Application Containers and System Services

Honza Horak <hhorak@redhat.com>
FLOCK, Krakow, August 2016

My experiences with Red Hat containers

- Input: sets of RPM packages
- Output: One set of container images
- Requirements:
  - usable on bare metal
  - designed for PaaS (OpenShift)

What this talk includes

1. Container basics
2. PostgreSQL container
3. Python container for building applications
4. System containers
5. Tools containers
6. Building infrastructure

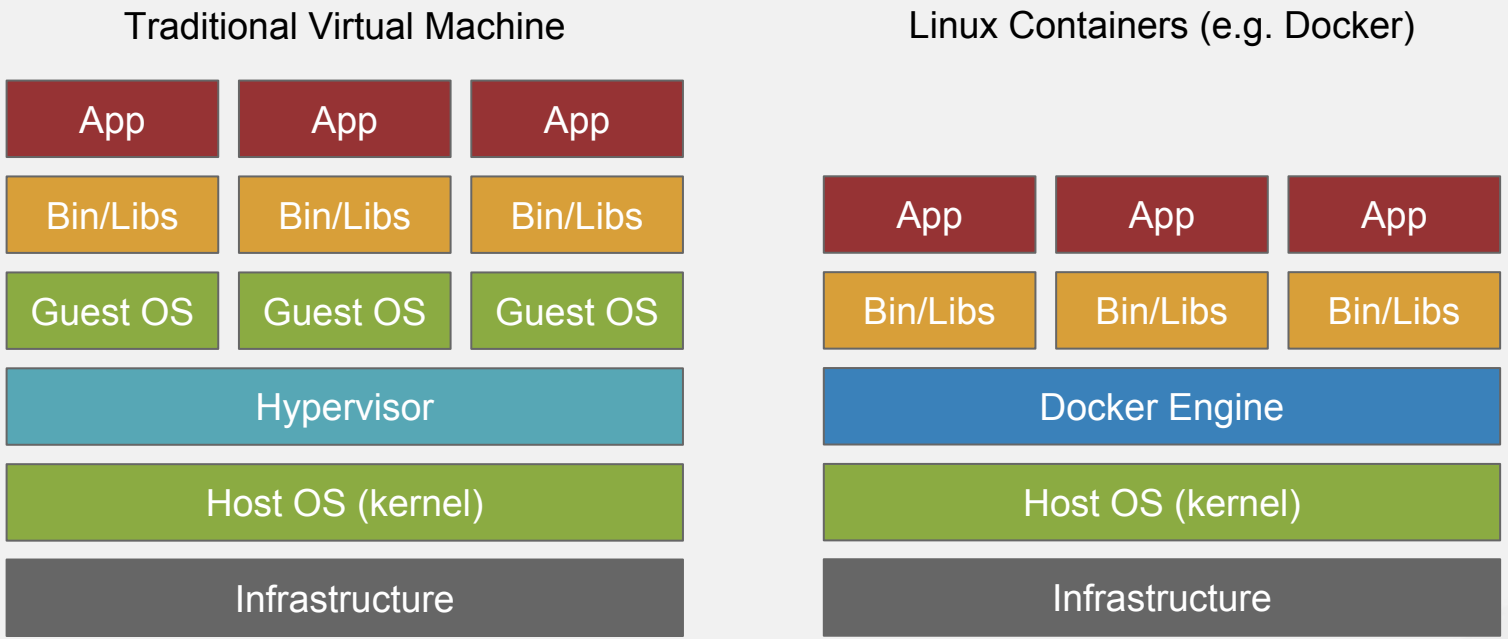
Disclaimer
docker ~ container
examples are simplified

1. CONTAINERS BASICS

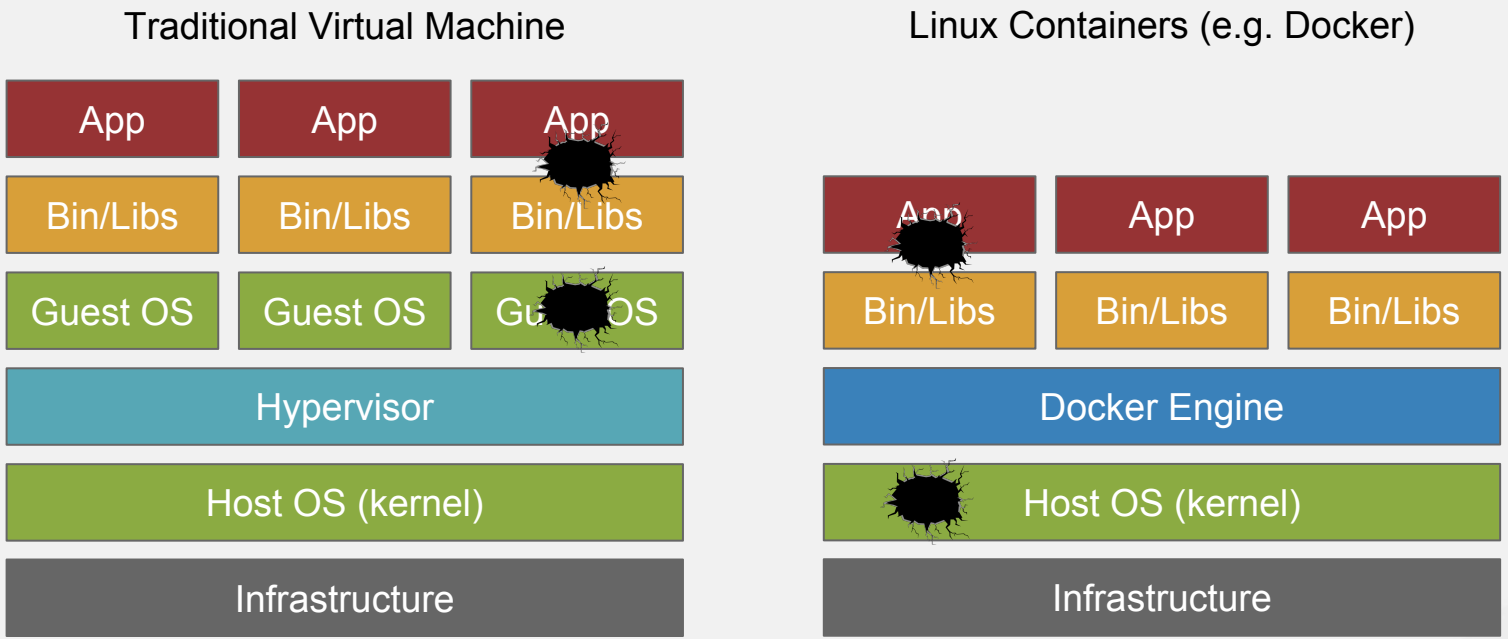

ВАТ "Київспецтранс"
ТЕЛ. 400-10-11

Tip #0:
Content matters.

