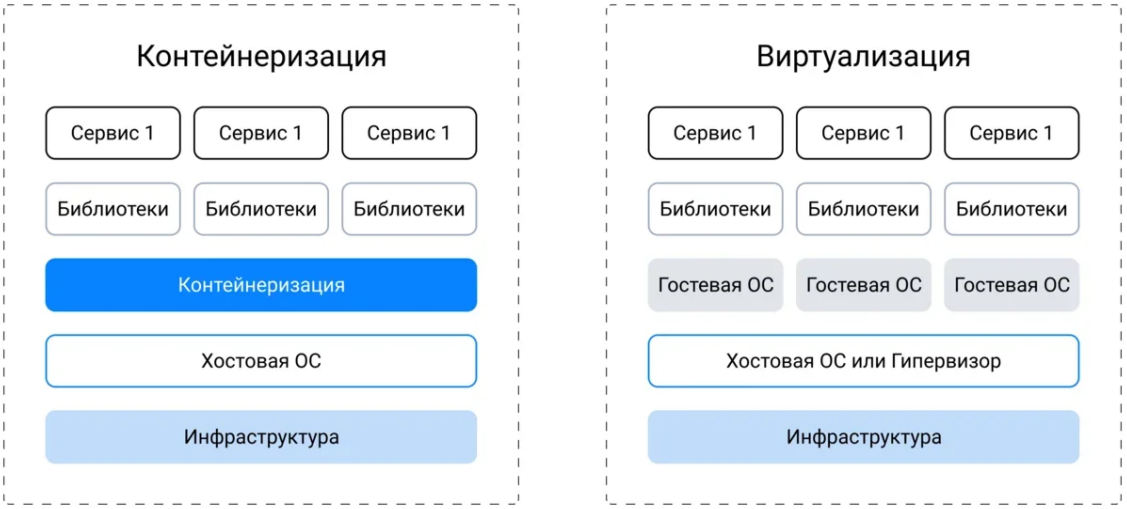
Теория.

Контейнеризация

Мы много говорили о пользе виртуальных машин. Они предоставляют изолированную среду, в которой приложения гарантированно работают, и поэтому упрощают перенос и тиражирование приложений. Но у ВМ есть и недостаток: они создают рабочую среду полностью, включая операционную систему (ОС) и весь установленный на ней софт. Когда на одном сервере создаются несколько ВМ, каждая запускает свои отдельные экземпляры ОС и прочих приложений. В результате ресурсы сервера — вычислительная мощность процессора, дисковое пространство и т. д. — расходуются неэффективно.

Новое решение: контейнеризация

Главное и принципиальное отличие **контейнера** от ВМ в том, что он использует ресурсы и ядро хостовой ОС. Несколько контейнеров, размещённых на одном сервере, используют ресурсы сервера совместно, тем самым экономя их.



Так же, как и ВМ, контейнер изолирован от других контейнеров и хостовой ОС. Он может содержать различные приложения и запускаться на различных платформах.

Хорошей практикой считается принцип «один контейнер — один сервис». Так проще обновлять приложения и создавать резервные копии. Например, если вы написали для веб-сервера NGINX веб-приложение на Python, поместите сервер и приложение в отдельные контейнеры.

Слоистая архитектура контейнеров

Говоря о контейнерах, часто употребляют термин **слой**. Любое изменение окружения — установка программы, создание директории — создаёт новый слой. Эти слои накладываются друг на друга.
Если на одном сервере оказываются несколько контейнеров с общими слоями (например, библиотеками), то слои не дублируются: они устанавливаются один раз и затем используются совместно.

Преимущества контейнеров

С контейнерами разработка стала эффективнее и проще. Чем же они хороши?

