



ОНЛАЙН-ОБРАЗОВАНИЕ



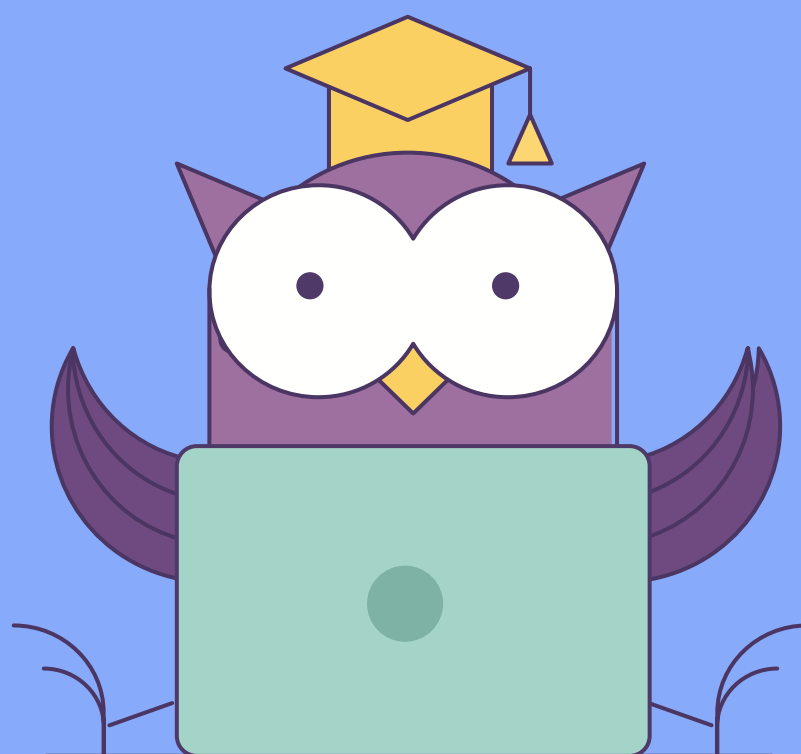
Управление пакетами

Курс «Администратор Linux»

