redhat.

HOW TO DEVELOP CONTAINERS IN ENTERPRISE WORLD

Honza Horak <hhorak@redhat.com>
Software Collections, Containers, CentOS, Fedora
DevConf, Brno, 6th Feb 2016

The goal

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

This talk is about...

What we learned on the path of building containers for OpenShift, aka don't do the same mistakes as we did.

1. Building the first container
2. PostgreSQL container
3. Python container to build applications
4. Software Collections in containers
5. Application Containers in Red Hat and CentOS Portfolio
6. Distribution of container apps

This talk is **not** about...

- Docker 101 (workshop on Friday)
- How to run containers in OpenShift

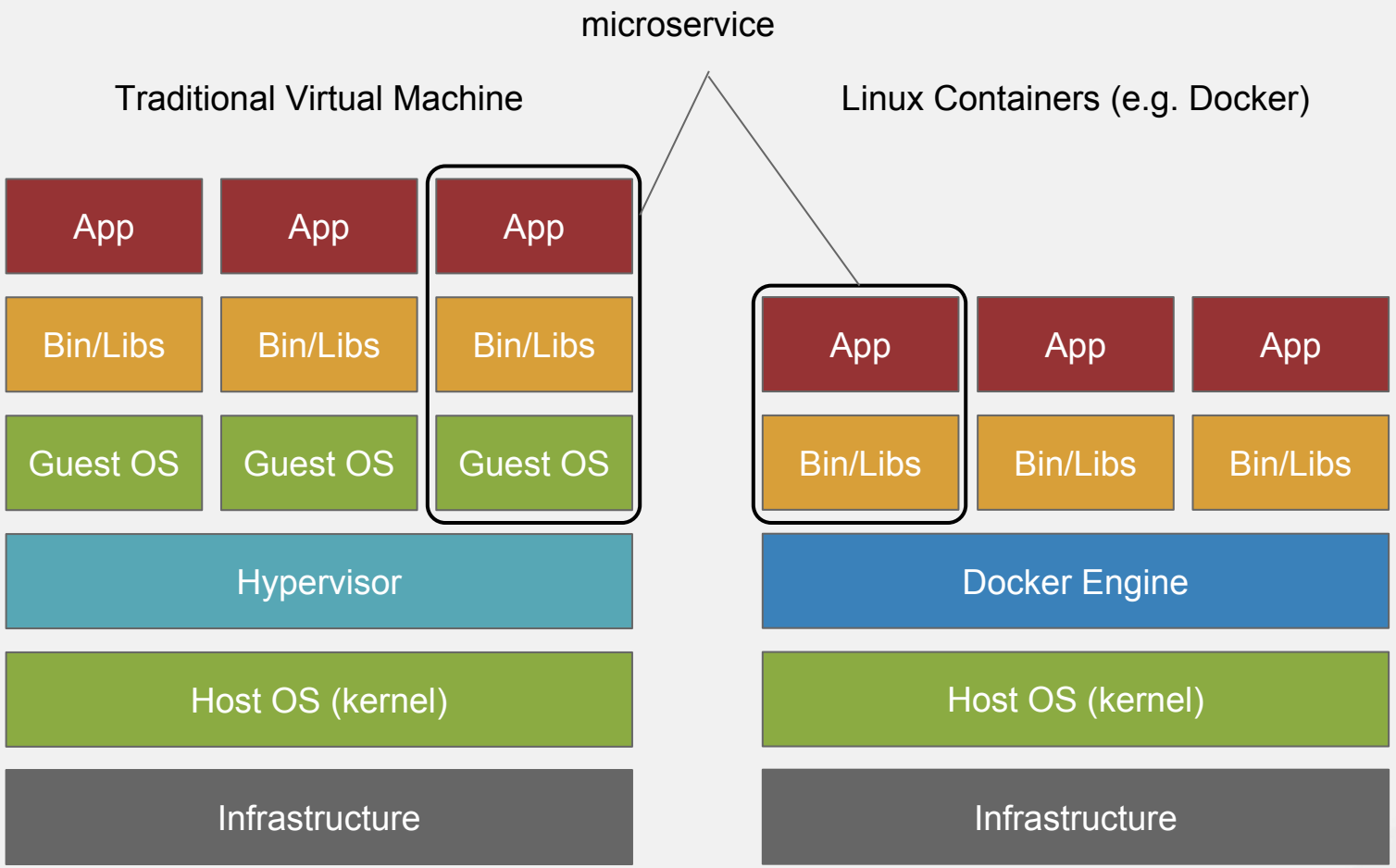
1. CONTAINERS BASICS


NUR FÜR ALTPAPIER
NUR FÜR ALTPAPIER
ALTPAPIER
Kleider & Schuhe
Kleider & Schuhe
Kleider & Schuhe

Tip of tips:

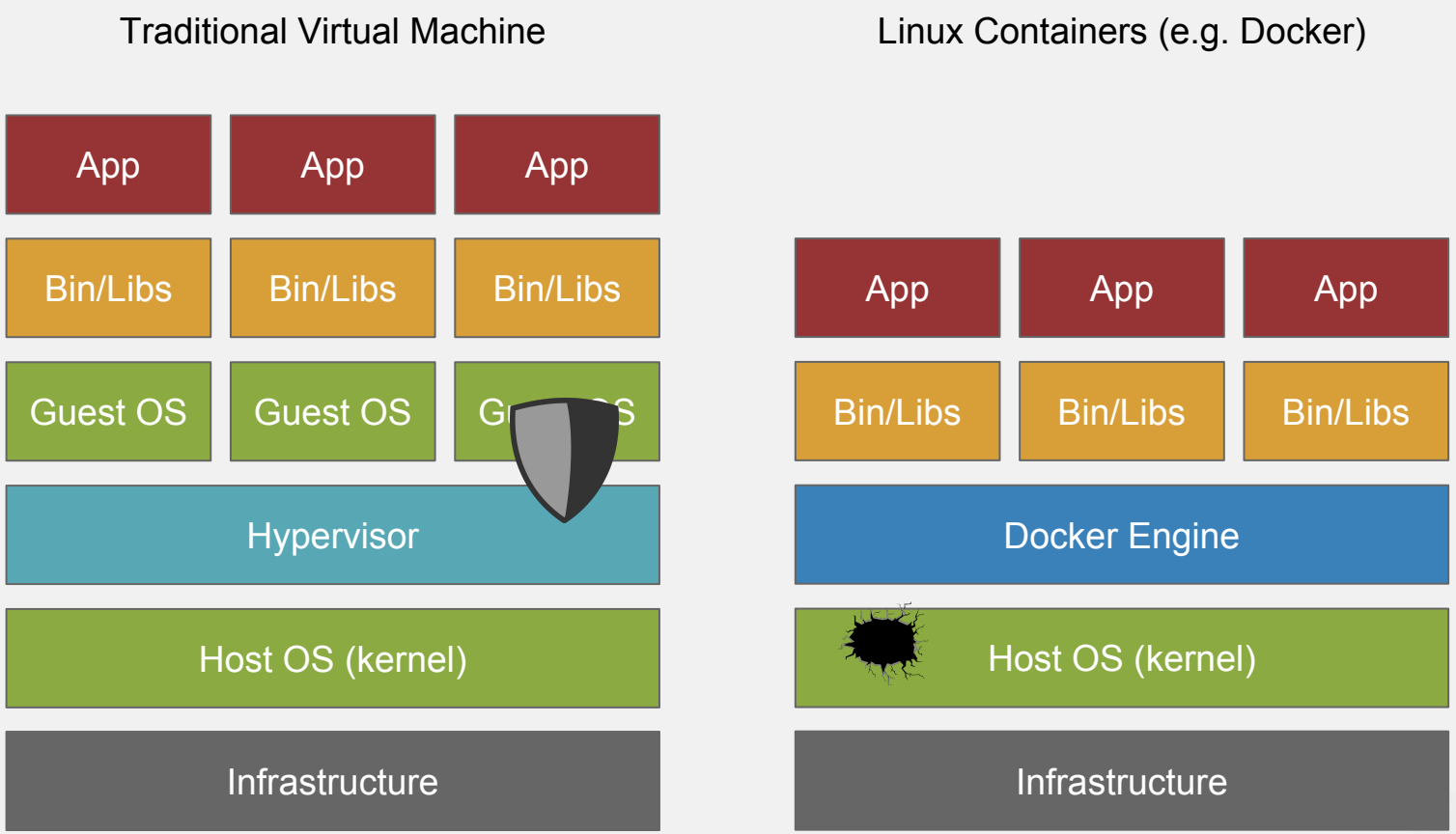
Content matters.

