выполненных с использованием Ansible.

Отчет: Подготовьте отчет, описывающий все действия, выполненные вами при создании и настройке виртуальной машины с использованием Ansible. Укажите, какие команды и плейбуки использовались, и какие результаты были получены. Также укажите, если возникли какие-либо проблемы и как вы их решали.

Контейнеры. Работа с Docker

Задача 2.4: Создание и запуск контейнера с веб-приложением в Docker

Цель этой задачи - ознакомиться с Docker, его основными командами и создать контейнер с простым веб-приложением.

Установка Docker: Если у вас ещё не установлен Docker, установите его на вашем компьютере.
Создание веб-приложения: Создайте простое веб-приложение (например, веб-страницу с "Hello, Docker!") или используйте какой-либо готовый образ веб-приложения из Docker Hub. * можно найти и использовать готовое приложение. Восполуйтесь поиском по github или другим поисковиком.
Создание Dockerfile: Создайте файл с именем Dockerfile в директории вашего веб-приложения. Dockerfile содержит инструкции по созданию образа Docker. Ваш Dockerfile должен включать следующие шаги:

- Использовать базовый образ, например, ubuntu:20.04.
- Установить необходимые компоненты и зависимости для запуска вашего веб-приложения.
- Скопировать файлы вашего веб-приложения в контейнер.
- Указать команду для запуска вашего веб-приложения (например, python app.py или npm start).
- Сборка Docker-образа: Используйте команду docker build для сборки Docker-образа на основе Dockerfile.

Запуск контейнера: Используйте команду docker run для запуска контейнера на основе созданного образа. Убедитесь, что ваше веб-приложение доступно в браузере.
Логирование: Ведите лог своих действий и операций, выполненных с использованием Docker.

Отчет: Подготовьте краткий отчет, в котором описаны все действия, выполненные вами с Docker, включая команды и результаты. Укажите, если возникли какие-либо проблемы и как вы их решали.

Задача: Создание многоконтейнерного приложения с использованием Docker и Docker Compose

Цель этой задачи - создать многоконтейнерное приложение с помощью Docker и Docker Compose, а также научиться управлять его контейнерами.

Задача 2.5:

Установка Docker и Docker Compose: Если у вас ещё не установлены Docker и Docker Compose, установите их на вашем компьютере.
Создание многоконтейнерного приложения: Создайте простое многоконтейнерное приложение, состоящее из двух контейнеров: веб-приложение и база данных. Вы можете использовать готовые образы (например, nginx и postgres), или создать собственные Dockerfile для каждого контейнера.
Настройка Docker Compose: Создайте файл docker-compose.yml, в котором опишите конфигурацию вашего многоконтейнерного приложения. Укажите сервисы (контейнеры),

их зависимости, порты, и другие параметры. Вот пример структуры docker-compose.yml файла:

```
yaml
Copy code
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./app:/usr/share/nginx/html
  db:
    image: postgres:latest
    env_file:
      - web-variables.env
```

Запуск приложения: Используйте команду docker-compose up для запуска вашего многоконтейнерного приложения. Убедитесь, что все контейнеры успешно запущены и могут взаимодействовать друг с другом.
Проверка веб-приложения: В откройте веб-браузере страницу вашего веб-приложения, которое работает в контейнере. Убедитесь, что оно доступно и функционирует корректно.
Логирование: Ведите лог своих действий и операций, выполненных с использованием Docker и Docker Compose.
Управление контейнерами: Остановите и удалите контейнеры с помощью команды docker-compose down. Затем снова запустите их с помощью docker-compose up.
Отчет: Подготовьте отчет, описывающий все действия, выполненные вами при создании и управлении многоконтейнерным приложением с использованием Docker и Docker Compose. Укажите, какие команды и конфигурации использовались, и какие результаты были получены. Также укажите, если возникли какие-либо проблемы и как вы их решали.

Задача 2.6: Знакомство с Kubernetes и Minikube

Цель этой задачи - ознакомиться с Kubernetes и Minikube, создать мини-кластер Kubernetes и выполнить несколько базовых операций с контейнерами.

Установка Minikube и kubectl: Если еще не установлены Minikube и kubectl, они должны быть установлены их на рабочих станциях.
Запуск Minikube: Создать и запустить локальный мини-кластер Kubernetes с помощью команды `minikube start`.
Проверка состояния кластера: Использовать команду kubectl cluster-info для проверки состояния кластера Kubernetes и доступности API сервера.
Создание и запуск пода: Создать файл манифеста для пода (например, my-pod.yaml) и запустить его в кластере с помощью kubectl create -f my-pod.yaml. Под должен содержать простое веб-приложение, которое будет доступно через сервис Kubernetes.
Проверка пода и сервиса: Использовать команды `kubectl get pods` и `kubectl`

