??????? ? ???????

- Общая информация и установка
- Образы
- Контейнеры
- Тома (volumes)
- Сеть (networking)
- Yaml формат

????? ??????????? ? ?????????

Установка

Доп. ссылка

Добавить пользователя, который впоследствии будет запускать контейнеры

```
apt-get install sudo
usermod -aG sudo sergey
sudo apt-get update
```

Добавить сертификат и новое хранилище

(Вариант 1 - Debian)

```
sudo apt-get install ca-certificates curl gnupg lsb-release
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

(Вариант 2 - Ubuntu)

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Установка docker и docker compose

(Вариант 1 Debian/Ubuntu)

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

(Вариант 2 Alt Linux, сертификаты ставить не надо)

```
sudo apt-get install -y docker-engine
sudo apt-get install -y docker-compose-v2
```

Дополнительные удобства

Запуск docker при старте системы

```
sudo systemctl enable docker
```

Добавить пользователя в группу docker

```
sudo usermod -aG docker $USER
```

Информация о текущем статусе docker

```
docker info
```

Для Alt Linux

```
sudo gpasswd -a USER docker
sudo systemctl enable --now docker
sudo init 6
```

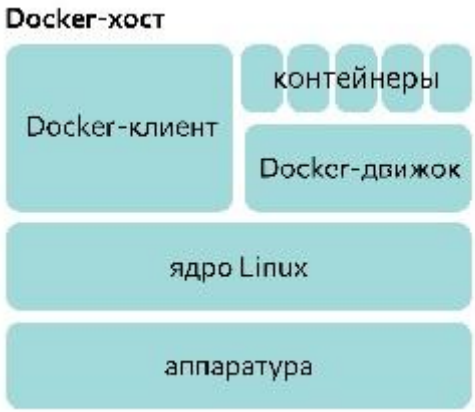
Размещение compose файлов.

Для dev: где угодно. Лучше определить, например ~/project_name

Для prod: /opt и /srv стандартизированы FHS (Filesystem Hierarchy Standard) для стороннего ПО. Я остановился на /srv/project_name/

Теория

Образы (Images): неизменяемые элементы, основанные как минимум на ядре и архитектуре. Контейнер: исполняемый образ. Назначение контейнера - запуск одного приложения. Аналогия: Образ=класс, контейнер=объект. Т.е. к образу в момент запуска добавляются параметры и он становится исполняемым контейнером, но изменения в контейнере не затрагивают образ. Контейнер в текущем состоянии можно превратить в образ, но после остановки (если не сохранить данные в файловой системе хоста или еще где-либо) все данные будут потеряны. Повторный запуск будет произведен как будто создался новый объект. В этом принципиальное отличие от виртуальной машины.



Может быть: архитектура ПК-ядро-образ-образ-образ-контейнер

Архитектура

Docker client -> Docker engine (daemon) -> containerd -> runc-> shim

runc легкая обертка libcontainer, задача - создание контейнеров. Для создании контейнера создается экземпляр runc, он создает контейнер, запускает и завершается. Контейнер передается shim. Shim сопоставляет процесс созданного контейнера с задачей в containerd, containerd логика выполнения контейнера, управления образами, томами, сетью, модульный, можно отключать

Пример процессов:

docker run --name ctr1 -it alpine:latest sh

- docker client преобразовывает в POST API запрос в Docker engine
- Запрос отправляется в сокет /var/run/docker.sock
- демон получает команду создания нового контейнера и передает вызов в containerd (CRUD-style API поверх gRPC)
- containerd преобразовывает Docker image и направляет запрос в runc для создания нового контейнера
- runc взаимодействует с ядром, создавая контейнер. Контейнер запускается как дочерний процесс runc. После запуска контейнера runc завершается.

Docker daemon может работать с сетью, non-TLS порт 2375, TLS port 2376

Контейнеры:

После запуска основным процессом становится приложение из параметров запуска

При отсутствии, используется приложение сконфигурированное в образе как точка запуска (Entrypoint)

Взаимоотношение между способами работы с Docker.