Applications as micro services

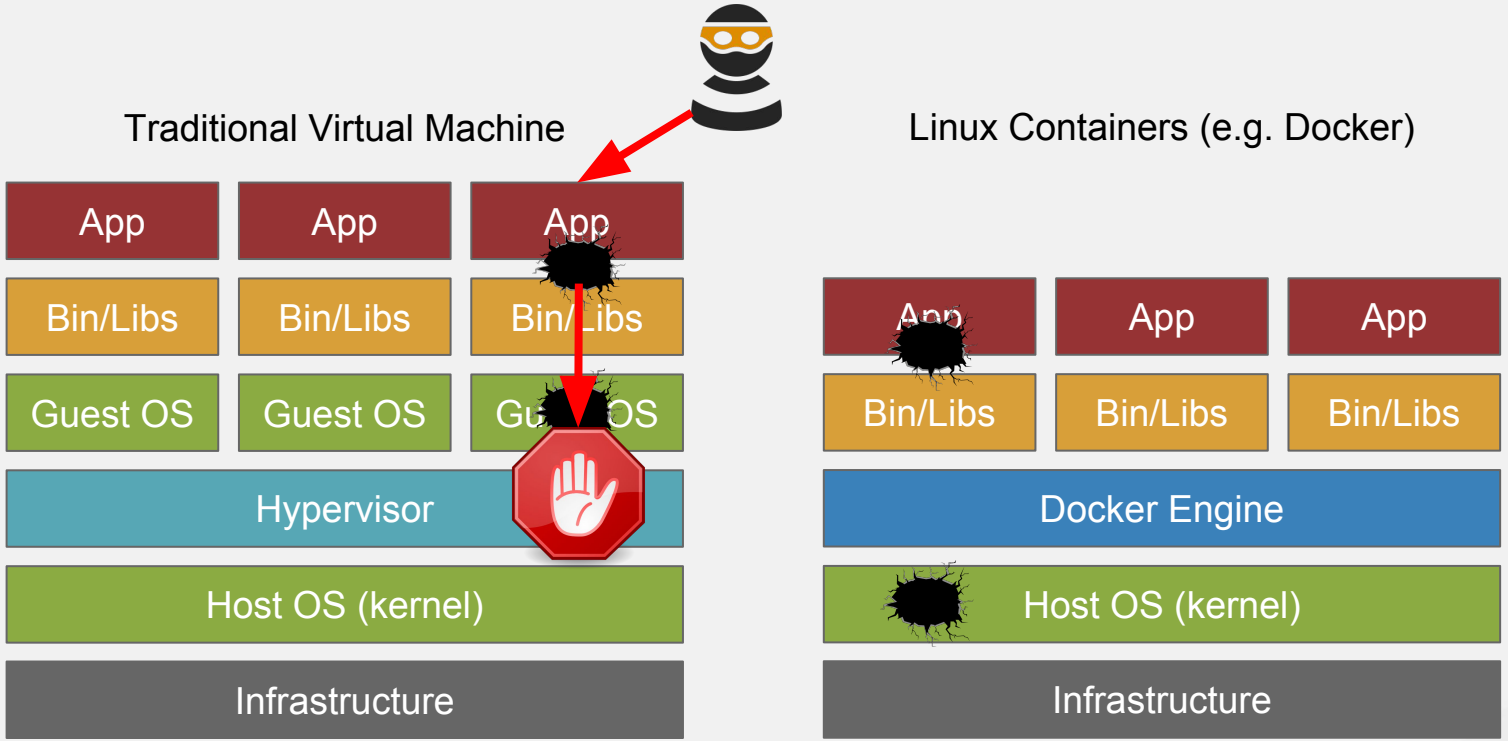


Container is not a virtual machine



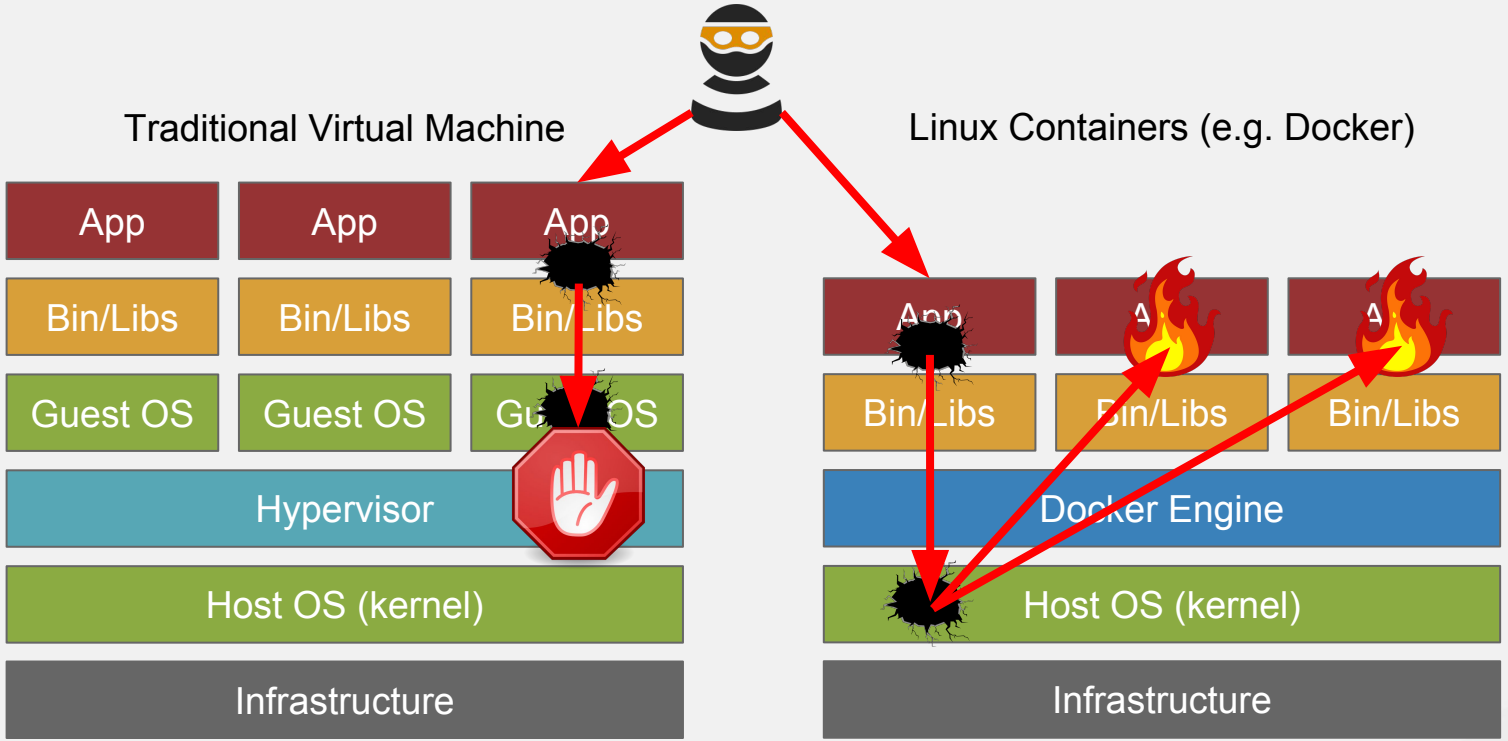
fedora

Container is not a virtual machine



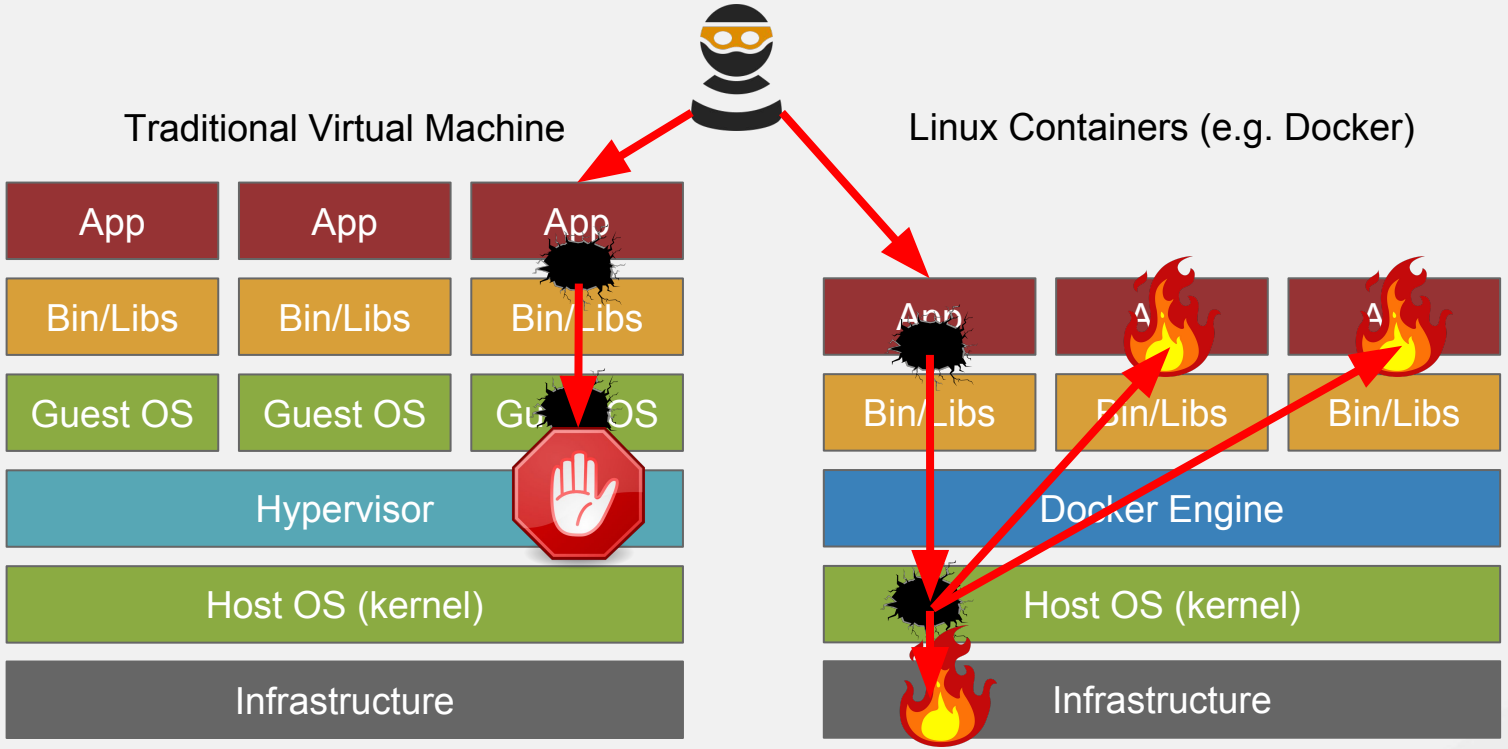
fedora

Container is not a virtual machine



fedora

Container is not a virtual machine



fedora

Tip #1:
Container is not a virtual machine.

Building the first container

```
#> dnf install -y docker
```

Building the first container

```