`get services` для проверки состояния своего пода и сервиса. Они также могут использовать команду `kubectl describe pod <имя пода>` для получения дополнительной информации о поде.
Журналы и отладка: Использовать команду `kubectl logs <имя пода>` для просмотра журналов пода и команду `kubectl exec -it <имя пода> -- /bin/bash` для отладки внутри контейнера пода.
Остановка и удаление ресурсов: Остановить и удалить свой под и сервис с помощью команд `kubectl delete pod <имя пода>` и `kubectl delete service <имя сервиса>`
Остановка Minikube: По окончании задачи, остановить локальный мини-кластер Kubernetes с помощью minikube stop.
Отчет: Подготовить отчет, описывающий все действия, выполненные ими при знакомстве с Kubernetes и Minikube, включая команды и результаты. Отчет также должен включать в себя выводы и впечатления от работы с Kubernetes.

3. Jenkins/TeamCity

В данном разделе рассматриваются инструменты для continuous integration на примере TeamCity или Jenkins. Они примерно в одинаковой степени используются на проектах и почти не отличаются друг от друга по своему назначению. Использовать можно любой инструмент. В зависимости от ментора на лекциях и практических занятиях может рассматриваться какой-то конкретный инструмент.

Материалы и литература

Установка и настройка

Jenkins:

- https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04
- https://www.digitalocean.com/community/tutorials/how-to-set-up-jenkins-for-continuous-development-integration-on-centos-7
- https://www.digitalocean.com/community/tutorials/how-to-set-up-continuous-integration-pipelines-in-jenkins-on-ubuntu-20-04

TeamCity:

- https://www.jetbrains.com/help/teamcity/installing-and-configuring-the-teamcity-server.html
- https://www.tutorialspoint.com/continuous_integration/continuous_integration_creating_project_teamcity.htm
- https://www.jetbrains.com/ru-ru/teamcity/documentation/
- https://www.jetbrains.com/help/teamcity/getting-started-with-teamcity.html#3.+Run+your+First+Build

Практическое задание:

1. Установить Jenkins или Teamcity server. Это может быть установка на ваш локальный компьютер или на инстансе в облаке, это не имеет значение, как не имеет значение и метод уставки (с использованием docker контейнера, playbook или установка вручную из репзитория и пр.).
2. Создать новый проект “Staging”, в нем добавить задачу для сборки простого приложения, например
   a. .net: https://github.com/chaitalidey6/HelloWorldAspNetCore/tree/master/HelloWorldAspNetCore
   b. Java: https://github.com/jenkins-docs/simple-java-maven-app
   c. Node JS: https://github.com/jenkins-docs/simple-node-js-react-npm-app

   Замечания:
   - Вы можете использовать любое привычное приложение на любом языке (.net, java, js, python, php).
   - Код приложения должен быть размещен в вашем собственном git-репозитории.
   - Должна использоваться ветка “staging”.
   - Приложение может быть собрано в контейнере (предпочтительный способ).
   - Задача по сборке должна запускаться с параметрами.
   - Результатом сборки обязательно должен быть артифакт (архив, docker-контейнер), который вы дальше будете использовать.
   - Необходимо самостоятельно подумать над тем, каким образом Jenkins/TeamCity получит доступ к git-репозиторию, при этом необходимо придумать наиболее безопасный на ваш взгляд способ.
3. Создать задачу в Jenkins /Teamcity для деплоя вашего артифакта на сервер и перезапуск приложения.
   Замечания:
   - Здесь артефакт может доставляться на удаленный сервер (например, на EC2 инстанс в AWS), либо на контейнер (при работе локально в Docker), либо на локальный сервер (при работе с Vagrant/VirtualBox).
   - Необходимо самостоятельно подумать над тем, каким образом будет организован доступ из Jenkins/Teamcity на сервер (дря загрузки артефактов), при этом необходимо придумать наиболее безопасный на ваш взгляд способ.
4. Настроить зависимость задачи деплоя от задачи сборки.
5. Настроить деплой артефакта в место, где он будет работать и запуск приложения.
6. Добавить задачу создания бэкапа артефактов на сервере.
7. Настроить пайплайн, где должны быть включены шаги: сборка, бэкап и деплой (опционально: тестирование).
8. Настроить автоматический запуск деплоя при добавлении нового commit’а в ветке “staging” git.
   ** При запуске локально – здесь могут быть проблемы с настройкой webhook, потому используйте другой метод взаимодействия с git.*

9. Создать новый проект “Production”, добавить задачу для сборки приложения, выполнить те же настройки, что и в Staging (п. 2), но с небольшими изменениями: должна использоваться ветка “master”.
10. Создать задачу для деплоя Production артефактов на сервер (здесь может использоваться тот же сервер, но приложения должны быть различными: «висеть» на разных портах или под разными доменами).
11. Настроить зависимость задачи деплоя от задачи сборки.
12. Настроить автоматический запуск деплоя при подтверждении pull request’а в ветке “master” в git.