- **Экономия ресурсов**. Во-первых, контейнеры занимают меньший объём, чем ВМ: они не содержат отдельных копий ОС и дополнительных программ и утилит. Во-вторых, благодаря общим слоям контейнеры оптимизируют использование ресурсов хоста.
- **Независимость**. Контейнер самодостаточен. Всё, что нужно для работы (библиотеки, настройки, среда запуска), находится внутри.
- **Переносимость**. Контейнер независим. Платформа, на которой его запускают, неважна: он везде будет работать одинаково. Можно спокойно переносить контейнер с одной платформы на другую.
- **Скорость** разворачивания контейнеров и работы в них. Это преимущество следует из предыдущих. Сервер не тратит время на эмуляцию гостевой ОС, а высвободившиеся ресурсы можно направить на увеличение производительности приложений и сервисов.
- **Тиражирование и масштабирование**. Собрали контейнер однажды — копируйте его сколько угодно раз. Запускайте

одновременно нужное количество копий контейнера. Всё будет работать одинаково.
- **Оркестрация**. Дирижёр одновременно управляет множеством музыкантов, играющих на разных инструментах. Вы можете создавать похожие системы из контейнеров, каждый из которых выполняет узкую задачу. Оркестрация — это управление такими системами, т. е. координация работы множества контейнеров.

Docker

Как и виртуальные машины, контейнеры создаются из образов. На сегодняшний день самая популярная и удобная платформа для создания и запуска образов — Docker.

Docker работает так. Предположим, вы с коллегами разработали приложение. Вы упаковываете его со всеми зависимостями — библиотеками, интерпретаторами, файлами и т. д. — в Docker-образ и отправляете в репозиторий (т. е. в хранилище). Чтобы развернуть приложение, нужно скачать из репозитория образ и создать из него контейнер на рабочем сервере.

Хранилища Docker-образов бывают публичные и приватные. Самое известное публичное хранилище — это Docker Hub. Однако если вы работаете с Yandex Cloud, лучше использовать собственное хранилище облака — Yandex Container Registry.

Как создаются образы

Docker-образы создаются с помощью инструкций, таких как запуск команды, добавление файла или директории, создание переменной окружения. Инструкции хранятся в `Dockerfile` — это обычный текстовый файл, который можно редактировать в любом текстовом редакторе (что соответствует принципам Infrastructure as Code).

Вот простой пример `Dockerfile` для образа, в котором есть только ОС Ubuntu и веб-сервер NGINX:

```
FROM ubuntu:latest

RUN apt-get update -y

RUN apt-get install -y nginx

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Каждая инструкция создаёт новый слой образа, и эти слои накладываются друг на друга. В конце вы задаёте команду — исполняемый файл, который будет запущен при старте Docker-контейнера.

В примере выше первая строка определяет исходный образ (публичный образ с последней версией Ubuntu), на основе которого мы строим свой.

Вторая и третья строки устанавливают веб-сервер NGINX.

Последняя строка задаёт точку входа — запускает NGINX.

Процесс создания образа — это считывание и выполнение инструкций из `Dockerfile`. Чтобы создать образ из `Dockerfile`, используется команда `build`. Если файл со спецификацией называется стандартно (`Dockerfile`), не указывайте название. Если иначе — напишите название после ключа `-f`. После ключа `-t` указывается имя образа, который будет создан:

```
docker image build -f my-dockerfile -t my-image .
```

Точка в примере означает, что для сборки используется текущая директория.

Как создаются контейнеры из образов

Для работы с хранилищем используются традиционные команды `push` и `pull`. Так мы помещаем образ в хранилище:

```
docker push my-image
```

Чтобы создать контейнер, загрузите из хранилища образ и запустите его:

```
docker pull my-image

docker run my-image
```

При создании контейнера из образа можно использовать параметры (флаги). Например, чтобы ограничить ресурсы памяти и процессора, загрузить свежую версию образа, передать значения переменных. Смотрите доступные флаги с помощью традиционного ключа `--help`.

Yandex Container Registry

Если вы работаете с Yandex.Cloud, лучше всего использовать сервис Yandex Container Registry.

Преимущества Yandex Container Registry

- **Бесплатный внутренний трафик**. Для создания контейнеров придётся скачивать образы, которые могут весить несколько гигабайтов. Если вы берёте образы из Docker Hub или другого внешнего реестра, трафик тарифицируется. А если из Yandex Container Registry — такой трафик считается внутренним и не оплачивается.
- **Приватный реестр**. В Docker Hub это платная возможность. В Yandex Container Registry ваш реестр по умолчанию приватный. Чтобы сделать его публичным, предоставьте права системной группе `allUsers`.
- **Политика автоматического удаления.** При CI/CD после каждого изменения исходного кода создаётся новый образ. В итоге образов становится слишком много, приходится вручную управлять ими и удалять лишние. В Yandex Container Registry можно настроить автоматическое удаление. Это упростит управление образами в рамках CI/CD и сэкономит дисковые

ресурсы и деньги, ведь стоимость хранения образов зависит от их объёма.
- **Удобство.** С Yandex Container Registry вы будете работать в привычном интерфейсе консоли управления и с командами утилиты `yc`.

Реестр, репозиторий и теги

Реестр в Yandex Container Registry — это хранилище Docker-образов, а **репозиторий** — набор образов с одинаковыми именами (т. е. версий образа).

Чтобы различать образы в репозитории и отбирать их по правилам, добавляйте к имени образа уникальный в рамках репозитория **тег**. Если тег не задан — последней версии образа автоматически присваивается тег `latest`.

При обращении к образу используется префикс `cr.yandex`. Он означает, что образ хранится в Yandex Container Registry.

Так выглядит запись для обращения к образу:

```
cr.yandex/<реестр>/<имя образа>:<тег>.
```

Пример полного имени: `cr.yandex/my-registry/my-app:latest`.

Регулярные выражения позволяют выбирать образы по правилам. Например, если тестовые образы приложения `my-app` создавались с тегами `testVersion1`, `testVersion2`, `testVersion3` и т. д., то вы отберёте все тестовые образы вот так:

```
cr.yandex/my-registry/my-app:test.*
```

Автоматическое удаление

Политики автоматического удаления настраиваются для каждого репозитория отдельно. Политика — это правила, по которым Docker-образы будут удаляться. Например, можно удалять все образы

с тегами `test.*` и все образы с тегами `prod.*`, созданные более месяца назад. При этом вы можете на всякий случай сохранить несколько образов, подходящих под условия.

Политики удаления описываются в JSON-файле в виде списка опций и их значений. Обычно используются опции:

- `tag_regexp` — тег Docker-образа для фильтрации.
- `untagged` — флаг для применения правила к Docker-образам без тегов.
- `expire_period` — время, кратное 24 часам, через которое Docker-образ попадает под политику удаления.
- `retained_top` — количество Docker-образов, которые не будут удалены, даже если подходят по правилу.

Вот пример файла `rules.json`:

```
[

    {

     "description": "Delete prod Docker images older than 30 days but
retain 20 last ones",

     "tag_regexp": "prod",

     "expire_period": "30d",

     "retained_top": 20

    },

    {

     "description": "delete all test Docker images except 10 last
ones",

     "tag_regexp": "test.*",

     "retained_top": 10

    },

    {
```

```
    "description": "delete all untagged Docker images older than 48
hours",

    "untagged": true,

    "expire_period": "48h"

   }

]
```

Удаление образа — это ответственное действие. Поэтому после настройки правил проверьте, как они будут работать в автоматическом режиме. Вам поможет тестовый запуск политики: `dry-run`.

Для репозитория можно настроить несколько политик, но активной будет только одна. Включайте и отключайте политики в зависимости от своих задач.

Container Optimized Image

Yandex Cloud позволяет создать из специального образа Container Optimized Image виртуальную машину, чтобы запустить на ней Docker-контейнер. При использовании Container Optimized Image не нужно устанавливать на машину Docker и скачивать образ с помощью команды `docker pull`.

Практическая работа. Создание докер-образа и загрузка его в Container Registry

В этой практической работе вы создадите реестр в Yandex Container Registry, подготовите Docker-образ виртуальной машины и поместите его в реестр, а затем создадите машину из этого образа.

1. Установите Docker.
2. Создайте реестр в Yandex Container Registry:

   Скопировать код

```
yc container registry create --name my-registry
```

3. Обратите внимание, что в выводе есть уникальный идентификатор (`id`) реестра. Он пригодится вам для следующих команд.