Занятие № 6



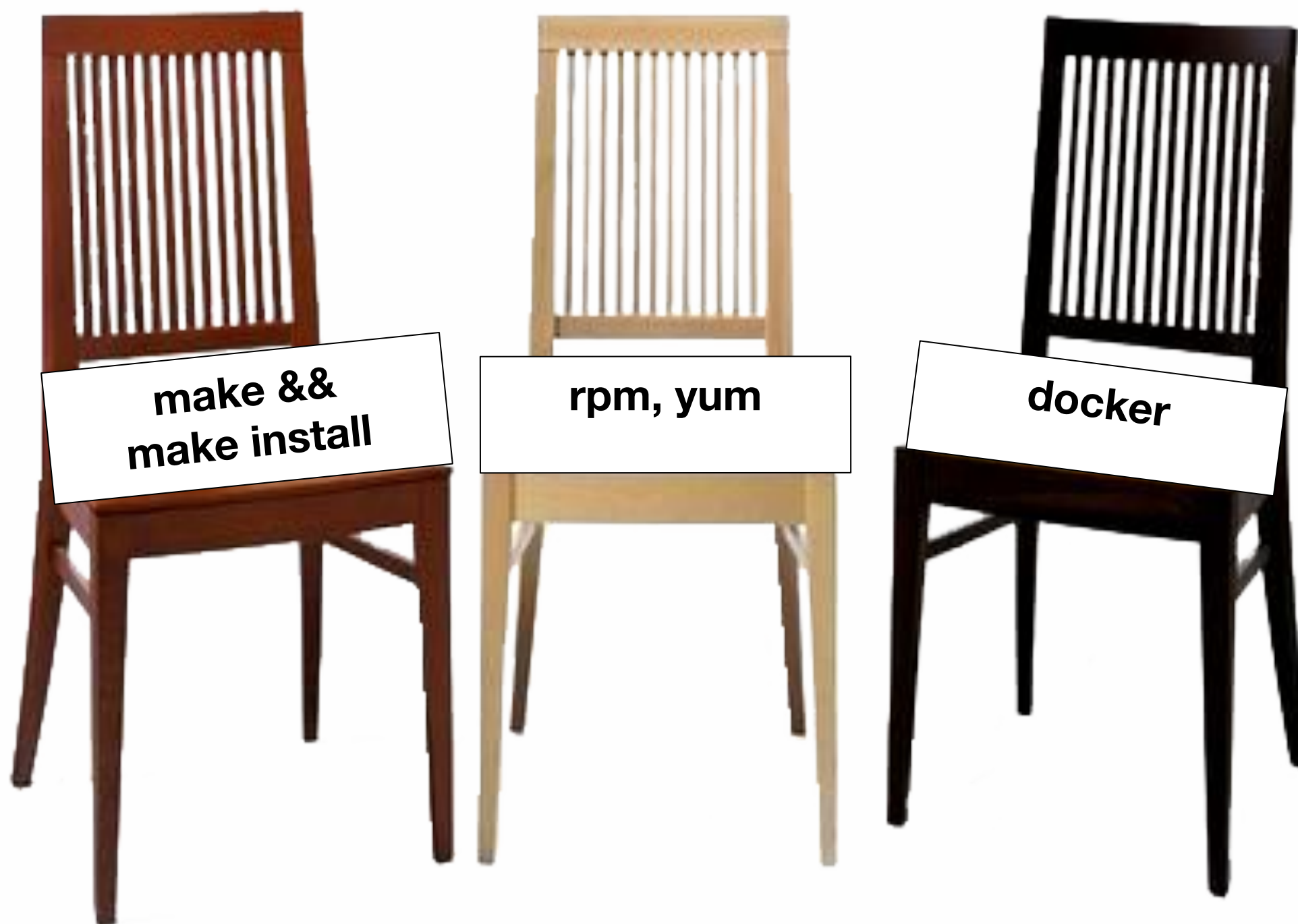
Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте  если все хорошо

Есть 3 стула





Дисклеймер:

Авторы абсолютно против использования этого метода в повседневной жизни. Бездумное использование «make install» опасно для операционной системы и вашей счастливой профессиональной жизни. Дальнейшее повествование имеет отрицательный эмоциональный окрас.

make && make install

O T U S

Инструмент контролирующей процесс сборки ПО из исходных файлов. Разработан в 1976 году в Bell Labs.

<https://www.gnu.org/software/make/manual/make.html>

make && make install

Самый простой метод, который описывается командой:

`./configure && make && make install`

Суть метода состоит просто в сборке и инсталляции необходимого ПО куда-то в операционную систему.

Make ожидает инструкции из текстового файла Makefile. В 99% случаев вы найдете этот файл рядом с исходниками программы.

```
make [ -f make-файл ] [ цель ] ...
```

Стандартные цели для сборки:

- all — выполнить сборку пакета;
- install — установить пакет из дистрибутива (производит копирование исполняемых файлов, библиотек и документации в системные каталоги);
- uninstall — удалить пакет (производит удаление исполняемых файлов и библиотек из системных каталогов);
- clean — очистить дистрибутив (удалить из дистрибутива объектные и исполняемые файлы, созданные в процессе компиляции);
- distclean — очистить все созданные при компиляции файлы и все вспомогательные файлы, созданные утилитой ./configure в процессе настройки параметров компиляции дистрибутива.

./configure --help - помощь в предварительной конфигурации сборки

Makefile синтаксис

```
all: program
```

```
program.o: program.c  
    gcc -c program.c -o program.
```

```
program: program.o  
    gcc program.o -o program
```

```
clean:  
    -rm -f program.o  
    -rm -f program
```

make & make install



Возможность использовать последнюю или, наоборот, устаревшую версию ПО, которой нет в репозиториях (но надо не лениться и написать spec'y)



Возможность собрать ПО с теми модулями/опциями, которые необходимы именно в вашем случае и для ваших целей (но надо не лениться и написать spec'y)