При работе с докером используются как минимум три способа (возможно больше) взаимодействий: прямое взаимодействие через консоль, dockerfile, docker compose. Терминология общая, но способы работы и результаты отличаются. Из-за того, что в целях упрощения консоли docker (групп команд docker ...) эти способы перемешаны, сначала возникает путаница. В статьях видимо по умолчанию считается, что читатель это отличие полностью осознал. Сравнение способов:

	Консоль	Dockerfile	docker compose

Мое понимание термина	Ввод команд настроек и запуска одного контейнера напрямую в консоли. Будет работать через bash скрипт, но удобнее так не делать.	Сохранение команд настроек одного контейнера в файл и запуск через консоль.	Настройка взаимодействия нескольких взаимосвязанных контейнеров при условии подготовленных образов. Запуск через консоль / планировщик
Применение	При тестовом запуске или для просмотра (корректировки) параметров работающего контейнера	Шаблон одного контейнера. Нужно при создании образа для последующего использования в compose.	Оркестрация образов.
Стадия использования	Тестовая, до MVP. Либо product для анализа проблем.	Тестирование, отладка одного образа. MVP - pre product.	Комплексный запуск сервиса. Pre product - product
Формат	Bash	ini	YAML
Точка использования	Преимущественно на ПК разработчика, при обучении технологии или для выявления проблем с работающим контейнером на всех стадиях.	ПК разработчика или тестовый сервер.	Тестовый / product сервер.
Необходимость серьезного изучения	Скорее администраторам.	Да, если для сервиса потребуется создание отдельных образов.	Да. Данный способ можно использовать для одного контейнера, для product все равно потребуется.*

* При сборке из чистых образов (типа debian для запуска nginx+...) требуется доп. установка и настройка ПО, а данный вопрос проще решать через dockerfile. К тому же существует например ansible, позволяющий решить ряд вопросов. В книгах для примеров часто используют консоль (быстро, акцент на одном изучаемом аспекте). А через ansible возможно настроить и запустить контейнер (и оркестр) без dockerfile/docker compose. Но в случае ansible для compose используется терминология compose, а вид команд другой.
Резюме: это одни яйца, только с разных концов. Нужно общее понимание происходящего. Хотя формат compose чуть предпочтительнее.

??????

Образ - контейнер только для чтения, содержащий все для запуска (минимальная ОС, приложение, зависимости, метаданные)
Образ состоит из слоев, существующие слои общие для использования и могут быть в разных образах.
Образ - manifest файл, в котором список слоев и метаданных.
Слой - целостный объект.
Каждая команда в dockerfile, модифицирующая файловую систему контейнера, создает новый слой.
Нельзя удалить образ, пока последний контейнер не будет удален.
Образы обычно хранятся в /var/lib/docker/<storage-driver>
!Нужно достаточно места в /var!
У образа есть hash (digest), у каждого из слоев есть digest
Параметр distribution hash - hash сжатого образа или образа для размещения в hub
Для поддержки разных архитектур и платформ для одного названия (напр. golang:latest) используются manifest list и manifests
Для каждого контейнера создается новый слой чтения/записи

Команды работы с образами

Базовая команда	Доп. парам.	Описание
docker images		Список установленных образов
	-q	только отображение ID контейнеров
	--digests	добавляет столбец с хэшем образа
	dangling: true/false	Образы без тега. Происходит если при создании нового образа сохраняется тег старого. У старого обнуляется тэг, у нового остается.