4. Ansible

Материалы и литература

- установка Ansible:

  https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-an-ubuntu-18-04-vps

  https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-18-04

- работа с playbooks

  https://www.digitalocean.com/community/tutorials/how-to-create-ansible-playbooks-to-automate-system-configuration-on-ubuntu

- настройка

  https://www.digitalocean.com/community/tutorials/how-to-configure-apache-using-ansible-on-ubuntu-14-04

  https://www.digitalocean.com/community/tutorials/how-to-deploy-a-basic-php-application-using-ansible-on-ubuntu-14-04

Замечание к практическому заданию

Практические задания по Ansible (раздел 3 и далее) не связаны с заданиями Vagrant/Docker-Jenkins (разделы 1 и 2). В случае досрочного завершения выполнения заданий из раздела 1 – можно перейти к разделу 3 или 4.

Практическое задание

4.1 Создать 3 репозитория в github или bitbucket с названием ab-haproxy, ab-logstash, ab-webui (все дальнейшие действия подразумевают выгрузку результатов в эти репозитории).
Важно: в решении необходимо использовать playbooks, а не ссылаться на репозиторий Galaxy.

4.2 Создать в Vagrant виртуальную машину Ubuntu 18.04

4.3 ab-haproxy

4.3.1 Создать следующие роли в Ansible (можно пользоваться репозиторием Galaxy):

- apt (добавление необходимых пакетов, обновление из репозиториев установленных по умолчанию пакетов)
- ntp (обновить время, настроить синхронизацию времени по cron 1 раз в сутки с любого общедоступного сервера времени)
- monit (установить и настроить monit для само-мониторинга виртуальной машины, правила можно составить любые, например: перезапуск haproxy)
- haproxy (можно адаптировать роли из https://galaxy.ansible.com/incubateurpe/haproxy, haproxy будет использоваться в качестве балансировщика для веб-иинтерфейса 4.5.)

4.3.2 На выходе Ansible так же должен быть сформирован ini-файл, содержащий любой уникальный ID виртуалки, например Packer ID, следующего формата:

```
[general]
uniqueID=
```

4.4 ab-logstash

4.4.1 Создать следующие роли в Ansible (можно пользоваться репозиторием Galaxy):

- apt (добавление необходимых пакетов, обновление из репозиториев установленных по умолчанию пакетов)
- ntp (обновить время, настроить синхронизацию времени по cron 1 раз в сутки с любого общедоступного сервера времени)
- monit (установить и настроить monit для само-мониторинга виртуальной машины)
- java (https://galaxy.ansible.com/andrewrothstein/openjdk для установки openjdk-8-jdk)
- logstash (https://www.elastic.co/downloads/logstash без локального syslog и web интерфейса)
- elasticsearch (https://www.elastic.co/downloads/elasticsearch)
- На выходе Ansible также должен быть сформирован ini-файл, содержащий любой уникальный ID виртуалки, например Packer ID

4.5 ab-webui

4.5.1 Создать следующие роли в Ansible (можно пользоваться репозиторием Galaxy):

- apt (добавление необходимых пакетов, обновление из репозиториев установленных по умолчанию пакетов)
- ntp (обнвоить время, настроить синхронизацию времени по cron 1 раз в сутки с любого общедоступного сервера времени)
- monit (установить и настроить monit для само-мониторинга виртуальной машины)
- rsyslog (для трансляции любых стандартных системных логов)
- kibana
- nginx (настроить reverse proxy на kibana)

4.5.2 На выходе Ansible так же должен быть сформирован ini-файл, содержащий любой уникальный ID виртуалки, например Packer ID

4.6 Настроить конфигурации следующим образом:

- logstash должен собирать логи с rsyslog на ВМ WebUI
- kibana должен предоставлять интерфейс для просмотра логов из logstash
- доступ к kibana должен быть возможен через ВМ haproxy

5. Дополнительные практические задания

Материалы и литература

- Nginx
    - Кеширование с Nginx
    - Использование if в Nginx
    - Пример настройки кеширования и изменения URI
    - ngx_http_log_module
- Apache
    - Кеширование в Apache
    - Mod_log_config
- Troubleshooting в Linux
    - ps
    - top htop
    - lsof
    - df
    - strace
    - log файлы
- Видеоматериалы:
    - 10 Advanced Linux Troubleshooting Tips
- Оптимизация в PostgreSQL
    - SQL-Vacuum
- Дисковые разделы в Linux
    - Файловые системы
    - LVM
- Server Version Disclosure
    - Fingerprint Web Server
    - Apache
    - Nginx
    - IIS