Containers are kind of virtualization



Container is not a virtual machine

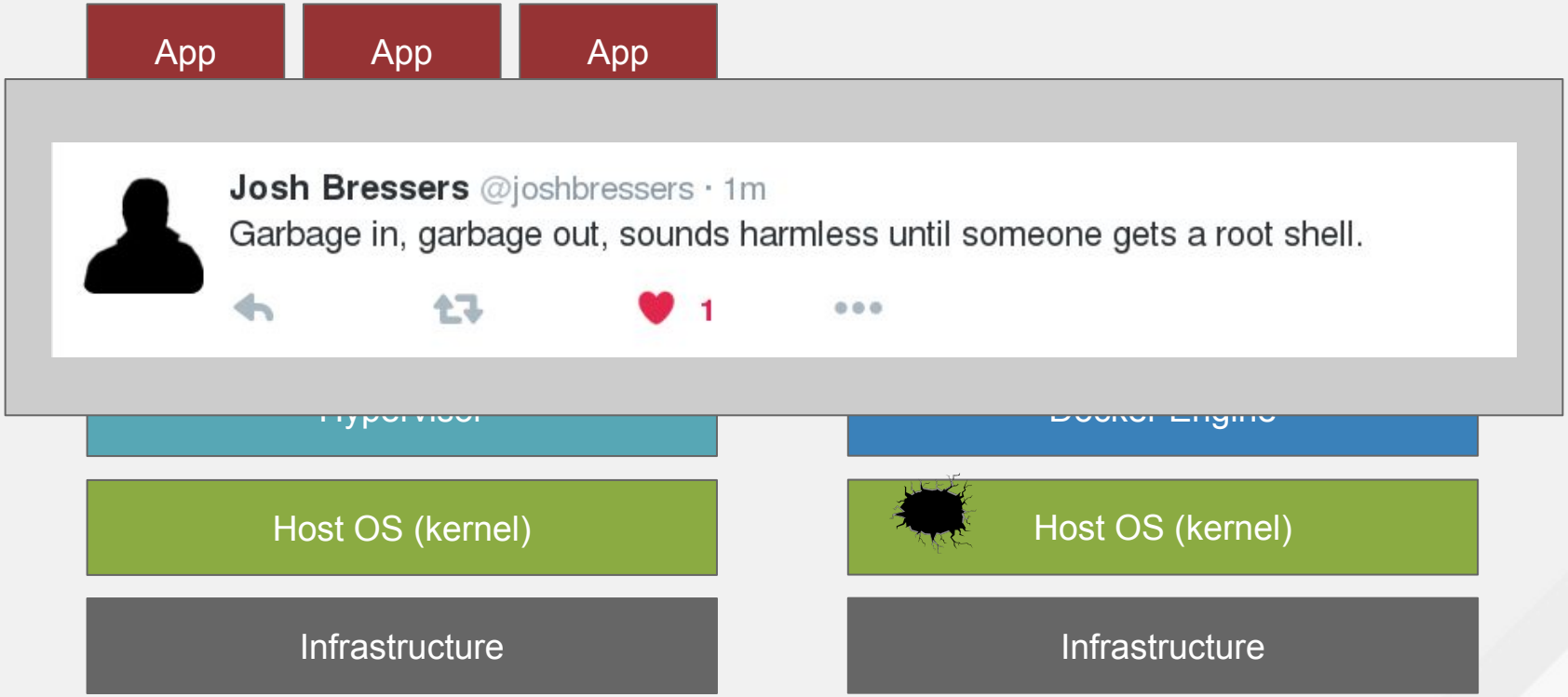
it's more a packaging format



Container is not a virtual machine

Traditional Virtual Machine

Linux Containers (e.g. Docker)



Tip #1:
Use only content you trust.

Building the first container

```
#> yum install -y docker
#> systemctl start docker
#> docker pull rhel7:7.2
```

Building the first container

```
#> yum install -y docker
#> systemctl start docker
#> docker pull rhel7:7.2

#> docker run -ti --name mycont rhel7:7.2 bash
[root@a1eefecdacfa /]# echo Hello Dojo > /root/greeting
[root@a1eefecdacfa /]# exit
```

Building the first container

```
#> yum install -y docker
#> systemctl start docker
#> docker pull rhel7:7.2

#> docker run -ti --name mycont rhel7:7.2 bash
[root@a1eefecdacfa /]# echo Hello Dojo > /root/greeting
[root@a1eefecdacfa /]# exit

#> docker commit mycont
0bdcfc5ba0602197e2ac4609b8101dc8eaa0d8ab114f542ab6b2f15220d0ab22
```

Tip #2:
Use reproducible builds.

Building the first container correctly

```
#> cat Dockerfile
FROM rhel7:7.2
RUN echo Hello Dojo > /root/greeting

#> docker build .
```

2. CREATING POSTGRESQL CONTAINER

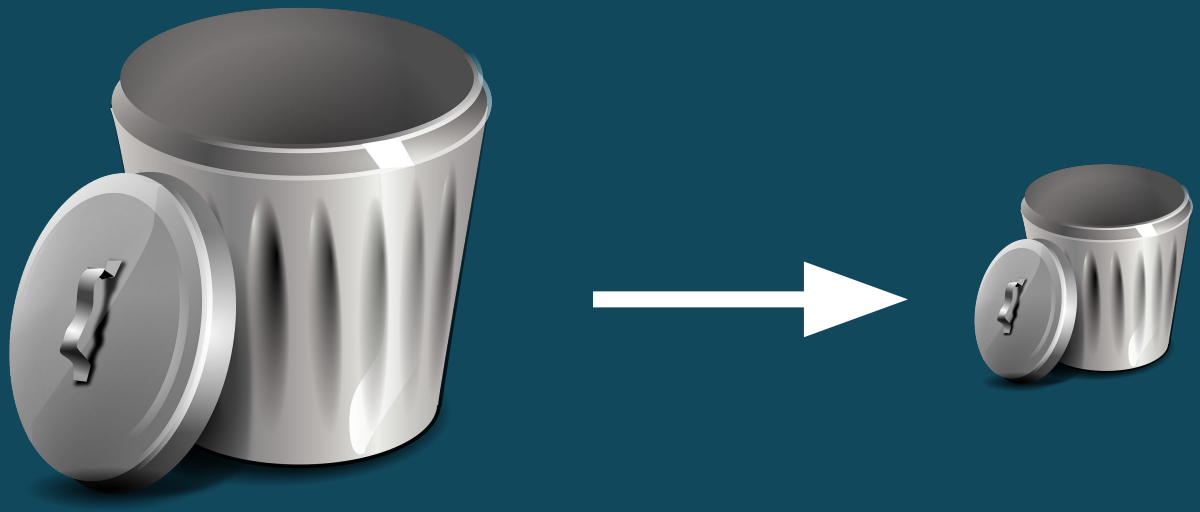
Installing RPMs in container

```
#> cat Dockerfile
FROM rhel7:7.2
RUN yum -y install postgresql-server

#> docker build .
```

Installing RPMs in container

```
#> docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM rhel7:7.2
Trying to pull repository docker.io/library/centos ... centos7: Pulling from
library/centos
Digest: sha256:0b0e2e8ff4ce5bb714fc30356f2a7f6ae29a1b84adef9f5cd22b388ffccb65d7
Status: Downloaded newer image for docker.io/centos:centos7

 ---> c8a648134623
Step 2 : RUN yum -y install postgresql-server
 ---> Running in 8b53d7337b55
<skipping yum output>
Complete!
 ---> 29036308c1ec
Removing intermediate container 8b53d7337b55
Successfully built 29036308c1ec
```

Tip #3:
Create small containers.