	label: <label>	Фильтрует на основе наличия метки или ярлыка и значения. Команда docker images не отображает метки в своих выходных данных.
docker search	<слово>	поиск в dockerhub по слову <слово>
	--filter ""	Доп. фильтр `docker search alpine --filter "is-official=true"`
	--limit число	кол-во выдачи, максимум 100

Базовая команда	Доп. парам.	Описание

docker image pull
Синоним:
docker pull
	<repository>:<tag>	Загрузка образа из репозитория repository:
		• В случае официального образа на docker hub: одно слово
		• В случае неофициального образа на docker hub: слово/слово
		`docker pull nigelpoulton/tu-demo:v2`
		• В случае внешнего хранилища: адрес хранилища без http(s)
		`docker pull gcr.io/google-containers/git-sync:v3.1.5`

	<digest>	Хэш образа
docker image push Синоним: **docker push**	<repository>:<tag>|<digest>	Загрузка образа на репозиторий, доп. параметры аналогичны pull
docker image prune	<repository>:<tag>	удаление образа из локального хранилища
	-a	удаление всех неиспользуемых контейнерами образов

docker build
docker image build
	-t тег образа. обычно имя:версия
	<путь к dockerfile> - обязательный
		Создание образа из dockerfile
		`docker build -t myapp:1.0 .`
		`docker build -t ${PROJECT_NAME}_db:${VERSION} ./database`

Базовая команда	Доп. парам.	Описание
docker save	-o имя_файла.tar имя_образа:тег	Сохранение существующего образа в файл `docker save -o имя_файла.tar имя_образа:тег`
docker load	-i имя_файла.tar	Загрузка образа из файла `docker load -i имя_файла.tar`
docker image tag **docker tag**		Создать тэг TARGET_IMAGE связанный с SOURCE_IMAGE Это нужно только для публикации образа в dockerhub `docker tag ddd-book:ch8.1 nigelpoulton/ddd-book:ch8.1`
docker rmi		Удаляет заданный образ или несколько образов. Удаление всех образов: `docker rmi $(docker images -q) -f` Эта команда удаляет старые теги без удаления образа, если есть еще образы, ссылающиеся на этот образ.
docker inspect <имя или ID образа>		информация по образу
docker info | grep Storage		При использовании docker pull или docker-compose up -d образы размещаются в /var/lib/docker/overlay2

??????????

Основные команды

Команда	Доп. пар.	Описание
docker ps		список работающих контейнеров
	-a	список остановленных но еще существующих контейнеров
docker logs <name>		список событий внутри контейнера name
docker start <name>		запуск остановленного контейнера
docker restart <name>		Перезапускает один или несколько контейнеров. Можно считать приблизительным аналогом выполнения для заданных контейнеров команды docker stop, за которой сразу следует команда docker start.
	-t	определяет интервал времени ожидания, необходимого для завершения работы контейнера, перед его остановом по сигналу SIGTERM.
docker stop <name>		Останавливает (но не удаляет) один или несколько контейнеров. После выполнения этой команды заданный контейнер переходит в состояние «остановлен».
	-t	Аналогично restart
docker kill <name>		Сигнал основному процессу (PID=1) в контейнере. По умолчанию SIGKILL (немедленное завершение работы). Возвращает идентификатор контейнера.
	-s	другой сигнал
docker rm <name или id>		удаление остановленного контейнера
	-f	позволяет удалять работающие контейнеры.

Команда	Доп. пар.	Описание
	-v	удалить тома, созданные удаляемым контейнером (если эти тома не смонтированы на каталоги и не используются другими контейнерами) `docker rm -v $(docker ps -aq -f status=exited)` удаляет все остановленные контейнеры
docker top <name или id>		информация о процессах внутри контейнера
docker port <name или id>		номера портов, назначенные механизмом Docker