PHP

- Kubernetes

Minikube

Helm

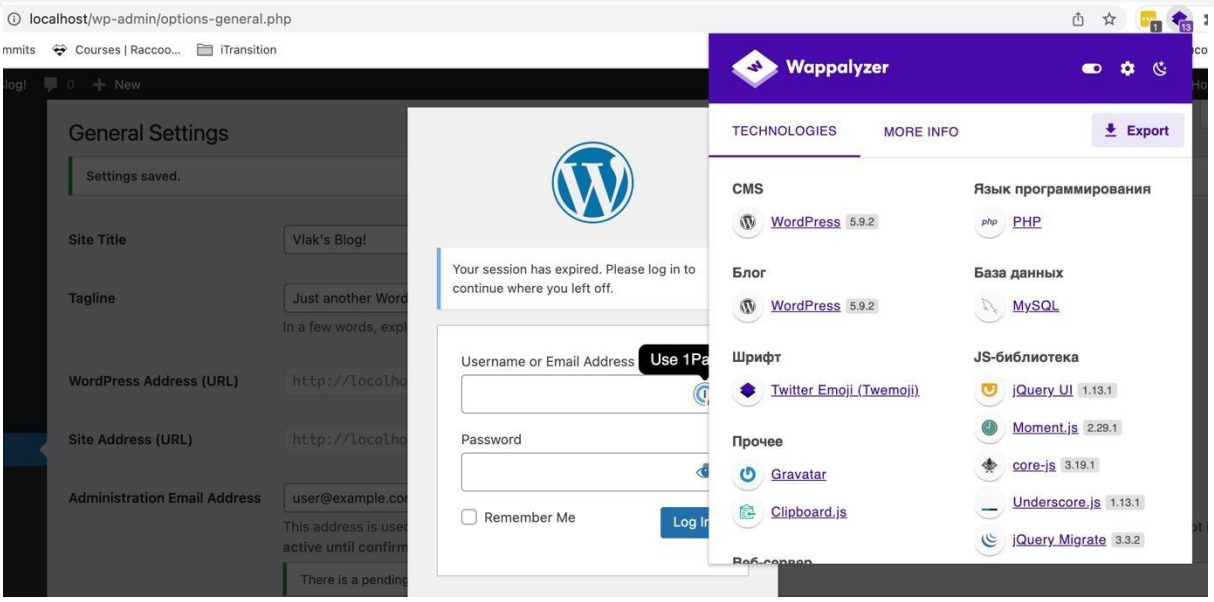
Практические задания

Замечание к практическому заданию

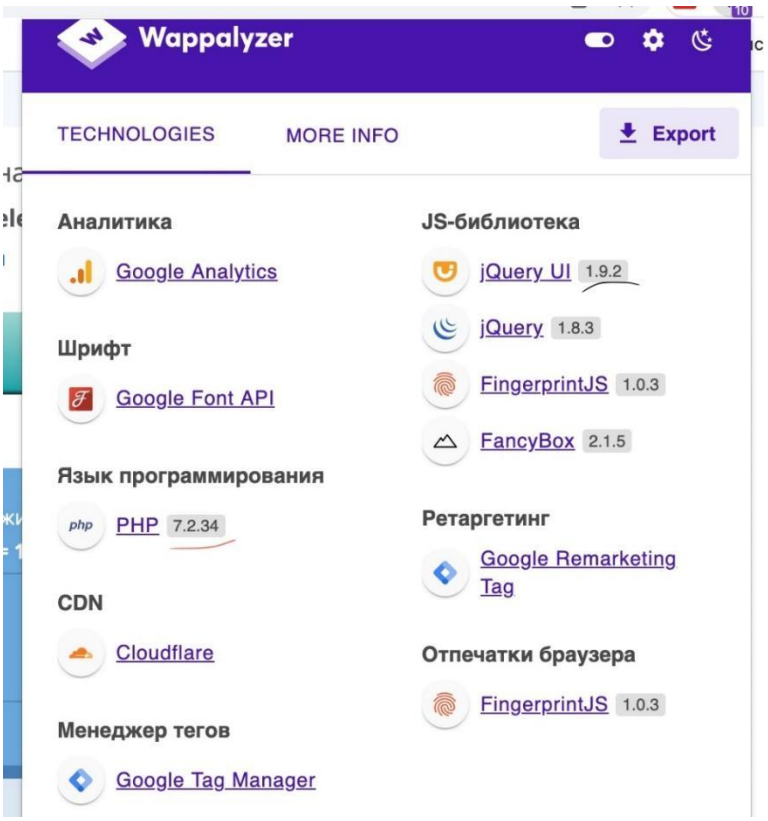
Перед выполнением заданий необходимо ознакомиться с документацией. Практически все задания можно выполнить более чем одним способом, любой рабочий вариант будет верным. Задания можно выполнять в виртуальных машинах, используя VirtualBox или VMWare, также можно использовать облачных провайдеров.