Installing RPMs in container

```
#> cat Dockerfile
FROM rhel7:7.2
RUN yum -y --setopt=tsflags=nodocs install postgresql-server && \
    yum clean all

#> docker build .
```

Installing RPMs in container

```
#> docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM rhel7:7.2
 ---> c8a648134623
Step 2 : RUN yum -y --setopt=tsflags=nodocs install postgresql-server
&& yum clean all
 ---> Using cache
 ---> 29036308c1ec
Successfully built 29036308c1ec
```

Installing RPMs in container

```
#> cat Dockerfile
FROM rhel7:7.2
RUN yum -y --setopt=tsflags=nodocs install postgresql-server && \
    yum clean all

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

Tip #4:
Be careful about docker cache.

Tip #4:
Be careful about docker cache.
Or use OpenShift Build Service.

Tip #4:
Be careful about docker cache.
Or use OpenShift Build Service.
Or see the Tomas' presentation
"Is it hard to build a docker image?"

Are we there yet?

It’s small, green and very dangerous.
What is it?

Make container more save

```
#> cat Dockerfile

FROM rhel7:7.2

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

ENV HOME=/var/lib/pgsql
ENV PGDATA=/var/lib/pgsql/data
ENV PGUSER=postgres
USER 26
```

Tip #5:
Use non-root user wherever possible.

Make container do something

```
#> cat Dockerfile

FROM rhel7:7.2

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

ENV HOME=/var/lib/pgsql
ENV PGDATA=/var/lib/pgsql/data
ENV PGUSER=postgres
USER 26

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

Tip #6:
Create microservices, not box of packages.

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 "$@"
```

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 "$@"
```

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.

Connecting to PostgreSQL container

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

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti -p 5432:5432 --name p1 postgresql
```

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti -p 5432:5432 --name p1 postgresql

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

Connecting to PostgreSQL container

```
#> docker build -t postgresql .

#> docker run -ti -p 5432:5432 --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 -p 5432:5432 --name p1 postgresql

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

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

Tip #8:
Do not use default passwords.

Let user to specify a password

```
#> 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

pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Let user to specify a password

```
#> 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

pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Let user to specify a password

```
#> 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

pg_ctl -w start -o "-h ''"
psql --command "ALTER USER \"postgres\" WITH ENCRYPTED PASSWORD
'${POSTGRESQL_ADMIN_PASSWORD}';"
pg_ctl stop
...
```

Let user to specify a password

```
#> 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

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 -p 5432:5432 --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 run -ti -p 5432:5432 --name p1 postgresql

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

#> psql -h 172.17.0.2 -U postgres
Password for user postgres:
psql (9.2.14, server 9.2.14)
Type "help" for help.

postgres=# _
```

How to configure such a database?

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

Example of PostgreSQL 9.4 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 \
          rhscl/postgresql-94-rhel7
```

Tip #9:
Support only most common configuration options.

Example of PostgreSQL 9.4 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 \
          rhscl/postgresql-94-rhel7
```

Tip #10:
Use expected paths

`-v /db:/var/lib/pgsql/data:Z`

Tip #11:
Allow users to extend the container easily.

3. PYTHON CONTAINER TO BUILD APPLICATIONS

3. PYTHON CONTAINER TO BUILD APPLICATIONS (AKA A BUILDER IMAGE)

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile

FROM rhel7:7.2

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

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile

FROM rhel7:7.2

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

Building simplest Python container

```
#> docker build -t rhscl/python-27-rhel7 .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM rhel7:7.2
 ---> c8a648134623
Step 2 : RUN yum install -y --setopt=tsflags=nodocs python python-setuptools
python-pip
 ---> Running in 067726695f58
...
Package python-2.7.5-34.el7.x86_64 already installed and latest version
No package python-pip available.
Resolving Dependencies
--> Running transaction check
...
Complete!
 ---> 45af014765cf
Removing intermediate container 067726695f58
Successfully built 45af014765cf
```

Tip #12:
Do not believe yum.

How to build application on top of it?

Building application container

```
#> cat Dockerfile
FROM rhscl/python-34-rhel7
ADD install-app /usr/bin/
RUN /usr/bin/install-app
CMD ["/usr/bin/python", "/opt/app-root/guestbook-pgsql/guestbook/bin.py"]

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

#> docker build -t guestbook .
```

Building application container

```
#> cat Dockerfile
FROM rhscl/python-34-rhel7
ADD install-app /usr/bin/
RUN /usr/bin/install-app
CMD ["/usr/bin/python", "/opt/app-root/guestbook-pgsql/guestbook/bin.py"]

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

#> docker build -t guestbook .
```

Tip #13:
Help users to be more effective.

Source-to-image (s2i) is a tool for building reproducible Docker images.

s2i produces ready-to-run images by injecting source code into a Docker image and **assembling** a new Docker image which incorporates the builder image and built source.

The result is then ready to use with docker **run**.

Building app container using s2i

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

#> s2i build /path/to/guestbook python-34-rhel7 guestbook
```

How source-to-image works

- builder container is started
- **assembly** script executed
- **run** script set as default CMD of resulting image
- container committed

Principles of source-to-image

1. assembly script

```
#> 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

```
#> cat assembly

#!/bin/bash

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

manage_file=$(find . -maxdepth 2 -type f -name 'manage.py' | head -1)

if is_django_installed; then
  exec python "$manage_file" runserver 0.0.0.0:8080
fi
```

Tip #14:
Let users use their favourite frameworks.

Microservices running on usual ports

```
#> cat Dockerfile

...
EXPOSE 8080
...

#> docker run -d -p 8080:1234 guestbook
```

Tip #15:
Do not just install software.
Build micro-services.

Tip #16:
Let users to run container as any UID.

Allow to use any UID

```
#> cat Dockerfile

...
RUN chown -R 1001:0 /opt/app-root && chmod -R g+rw /opt/app-root
USER 1001
...

#> docker run -ti -u 1483 python-27-centos7 bash
```

Source-to-image in practice

```
$> s2i build https://github.com/hhorak/guestbook-pgsql.git \
       --context-dir=app/ rhscl/python-34-rhel7 guestbook

$> docker run -p 8080:8080 guestbook
```

Co . . . . t matters.