docker create <name или id>		Создает контейнер из образа, но не запускает его. Аргументы как у docker run.
docker run		создание и запуск контейнера из образа
	-it или -d	сеанс интерактивной работы или запуск в фоновом режиме
	--link <namecur:nameinnew>	соединение между новым контейнером и существующим контейнером myredis, в новом контейнере ссылка на существующий обозначена именем redis `--link myredis:redis`
	--name	имя для дальнейшего взаимодействия
	--hostname	имя для обращения
	--mount	монтирование
	--volumes-from CONTAINER	использование томов контейнера
	-p HostPort:ContPort	перенаправление с портов хоста на порт контейнера
	--rm	удаление остановленного контейнера и файловой системы после запуска и выполнения команды. Несовместим с ключом -d.

Команда	Доп. пар.	Описание
	--restart	Позволяет настроить образ действий при попытке Docker перезапустить остановленный контейнер. • no запрещает любые попытки перезапуска контейнера. • unless-stopped Будет перезагружаться до остановки. После перезагрузки демона Docker контейнер останется выключенным. • always попытки перезапуска выполняются в любом случае вне зависимости от состояния контейнера после выхода. После перезагрузки демона Docker контейнер опять запустится. • on-failure: перезапуск выполняются для контейнера, завершившего работу с ненулевым статусом. После перезагрузки демона Docker контейнер опять запустится. Может быть задан аргумент, определяющий максимальное количество попыток перезапуска (иначе попытки будут выполняться бесконечно). `docker run --restart on-failure:10 postgres`
	-t, --tty	Создает псевдоустройство TTY (терминал). Как правило, используется вместе с ключом -i для запуска контейнера в интерактивном режиме.
	-e, --env	Определяет переменные среды внутри контейнера.

Команда	Доп. пар.	Описание
	--entrypoint	Определяет точку входа для запускаемого контейнера в соответствии с заданным аргументом, заменяя содержимое любой инструкции ENTRYPOINT из Dockerfile.
	-u, --user	Определяет пользователя, от имени которого выполняются команды. Может быть задано как символьное имя пользователя или как числовой идентификатор UID. Заменяет содержимое инструкции USER из Dockerfile.
	-w, --workdir	Устанавливает рабочий каталог в контейнере в соответствии с заданным путевым именем. Заменяет любые значения, определенные в файле Dockerfile.
	--add-host=docker:10.180.0.1	ХЗ, узнать
docker pause/unpause		Временно приостанавливает/запускает все процессы внутри контейнера. Процессы не получают никаких сигналов приостановки(не могут быть остановлены и завершены или удалены).
docker commit <container name> <repo name>		образ контейнера. По умолчанию приостанавливаются, без приостановки --pause=false.
docker network ls		список сетей

Взаимодействие реального времени

Ctrl-PQ (или Ctrl+P, Ctrl+Q) отключение от контейнера без остановки

Команда	Доп. пар.	Описание
docker attach <name или id>		наблюдать или взаимодействовать с основным процессом внутри контейнера.

Команда	Доп. пар.	Описание
docker cp		Позволяет копировать файлы между файловыми системами контейнера и хоста. `docker cp /tmp/config.ini grafana:/usr/share/grafana/conf/`
docker exec		Запускает заданную команду внутри контейнера (может быть работающий сейчас). Может использоваться для выполнения задач сопровождения или в качестве замены ssh при входе (регистрации) в контейнер. `docker exec -it ubuntu:latest bash`
docker events		Выводит в реальном времени события от демона демону. Выхода Ctrl-C.

Примеры

- Запуск контейнера образа ubuntu:latest в интерактивном режиме с bash