Категория Nginx/Apache

1. Создать виртуальную машину, установить nginx. Настроить в nginx кэш по условиям: кэшировать только те запросы, где есть utm_source=
2. Создать виртуальную машину, установить apache web server. Настроить в apache кэш по условиям: кэшировать только те запросы, где есть utm_source=. Можно использовать mod_lua, mod_cache, mod_disk_cache, mod_alias и другие модули.
3. Изменить конфигурации apache и nginx: запустить их на порту 8080. Установить дополнительный nginx (можно virtualhost, в случае, когда nginx будет бэком) на порту 80 и настроить на нем reverse proxy. Настройка логгирования на бэк веб серверах, чтобы отображались x-forwarded-for, server hostname в логах серверов.
4. Добавить в cron задачу, которая будет находить и удалять файлы в папке /tmp, с даты создания которых прошло больше 14 дней и размер которых больше 5 Мб.
5. Установить Minikube, Helm. Установить bitnami/wordpress helm chart. При помощи wappalyzer посмотреть, какие технологии используются. Сделать скриншот вывода и залить его в репозиторий. Скриншот должен выглядеть примерно так:

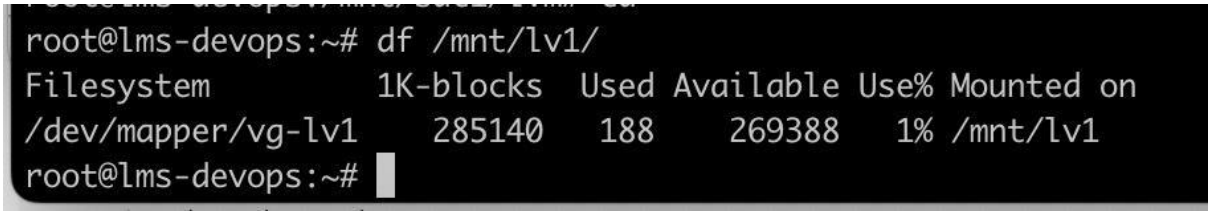


6. Скрыть версии apache, nginx, php на серверах, используемых для выполнения заданий. Поставить расширение Wappalyzer для Chrome, проверить информацию по версиям. Сделать скриншоты до скрытия версий и после. Залить их в репозиторий или ClickUp. Пример скриншота:



Категория Linux

7. Создать 4 файла размером 1Гб каждый, создать loopback устройства из файлов при помощи losetup. Создать физические разделы на этих устройствах при помощи pvcreate. Создать volume group из первых двух девайсов. На ней создать logical volume при помощи lvcreate. Создать файловую систему при помощи mkfs.ext3, подмонтировать её, посмотреть какой размер. Добавить оставшиеся два устройства в группу. Изменить размер логического тома, затем размер файловой системы. Проверить размер при помощи df. В качестве результата сделать два скриншота команд. Пример скриншота:



Категория Jenkins

1. Создать в AWS два бесплатных инстанса – t2.micro с Ubuntu.
2. Организовать доступ к ним через белый список IP (security group, SSH)
3. Установить Jenkins на один из инстансов, не забыв открыть порт 8080 (по умолчанию) для своего IP.
4. Между инстансами установить SSH соединение. Для этого необходимо прокинуть public key пользователя Jenkins (Jenkins instance) в доверенные ключи пользователя ubuntu (Staging instance).
5. Установить докер в Staging instance. Добавить пользователя ubuntu в группу докера (sudo usermod -aG docker ubuntu и перелогин).
6. Запустить docker run hello-world в Staging instance.
7. Создать репозиторий в GitHub. Положить в него Jenkinsfile.
8. В Jenkinsfile указать stages со скриптом проброса туннеля docker socket (https://medium.com/@dperny/forwarding-the-docker-socket-over-ssh-e6567cfab160) для управления контейнерами в Staging.

6. Amazon

Материалы и литература

1. https://aws.amazon.com/training/intro_series/ - вводные демо-видео по основным сервисам.
2. https://qwiklabs.com/ - Amazon рекомендует этот источник как официальный сервис для самообучения.
3. AWS Well-Architected Framework – общее описание сервисов Amazon и для чего их следует использовать.
4. AWS Best Practices: five key approaches to get you started – Рекомендации по использованию ключей
5. https://aws.amazon.com/products/security/
6. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html
7. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html
8. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs.html
9. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_Ruby.html
10. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html
11. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_GetStarted_EC2.html
12. FAQ разделы для каждого конкретного сервиса