Where to get some new bits?

Software Collections already deliver recent versions.

Software Collections already deliver recent versions.
And packages, like pip :)

4. SOFTWARE COLLECTIONS IN CONTAINERS

Software collections are available

and do not conflict with system packages

```
#> yum -y install rh-postgresql94
 ...
Installed:
  rh-postgresql94.x86_64 0:2.0-9.el7
Dependency Installed:
  rh-postgresql94-postgresql.x86_64 0:9.4.4-1.el7
  rh-postgresql94-postgresql-libs.x86_64 0:9.4.4-1.el7
  rh-postgresql94-postgresql-server.x86_64 0:9.4.4-1.el7
  rh-postgresql94-runtime.x86_64 0:2.0-9.el7
Complete!
```

Example of running SCL

The whole magic is in changing environment variables

```
#> scl enable rh-python34 'python -V'
Python 3.4.3
```


Which container includes a collection?

Tip #17:
Do not be afraid to combine containers & Software Collections.

How SCL may be handy in container

- OS containers (in comparison to one-process containers)
  - what if we need two versions of something inside a container?
- same problems in container as outside
  - python 2.7 is always needed for YUM
- one binary for both (develop once + test once)
  - saving resources
  - same content on traditional Linux and in containers
  - easy transition from traditional environment to containers

Tip #18:

Make sure SCL is enabled in container.

(hide that we’re using SCL underneath)

Make sure SCL is enabled in container

```
#> docker run -ti registry.access.redhat.com/rhscl/python-34-rhel7
bash
```

Make sure SCL is enabled in container

```
#> docker run -ti registry.access.redhat.com/rhscl/python-34-rhel7
bash

bash-4.2$ cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.2 (Maipo)
```

Make sure SCL is enabled in container

```
#> docker run -ti registry.access.redhat.com/rhscl/python-34-rhel7
bash

bash-4.2$ cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.2 (Maipo)

bash-4.2$ python -V
Python 3.4.2
```

How to enable SCL in container

```
#> cat Dockerfile
…
ADD scl_enable /usr/share/container-scripts/
ENV BASH_ENV=/usr/share/container-scripts/scl_enable \
    ENV=/usr/share/container-scripts/scl_enable \
    PROMPT_COMMAND=". /usr/share/container-scripts/scl_enable"
ENTRYPOINT ["container-entrypoint"]
…

#> cat scl_enable
unset BASH_ENV PROMPT_COMMAND ENV
source scl_source enable rh-python34
```

How to enable SCL in container

(or change environment that cannot be changed using ENV)

```
#> cat container-entrypoint

#!/bin/bash
exec "$@"
```

Tip #19:
Do not use ENTRYPOINT for anything else than environment change.

5. APPLICATION CONTAINERS IN RED HAT AND CENTOS PORTFOLIOS

Our focus in containerized world

- one solution across products (CentOS, RHEL, Atomic, OpenShift, ...)
- make containers look same (PostgreSQL, MariaDB)
- support specific use cases (not too many, not too few)

openshift/mongodb-24-rhel7 openshift/mysql-55-rhel7 openshift/postgresql-92-rhel7 **rhscl/mariadb-100-rhel7** rhscl/mongodb-26-rhel7 rhscl/mysql-56-rhel7 **rhscl/postgresql-94-rhel7** openshift/nodejs-010-rhel7 openshift/perl-516-rhel7 openshift/php-55-rhel7 rhscl/python-27-rhel7 openshift/python-33-rhel7 rhscl/perl-520-rhel7 **rhscl/php-56-rhel7 rhscl/python-34-rhel7** rhscl/ror-41-rhel7 rhscl/ruby-22-rhel7 openshift/ruby-20-rhel7 rhscl/passenger-40-rhel7 rhscl/httpd-24-rhel7 rhscl/nginx-16-rhel7 rhel7/devtoolset-4-toolchain

Available at **http://registry.access.redhat.com**

redhat.

openshift/mongodb-24-centos7 openshift/mysql-55-centos7
openshift/postgresql-92-centos7 **centos/mariadb-100-centos7**
centos/mongodb-26-centos7 centos/mysql-56-centos7
centos/postgresql-94-centos7 openshift/nodejs-010-centos7
openshift/perl-516-centos7 openshift/php55-centos7 centos/python-27-centos7 openshift/python-33-centos7 centos/perl-520-centos7
centos/php-56-centos7 centos/python-34-centos7
centos/ror-41-centos7 centos/ruby-22-centos7 openshift/ruby-200-centos7 centos/passenger-40-centos7 centos/httpd-24-centos7
centos/nginx-16-centos7