#> dnf install -y docker
#> systemctl start docker
```

Building the first container

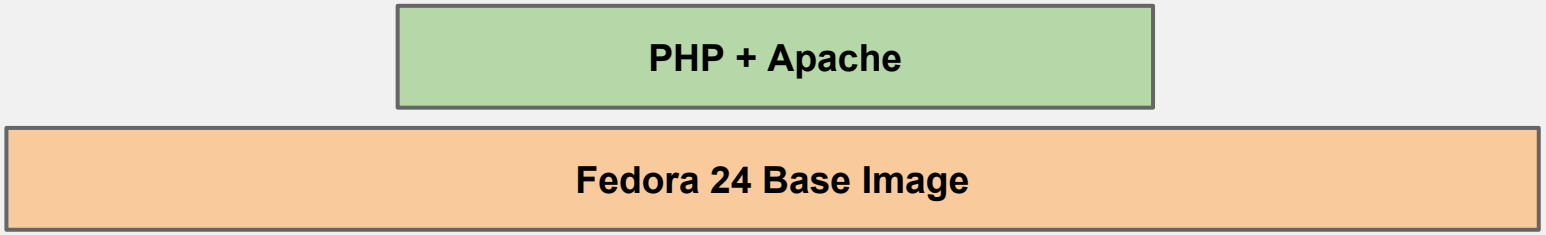
```
#> dnf install -y docker
#> systemctl start docker
#> docker pull fedora:24
```

Docker and layers

Base image provides smallest distro.

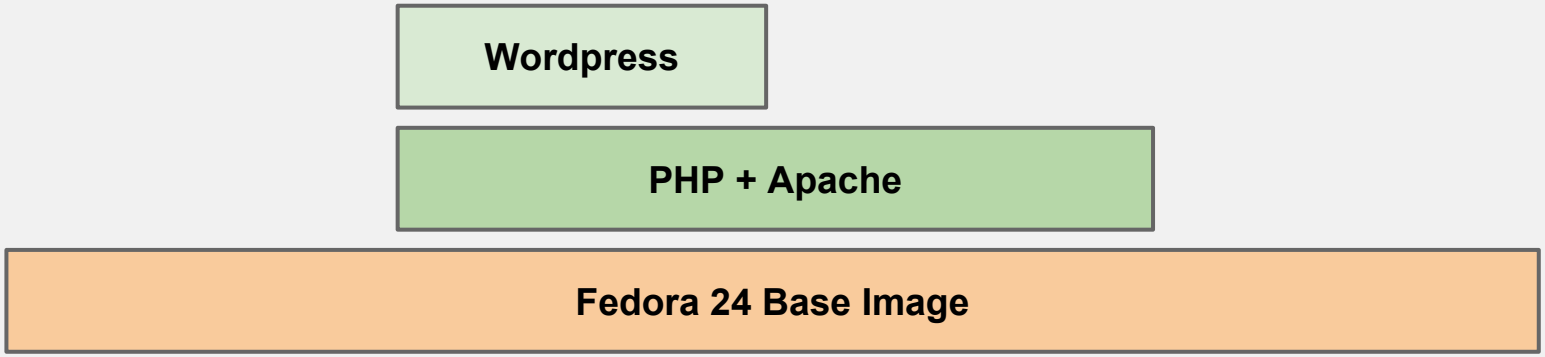
Docker and layers

A layered image builds extends existing base image.



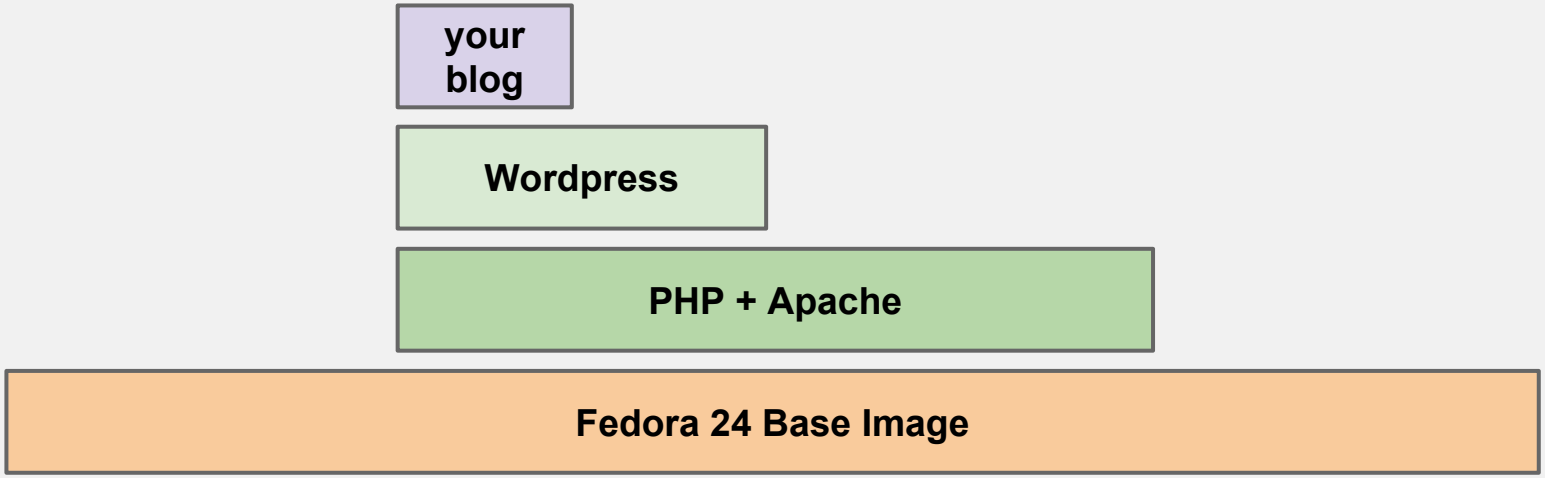
Docker and layers

Common data are stored only once.