Отсутствие необходимости разбираться с созданием пакета и делать свои сложные «обертки» для сборки (но надо не лениться и написать spec'y, которая «make install» выполнит внутри себя)

make & make install

- 👍 Возможность использовать последнюю или, наоборот, устаревшую версию ПО, которой нет в репозиториях (но надо не лениться и написать spec'y)
- 👍 Возможность собрать ПО с теми модулями/опциями, которые необходимы именно в вашем случае и для ваших целей (но надо не лениться и написать spec'y)
- 👍 Отсутствие необходимости разбираться с созданием пакета и делать свои сложные «обертки» для сборки (но надо не лениться и написать spec'y, которая «make install» выполнит внутри себя)
- 👎 Требует наличия сборочного окружения на сервере
- 👎 *в случае компрометации сервера - облегчает задачу по сборке эксплоита злоумышленнику*
- 👎 *это окружение не является необходимым для выполнения приложения, оно нужно только для подготовки окружения*
- 👎 Занимает много времени.
- 👎 **Ручное разрешение зависимостей!!!!**
- 👎 Сложность тиражирования.
- 👎 Возможны затруднения с unit'ами и init-скриптами
- 👎 Больше затрат на поддержание системы в целом за счет необходимости самостоятельно следить за обновлениями безопасности

make & make install

- 👍 Возможность использовать последнюю или, наоборот, устаревшую версию ПО, которой нет в репозиториях (но надо не лениться и написать spec'y)
- 👍 Возможность собрать ПО с теми модулями/опциями, которые необходимы именно в вашем случае и для ваших целей (но надо не лениться и написать spec'y)
- 👍 Отсутствие необходимости разбираться с созданием пакета и делать свои сложные «обертки» для сборки (но надо не лениться и написать spec'y, которая «make install» выполнит внутри себя)
- 👎 Требует наличия сборочного окружения на сервере
- 👎 *в случае компрометации сервера - облегчает задачу по сборке эксплоита злоумышленнику*
- 👎 *это окружение не является необходимым для выполнения приложения, оно нужно только для подготовки окружения*
- 👎 Занимает много времени.
- 👎 **Ручное разрешение зависимостей!!!**
- 👎 Сложность тиражирования.
- 👎 Возможны затруднения с unit'ами и init-скриптами
- 👎 Больше затрат на поддержание системы в целом за счет необходимости самостоятельно следить за обновлениями

На macOS также не стоит использовать «make & make install» для установки ПО при наличии пакетного менеджера brew 🍺. С помощью «brew edit» можно отредактировать формулу, например добавить флаги сборки, и переустановить пакет.

make && make install

Самая ключевая проблема этого способа установки ПО заключается в **стремлении к хаосу**.

Подробнейший разбор и холивар есть в очень хорошей статье:

<https://otus.pw/ulZn/>





RPM

- RedHat Package Manager - написан на C+Perl, первая версия увидела свет в 1997 году
- В ОС устанавливается уже собранное ПО, поддерживаемое командой поддержки репозитория/дистрибутива.
- RPM-пакет можно рассматривать как умный gzip-архив
- Важно понимать, что при установке не происходит и не должно происходить сборки/компиляции ПО

RPM

- 👍 Экстремально низкие трудозатраты на установку ПО
- 👍 ПО поддерживается maintainer'ами репозитория/дистрибутива
- 👍 Большинство пакетов разбито на binary/dev-подпакеты, нет мусора в ОС
- 👍 Для установки ПО не нужно окружение для сборки, меньше софта в ОС
- 👍 Возможность автоматически выполнять скрипты при установке/удалении
- 👎 Замороженные версии софта, не всегда последние
- 👎 Ручное разрешение зависимостей!!!

Пакеты представляют из себя архив определенного формата, в нем содержится:

- Метаинформация:
 - Имя
 - Версия
 - Релиз
 - Архитектура
 - Зависимости
 - Ресурсы
 - ...
- Файлы
- Скриптлеты
 - pre-install
 - post-install
 - pre-remove
 - post-remove
 - ...