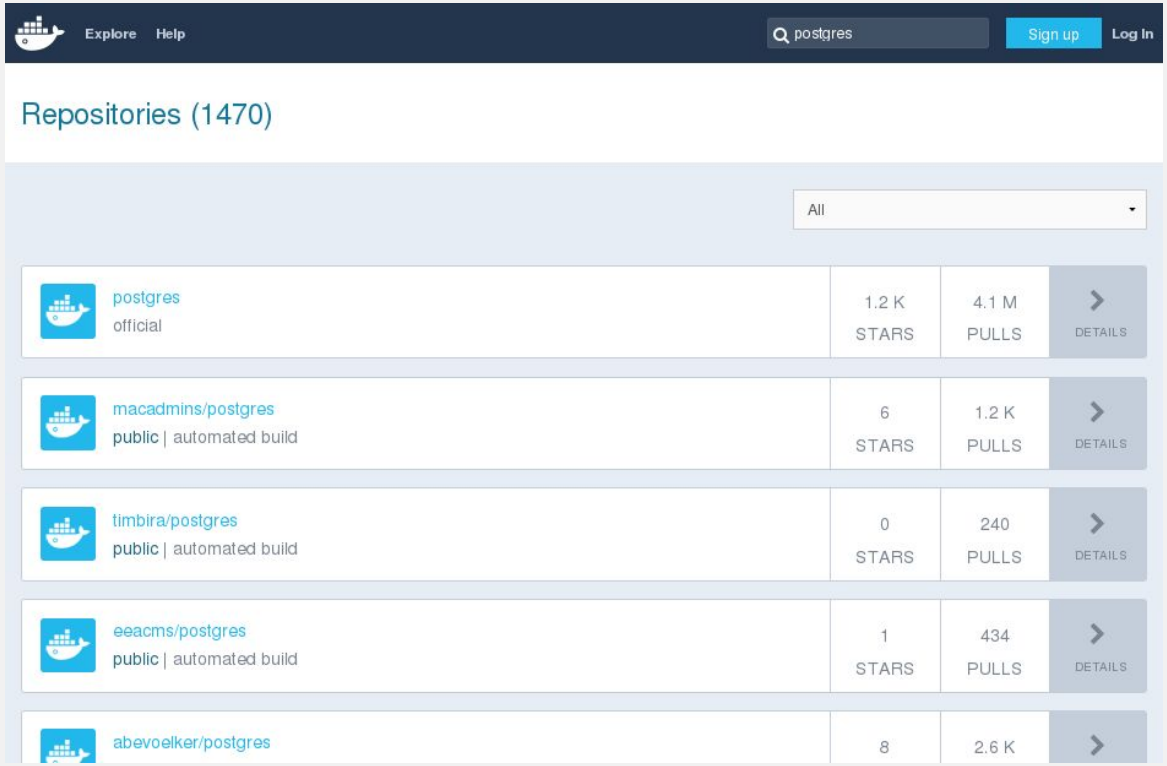
Available at **https://hub.docker.com**

CentOS

Tip #20:
Use containers from reliable provider.

Tip #20:
Use containers from reliable provider.
like Red Hat or CentOS :)

Which one?



Tip #21:
Think about what the name promises.

Containers naming questions

- include major version?
- include platform version underneath?
- examples:
  - rhscl/postgresql-94-rhel7 ?
  - centos/postgresql-94-centos7 ?
  - or is just centos/postgresql enough ?
  - or centos/postgresql-94 ?

https://github.com/projectatomic/ContainerApplicationGenericLabels/blob/master/vendor/redhat/names.md

Tip #22:
Consider what is part of image's API.

Paths, variables

what not to change (too often) once introduced

- /usr rather than /usr/local
- hide the /opt/rh (Software Collections specifics)
- expected paths for volumes (`/var/lib/pgsql`), configuration (`/etc`)
- environment variable names
- content of the image

Tip #23:
Set metadata to containers.

OpenShift and Kubernetes labels

```
LABEL io.k8s.description="PostgreSQL is an advanced Object-Relational DBMS" \
      io.k8s.display-name="PostgreSQL 9.4" \
      io.openshift.expose-services="5432:postgresql" \
      io.openshift.tags="database,postgresql,postgresql94,rh-postgresql94"
```

Tip #24:
Take security seriously.

Security

“Containers do not contain”

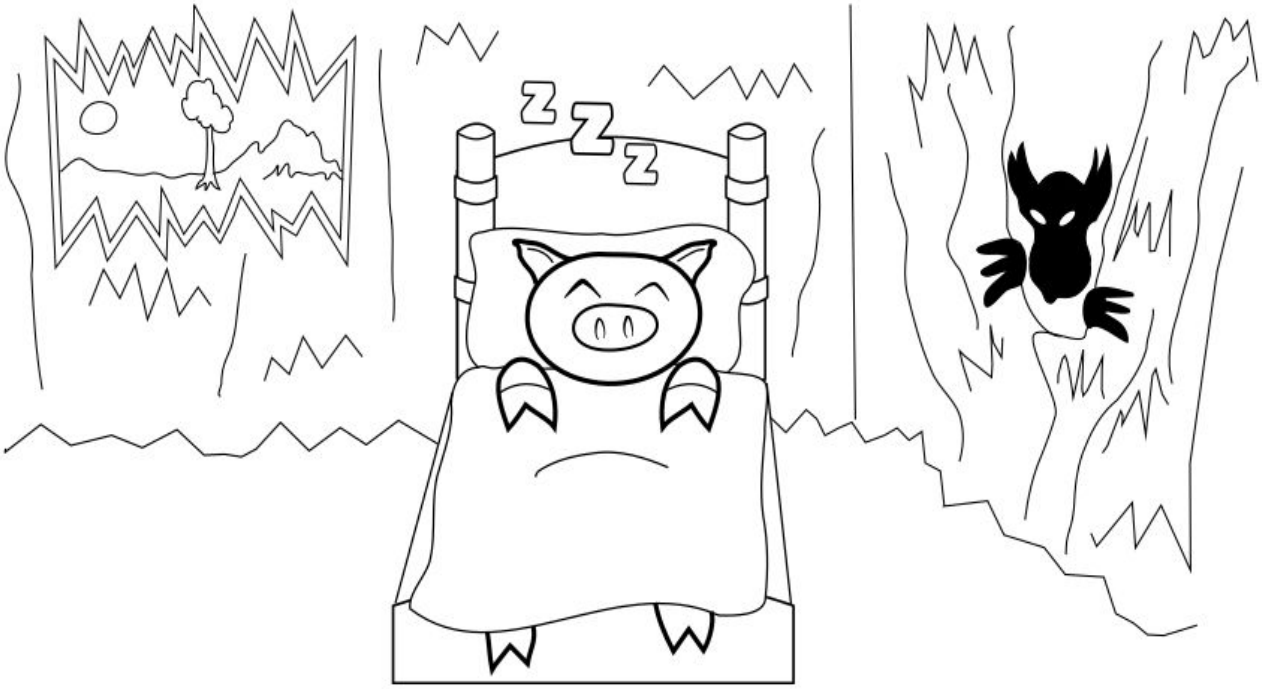
- colouring books by Dan and Máirín

  https://github.com/mairin/selinux-coloring-book

  https://github.com/fedoradesign/coloringbook-containers/raw/master/Print-Ready/Pages.pdf

