АННОТАЦИЯ

Цель выпускной квалификационной работы – исследование и создание кластерной инфраструктуры, которая позовлит разрабатывать и развертывать программные проекты множеством команд из нескольких человек.

В работе составлен обзор по архитектурам приложений и инфраструктур. Произведен анализ инструментов для создания инфраструктуры.

В ходе работы в рамках проекта был развернут кластер Kubernetes, его вспомогательные компоненты, настроено их взаимодействие.

Созданная инфраструктура внедрена в МИЭМ НИУ ВШЭ и происходит поэтапный процесс переноса проекта в нее.

В структуру работы входит введение, ??? глав, заключение, список использованных источников , 1 рис. и 4 источн.

Общий объем проекта составляет 24 стр.

ABSTRACT

blablabla

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

Ansible — blablabla.

Block input and output — обращение на считывание и запись с хранителей данных.

Central processing unit — процессор.

Control group — управление (контроль) групп.

Dashboard — blablabla.

DLL Hell — сбой, возникающий, когда одна часть программного обеспечения ведет себя не так, как ожидалось второй частью программного обеспечения, что в некотором роде зависит от действия первого [3].

Docker — программное обеспечение для автоматизации развертывания и управления приложениями в среде виртуализации на уровне операционной системы [4].

etcd — blablabla.

Helm — blablabla.

Image или образ контейнера — архив файловой системы, который используется для запуска контейнера из него.

Ingress Controller — blablabla.

kubeadm — blablabla.

kubelet — blablabla.

Kubernetes — blablabla.

kubespray — blablabla.

master — blablabla.

Memory — память.

Mount — монтирование, например дисков как директорий. Способ подключения различных хранителей данных в OS GNU/Linux.

Namespaces — пространства имен.

Networking или network — сеть.

nginx — blablabla.

NodeSelector — blablabla.

PersistentVolume — blablabla.

PersistentVolumeClaim — blablabla.

playbook — blablabla.

Pod — blablabla.

PodSecurityPolicy — blablabla.

Process ID — идентификатор процесса.

Resource management — управление и ограничение ресурсов машины.

roles — blablabla.

Service — blablabla.

Storage Provisioner — blablabla.

StorageClass — blablabla.

tasks — blablabla.

TLS — blablabla.

Traefik — blablabla.

User — пользователь.

volumes — blablabla.

Wildcard сертификат — blablabla.

worker — blablabla.

YAML — blablabla.

Виртуальная машина — программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы (target — целевая, или гостевая платформа) и исполняющая программы для target-платформы на host-платформе (host — хост-платформа, платформа-хозяин) или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы [1].

Гипервизор — программа или аппаратная схема, обеспечивающая или позволяющая одновременное, параллельное выполнение нескольких операционных систем на одном и том же хост-компьютере [2].

Инстанс — экземлпяр объекта.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

VM — virtual machine (виртуальная машина).

OS или ОС — операционная система.

ПО — программное обеспечение.

PID — Proccess ID.

Cgroup — control group.

CPU — central processing unit.

I/O — input and output.

K8s — Kubernetes.

IP — blablabla.

HTTP — blablabla.

HTTPS — blablabla.

DNS — blablabla.

NFS — blablabla.

SSD — blablabla.

HDD — blablabla.

ВВЕДЕНИЕ

В настоящее время существует спрос на реализацию решений по автоматизации процессов разработки и развертывания программных продуктов. Для этого используются такие средства, как:

— Система контроля версий (Version Control System, далее – VCS).

— Средства непрерывной интеграции и доставки артефактов (Continuous Integration, Continuous Delivery или Deployment, далее – CI/CD).

— Система контенеризации.

— Оркестрация контейнеров.

— Вспомогательные инструменты.

Цель работы – исследование и создание кластерной инфраструктуры на основе контейнеров.

СТАРАЯ ТЕРМИНОЛОГИЯ

Immutable - неизменный.

Self-healing - самолечение.

Declarative - декларативность.

SLA - доступность.

Daemon или **Демон** - процесс, запущенный в фоне.

CLI (Command line interface) - консольный интерфейс приложения.

FS (File system) - файловая система раздела.

Hash или **Хэш** - результат выполнения некоторой математической хэш-функции.

Tag или **Тэг** - некоторая отметка на чем-либо для более удобного поиска объектов.

Yaml (YAML Ain't Markup Language) -"дружественный"формат сериализации данных, концептуально близкий к языкам разметки.