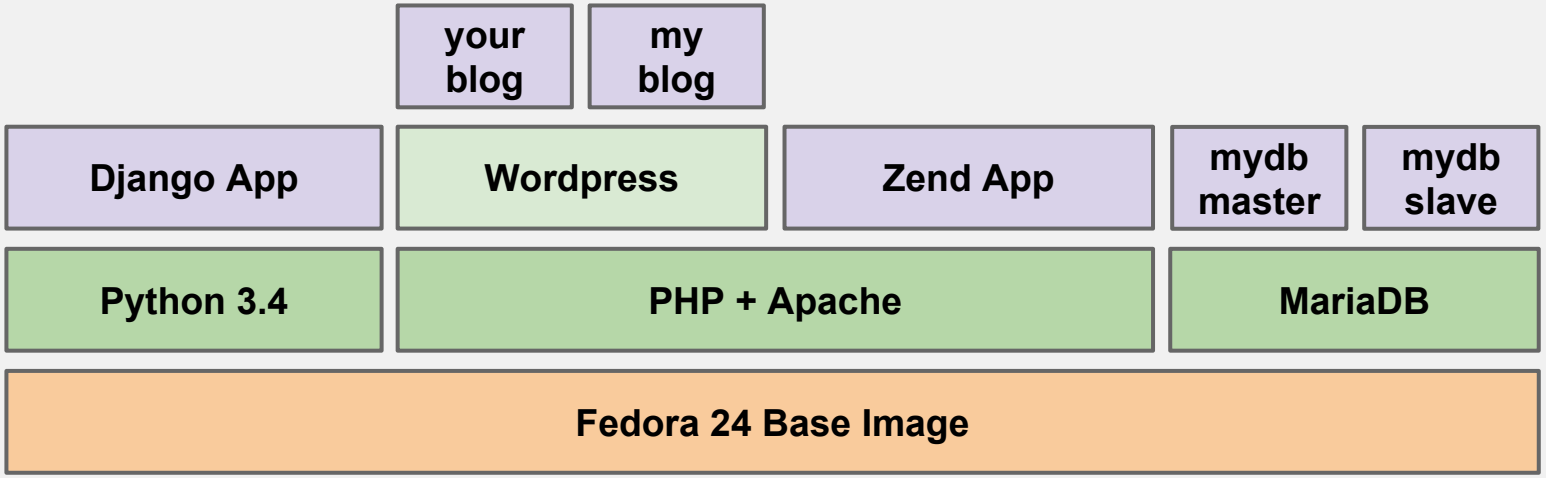
Docker and layers

Users are free to create another flavor for their specific needs.



Docker and layers

“Your mother was right, it’s better to share.”



fedora

Building the first container

```
#> dnf install -y docker
#> systemctl start docker
#> docker pull fedora:24

#> docker run -ti --name mycont fedora:24 bash
[root@a1eefecdacfa /]# |
```

Building the first container

```
#> dnf install -y docker
#> systemctl start docker
#> docker pull fedora:24

#> docker run -ti --name mycont fedora:24 bash
[root@a1eefecdacfa /]# echo Hello FLOCK > /root/greeting
[root@a1eefecdacfa /]# |
```

Building the first container

```
#> dnf install -y docker
#> systemctl start docker
#> docker pull fedora:24

#> docker run -ti --name mycont fedora:24 bash
[root@a1eefecdacfa /]# echo Hello FLOCK > /root/greeting
[root@a1eefecdacfa /]# exit

#> docker commit mycont
|
```

Building the first container

```
#> dnf install -y docker
#> systemctl start docker
#> docker pull fedora:24

#> docker run -ti --name mycont fedora:24 bash
[root@a1eefecdacfa /]# echo Hello FLOCK > /root/greeting
[root@a1eefecdacfa /]# exit

#> docker commit mycont
0bdcfc5ba0602197e2ac4609b8101dc8eaa0d8ab114f542ab6b2f15220d0ab22
```

Tip #2:
Use reproducible builds.

Building the first container correctly

```
#> cat Dockerfile
FROM fedora:24
RUN echo Hello FLOCK > /root/greeting

#>
```

Building the first container correctly

```
#> cat Dockerfile
FROM fedora:24
RUN echo Hello FLOCK > /root/greeting

#> docker build .
#>
```

2. CREATING POSTGRESQL CONTAINER

Installing RPMs in a container

```
#> cat Dockerfile
FROM fedora:24
RUN dnf -y install postgresql-server

#> docker build .
```

Installing RPMs in a container

```
#> docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM fedora:24
 ---> c453594215e4

...

Installed:
  postgresql-server.x86_64 0:9.5.3-1.fc24

Dependency Installed:
  postgresql.x86_64 0:9.5.3-1.fc24    postgresql-libs.x86_64 0:9.5.3-1.fc24
  systemd-sysv.x86_64 0:229-1.fc24

Complete!
 ---> 1ff2f2d0bc66
Removing intermediate container 2a86d8a78b75
Successfully built 1ff2f2d0bc66
```

Tip #3:
Make containers small.

Installing RPMs in a container

```
#> cat Dockerfile
FROM fedora:24
RUN dnf -y --setopt=tsflags=nodocs install postgresql-server && \
    dnf clean all

#> docker build .

#> docker build -t postgresql .
```

Installing RPMs in a container

```
#> docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM fedora:24
 ---> c8a648134623
Step 2 : RUN dnf -y --setopt=tsflags=nodocs install postgresql-server
&& dnf clean all
 ---> Using cache
 ---> 29036308c1ec
Successfully built 29036308c1ec
```

Tip #4:
Be careful about docker cache.

Installing RPMs in a container

```
#> cat Dockerfile
FROM fedora:24
RUN dnf -y --setopt=tsflags=nodocs install postgresql-server && \
    dnf clean all

#> docker build --no-cache=true .
```

Tip #5:
Squash images.

Installing RPMs in a container

```
#> docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM fedora:24
 ---> c8a648134623
Step 2 : RUN dnf -y --setopt=tsflags=nodocs install postgresql-server
&& dnf clean all
 ---> Using cache
 ---> 29036308c1ec
Successfully built 29036308c1ec
```

Squashing containers

Have only one layer instead of dozen of intermediate layers.

```
#> docker build -t fedora/postgresql:9.5 .

#> dnf -y install python3-docker-squash

#> docker-squash -f fedora:24 fedora/postgresql:9.5
```

Are we there yet?

Tip #6:
Do something clever when starting container.

Make container do something

```
#> cat Dockerfile

FROM fedora:24

RUN dnf -y --setopt=tsflags=nodocs install postgresql-server && \
    dnf clean all

ENV HOME=/var/lib/pgsql \
    PGDATA=/var/lib/pgsql/data \
    PGUSER=postgres

ADD run-postgresql /usr/bin/
CMD [ "/usr/bin/run-postgresql" ]
```

Make container do something

```
#> cat run-postgresql

#!/bin/bash

initdb

echo "host all all 0.0.0.0/0 md5" >${PGDATA}/pg_hba.conf
echo "listen_addresses = '*'" >${PGDATA}/postgresql.conf

exec postgres "$@"
```

Tip #7:
Use exec for final process.

Tip #7:
Use exec for final process.
Or use systemd inside container?

Systemd inside container?

+ collects **zombies**
+ allows to run system containers (with more daemons in one container)
+ logging to journald

- not straightforward, requires additional docker arguments
- requires running containers as **root** (right?)

```
$> cat Dockerfile
...
RUN systemctl enable httpd.service
CMD ["/sbin/init"]
```

Tip #8:
Use non-root user wherever possible.

Run as non-root user by default

```
#> cat Dockerfile

FROM fedora:24

RUN dnf -y --setopt=tsflags=nodocs install postgresql-server && \
    dnf clean all

ENV HOME=/var/lib/pgsql
ENV PGDATA=/var/lib/pgsql/data
ENV PGUSER=postgres
USER postgres

ADD run-postgresql /usr/bin/
CMD [ "/usr/bin/run-postgresql" ]
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> |
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti --name p1 postgresql

#> |
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti --name p1 postgresql

#> docker inspect --format='{{.NetworkSettings.IPAddress}}' p1
172.17.0.2

#> |
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti --name p1 postgresql

#> docker inspect --format='{{.NetworkSettings.IPAddress}}' p1
172.17.0.2

#> psql -h 172.17.0.2
Password: _
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti --name p1 postgresql

#> docker inspect --format='{{.NetworkSettings.IPAddress}}' p1
172.17.0.2

#> psql -h 172.17.0.2
Password: _
```

Tip #9:
Do not use default passwords.