SECURITY

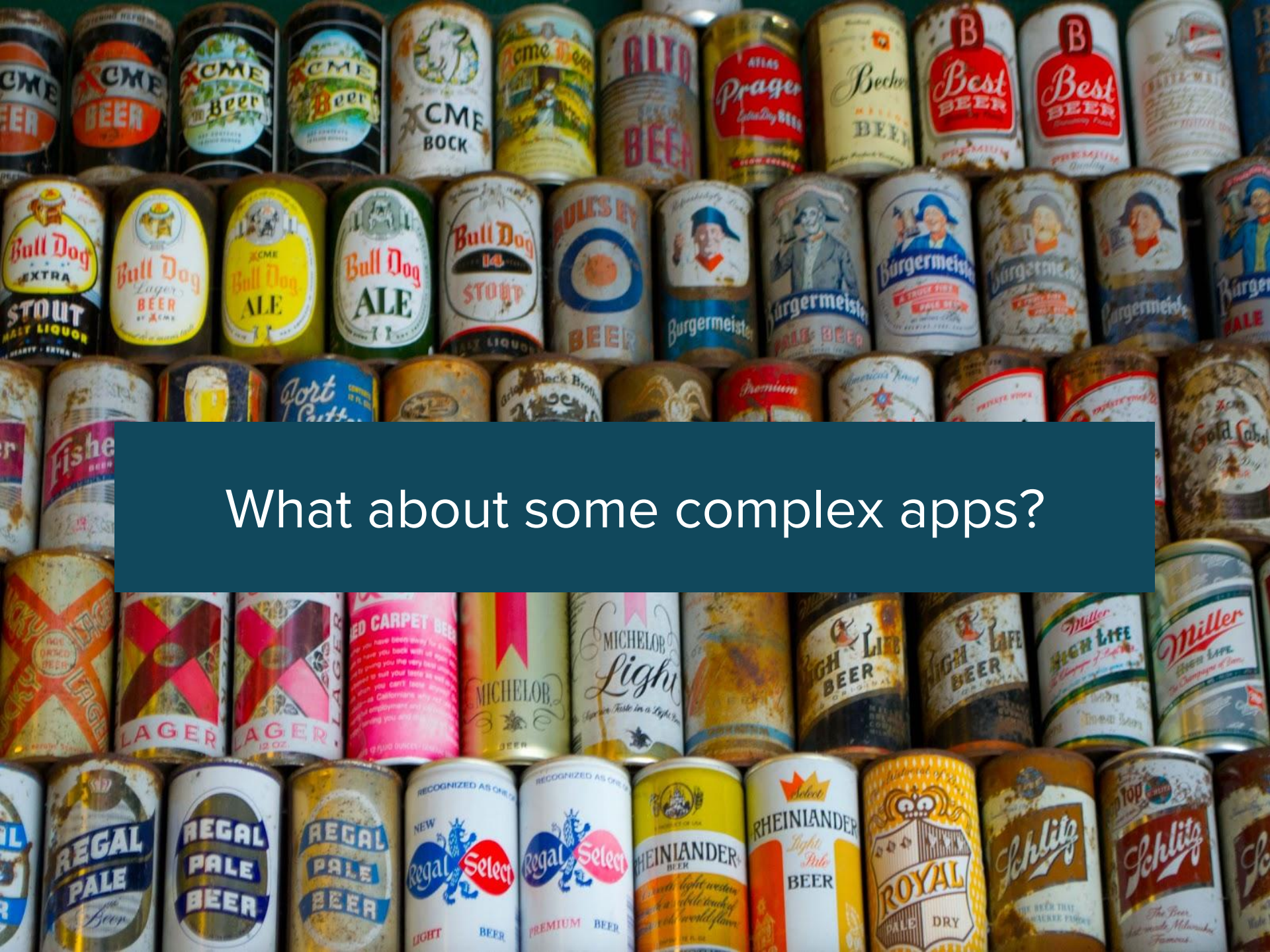
As with apartments, the most secure containers have strong walls between them. You don't want one compromised container to result in the whole system being compromised.