SSD (Solid State Drive) или *Твердотельный диск* - вид накопителя, хранящего данные.

REST API - архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

Garbage Collector - "сборщик мусора".

QoS (Quality of Service) - органичения на что-либо, разграничающие по ролям.

Requested resources - запрошенные ресурсы.

Netwrok Policies - сетевые политики.

Iptables - обертка над netfilter.

Ipvs - обертка над netfilter.

OSI - сетевая модель стека сетевых протоколов.

VRRP (Virtual Router Redundancy Protocol) - сетевой протокол, предназначенный для увеличения доступности маршрутизаторов.

SSL/TLS (Secure Sockets Layer) (Transport Layer Security) - криптографические протоколы, обеспечивающие защищенную передачу данных между узлами в сети Интернет.

Hook - перехват чего-либо.

Git - одна из систем контроля версий.

HTTP (HyperText Transfer Protocol) - протокол прикладного уровня передачи данных.

Tar - архиватор.

GET Requests - метод *HTTP*.

1 Исследовательская часть работы

1.1 Виды архитектур инфраструктуры и приложений в ней

На текущий момент существует множество решений со стороны архитектуры инфраструктуры и, как следствие, приложений в ней. Для выбора конкретного решения под вышеуказанные **Указать цели выше** цели необходимо рассмотреть основные из них. Далее они будут описаны по хронологии появления.

а) Монолитная эра. Ей свойственны следующие аспекты:

- Приложения монолитные.
- Куча зависимостей.
- Долгая разработка до релиза.
- Все инстансы известны по именам.
- Использование виртуализации. Это означает:
  - Один сервер – несколько VM [1].
  - Resource Management.
  - Изоляция окружений.

Соответсвенно использовались VM:

- VMWare.
- Microsoft Hyper-V.
- VirtualBox.
- Qemu.

Подход был следующий: один большой сервер делили на несколько виртуальных машин. Это давало полную изоляцию, но недостатками были:

- Hypervisor [2].

— Большие образы.

— Как следствие больших образов с разным ПО – медленное управление VM.

б) На смену им пришла виртуализация на уровне ядра с помощью следующих инструментов:

— OpenVZ.

— Systemd-nspawn.

— LXC.

Но остались прежние проблемы:

— Большие образы с OS с большим количеством ПО.

— Нет стандарта упаковки и доставки.

— DLL Hell [3].

в) Но далее пришли контейнеры. Разница между VM и контейнером:

— Виртуальная машина подразумевает виртуализацию железа для запуска гостевой ОС.

— Контейнер использует ядро хостовой ОС.

— В VM может работать любая ОС.

— В контейнере может работать только GNU/Linux (с недавних пор и Windows).

— VM хороша для изоляции.

— Контейнеры не подходят для изоляции.

В итоге мы приходим к ситуации, изображенной на (Рис. 1.1):