Практические задания

Часть 1 - Основы AWS

1. Создать выделенную сеть Amazon Virtual Private Cloud с тремя подсетями, как минимум в двух разных зонах (например, обычно обозначаются как us-west-1a, us-west-1c). Две подсети (расположенные в разных зонах) должны быть публичными. Третья подсеть – приватная, ограничить доступ на ACL.
2. Создать Security Group (назовем ее web-sg):
   - Разрешить входящий SSH трафик только со своего IP (или доверенных IPs).
   - Разрешить входящий HTTP/HTTPS трафик со своего IP (или доверенных IPs).
   - Разрешить весь исходящий трафик во все 3 подсети.
   - Остальной трафик запретить.
3. Сгенерировать собственный RSA ключ (Key Pairs) для использования в дальнейшем при создании инстансов (необходимо для подключения по SSH).
4. EC2:
   a. Создать один t2.micro инстанс в созданной в п 6.1 VPC и одной из публичных подсетей. Использовать Security Group из п 6.1.1.
   b. Создать второй t2.micro инстанс в созданной в п 6.1 VPC и второй публичной подсети. Использовать Security Group из п 6.1.1.
   c. На оба инстанса установить Nginx и создать простую страницу-заглушку (index.html) на 80-ом порту.
5. ELB:
   a. Создать один ELB с поддержкой созданных Availability Zones.
   b. Разрешить HTTP трафик на ELB с любого IP адреса.
   c. Добавить в ELB оба инстанса.
   d. Настроить Health Check на протокол HTTP, порт 80, страница index.html с минимальными интервалами проверки.
   e. Обновить security group созданную в пункте 2., так чтобы доступ по http/https был возможен только с ELB.

f. Принудительно остановить веб-сервер на одном из инстансов и проверить доступность сайта.

6. RDS:
   a. Создать инстанс PostgreSQL в выделенной VPC и приватной подсети с типом хранилища как General Purpose и объёмом в 20 Гб. Использовать Security Group из п 6.1.2.
   b. Разрешить входящий трафик только от web-sg. Как результат должны продемонстрировать возможность подключения к RDS как минимум с двух исходных точек (серверов)
7. ElastiCache:
   a. Создать один инстанс ElastiCache (Redis) в выделенной VPC.
   b. Разрешить трафик только внутри выделенной VPC. Как результат: должны продемонстрировать возможность подключения к Redis как минимум с двух исходных точек (серверов)
   c. Создать один инстанс ElastiCache (Memcached) в выделенной VPC.
   d. Разрешить трафик только от серверов созданных в пункте 5.3. Как результат: должны продемонстрировать возможность подключения к Memcached как минимум с двух исходных точек (серверов)
8. Создать CloudFront Distribution с параметрами по умолчанию.
   a. Сгенерировать 100 небольших файлов (< 512 Kb) и заполнить ими созданный бакет в S3. К файлам сторонние лица не должны иметь доступ.
   b. Настроить политику хранения объектов в данном бакете S3 следующим образом:
   - по истечении 30 дней – отправлять объекты в Glacier;
   - после 6 месяцев хранения – полностью удалять с Glacier.
9. *На одном из серверов созданных в пункте 6.3. подготовьте скрипт (с использование aws cli) для загрузки/выгрузки/удаления файлов в S3 бакете созданном в пункте 6.7.1 в соответствии с best practice и наибольшим уровнем безопасности окружения.
10. *Заменить инстансы (ELB должно будет ссылаться на новые инстансы созданные через autoscaling group) созданные в пункте 6.3 на autoscalinng group, со следующими правилами:
    - Если CPU Utilization > 70%, то добавить инстанс
    - Если CPU Utilization < 15%, то удалить инстанс
    - Минимальное количество запущенных инстансов = 1
    - Максимальное количество запущенных инстансов = 4
    - Autoscaling group должна разворачивать инстанс вместе с установленным nginx и страницей заглушкой из пункта 6.3

Часть 2 — Работа с контейнерами

11. Установите и настройте EB CLI
    a. Создайте приложение и окружение, используя Dockerrun.aws.json v2 создайте 2 контейнера:
    - Nginx
    - Rails (можно использовать приложение на другом языке программирования, который поддерживает EB, но следующие подпункты обязательны)
    b. Установите на rails порт 3000 и настройте nginx на фронт

c. Создайте простое Rails приложение выводящее строку "Hello world"
d. Проверьте используя URL: <имя_приложения>.<регион>.elasticbeanstalk.com работу beanstalk, вы должны увидеть содержимое index.html

Часть 3 — Работа с контейнерами

1. Создать в AWS два бесплатных инстанса – t2.micro с Ubuntu.
2. Организовать доступ к ним через белый список IP (security group, SSH)
3. Установить Jenkins на один из инстансов, не забыв открыть порт 8080 (по умолчанию) для своего IP.
4. Между инстансами установить SSH соединение. Для этого необходимо добавить public key пользователя Jenkins (Jenkins instance) в доверенные ключи пользователя ubuntu (Staging instance).
5. Установить Docker на Staging instance. Добавить пользователя ubuntu в группу Docker (sudo usermod -aG docker ubuntu).
6. Запустить docker run hello-world в Staging instance.
7. Создать репозиторий в GitHub. Положить в него Jenkinsfile.
8. В Jenkinsfile указать stages со скриптом проброса туннеля docker socket (https://medium.com/@dperny/forwarding-the-docker-socket-over-ssh-e6567cfab160) для управления контейнерами в Staging.

```
pipeline {
    agent any

    environment {
        DOCKER_HOST="unix://\$(pwd)/docker.sock"
        STAGE_INSTANCE="ubuntu@aws-dns"
    }

    stages {
        stage('Setup SSH tunnel') {
            steps {
                script {
                    sh "ssh -nNT -L
\$(pwd)/docker.sock:/var/run/docker.sock ${STAGE_INSTANCE} & echo \$! >
/tmp/tunnel.pid"
                    // sometimes it's not enough time to make a tunnel, add
sleep
                    sleep 5
                }
            }
        }

        stage('Deploy') {
            steps {
                script {
                    sh "DOCKER_HOST=${DOCKER_HOST} docker ps -a"
                }
            }
        }
    }
```

```
    post {
        always {
            script {
                sh "pkill -F /tmp/tunnel.pid"
            }
        }
    }
}
```