```
docker run -it ubuntu:latest /bin/bash
```

- Подключение к работающему контейнеру и запуск в нем bash

```
docker exec -it e37f24dc7e0a bash
```

???? (volumes)

Тома (volumes) – файлы или каталоги, смонтированные на хосте и не являющиеся частью каскадно-объединенной файловой системы.
Другие контейнеры могут совместно использовать их, и все изменения будут сразу же фиксироваться в файловой системе хоста.

Устаревшее: В качестве точки монтирования можно определить любой другой каталог хоста в команде docker run (например, docker run -d -v /host/dir:/container/dir test/web-server).

В Dockerfile bind mounts не работает - а это и не надо делать, т к это определяется при старте образа/через compose.

Способы хранения данных:

1. Временное (удаляется при остановке контейнера)
    1. по умолчанию изолировано на диске
    2. tmpfs в оперативной памяти
2. Постоянное
    1. обычные тома Docker
    2. bind mounts - прямое монтирование папки в контейнер

Драйвера volumes

Драйвер	Описание
local	Драйвер по умолчанию. Только точки монтирования, доступные на хосте.
	И еще штук 30 драйверов, список драйверов

Управление томами при запуске контейнера из консоли:

Если тома нет - будет создан

Основная команда	Параметр	Описание
docker run ... --mount	type=	Тип тома: • volume • bind • tmpfs

Основная команда	Параметр	Описание
	source(src)=	Имя тома или не указывается для анонимных
	destination(dst)=	точка монтирования в контейнере
	volume-driver=	local по умолчанию, локальный том
	volume-opt=	опция=значение `volume-opt=type=nfs,volume-opt=device=<nfs-server>:<nfs-path>`
	readonly	только для чтения
docker run ... --volumes-from ContID		связывание с томами контейнера

Пример:

```
--mount 'type=volume,src=data-volume,dst=/var/opt/project,volume-driver=local,readonly'
```

Без пробелов после запятых.

Создание и управление томами независимо от контейнеров

Основная команда	Параметр	Описание
docker volume	create --name my_volume create my_volume	Создание тома. /var/lib/docker/volumes/имя тома/_data - расположение файлов По умолчанию на хосте в каталоге установки Docker (обычно каталог /var/lib/docker/). /var/lib/docker/volumes/
	ls	местоположение определенных томов, по имени или ID тома.
	inspect my_volume	информация о томе
	rm my_volume	удаление тома
	prune	удаление всех томов, которыми не пользуются контейнерами. Но иногда после удаления контейнера данные не обновляются

Основная команда	Параметр	Описание
docker system prune		очистка ресурсов docker. Потом - повторное удаление томов.

Примеры:
Автоматическое создание том с именем test-data в /var/lib/docker/volumes/test-data/_data

```
docker run -ti -d --name alpine-container -v test-data:/var/lib/app/content alpine
```

Монтирование в другую точку.
Нужно создать директорию.

```
mkdir /home/avimanyu/test-data
docker run -ti -d --name alpine-container -v /home/avimanyu/test-data:/var/lib/app/content
alpine
```

Архивировать том datavol в dbdata и восстановить его в том dbdata2 контейнера dbstore2
Создание тома datavol в контейнере dbdata

```
docker run -it --mount 'type=volume,src=datavol,dst=/datavol' --name dbstore ubuntu /bin/bash
```

Создание временного контейнера, примонтирование тома из dbdata, создание тома backup и сохранение архива datavol в backup и удаление временного контейнера

```
docker run --rm --volumes-from dbstore --mount 'type=volume,src=backup,dst=/backup' --name
tmpubn ubuntu tar cvf /backup/backup.tar /datavol
```

Создание тома datavol2 в контейнере dbdata2

```
docker run -it --mount 'type=volume,src=datavol2,dst=/datavol' --name dbstore2 ubuntu
/bin/bash
```

Создание временного контейнера, примонтирование тома из dbstore2, создание тома backup и сохранение архива datavol в backup и удаление временного контейнера

```
docker run --rm --volumes-from dbstore2 --mount 'type=volume,src=backup,dst=/backup' ubuntu
bash -c "cd /datavol && tar xvf /backup/backup.tar --strip 1"
```

???? (networking)

Docker поставляется со следующими сетевыми драйверами в рамках библиотеки libnetwork:

- single-host bridge networks (bridge)
- multi-host overlays (overlay)
- options for plugging into existing VLANs (macvlan)

Для тестов нужно установить