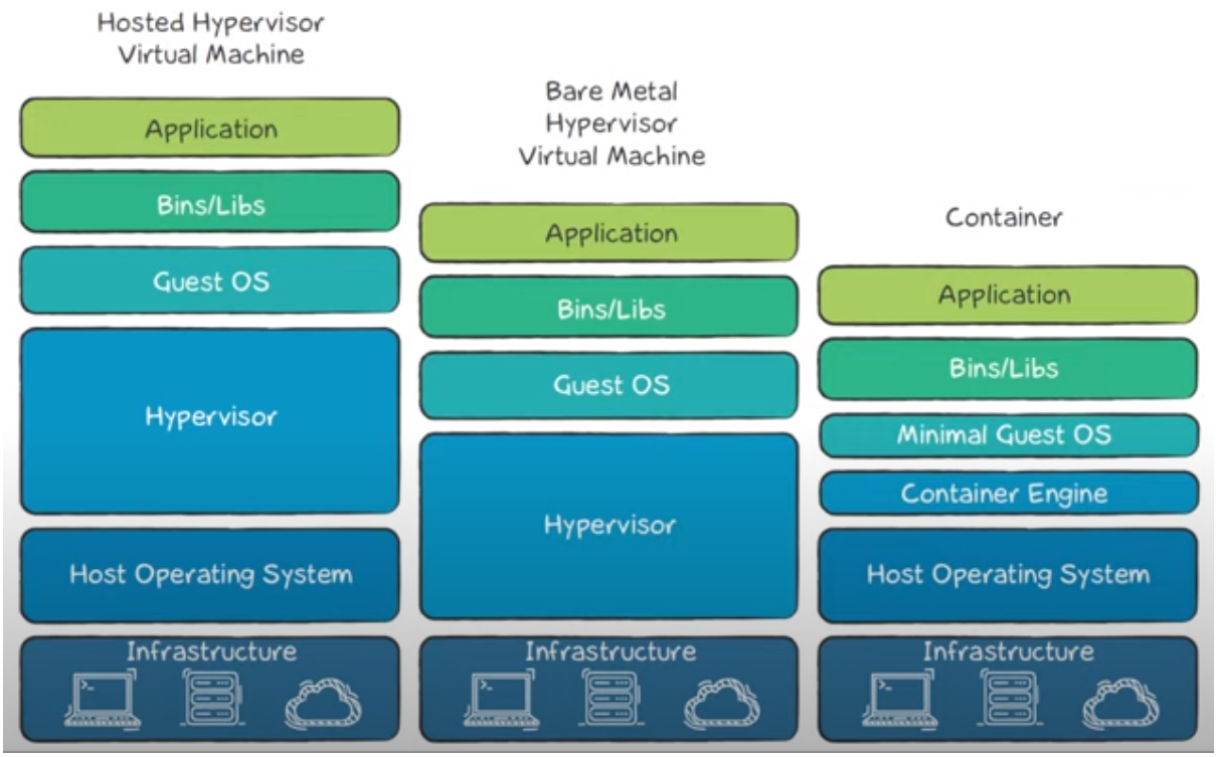

Рисунок 1.1 — Сравнение VM и контейнеров

Способ реализации контейнеризации:

— Namespaces:
  — PID.
  — Networking.
  — Mount.
  — User;
— Control Groups:
  — Memory. .
  — CPU
  — Block I/O;
  — Network;

В итоге получается следующая логика:

— Один процесс – один контейнер.

— Все зависимости в контейнере.

— Чем меньше образ контейнера – тем лучше.

— Инстансы становятся эфемерными.

И в 2014-2015 годах пришел расцвет Docker [4], который:

— Меняет философию подхода к разработке.

— Стандартизует упаковку приложения.

— Решает вопрос зависимостей.

— Гарантирует воспроизводимость.

— Обеспечивает минимум дополнительных средств для использования.

1.2 Область применения

Для большого количества проектов с множеством команд

1.3 Обзор литературы

Обзор на статьи Фланта, SouthBridge etc

1.4 Разработка архитектуры и выбор инструментария

Картинки с разными уровнями погружений и т.д. в новую архитектуру с пояснениями

1.5 Выстроение этапов реализации инфраструктуры

Что-то типа roadmap

2 Практическая часть работы

2.1 GitLab

Первым этапом было необходимо развренуть систему контроля версий. Ранее был выбран GitLab как комплексное решение данной задачи.

2.2 Kubernetes

Далее необходимо было развернуть кластер для контейнеров. Основным инструментом, решающим эту задачу был выбран Kubernetes. Официальным инструментом для выполнения этой задачи является kubeadm, но он имеет ряд недостатков. Среди них стоит отметить сложность настройки нестандартных конфигураций, не говоря уже о сложности обновления кластера или подключения дополнительных нод. Поэтому был выбран один из самых популярных инструментов – kubespray. Этот инструмент, по своей сути – YAML-манифесты, с помощью которого описаны playbook для Ansible – средства автоматизации по развертыванию инфраструктур. В свою очередь, tasks и roles в playbook используют kubeadm, но управлять им через данные абстракции в разы проще и удобнее. Также это позволяет иметь готовый артефакт, чтобы повторно развернуть кластер в другом месте или обновить его, внеся небольшие изменения.

Далее будут рассмотрены переменные, определенные в файле для развернутого кластера, отличающиеся от заданных по умолчанию:

а) Файл `inventory.ini`:

Данный файл кардинально отличается от оригинала, так как тот является лишь примером. В нем указаны в общей группе все инстансы, которые будут использоваться как ноды кластера. Среди них master и worker ноды, за названием каждой ноды должен быть

указан ее IP адрес для подключения. Также master ноды должны быть перечислены в группе kube-master. Если etcd будет располагаться на master нодах, то их надо указать в группе etcd, иначе необходимо указать отдельные инстансы в общей группе под хранилище манифестов, и, соотвественно, указать их в группе хранилища. Worker ноды указываются в группе kube-node. Детальнее можно ознакомиться в ПРИЛОЖЕНИИ ТАКОМ-ТО.