Connecting to PostgreSQL container

```
#> cat run-postgresql

...
echo "host all all 0.0.0.0/0 md5" >${PGDATA}/pg_hba.conf
echo "local all postgres peer" >>${PGDATA}/pg_hba.conf
echo "listen_addresses = '*'" >${PGDATA}/postgresql.conf

# start postgres daemon listening on local socket only
pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Connecting to PostgreSQL container

```
#> cat run-postgresql

...
echo "host all all 0.0.0.0/0 md5" >${PGDATA}/pg_hba.conf
echo "local all postgres peer" >>${PGDATA}/pg_hba.conf
echo "listen_addresses = '*'" >${PGDATA}/postgresql.conf

# start postgres daemon listening on local socket only
pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Connecting to PostgreSQL container

```
#> cat run-postgresql

...
echo "host all all 0.0.0.0/0 md5" >${PGDATA}/pg_hba.conf
echo "local all postgres peer" >>${PGDATA}/pg_hba.conf
echo "listen_addresses = '*'" >${PGDATA}/postgresql.conf

# start postgres daemon listening on local socket only
pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Connecting to PostgreSQL container

```
#> cat run-postgresql

...
echo "host all all 0.0.0.0/0 md5" >${PGDATA}/pg_hba.conf
echo "local all postgres peer" >>${PGDATA}/pg_hba.conf
echo "listen_addresses = '*'" >${PGDATA}/postgresql.conf

# start postgres daemon listening on local socket only
pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Connecting to PostgreSQL container

```
#> docker run -ti -d --name p1 \
              -e POSTGRESQL_ADMIN_PASSWORD=pass postgresql
b1e23c844346d2788d7b7891d8f78244788f71b19dcf291b05cdf1d7685ef556

#>
```

Connecting to PostgreSQL container

```
#> docker run -ti -d --name p1 \
              -e POSTGRESQL_ADMIN_PASSWORD=pass postgresql
b1e23c844346d2788d7b7891d8f78244788f71b19dcf291b05cdf1d7685ef556

#> psql -h 172.17.0.2 -U postgres
Password for user postgres: |
```

Connecting to PostgreSQL container

```
#> docker run -ti -d --name p1 \
              -e POSTGRESQL_ADMIN_PASSWORD=pass postgresql
b1e23c844346d2788d7b7891d8f78244788f71b19dcf291b05cdf1d7685ef556

#> psql -h 172.17.0.2 -U postgres
Password for user postgres:
psql (9.5.3, server 9.5.3)
Type "help" for help.

postgres=# _
```

...or mount file with password

1. User creates a file with password
2. File is bind-mounted (docker run -v) to the container
3. Password is not in plain-text in the variable (seen in docker daemon)

```
#> read -s mypass

#> echo "$mypass" >/root/pgpass

#> docker run -v /root/pgpass:/var/lib/pgsql/password ...
```

How to configure such a database?

Configuring PostgreSQL container

```
#> cat run-postgresql

...

echo "max_connections = ${POSTGRESQL_MAX_CONNECTIONS}"
>>${PGDATA}/postgresql.conf

...
```

Example of PostgreSQL 9.5 container

```
#> docker run -d \
          -p 5432:5432 \
          -e POSTGRESQL_ADMIN_PASSWORD=secret \
          -e POSTGRESQL_MAX_CONNECTIONS=10 \
          -e POSTGRESQL_USER=guestbook \
          -e POSTGRESQL_PASSWORD=pass \
          -e POSTGRESQL_DATABASE=guestbook \
          -v /db:/var/lib/pgsql/data:Z \
          fedora/postgresql:9.5
```

Tip #10:
Support the **most common** configuration options

Tip #10:
Support the **most common** configuration options
and **allow users to build their own flavours.**

Allow to extend the container

```
#> cat post-init-hooks/post-init
echo "lock_timeout=${POSTGRESQL_LOCK_TIMEOUT}" >>
"${POSTGRESQL_CONFIG_FILE}"
```

Allow to extend the container

```
#> cat post-init-hooks/post-init
echo "lock_timeout=${POSTGRESQL_LOCK_TIMEOUT}" >>
"${POSTGRESQL_CONFIG_FILE}"

#> cat Dockerfile
FROM fedora/postgresql:9.5
COPY post-init-hooks ${CONTAINER_SCRIPTS_PATH}/hooks.d
```

Allow to extend the container

```
#> cat post-init-hooks/post-init
echo "lock_timeout=${POSTGRESQL_LOCK_TIMEOUT}" >>
"${POSTGRESQL_CONFIG_FILE}"

#> cat Dockerfile
FROM fedora/postgresql:9.5
COPY post-init-hooks ${CONTAINER_SCRIPTS_PATH}/hooks.d

#> docker build -t hhorak-postgresql-95 .
```

Tip #11:

See more at:

https://github.com/sclorg/postgresql-container

3. PYTHON CONTAINER FOR BUILDING APPLICATIONS

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile

FROM fedora:24

RUN dnf install -y --setopt=tsflags=nodocs python python-setuptools
python-pip
```

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile

FROM fedora:24

RUN dnf install -y --setopt=tsflags=nodocs python python-setuptools
python-pip && dnf clean all
```

So we have python in container.
How to build application on top of it?

Building app container

We might document something like this, which users will love to do..

```
#> cat Dockerfile
FROM fedora/python-35
ADD install-app /usr/bin/
RUN /usr/bin/install-app
CMD ["/usr/bin/python", "/opt/app-root/guestbook/guestbook/bin.py"]

#> cat install-app
#!/bin/bash
git clone https://github.com/hhorak/guestbook-pgsql.git /opt/app-root/guestbook
cd /opt/app-root/guestbook/guestbook
./setup.py

#> docker build -t guestbook1 .
```

Tip #12:
Help users to be more effective.

Building app container using s2i

```
#> dnf -y install source-to-image

#> s2i build /path/to/guestbook-app fedora/python-35 guestbook1

#> docker run -p 8080:8080 guestbook1
```

How source-to-image (s2i) works

1. builder container (e.g fedora/python) is run
   a. application is pulled or bind-mounted under **/tmp/src/** in container
   b. **/usr/libexec/s2i/assembly** script executed
   c. **/usr/libexec/s2i/run** script set as default CMD of resulting image
2. container is committed

Principles of source-to-image

1. **assembly** script is run to create image with application

```
#> cat assembly

#!/bin/bash

cp -Rf /tmp/src/. ./

if [[ -f requirements.txt ]]; then
  pip install --user -r requirements.txt
fi
```

Principles of source-to-image

2. **run** script is executed as default CMD of resulting image

```
    #> cat run

    #!/bin/bash

    function is_django_installed() {
      python -c "import django" &>/dev/null
    }

    if is_django_installed; then
      manage_file=$(find . -maxdepth 2 -type f -name 'manage.py' | head
-1)
      exec python "$manage_file" runserver 0.0.0.0:8080
    fi
```

Tip #13:
Let users use their favourite frameworks.