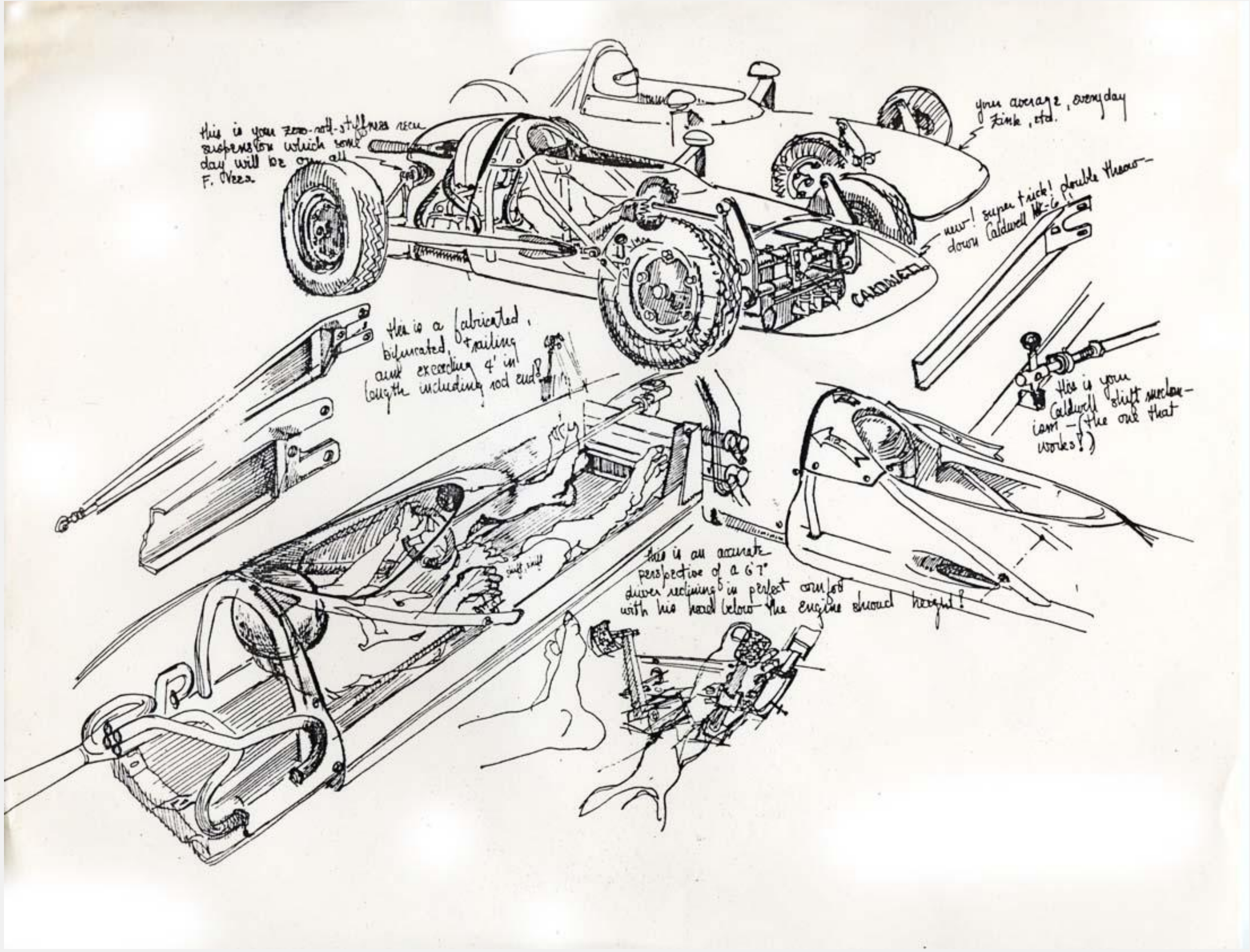
This is very important with containers, because the kernel is shared. What makes the Red Hat "Brick Apartment Building" more secure? SELinux, for one...

https://github.com/fedoradesign/coloringbook-containers/raw/master/Print-Ready/Pages.pdf

What is the thing that matters?


What about some complex apps?

6. DISTRIBUTION OF COMPLEX CONTAINER APPS


this is your zero-roll-stiffness rear suspension which some day will be on all F. Vees.
your average, everyday Fink, etc.
new! super trick! double throw-down Caldwell HK-6!!
this is a fabricated, bifurcated, trailing arm exceeding 4' in length including rod ends!
this is your Caldwell shift mechanism — (the one that works!)
this is an accurate perspective of a 6'7" driver reclining in perfect comfort with his head below the engine shroud height!

How to distribute...

- “how to run” instructions
  - readme
  - bash scripts
  - orchestration specs (kubernetes)

```
#> curl http://some-random.web/run | bash
```

“**Nulecule** is a standard way of defining multi-container application’s configuration without need to distribute instructions to end-user”

“Nulecule is a **standard** way of defining multi-container application’s configuration without need to distribute instructions to end-user”

“Nulecule is a standard way of **defining** multi-container **application’s** configuration without need to distribute instructions to end-user”

“Nulecule is a standard way of defining **multi-container application**’s configuration without need to distribute instructions to end-user”

“Nulecule is a standard way of defining multi-container application’s **configuration** without need to distribute instructions to end-user”

“Nulecule is a standard way of defining multi-container application’s configuration without need to **distribute** instructions **to end-user**”

Tip #25:
Use Nulecule to deliver artefacts to run container applications.

Further reading for long winter evenings...

redhat.

Software Collections: https://www.softwarecollections.org/en/docs/guide/
Mailing list about Software Collections: <sclorg@redhat.com>
SCLo SIG @ CentOS: http://wiki.centos.org/SpecialInterestGroup/SCLo
Nulecule home: https://github.com/projectatomic/nulecule
Sources of Docker images: https://github.com/sclorg/rhscl-dockerfiles
Example of Nulecule app: https://github.com/hhorak/guestbook-pgsql
Documentation for RHSCL's images: https://access.redhat.com/articles/1752723

Container is not a virtual machine.
Avoid root access.
Focus on common use cases.
Support source-to-image tool.
Use Nulecule for distribution.

Do not forget, content does matter.

Honza Horak <hhorak@redhat.com>
@HonzaHorak

https://hhorak.fedorapeople.org/2016/160206_How_to_Develop_Containers_in_Enterprise_World_DevConf_2016.pdf