RPM: управление

- `rpm -q {name}` — проверить установлен ли пакет {name}
- `rpm -qi {name}/{pkg}` — показать мета-информацию о пакете
- `rpm -qp --queryformat %{VERSION}-%{RELEASE} {pkg}` — формат вывода информации
- `rpm -e {name}` — удалить пакет
- `rpm -i {pkg}` — установить пакет
- `rpm -ql {name}/{pkg}` — вывести список файлов пакета
- `rpm -q --scripts` — показать скриптлеты
- `rpm -qR` — показать от каких пакетов зависит этот
- `rpm -q --provides {name}` — показать ресурсы предоставляемые пакетом
- `rpm -qf {file}` — показать какому пакету принадлежит {file}.
- `rpm2cpio {pkg} | cpio -idmv` — распаковать содержимое пакета

RPM: верификация

- `rpm -Vf tree.rpm`
 - `rpm -Va`
 - `rpm -Vp tree.rpm`
 - `rpm -Vv mc`
-
- 5 — контрольная сумма MD5
 - S — размер
 - L — символическая ссылка
 - T — дата изменения файла
 - D — устройство
 - U — пользователь
 - G — группа
 - M — режим (включая разрешения и тип файла)
 - ? — файл не удалось прочитать

RPM: пример из жизни

```
rm -f /var/lib/rpm/__db*  
db_verify /var/lib/rpm/Packages  
rpm --rebuilddb  
yum clean all
```

RPM:сборка

```
sudo yum install rpmdevtools rpm-build
```

```
rpmdev-setuptree
```

```
tree -d -L 1 ~/rpmbuild
```

```
~/rpmbuild
```

- |— BUILD # директория в которой происходит сборка
- |— RPMS # директория с собранными пакетами
- |— SOURCES # директория с исходными файлами
- |— SPECS # директория с спес-файлами
- |— SRPMS # директория с SRPM-пакетами

RPM: спец-файл

Это описание ПО вместе с инструкциями как построить пакет и списком файлов для всех устанавливаемых файлов.

имя пакета - тире - номер версии - тире - номер выпуска (релиза) - точка - спец

vim-8.1.2.spec

Заголовок

Summary: Это однострочное описание пакета.

Name: Это должна быть строка имени из имени файла rpm, которое вы планируете использовать.

Version: Это должна быть строка версии из имени файла rpm, которое вы планируете использовать.

Release: Это номер выпуска для пакета с той же самой версией (например, если мы сделали пакет и обнаружили, что он незначительно неисправный и нам необходимо сделать его заново, то следующий пакет будет номер выпуска 2).

Icon: можно установить иконку по умолчанию

Source: указание на файл с исходниками. Можно указывать несколько файлов через Source0, Source1, etc.

Path: указание на файлы с патчами

RPM: spec-файл

Раздел Prep

Здесь располагаются команды для подготовки пакета к команде make (исправления в исходниках, установка доп. пакетов). По сути сюда можно вставить скрипт и он все сделает сам. НО! Есть встроенные макросы для удобства:

%setup - распаковывает исходные тексты и делает cd к ним

%patch - накладывает патчи автоматически

Раздел Build

Раздел для построения ПО. Макросов нет, но так же можно просто запустить скрипт для сборки

Раздел Install

Раздел для установки ПО в операционную систему. Если есть make install, то можно подставить сюда

- %pre макрос для выполнения предустановочного скрипта.
- %post макрос для выполнения послеустановочного скрипта.
- %preun макрос для выполнения скрипта перед удалением пакета.
- %postun макрос для скрипта выполняемого после удаления пакета.

RPM: спец-файл

Раздел File

Это раздел где вы обязаны перечислить все файлы для двоичного пакета. У RPM нет механизма самостоятельно узнать какие файлы были установлены с помощью `make install`.

`%doc` - обозначение для документации. Будут установлены в директорию `/usr/doc/$NAME-$VERSION-$RELEASE`

`$config` - обозначение для конфигурационных файлов в пакете.

`$dir` - директория, которой “владеет” пакет.

