



PATH FROM SOFTWARE COLLECTIONS TO CONTAINERS FOR OPENSIFT

Honza Horak <hhorak@redhat.com>
CentOS Dojo, Brussels, 29th Jan 2016

The goal

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



What this talk includes

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

1. CONTAINERS BASICS



Установка
санитарных
услуг
по 8001
1047028

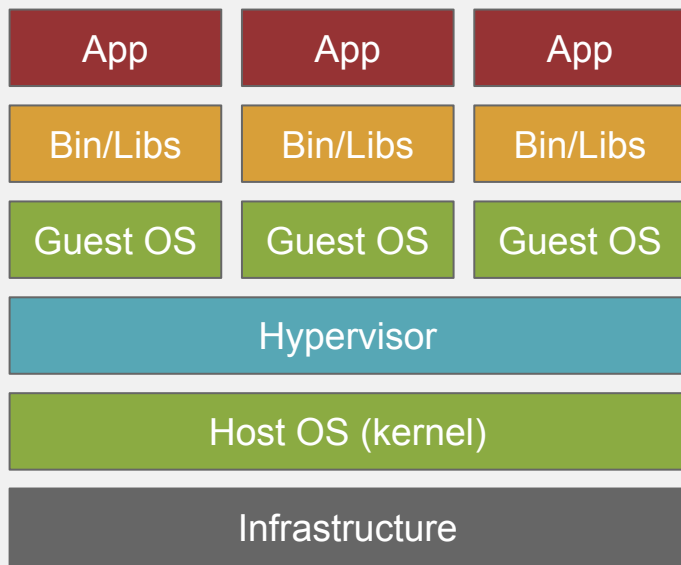
1047028

РАСКРУСЦЕНТРАНС
ТЕЛ. 400-10-11

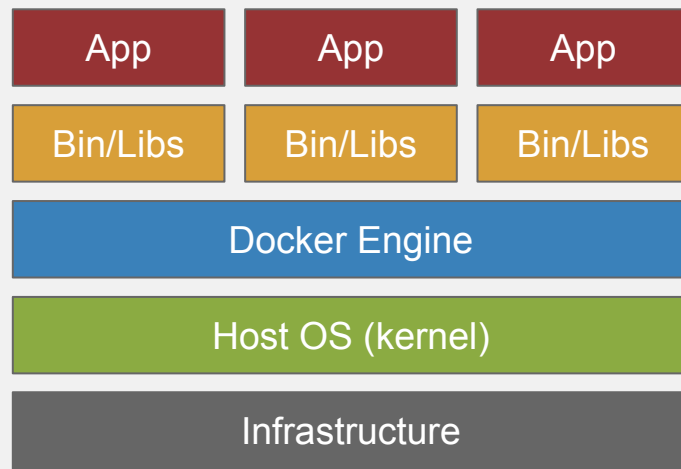
Tip #0:
Content matters.

Applications as micro services

Traditional Virtual Machine

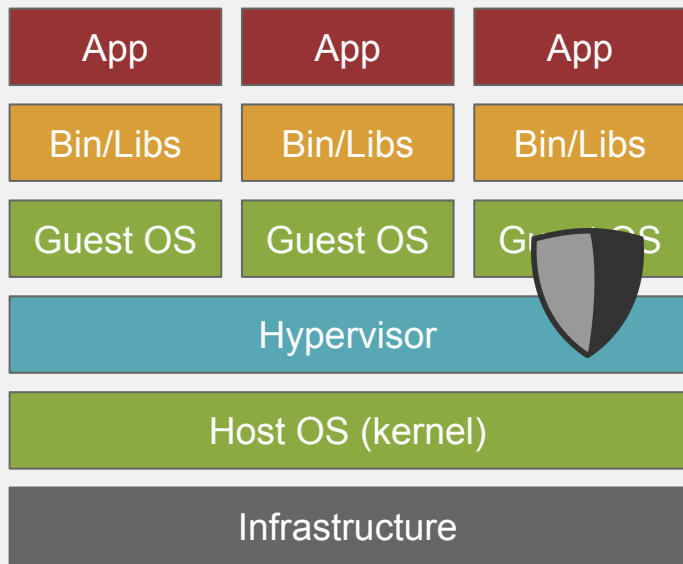


Linux Containers (e.g. Docker)

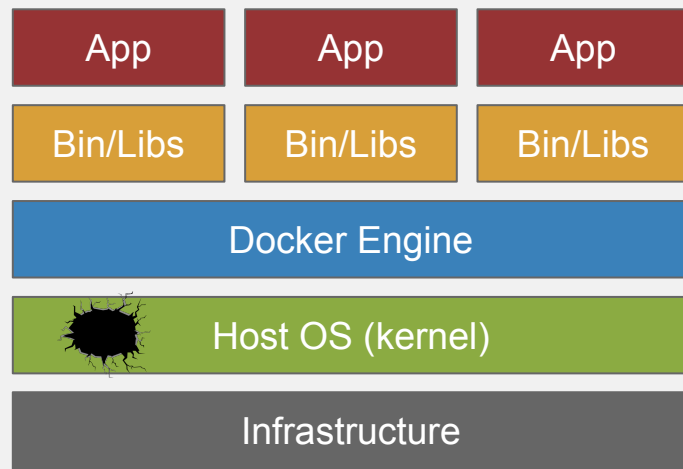


Container is not a virtual machine

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 centos:7

#> docker run -ti --name mycont centos:7 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 centos:centos7
RUN echo Hello Dojo > /root/greeting

#> docker build .
```

2. CREATING POSTGRESQL CONTAINER

Installing RPMs in container

```
#> cat Dockerfile
FROM centos:centos7
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 centos:centos7  
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:
Make small containers.