Source-to-image in practice

```
#> s2i build https://github.com/joe/guestbook.git \
       --context-dir=app/ fedora/python guestbook1

#> docker run -p 8080:8080 guestbook1
```

Tip #14:
See more at:
https://github.com/sclorg/s2i-python-container

4. SYSTEM CONTAINERS

Run container as systemd service

i.e. replace daemon with container

We need to:

1. Create Docker container
2. Create systemd unit file for the service
3. Work with the systemd unit as usually

Run container instead of process

1. Create Docker container (but not run)

```
#> docker create --name rsyslog-cont fedora/rsyslog
```

Run container instead of process

2. Create systemd unit file for the service

```
# cat /etc/systemd/system/rsyslog-cont.service
[Unit]
Description=rsyslog service as a docker
container
After=docker.service

[Service]
ExecStart=/usr/bin/docker start rsyslog-cont
ExecStop=/usr/bin/docker stop
rsyslog-cont

[Install]
WantedBy=multi-user.target
```

Run container instead of process

3. Work with the systemd service as usually

```
#> systemctl enable rsyslog-cont.service
#> systemctl start rsyslog-cont.service
```

Tip #15:
Support `atomic` command to allow users to create system container easily.

`atomic` command

Available on Fedora Atomic Host to handle system containers.

```
# 1. create container
# 2. create and enable systemd service
#> atomic install -n rsyslog-cont fedora/rsyslog
```

```
# 3. start systemd service
#> systemctl start rsyslog-cont.service
```

See more at http://www.projectatomic.io/docs/usr-bin-atomic/

5. TOOLS CONTAINERS

Containers that are not daemons

- Tools to manage daemons
- System tools not available on Fedora Atomic Host
- Desktop applications
- ...

Tools to manage daemons

(that are not part of the daemon image)

```
#> docker run -ti fedora/mysql-utilities mysqldbcopy -h 172.16.3.2 ...

#> docker run -ti fedora/mongo-tools mongodump -h 172.16.3.3 ...
```

Interaction is easy, we can use network socket to work with daemon.

System tools not available on Atomic Host

For example whole build toolchain

```
#> docker run -ti -v /:/host fedora/toolchain bash
bash-4.2$ gcc -v
bash-4.2$ cd /host/home/hhorak/myprog
bash-4.2$ make
bash-4.2$ ...
```

System tools not available on Atomic Host

Some may need to be run as super-privileged container

Either manually

```
#> docker run -ti --privileged --ipc=host --net=host --pid=host [...] \
            -v /:/host fedora/perftools bash
bash-4.2$ ocount ls
bash-4.2$ operf ls
```

Tip #16:
`atomic` command is especially useful for SPC containers.

System tools not available on Atomic Host

Some may need to be run as super-privileged container

Either manually

```
#> docker run -ti --privileged --ipc=host --net=host --pid=host [...] \
           -v /:/host fedora/perftools bash
bash-4.2$ ocount ls
bash-4.2$ operf ls
```

Or more conveniently using `atomic --spc`

```
#> atomic run --spc fedora/perftools bash
bash-4.2$ ocount ls
bash-4.2$ operf ls
```

Desktop applications

It is possible to run desktop applications in docker.

```
docker run -ti --rm \
       -e DISPLAY=$DISPLAY \
       -v /tmp/.X11-unix:/tmp/.X11-unix \
       fedora/firefox
```

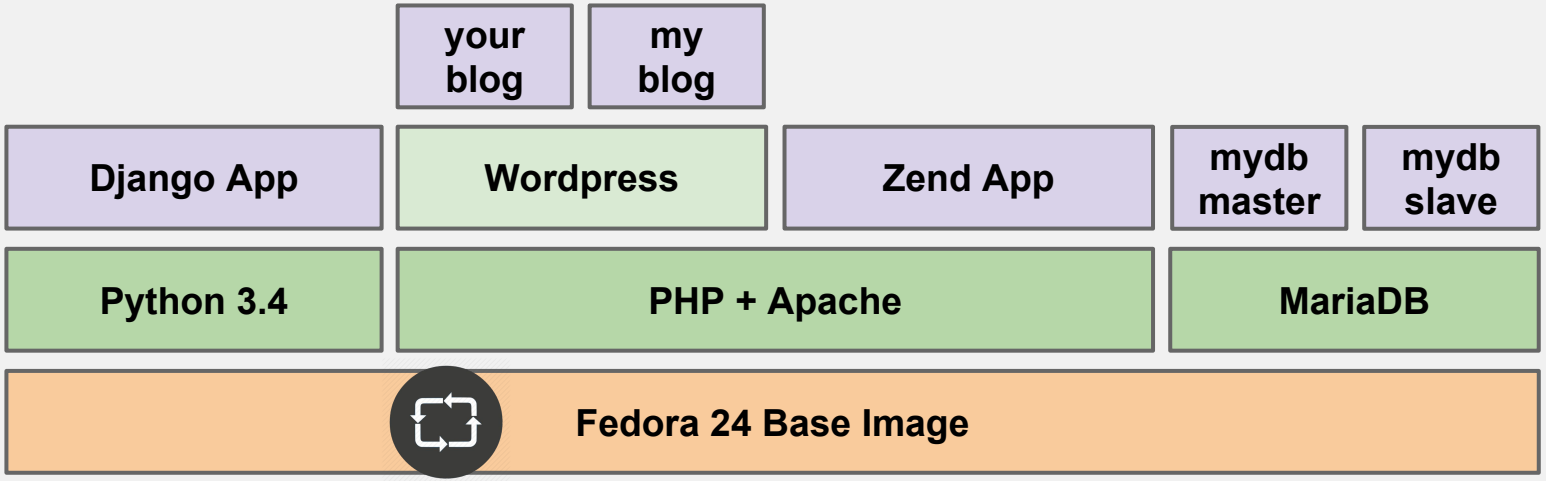
Tip #17:
Consider using Flatpak for desktop applications.

6. BUILDING INFRASTRUCTURE

Tip #18:
Rebuild container images once base images is updated.

Docker layers = bundling

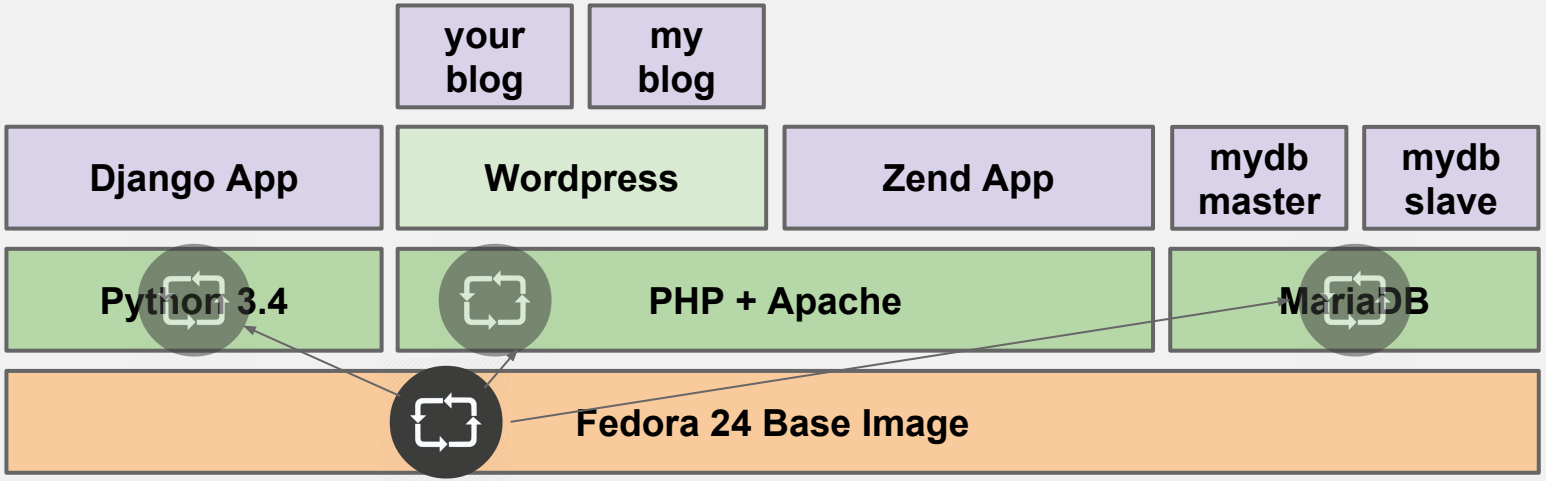
So we need to rebuild all layers above.