`%files -f <filename>` - можно взять список файлов из заранее подготовленного файла, а не перечислять руками

RPM: сборка

`rpmbuild -bb otus.spec` — сборка RPM

`rpmbuild -bs otus.spec` — сборка SRPM

`rpmbuild -ba otus.spec` — сборка RPM+SRPM

RPM: hint

```
sudo yum install -y yum-utils
```

```
yumdownloader --source <package>
```

```
sudo yum-builddep <package|specfile>
```

RPM: mock

- `sudo yum install mock mock-rpmsfusion-free`
- `sudo usermod -a -G mock $(whoami)`
- `newgrp mock`
- `spectool -g -R clogtail.spec`
- `rpmbuild -bs clogtail.spec`
- `ls /etc/mock`
- `mock -r epel-7-x86_64 --rebuild
~/rpmbuild/SRPMS/clogtail-0.3.0-2.fc28.src.rpm`

Yum

Yum является программой управления и автоматизации работы с пакетным менеджером **rpm (Redhat Packet Manager)**.

Сам по себе rpm способен только проинсталлировать пакет в систему, проверив установлены ли пакеты-зависимости.

Yum позволяет реализовать скачивание пакета, автоматическое разрешение зависимостей, обновления, добавление репозиториев.

Yum: приготовление

/etc/yum.conf — конфиг в ini-формате

[root@otus ~]# cat /etc/yum.conf

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=1
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
gpgcheck=1
plugins=1
proxy=http://proxy.mydomain.com:3128
proxy_username=user_proxy
proxy_password=pass_user_proxy
```

**/etc/yum.repos.d/*.repo — файлы в ini-формате
описывающие репозитории**

[root@otus ~]# cat /etc/yum.conf.d/go.repo