```
apt-get install bridge-utils
```

Docker регистрирует DNS сервис в пределах бриджа. Но в сети по умолчанию (docker0) DNS сервиса нет.

Команда	Описание

brctl show	Список бриджей.
	`sudo brctl show`
	`bridge name bridge id STP enabled interfaces`
	`br-62694f46296d 8000.7ee73b5c2894 no`
	`br-8af5ede4ffdc 8000.aee56ab6b984 no`
	`br-96dd8dcd216d 8000.5a64a8825202 no`
	`docker0 8000.f220300c62b1 no`

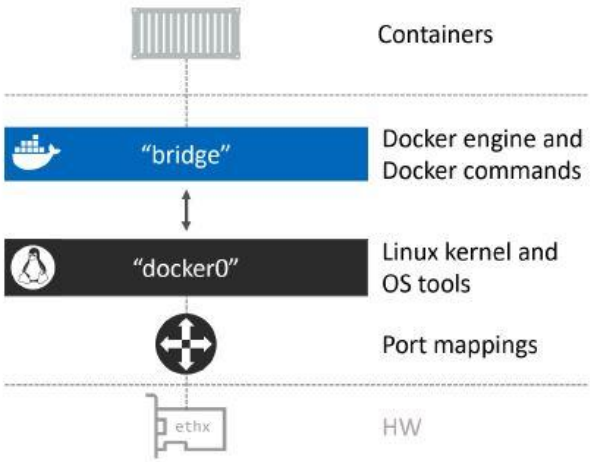
Также есть опция поиска сервисов и балансировка входной нагрузки.

Основная команда	Параметр	Описание
docker network	ls	Список сетей
docker inspect ИмяСети		Выводит информацию по указанной сети. bridge - сеть по умолчанию.

Основная команда	Параметр	Описание
docker network create	-d драйвер	Создает сеть `docker network create -d bridge localnet`
	название сети	
docker port ContName		Выводит map портов внутрь контейнера
docker network prune		Удаляет неиспользуемые сети
docker network rm	Название сети	Удаляет конкретную сеть

Single-host bridge networks

Создается интерфейс на хосте docker.



Один порт может занимать только один контейнер. Взаимодействие контейнеров внутри хоста.

Multi-host overlay networks

Для взаимодействия контейнеров внутри виртуальной сети нескольких хостов. Могут быть расположены на разных нодах swarm. Сеть второго уровня, распределяющаяся по нужным нодам с dns сервисом и распределением нагрузки.

Plugging into existing VLANs

Для прямого подключения к сетевому интерфейсу соответственно с независимым адресом. Необходим promiscuous mode на интерфейсе хоста (неразборчивый режим).

ipvlan с возможностью независимого адреса в пределах диапазона сетевой карты хоста заработал

```
services:
  condb:
    image: nginx
    networks:
      my_ipvlan:
        ipv4_address: 192.168.1.40

networks:
  my_ipvlan:
    driver: ipvlan
    driver_opts:
      parent: enp0s3
      ipvlan_mode: l2
    ipam:
      config:
        - subnet: 192.168.1.0/24
          gateway: 192.168.1.1
```

macvlan напрямую не заработал. Из интернетов:

```
sudo ip link add mymacvlan link enp0s3 type macvlan mode bridge
sudo ip addr add 192.168.1.99/24 dev mymacvlan
sudo ip link set mymacvlan up
```

Docker и firewall

Тут у меня началась веселуха. Первая проблема в том, что в литературе по докеру сетевая подсистема описана слабо, и не говорится самого главного: один хрен ip/nf tables (для определенности далее будем использовать iptables, помня обо всех нюансах). Docker самостоятельно управляет записями в iptables, за счет них организуется сетевая подсистема. Поэтому

1. Все службы вне докера управляются цепочкой INPUT
2. Службы докера, транслируемые наружу (ports в compose файле) по умолчанию попадают в цепочку FORWARD, которая сначала отдает пакеты в цепочку DOCKER-USER, затем в DOCKER-FORWARD и далее блокируется

Цепочка DOCKER-USER служит для пользовательского управления пакетами. Автоматически самим Docker не изменяется. Поэтому в ней можно блокировать например внешние запросы к внутренним сервисам, оставляя доступ для определенных адресов.

Yaml ??????

- комментарии

Ключ-значение