fedora

Docker layers = bundling

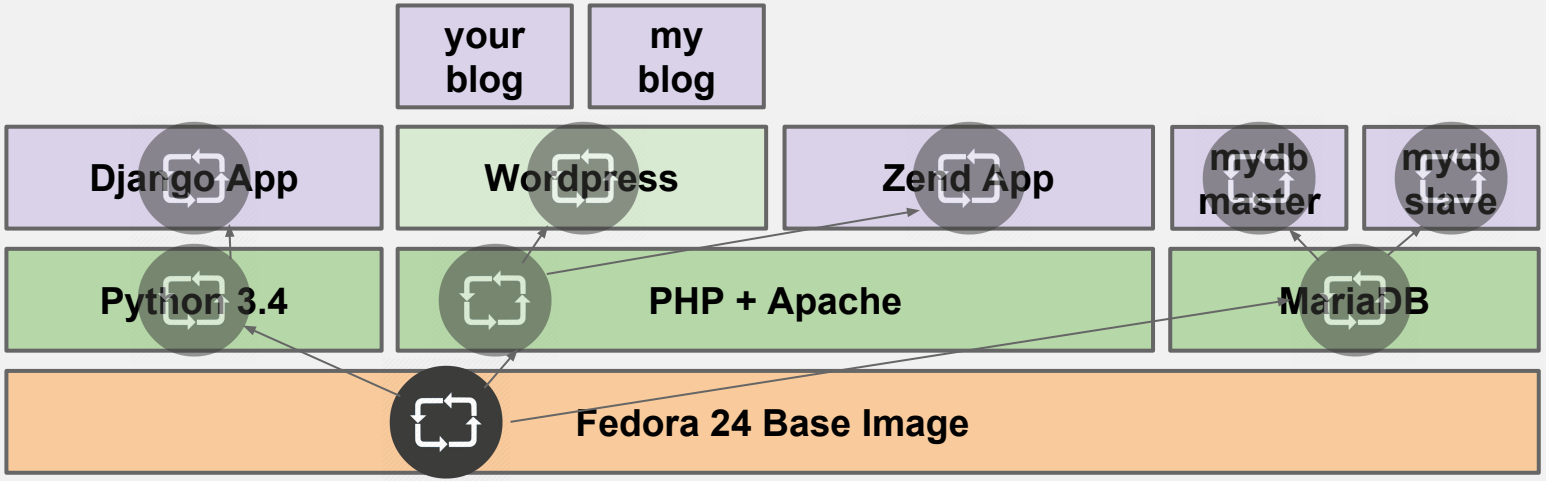
So we need to rebuild all layers above.



fedora

Docker layers = bundling

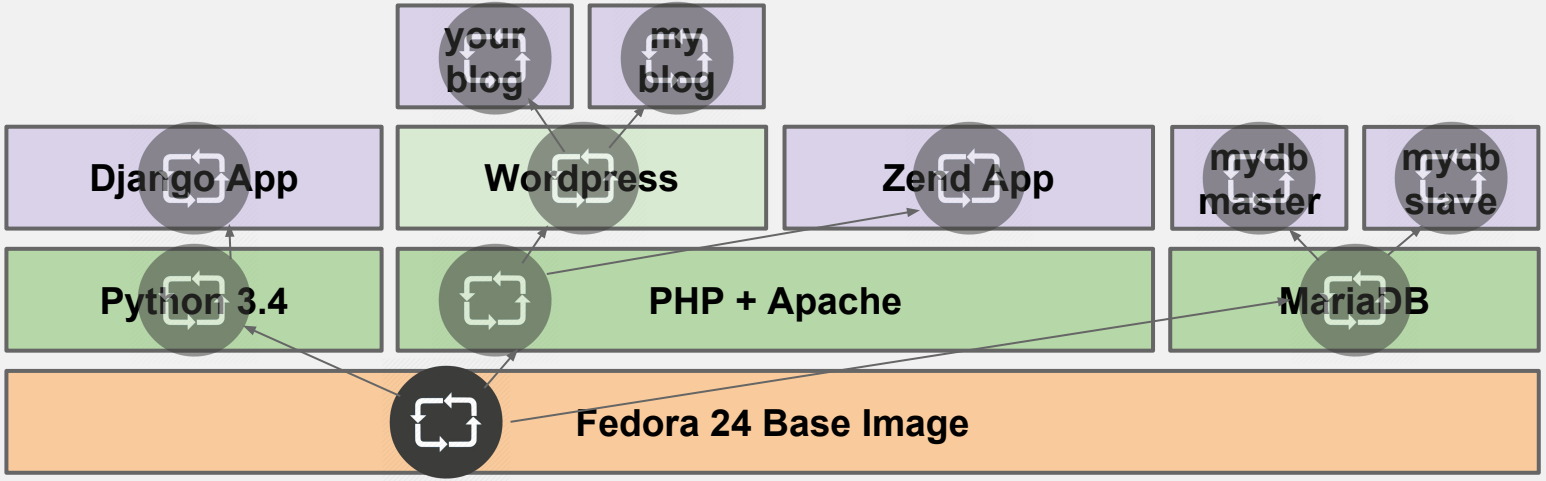
So we need to rebuild all layers above.



fedora

Docker layers = bundling

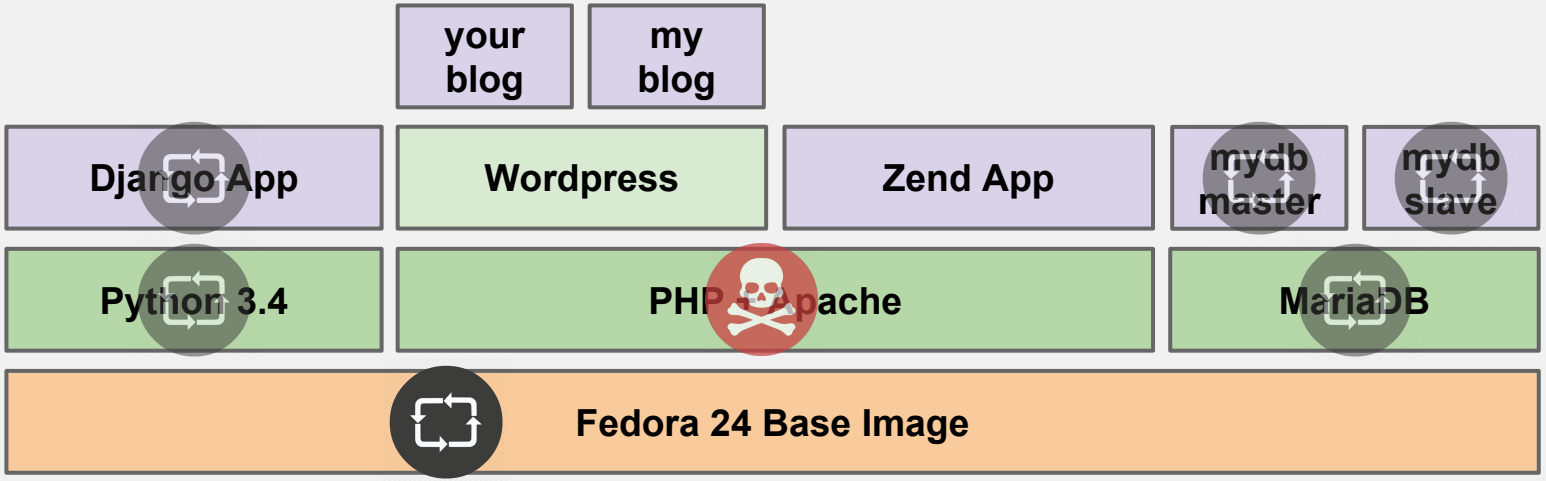
So we need to rebuild all layers above.