Примечания:
Команда sleep 5, как видно из комментария, нужна для задержки следующего шага на 5 секунд, т.к. открытие туннеля происходит не моментально и может занять некоторое время.
Шаг post необходим для удаления туннеля.

9. Создать в Jenkins New Item –> Pipeline. В разделе Pipeline->SCM выбираем Git, вводим адрес репозитория (как в clone).
   Заодно увидим ошибку Failed to connect to repository, значит нужно положить содержимое публичного ключа Jenkins instance в ssh ключи Github. Если всё произведено корректно, то ошибка пропадёт.
10. В разделе Script Path вводим путь к Jenkinsfile. Сохраняем настройки.
11. Нажимаем Build Now - должна запуститься задача. Заходим в задачу, а затем в Console output. Если всё прошло хорошо, то в логах мы должны увидеть результат выполнения команды docker ps -a на Staging instance.

7. CloudFormation and Terraform. IaaC (Infrastructure as a Code).

Материалы и литература

1. Best Practices for Creating Amazon CloudFormation Templates – рекомендации по использованию CloudFormation
2. AWS CloudFormation (UserGuide) — официальная документация по AWS CF
3. Hashicorp Terraform - Getting started-AWS
4. Terraform Registry

Практические задания:

1. Подготовить шаблон CloudFormation используя Nested Stacks для автоматизации создания ресурсов, знакомство с которыми было в части «Основы AWS».
2. Подготовить шаблон Terraform для автоматизации создания ресурсов, знакомство с которыми было в части «Основы AWS».

Задание на отлично: Создать все ресурсы, включая автоматизацию создания сайта из первой части.

Создание CloudFront и S3 можно не использовать в связи с продолжительным временем создания Cloudfront Distribution.

8. Azure

Материалы и литература

Материалы и литература будут выданы на практическом занятии.

Практические задания

1. Вам необходимо развернуть две виртуальные машины Azure с именами VMLU01 и VMLU02 на основе образа Ubuntu. Развертывание должно соответствовать следующим требованиям: Обеспечение SLA 99,95%.Использование managed disks.
2. Вы планируете забэкапить файлы и документы с on-premise Windows file server в хранилище Azure. Бэкап файлы будут храниться в виде блобов.
   Вам необходимо создать storage account с именем CorpStorage01. Решение должно соответствовать следующим требованиям:
   - Убедитесь, что документы доступны через drive mapping с виртуальных машин Azure под управлением Windows Server.
   - Обеспечьте максимально возможное redundancy документов.
   - Минимизируйте затраты на storage account.

3. Вы планируете развернуть Application Gateway с именем AppGw01 для балансировки нагрузки внутреннего IP-трафика на виртуальные машины Azure, подключенных к subnet0.
   Вам необходимо настроить виртуальную сеть с именем VNET01 для поддержки Application Gateway.
4. Вы планируете разместить несколько защищенных веб-сайтов на Web01. Вам необходимо разрешить HTTPS через TCP-порт 443 на Web01 и запретить HTTP через TCP-порт 80 на Web01.
5. Вам нужно создать веб-приложение с именем WabApp01, которое можно горизонтально скалировать. Решение должно использовать самый низкий возможный ценовой уровень app Service Plan.

9. Раздел со звездочкой: сборка сайтов в AWS

Условия:

1. Задачи должны выполняться на Centos или Amazon Linux
2. Необходимо разобраться с установкой Apache + PHP 7.3 + MySQL + NPM, если получится - автоматизировать этот процесс (достаточно все оформить в bash, ansible)
3. Зарегистрировать бесплатную учетную запись в AWS
4. Разобраться со стоимостью сервисов в AWS и все дальнейшие действия делать только на бесплатном окружении!
5. Создать новый Инстанс, автоматически установить на нем софт из п.2, сохранить как AMI и удалить.
6. Используя Cloudformation или Terraform:
    a. Запустить инстанс, с предустановленным ПО из AMI из п.5.
    b. Создать Application load balancer, добавить маршрутизацию на сервер из п.5
    c. Входные параметры Cloudformation/Terraform:
        i. Размер инстанса (выпадающий список)
        ii. SSH Ключ
        iii. Использование публичного IP адреса (true/false)
        iv. VPC, где будет размещен инстанс
        v. Имя инстанса, которое будет задано как тег Name.
    d. Выходные параметры:
        i. Публичный IP инстанса
        ii. DNS, полученный в Load Balancer
7. На сервере сделать два сайта (через виртуальную директорию, или через хосты - на усмотрение)
    a. 1й сайт: установить самый Wordpress без дополнительной конфигурации
    b. 2й сайт: установить статический сайт https://github.com/gatsbyjs/gatsby-starter-hello-world