б) Файл `group_vars/k8s-cluster/addons.yml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `dashboard_enabled: true`

Это позволяет получить базовый dashboard с состоянием кластера до момента поднятия полноценного мониторинга.

— `helm_enabled: true`

Эта опция устанавливает Helm вместе с кластером Kubernetes.

— `metrics_server_enabled: true`

Включает отдачу метрик.

— `ingress_nginx_enabled: false`

Это значение по умолчанию, но на первых этапах его стоит поменять на true, чтобы иметь в базовой установке кластера Ingress Controller на базе nginx.

— `ingress_nginx_host_network: true`

Данный параметр не будет задействован без предыдущего, но его стоит выставить в значение true, чтобы при включении базового контроллера он был работоспособен за счет обработки трафика с хостовой ноды.

— `ingress_nginx_namespace: "ingress-nginx"`

Здесь указано пространство в кластере, в котором будет находится контроллер. Этот параметр стоит раскомментировать, чтобы четко задать пространство, не предоставляя возможности разместить контроллер в пространстве по умолчанию или же служебном пространстве.

— `ingress_nginx_insecure_port: 80`

Порт, на который должны приходить HTTP-данные.

— `ingress_nginx_secure_port: 443`

Порт, на который должны приходить HTTPS-данные.

в) Файл `group_vars/k8s-cluster/k8-cluster.yml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `kube_version: v1.19.6`

На момент развертывания кластера последняя из стабильных версий kubelet.

— `kube_service_addresses: 10.10.0.0/16`

Более широкая по диапазону адресов сеть, нежели предложенная по-умолчанию. Также необходимо отслеживать, чтобы виртуальные сети кластера не пересекались с сетями на хостовых нодах.

— `kube_pods_subnet: 10.15.0.0/16`

Аналогично предыдущему параметру, указана более широкая сеть.

— `cluster_name: cluster.itsoft`

Задание имени кластера, что также влияет на внутренние DNS-имена.

— `podsecuritypolicy_enabled: true`

Включение PodSecurityPolicy в кластере, что позволит настроить ограничения работы контейнеров и других сущностей для большей безопасности.

— `kubeconfig_localhost: true`

Установка служебной утилиты на машину, с которой производится развертывание кластера.

— `kubectl_localhost: true`

Установка служебной утилиты для управления кластером на машину, с которой производится развертывание кластера.

г) Файл `group_vars/k8s-cluster/k8-net-calico.yml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `calico_ip_auto_method: “interface=ens3”`

Опция, позволяющая сетевому плагину для кластера автоматически определить, какую сеть использовать для работы между нодами, на основе указанного сетевого интерфейса.

д) Файл `group_vars/etcd.yml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `etcd_memory_limit: 0`

Такое задание значения параметра снимает ограничения по памяти на хранилище манифестов etcd, что критично для его работы.

е) Файл `group_vars/all/docker.yml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `docker_storage_options: -s overlay2`

Так как на хостах используется свежая версия ядра ОС, то желательно использование более современного драйвера хранения данных для Docker.

После адаптации конфигурационных файлов под задачи и доступные вычислительные ресурсы, необходимо запустить playbook `cluster.yml`. При дальнейших обновлениях кластера необходимо использовать файл `upgrade-cluster.yml`.

2.3 NFS

Следующим этапом стало подключение системы хранения данных для контейнеров. Для этого была создана NFS – сетевая файловая система. Она подключена с помощью проекта NFS Subdir External Provisioner для кластера Kubernetes. Проект также пришлось адаптировать под созданные хранилища отдельно на SSD и HDD:

а) Файл `deploy/helm/values.yaml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

На месте примера с одной подключаемой NFS была переписана структура YAML-манифеста, которая теперь включает подразделы ssd и hdd, в которых уже, в свою очередь, указывается IP адрес сервера, путь до директории на сервере и опции монтирования.

б) Файл `deploy/helm/templates/deployment.yaml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

Из-за подключения двух NFS у Storage Provisioner для Kubernetes необходимо было создать, соотвественно, два контейнера: один для хранилища на SSD, второй для HDD. В связи с этим были продублирован раздел с контейнером, изменены их имена и метки, подправлены обращения к переменным в `deploy/helm/values.yaml`, и добавлены необходимые volumes для монтирования.

в) Файл `deploy/helm/templates/storageclass.yaml` (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

Также, как и в `deploy/helm/templates/deployment.yaml`, были скорректированы обращения к переменным в `deploy/helm/values.yaml`, продублирован StorageClass, поправлены имена во избежание коллизий.