```
[c7-go-repo]
name=go-repo - CentOS
baseurl=https://mirror.go-repo.io/centos/7/$basearch/
gpgcheck=0
enabled=1
```

Yum: управление

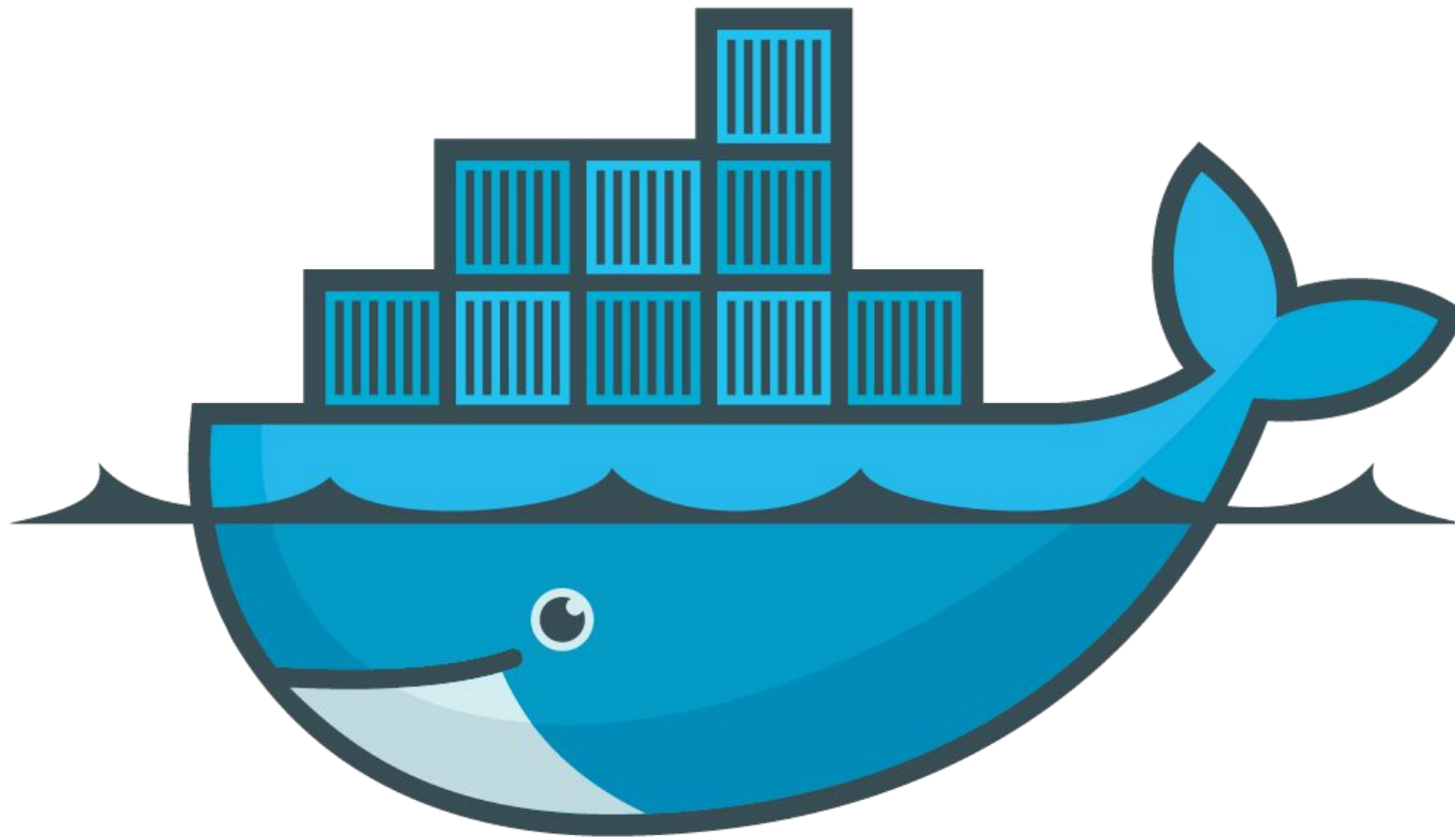
- `search` — поиск пакета
- `install` — установка пакета(ов)
- `update` — обновление (до версии)
- `downgrade` — откат до до версии
- `check-update` — проверка обновлений
- `remove` — удаление пакета
- `info` — информация о пакете
- `provides` — найти из какого пакета файл
- `shell` — CLI

Yum: полезное

- `yum updateinfo list security all`
- `yum updateinfo list security installed`
- `yum -y update --security`
- `yum update-minimal --security -y`
- `yum update --cve <CVE>`
- `yum updateinfo list cves`
- `yum updateinfo list`