    Оба сайта должны работать через балансировщик

В качестве успешно выполненного задания необходимо показать:

a. Скрипт по конфигурированию сервере (установке необходимого софта из п.2) – bash или playbook
b. Шаблон cloudformation для разворачивания окружения
c. Сайт на WP
d. Сайт на gatsby

10 Мониторинг

Мониторинг в DevOps: Основные Концепции и Цели

Мониторинг - это процесс сбора, анализа и визуализации данных о состоянии систем и приложений. Он играет ключевую роль в методологии DevOps, обеспечивая непрерывное отслеживание и управление системами. Мониторинг призван решать следующие проблемы:

- Раннее Обнаружение Проблем: Мониторинг позволяет выявлять проблемы и сбои в системе на ранних этапах, до того как они начнут сильно влиять на бизнес-процессы.
- Улучшение Производительности: Путем анализа метрик производительности, можно оптимизировать работу системы, устранить узкие места и повысить эффективность.
- Планирование Ресурсов: Мониторинг предоставляет информацию о потреблении ресурсов, что помогает в планировании масштабирования и выделения ресурсов.
- Соблюдение Соглашений об Уровне Обслуживания (SLA): Мониторинг помогает отслеживать выполнение SLA и своевременно реагировать на их нарушения.

Стратегии Мониторинга:

- Мониторинг Инфраструктуры: Отслеживание состояния серверов, сетей и хранилищ данных.
- Мониторинг Приложений: Отслеживание работоспособности приложений, включая производительность и ошибки.
- Мониторинг Логов: Сбор и анализ журналов для выявления проблем и аудита.
- Мониторинг Производительности: Измерение и анализ метрик производительности.
- Мониторинг Безопасности: Отслеживание потенциальных угроз и нарушений в безопасности.

Современные Инструменты Мониторинга:

- Prometheus: Открытая система мониторинга и алертинга с поддержкой множества языков и интеграцией с Kubernetes.
- Grafana: Платформа визуализации данных, которая часто используется с Prometheus для создания графиков и дашбордов.
- ELK Stack (Elasticsearch, Logstash, Kibana): Инструменты для сбора, хранения, анализа и визуализации логов.
- Zabbix: Классический инструмент для мониторинга инфраструктуры и приложений.
- Datadog: Облачный сервис для мониторинга и анализа данных о производительности и безопасности.
- NewRelic: Облачный сервис для мониторинга и анализа данных о производительности и безопасности.

Задача 1: Установка и Настройка Prometheus и Grafana

Цель: Установить и настроить Prometheus и Grafana в кластере Minikube.

- Установите Prometheus в кластере Minikube, используя Helm Chart.
- Установите Grafana также с использованием Helm Chart.
- Настройте Prometheus для мониторинга самого себя (self-monitoring).
- Создайте источник данных (data source) Prometheus в Grafana.

Задача 2: Создание Дашборда в Grafana

Цель: Создать пользовательский дашборд в Grafana и добавить на него панели с данными из Prometheus.

- Войдите в интерфейс Grafana и создайте новый дашборд.
- Добавьте на дашборд несколько панелей, используя метрики из Prometheus. Например, можно добавить график с использованием метрики prometheus_http_requests_total.
- Настройте панели, чтобы они отображали интересующие вас метрики и временной интервал.

Задача 3: Создание Алертов в Prometheus и Интеграция с Grafana

Цель: Создать алерты в Prometheus и интегрировать их с Grafana для мониторинга состояния кластера Minikube.

- Создайте правило алерта в файле конфигурации Prometheus. Например, правило может отслеживать использование CPU или памяти.
- Настройте интеграцию алерта Prometheus с Grafana, чтобы алерты отображались в интерфейсе Grafana.
- Создайте панель в дашборде Grafana, которая будет отображать статус алертов.

Задача 4: Масштабирование Minikube и Мониторинг с Prometheus

Цель: Исследовать, как Minikube масштабирует приложения и как это влияет на мониторинг в Prometheus и Grafana.

- Увеличьте количество реплик какого-либо пода в вашем Minikube кластере.
- Наблюдайте, как изменения в масштабе влияют на метрики в Prometheus и их отображение в Grafana.
- Отобразите изменения на графиках и панелях в Grafana.

Terms of use