После выполнения подготовки и настройки конфигурации под имеющиеся NFS, необходимые сущности Kubernetes разворачиваются с помощью Helm по инструкции проекта. Также, после развертывания проекта, необходимо написать YAML-манифесты для создания набора тестовых экземпляров сущностей Kubernetes, чтобы проверить работу хранилища: PersistentVolumeClaim, обращающегося к необходимому StorageClass, несколько Pod с PersistentVolume, который обращается к указанному ранее PersistentVolumeClaim, один из которых записывает в него данные, а другой позволяет их прочесть (ПРИЛОЖЕНИЕ ТАКОЕ-ТО).

2.4 Traefik

Далее необходимо было настроить Ingress Controller для промышленного использования, то есть обработки сетевого трафика. В качестве этого инструмента был выбран Traefik. Он, в сравнении с предлагаемым по умолчанию nginx, является намного более современным, лишенным некоторых ощутимых недостатков, которые присущи nginx. Его настройка также заслуживает внимания:

Файл values.yaml (ПРИЛОЖЕНИЕ ТАКОЕ-ТО):

— `ingressClass[enabled]: true`

Этот параметр означает, что будет создана сущность IngressClass в кластере Kubernetes для того, чтобы он корректно увидел новый Ingress Controller.

— `ingressClass[isDefaultClass]: true`

Задание Traefik как Ingress Controller по умолчанию.

— `pilot[enabled]: true`

Эта опция включает возможность управления и мониторинга активности Traefik посредством веб-интерфейса. За ней следует еще одна, содержащая токен.

— `volumes`

Этот раздел параметров указывает сущности Kubernetes, которые необходимо подключить к Pod Traefik. На данном этапе это используется, чтобы подключить wildcard сертификаты для сайтов.

— `ports[web][hostPort]: 80`

Это значение позволяет включить проброс сетевого трафика с 80-го порта инстанса, на котором будет находиться Pod Traefik на заданный по умолчанию порт самого Pod Traefik.

— `ports[web][redirectTo]: websecure`

Эта опция включает перенаправления трафика с HTTP на HTTPS.

— `ports[websecure][hostPort]: 443`

Аналогично подобной опции в разделе `web`, позволяет проброс трафика с хостовой машины внутрь Pod Traefik.

— `ports[websecure][tls][enabled]: true`

Эту опцию необходимо включать для того, чтобы работали сертификаты для HTTPS. Далее, в разделе tls, указываются пути до сертификатов и обслуживаемые доменные имена.

— `service[enabled]: false`

Так как у нас Traefik является входным узлом, то для него не требуется сущность Service Kubernetes.

— `persistence[enabled]: true`

Использование PerstistentVolumeClaim Kubernetes для хранения TLS сертификатов. Далее также необходимо указать корректный StorageClass, который был создан ранее.

— `podSecurityPolicy[enabled]: true`

Включение аспектов безопасности Pod Traefik.

— `resources`

Этот раздел аргументов позволяет задать минимальные требования по ресурсам для запуска Traefik, а также его лимиты, за которые он не сможет выйти по потреблению.

— `nodeSelector`

С помощью сущности NodeSelector можно указать, на каком хосте или группе хостов должен оказаться конкретный Pod. В данном случае, это позволяет расположить Pod Traefik именно там, куда проброшен трафик из Интернета.

— `tolerations`

С помощью данной сущности Kubernetes можно обойти какие-либо ограничения. В текущем случае, это ограничение на планирование запуска Pod на master ноде.

2.5 CI/CD

2.6 Certs

3 Документация

3.1 Инструкция для пользователя

3.2 Руководство администратора

3.3 Документация разработчика

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Wikipedia. Виртуальная машина — Википедия. — 2021. — Режим доступа: https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0&stable=1 (дата обращения: 07.05.2021).

2. Wikipedia. Гипервизор — Википедия. — 2021. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D0%B2%D0%B8%D0%B7%D0%BE%D1%80 (дата обращения: 07.05.2021).

3. Dick Stephanie, Volmar Daniel. DLL hell: software dependencies, failure, and the maintenance of microsoft windows // IEEE Annals of the History of Computing. — 2018. — Vol. 40, no. 4. — P. 28–51.

4. Демидова ТС, Соболев АА. Использование Docker для развёртывания вычислительного программного комплекса // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем. — 2019. — P. 432–435.