```
first: second
```

Обязательный пробел после :

Вложенные ресурсы в python виде, два пробела

Типы данных

Можно определять самим, можно отдавать на определение интерпретатору. Способ самостоятельного определения:

Тип данных	Без определения типа	Пример с определением
строка	myparam: 'first' myparam: first myparam: "first" Может быть без кавычек если хотя-бы один нечисловой символ	!!str myparam: !!str 'first'
Многостроковая переменная:	С сохранением переходов на новую строку: config: | server.port=8443 logfile=/var/log Без сохранения: config: > server.port=8443 logfile=/var/log	
целое число	myparam: 42	!!int myparam: !!int 42
число с плавающей точкой	myparam: 3.14	!!float myparam: !!float 3.14
булев тип	myparam: true (или false, True, False, yes, no, on, off)	!!bool myparam: !!bool true
пустое значение	myparam: null (или ~, пустая строка myparam:)	!!null myparam: !!null null

Тип данных	Без определения типа	Пример с определением
последовательность (список)	myparam: [1, 2, 3] myparam: - 1 - 2 - 3 Список словарей. env: - name: FIRSTVAR value: ONE - name: SECONDVAR value: TWO	!!seq myparam: !!seq [1, 2, 3]
словарь (отображение)	myparam: {key: value} myparam: key: value	!!map myparam: !!map {key: value}
дата и время	myparam: 2024-01-15T14:30:00Z myparam: 2024-01-15 14:30:00	!!timestamp myparam: !!timestamp 2024-01-15T14:30:00Z
двоичные данные (base64)	myparam: R0lGODlh... (YAML автоматически не распознает, лучше явно указывать !!binary)	!!binary myparam: !!binary R0lGODlh...
множество	В явном виде без !!set не поддерживается — нужно использовать список с уникальностью вне YAML.	!!set myparam: !!set {a, b, c}
упорядоченная карта	Без !!omap будет обычным словарём. Для порядка используют список пар: myparam: - a: 1 - b: 2	!!omap myparam: !!omap [{a: 1}, {b: 2}]
список пар ключ-значение	Без !!pairs — обычный список с одним элементом-словарём.	!!pairs myparam: !!pairs [key1: val1, key2: val2]
слияние карт	Без !!merge не работает — ключ << интерпретируется как обычная строка.	!!merge myparam: !!merge <<: *defaults

Есть типы данных python. Работают только в Python-парсерах YAML (PyYAML, ruamel.yaml) и не являются частью стандарта YAML 1.1/1.2. Использование в других языках приведёт к ошибкам или игнорированию:

Тип данных	Пример с определением
строка	!!python/str myparam: !!python/str 'hello'

Тип данных	Пример с определением
юникод-строка	!!python/unicode myparam: !!python/unicode 'привет'
байтовая строка	!!python/bytes myparam: !!python/bytes 'hello' (в base64)
целое число	!!python/int myparam: !!python/int 12345678901234567890 !!python/long длинное целое (устар.) myparam: !!python/long 42L
число с плавающей точкой	!!python/float myparam: !!python/float 3.14
комплексное число	!!python/complex myparam: !!python/complex 2+3j
булев тип	!!python/bool myparam: !!python/bool True
пустое значение	!!python/none myparam: !!python/none None
последовательность (список)	!!python/list myparam: !!python/list [1, 2, 3]
кортеж	!!python/tuple myparam: !!python/tuple (1, 2, 3) (в YAML запись: !!python/tuple [1, 2, 3])
словарь (отображение)	!!python/dict myparam: !!python/dict {a: 1, b: 2}
множество	!!python/set myparam: !!python/set {1, 2, 3} (в YAML: !!python/set [1, 2, 3])
произвольный объект класса	!!python/object myparam: !!python/object:module.Class {attr: value}
создание объекта через конструктор	!!python/object/new myparam: !!python/object/new:module.Class [arg1, arg2]
вызов функции/конструктора	!!python/object/apply myparam: !!python/object/apply:module.func [arg1, arg2]
ссылка на имя объекта	!!python/name myparam: !!python/name:module.func

Тип данных	Пример с определением
ссылка на модуль	!!python/module myparam: !!python/module:math
объект-строка	!!python/object/str myparam: !!python/object/str 'hello'
объект-список	!!python/object/list myparam: !!python/object/list [1, 2, 3]

Пример кода:

```
import yaml

# ВНИМАНИЕ: Этот код потенциально опасен!

yaml_str = """
functions:
  - !!python/name:builtins.print
    args: ["Hello from YAML!"]
  - !!python/name:__main__.my_custom_func
    args: [10, 20]
"""

def my_custom_func(x, y):
    return f"Custom result: {x * y}"

# Используем unsafe_load для поддержки !!python/name
data = yaml.unsafe_load(yaml_str)

for func in data['functions']:
    # func здесь - это объект функции
    args = func.get('args', []) if isinstance(func, dict) else []

    if callable(func):
        result = func(*args)
        print(f"Результат: {result}")
```

Однако использование python/... как я понял считается опасным. Лучше без этого.

Переменные

Есть подходы по повторному использованию значений, в классическом понимании переменных нет (да и это не нужно). Определение переменных.

```
# Определение якоря
defaults_something: &defaults
  timeout: 30
  retries: 3
  debug: false

# Использование алиаса
service1:
  <<: *defaults
  name: "service1"
  port: 8080

service2:
  <<: *defaults
  name: "service2"
  timeout: 60  # переопределяет значение из defaults
  port: 8081

# Пример с отдельными значениями
database: &db_config
  host: localhost
  port: 5432

development:
  database: *db_config
# в этом случае
developmen:
  database:
    host: localhost
    port: 5432

#Вложенное слияние
db_defaults: &db_defaults
  host: localhost
  port: 5432
  settings:
    pool_size: 10
```

```
    timeout: 30

production:
  database:
    <<: *db_defaults
    host: prod-db.com
    settings:
      <<: *db_defaults['settings']  # так нельзя
      # правильный способ:
      pool_size: 20
```

- Merge key (<<) работает только со словарями (maps)
- Порядок важен: переопределения должны идти после <<
- Не все парсеры YAML поддерживают << (стандарт YAML 1.2 его исключил, но большинство парсеров поддерживают)
- Алиасы (*name) и якоря (&name) работают везде

Множественные документы

Возможность хранить несколько независимых YAML-документов в одном файле, разделяя их тремя дефисами ---. Пример загрузки файла настроек с множественными документами:

```
import yaml

# Строка с несколькими документами
yaml_str = """
name: John
age: 30
---
users:
  - Alice
  - Bob
---
status: success
code: 200
"""

# Загрузка всех документов
documents = list(yaml.safe_load_all(yaml_str))

print(f"Загружено документов: {len(documents)}")
```

```
print(documents[0])  # {'name': 'John', 'age': 30}
print(documents[1])  # {'users': ['Alice', 'Bob']}
print(documents[2])  # {'status': 'success', 'code': 200}
```

Пример: конфигурация различных окружений

```
# Документ 1: Общие настройки
environment: common
database:
  host: localhost
  port: 5432
---
# Документ 2: Настройки разработки
environment: development
debug: true
database:
  host: dev-db.local
---
# Документ 3: Продакшн настройки
environment: production
debug: false
database:
  host: prod-db.example.com
  pool_size: 20
```

```
import yaml

def load_env_config(env_name):
    with open('config.yaml', 'r') as file:
        for doc in yaml.safe_load_all(file):
            if doc and doc.get('environment') == env_name:
                return doc
    return None

dev_config = load_env_config('development')
prod_config = load_env_config('production')

print(dev_config['debug'])   # True
print(prod_config['debug'])  # False
```

Перекрестные ссылки и якоря не работают между документами. Это полностью независимые документы.