- `yum info-sec`

Yum: свой репозиторий

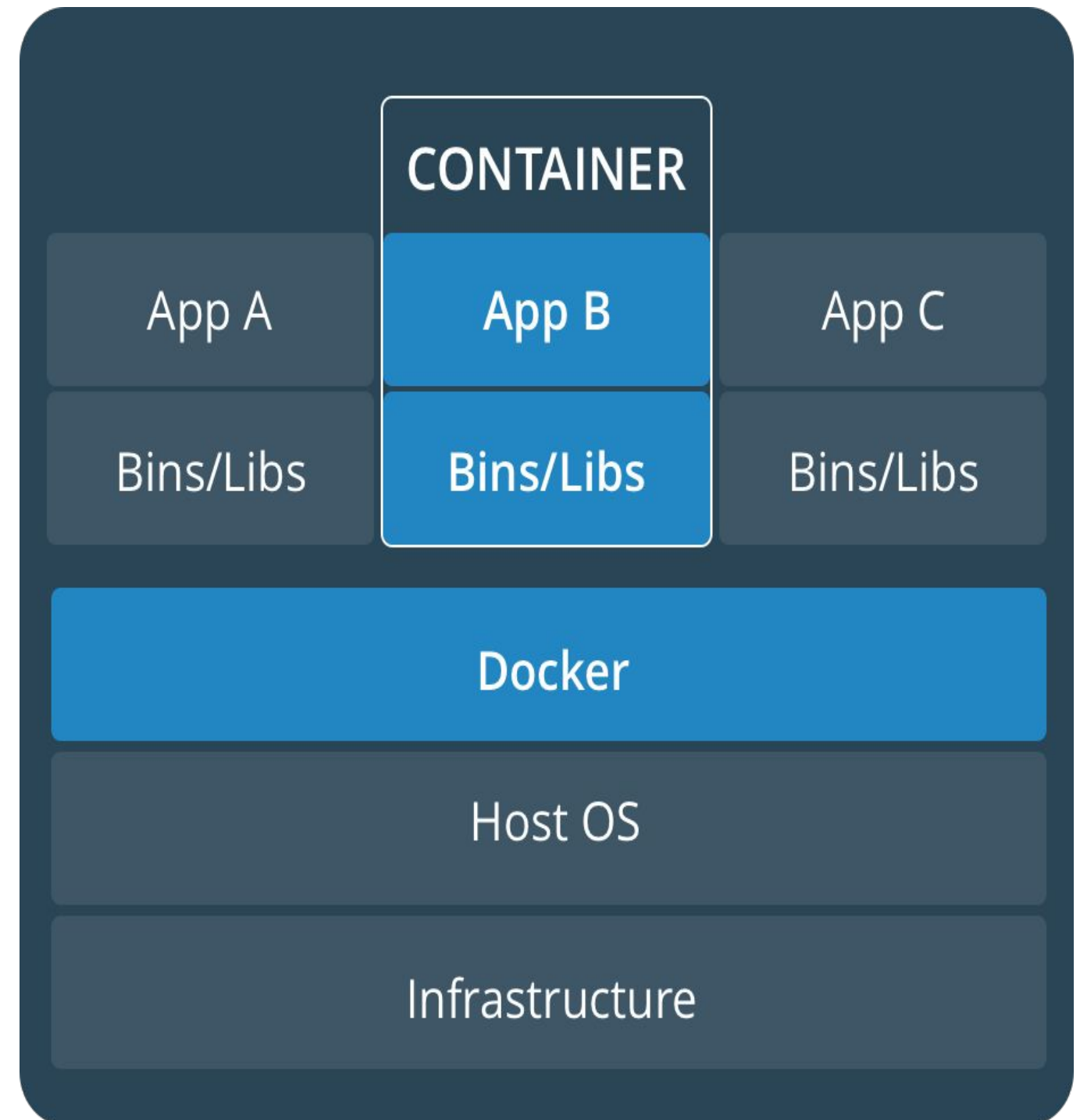
- `sudo yum install createrepo`
- `sudo mkdir -p /repos/CentOS/7/`
- `sudo createrepo /repos/CentOS/7/`
- `rsync -avz rsync://mirror.truenetwork.ru/centos/7.5.1804/
/repos/CentOS/7/`
- `[local-base]`
- `name=Local-Base`
- `baseurl=file:///repos/CentOS/7/$basearch/`
- `enabled=0`



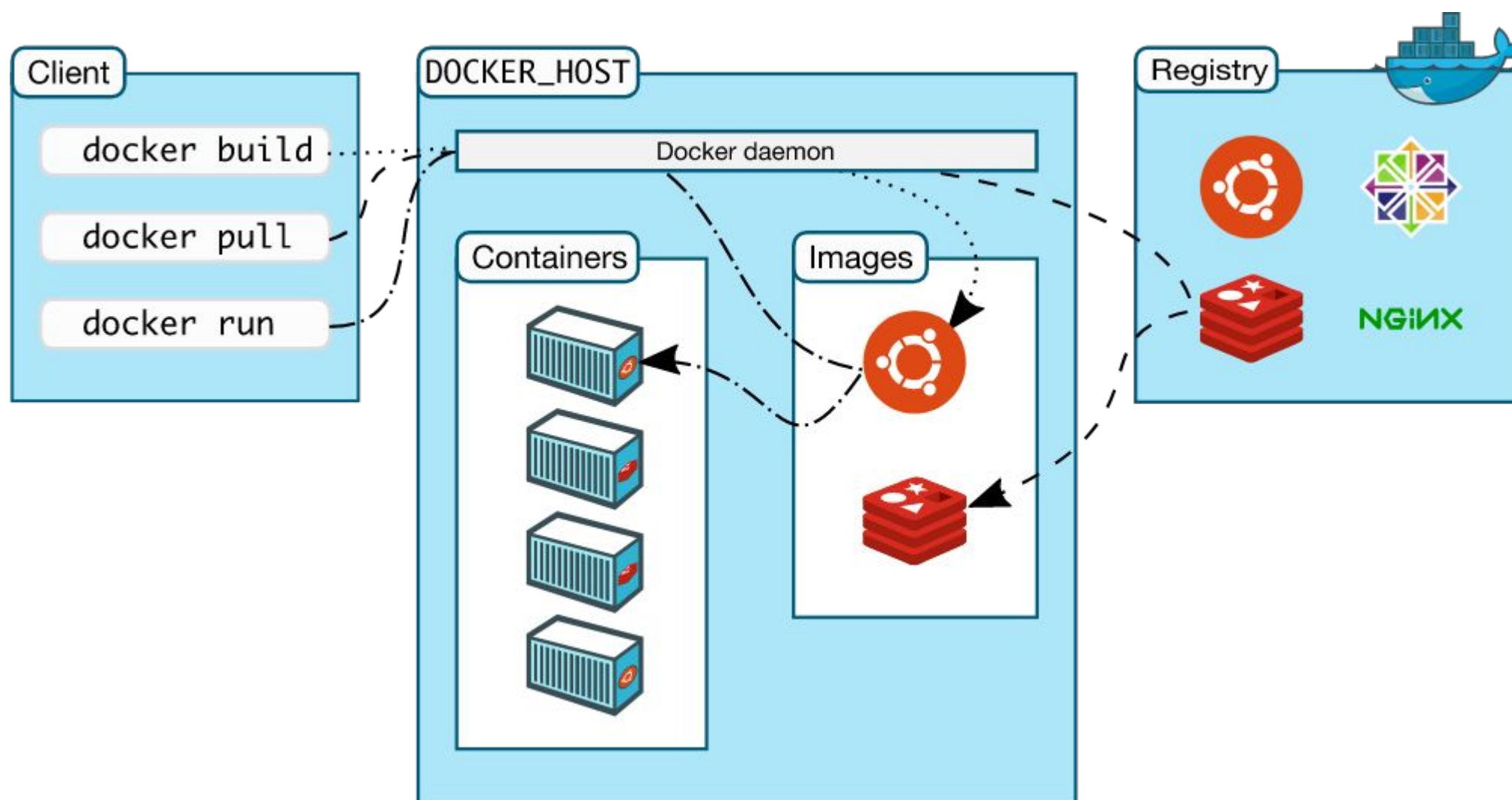
docker

Yum: свой репозиторий

- Docker - это про стандартизацию, иммутабельность и воспроизводимость
- Контейнер это не виртуальная машина. Это приложение и его окружение, упакованные в изолированное окружение
- НО! Мы можем ограничивать ресурсы потребляемые контейнером.



Yum: свой репозиторий