```
id: crpfpd8jhhldiqah91rc

folder_id: b1gfdbij3ijgopgqv9m9

name: my-registry

status: ACTIVE

created_at: "2021-04-06T00:46:48.150Z"
```

4. Аутентифицируйтесь в Yandex Container Registry с помощью Docker Credential helper. Это нужно для того, чтобы внешняя платформа Docker могла от вашего имени отправить образ в ваш приватный реестр в Yandex Cloud.

   Скопировать код

```
yc container registry configure-docker
```

5. Подготовьте `Dockerfile`.

   Скопировать код

```
FROM ubuntu:latest

RUN apt-get update -y

RUN apt-get install -y nginx

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

6. По умолчанию Docker использует файл с именем `Dockerfile` и без расширения.
7. Перейдите в папку с `Dockerfile` и соберите образ (не забудьте подставить идентификатор своего реестра):

   Скопировать код

```
docker build . -t cr.yandex/<идентификатор_реестра>/ubuntu-nginx:latest
```

8. Ключ `-t` позволяет задать образу имя.
   Напоминаем, что в Yandex Container Registry можно загрузить только образы, названные по такому шаблону:

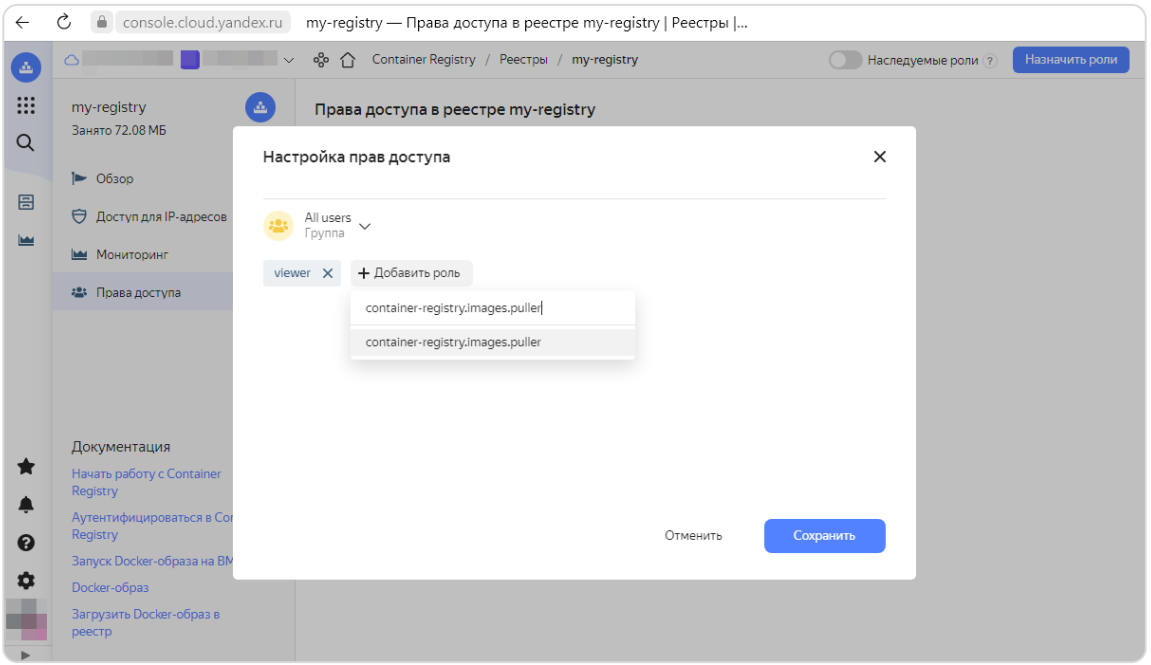
```
cr.yandex/<ID реестра>/<имя Docker-образа>:<тег>
```

9. Загрузите Docker-образ в реестр:

Скопировать код

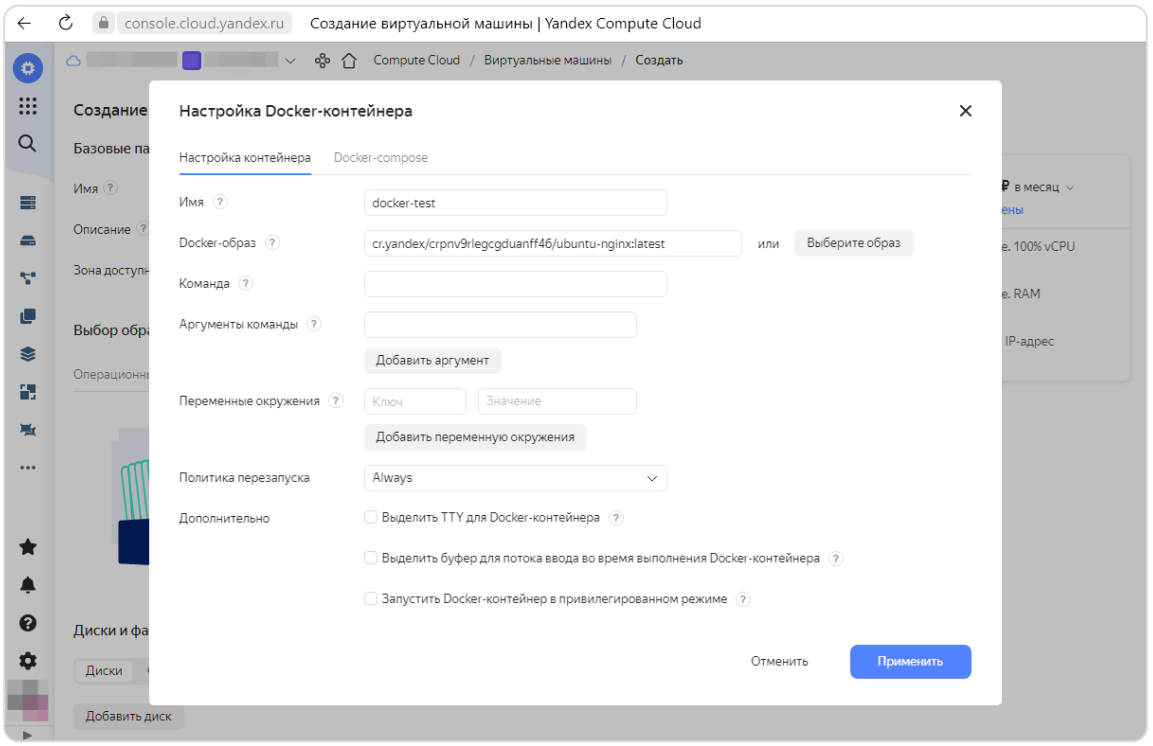
```
docker push cr.yandex/<идентификатор_реестра>/ubuntu-nginx:latest
```

10. В консоли управления перейдите в реестр и предоставьте всем пользователям право использовать хранящиеся образы. Для этого перейдите на вкладку **Права доступа**, в правом верхнем углу нажмите кнопку **Назначить роли**. В открывшемся окне нажмите кнопку **Выбрать пользователя**, на вкладке **Группы** выберите `All users`. Нажмите кнопку **Добавить роль** и последовательно введите `viewer` и `container-registry.images.puller`. Нажмите кнопку **Сохранить**.



10. В консоли управления создайте ВМ с помощью Container Optimized Image.
При создании машины в разделе **Выбор образа загрузочного диска** переключитесь на вкладку **Container Solution** и нажмите **Настроить**. Выберите из реестра созданный образ, остальные

настройки оставьте по умолчанию и нажмите **Применить**.



Другие настройки ВМ мы уже разбирали.

11. Когда новая ВМ получит статус `Running`, найдите её внешний IP адрес в консоли управления и убедитесь, что по этому адресу отображается приветственная страница NGINX.

Обратите внимание! С помощью Docker-образа вы создали и запустили виртуальную машину с предустановленным, нужным вам ПО. При этом вам даже не потребовалось заходить внутрь ВМ и выполнять установку или настройку ПО вручную.