fedora

Docker layers = bundling

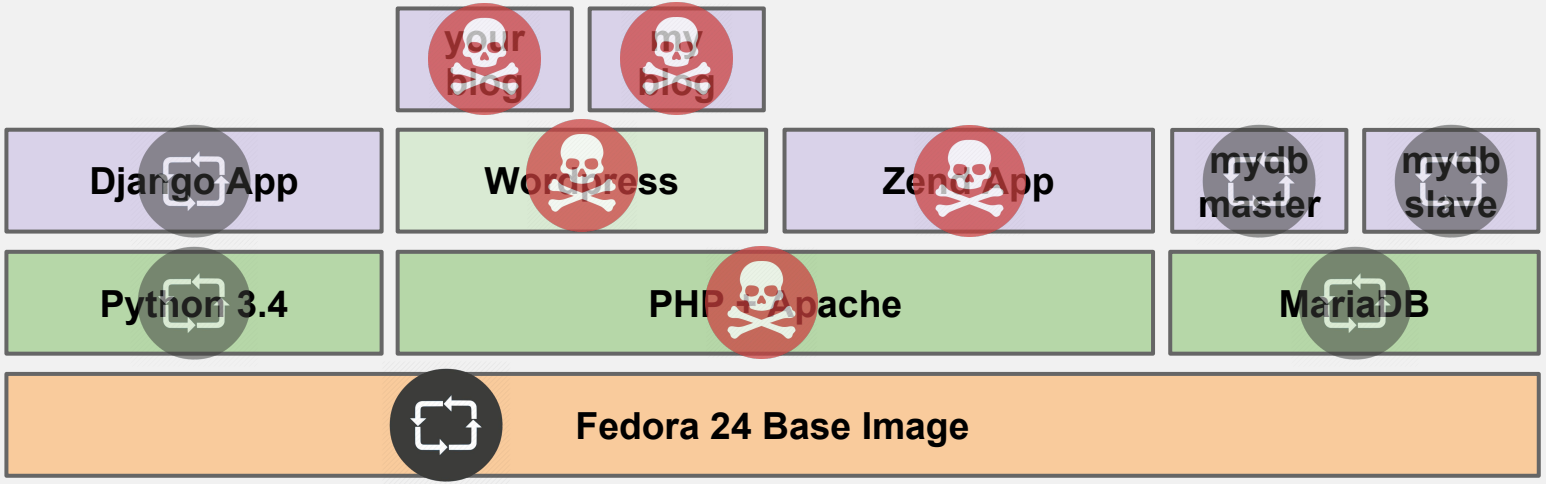
So we need to rebuild all layers above.



fedora

Docker layers = bundling

So we need to rebuild all layers above.



fedora

Tip #19:

Automate rebuilding.

Tip #19:
Automate rebuilding.
And automate testing as well.

Tip #20:
Keep and maintain **sanity tests** together with image source.

Tests that verify image

should be part of the image source

```
#> find -type f
./Dockerfile
./root/usr/share/container-scripts/mysql/passwd-change.sh
./root/usr/share/container-scripts/mysql/post-init.sh
./root/usr/share/container-scripts/mysql/common.sh
./root/usr/bin/run-mysqld
./root/usr/bin/run-mysqld-slave
./root/usr/bin/container-entrypoint
./root/etc
./root/etc/my.cnf
./examples
./README.md
./test/run

#> docker build -t testimg .
#> IMAGE=testimg ./test/run
```

Tests that verify image

Docker itself is enough for running sanity tests.

```
#!/bin/sh

# run the daemon
IMAGE=${IMAGE:-fedora/mysql-57}
docker run --rm MYSQL_USER=user -e MYSQL_PASSWORD=pass --cidfile cid $IMAGE
CONT_IP=$(docker inspect --format='{{.NetworkSettings.IPAddress}}' `cat cid`)

# wait till daemon is started
for i in 10 do
  sleep 3
  docker run --rm $IMAGE mysql --host $CONT_IP -uuser -ppass <<< "SELECT 1;"
  [ $? -eq 0 ] && echo "Success!" && return 0
done
echo "Giving up: Failed to connect. Logs:"
docker logs `cat cid`
```

Tests that verify image

Docker itself is enough for running sanity tests.

```
#!/bin/sh

# run the daemon
IMAGE=${IMAGE:-fedora/mysql-57}
docker run --rm MYSQL_USER=user -e MYSQL_PASSWORD=pass --cidfile cid $IMAGE
CONT_IP=$(docker inspect --format='{{.NetworkSettings.IPAddress}}' `cat cid`)

# wait till daemon is started
for i in 10 do
  sleep 3
  docker run --rm $IMAGE mysql --host $CONT_IP -uuser -ppass <<< "SELECT 1;"
  [ $? -eq 0 ] && echo "Success!" && return 0
done
echo "Giving up: Failed to connect. Logs:"
docker logs `cat cid`
```

Tests that verify image

```
#!/bin/sh

# run the daemon
IMAGE=${IMAGE:-fedora/mysql-57}
docker run --rm MYSQL_USER=user -e MYSQL_PASSWORD=pass --cidfile cid $IMAGE
CONT_IP=$(docker inspect --format='{{.NetworkSettings.IPAddress}}' `cat cid`)

# wait till daemon is started
for i in 10 do
  sleep 3
  docker run --rm $IMAGE mysql --host $CONT_IP -uuser -ppass <<< "SELECT 1;"
  [ $? -eq 0 ] && echo "Success!" && return 0
done
echo "Giving up: Failed to connect. Logs:"
docker logs `cat cid`
```

Tests that verify image

```
#!/bin/sh

# run the daemon
IMAGE=${IMAGE:-fedora/mysql-57}
docker run --rm MYSQL_USER=user -e MYSQL_PASSWORD=pass --cidfile cid $IMAGE
CONT_IP=$(docker inspect --format='{{.NetworkSettings.IPAddress}}' `cat cid`)

# wait till daemon is started
for i in 10 do
  sleep 3
  docker run --rm $IMAGE mysql --host $CONT_IP -uuser -ppass <<< "SELECT 1;"
  [ $? -eq 0 ] && echo "Success!" && return 0
done
echo "Giving up: Failed to connect. Logs:"
docker logs `cat cid`
```

General tests for container images

- Sanity tests pass
- Includes signed RPMs
- LABELs (metadata for Open Shift, build info, source URL)
  - https://github.com/projectatomic/ContainerApplicationGenericLabels
- Based on latest base image
- Includes README.md, usage message
- No errors reported during image build
- Software Collection is enabled automatically inside container
- API does not change (what API?)

Tip #21:
Consider what is part of image's API.

Keep stable API

- paths or variables used when extending image
  - `COPY post-init-hooks ${CONTAINER_SCRIPTS_PATH}/hooks.d`
- volume paths
  - volumes e.g. `-v /mydata:/var/lib/mysql:Z`
- which **port** the service listens on by default
- which **commands** are available (not only default)
  - `#> docker run mysql-57 run-mysql-master`

Tip #22:
Use similar rules across similar images.

Use similar rules across similar images

And extend http://docs.projectatomic.io/container-best-practices/

- **`/usr`** rather than `/usr/local` or even `/randomdir`
- expected **paths** for
  - volumes e.g. `/var/lib/mysql`
  - configuration, e.g. `/etc/my.cnf`
- expected **port** the service listens on by default (3306 for MariaDB)
- default user (1001 if no special user exists for the service)

Remember some tips?

Content matters.

Content matters.

Do not run processes as root in containers.

Content matters.
Do not run processes as root in containers.
Allow users to extend images easily.

Content matters.
Do not run processes as root in containers.
Allow users to extend images easily.
Support `atomic` where reasonable.

Content matters.
Do not run processes as root in containers.
Allow users to extend images easily.
Support `atomic` where reasonable.
Automate.

Questions?

https://github.com/sclorg/
http://docs.projectatomic.io/container-best-practices/
https://github.com/fedora-cloud/Fedora-Dockerfiles
http://www.projectatomic.io/

Honza Horak <hhorak@redhat.com>
@HonzaHorak

Thanks.

Honza Horak <hhorak@redhat.com>
@HonzaHorak

https://hhorak.fedorapeople.org/2016/160802_application_containers_and_system_services.pdf