```
# Родительский образ
FROM python:2.7-slim

# Устанавливаем рабочую директорию /app
WORKDIR /app

# Копируем содержимое в директорию /app
COPY . /app

# Установка необходимых зависимостей
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Порт 80 будет доступен снаружи
EXPOSE 80

# Установка переменной окружения
ENV NAME World

# Команда которая будет выполнена при запуске контейнера
CMD ["python", "app.py"]
```

Таким образом по аналогии с BUILDROOT формируется слепок ФС, который разворачивается в виде контейнера:

```
docker build -t registry.otus.ru/otus/otus -f Dockerfile .  
    docker push registry.otus.ru/otus/otus  
    docker run registry.otus.ru/otus/otus:latest
```

Ваши вопросы?

Yum: свой репозиторий

- 1) Создать свой RPM пакет (можно взять свое приложение, либо собрать, например, апач с определенными опциями)
- 2) Создать свой репозиторий и разместить там ранее собранный RPM

Реализовать это все либо в Vagrant, либо развернуть у себя через nginx и дать ссылку на репозиторий.

* Написать свой Dockerfile, собрать Image и разместить его в Docker Registry. Как результат прислать ссылку на Image и краткую инструкцию.

**Заполните, пожалуйста,
опрос в ЛК о занятии**

**Спасибо
за внимание!**

До встречи в Slack и на вебинаре