Installing RPMs in container

```
#> cat Dockerfile
FROM centos:centos7
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 centos:centos7  
----> c8a648134623  
Step 2 : RUN yum -y --setopt=tsflags=nodocs install postgresql-server  
&& yum clean all  
----> Using cache  
----> 29036308c1ec  
Successfully built 29036308c1ec
```

Tip #4:
Be careful about docker cache.

Installing RPMs in container

```
#> cat Dockerfile
FROM centos:centos7
RUN yum -y --setopt=tsflags=nodocs install postgresql-server && \
    yum clean all

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

Are we there yet?

Make container do something

```
#> cat Dockerfile

FROM centos:centos7

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 #5:
Use non-root user wherever possible.

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 #6:
Use `exec` for final process.

Running PostgreSQL container

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

```
#> docker run -ti postgresql
```

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  
Password: _
```



Tip #7:
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

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 -p 5432:5432 --name p1 \  
-e POSTGRESQL_ADMIN_PASSWORD=pass postgresql  
b1e23c844346d2788d7b7891d8f78244788f71b19dcf291b05cdf1d7685ef556
```

```
#> 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 "max_connections = ${POSTGRES_MAX_CONNECTIONS}" >>${PGDATA}
/postgresql.conf
...
```

Tip #8:
Support only most common
configuration options.

Tip #9: Use expected paths

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

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 \  
    postgresql-94-centos7
```

3. PYTHON CONTAINER TO BUILD APPLICATIONS

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile
```

```
FROM centos:7
```

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

Simplest Python container

Spotted an issue?

```
#> cat Dockerfile
```

```
FROM centos:7
```

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

Building simplest Python container

```
#> docker build -t python-27-centos7 .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM centos:7
---> 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 #10:
Do not believe yum.

How to build application on top of it?

Building app container

```
#> cat Dockerfile
FROM python-27-centos7
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 #11:
Help users to be more effective.

Building app container using s2i

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

```
#> s2i build /path/to/guestbook python-27-centos7 guestbook
```

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`.

How source-to-image works

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

Tip #12:
Let users use their favourite frameworks.

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 #13:
Do not just install software.
Build micro-services.

Microservices running on usual ports

```
#> cat Dockerfile
```

```
...
```

```
EXPOSE 8080
```

```
...
```

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

Tip #14:
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/joe/guestbook.git \  
      --context-dir=app/ centos/python-34-centos7 guestbook  
  
$> docker run -p 8080:8080 guestbook
```

Content 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
```

Tip #15:
Do not be afraid to combine
containers & Software Collections.

A large, dense collection of vintage beer cans is displayed in several rows. The cans feature various labels and designs, including "CME BEER", "Bull Dog", "Buller's", "Burgermeister", "Miller", "Schlitz", and "Regal". The cans are arranged in a way that shows a variety of historical beer branding. A dark blue rectangular box is superimposed over the middle of the image, containing the text "Which container includes a collection?".

Which container includes a collection?

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 needed for YUM

How SCL may be handy in container

- 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

5. APPLICATION CONTAINERS IN RED HAT AND CENTOS PORTFOLIO

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)

Images based on Software Collections

made available by SCLo SIG in CentOS

Databases Collections	Red Hat's registry	docker.io
mariadb55		
mongodb24	openshift/mongodb-24-rhel7	openshift/mongodb-24-centos7
mysql55	openshift/mysql-55-rhel7	openshift/mysql-55-centos7
postgresql92	openshift/postgresql-92-rhel7	openshift/postgresql-92-centos7
rh-mariadb100	rhsc1/mariadb-100-rhel7	centos/mariadb-100-centos7
rh-mongodb26	rhsc1/mongodb-26-rhel7	centos/mongodb-26-centos7
rh-mysql56	rhsc1/mysql-55-rhel7	centos/mysql-56-centos7
rh-postgresql94	rhsc1/postgresql-94-rhel7	centos/postgresql-94-centos7

Images based on Software Collections

made available by SCLo SIG in CentOS

Language Collections (1/2)	Red Hat's registry	docker.io
nodejs010	openshift/nodejs-010-rhel7	openshift/nodejs-010-centos7
perl516	openshift/perl-516-rhel7	openshift/perl-516-centos7
php54		
php55	openshift/php-55-rhel7	openshift/php55-centos7
python27	rhscs/python-27-rhel7	centos/python-27-centos7
python33	openshift/python-33-rhel7	openshift/python-33-centos7
rh-perl520	rhscs/perl-520-rhel7	centos/perl-520-centos7
rh-php56	rhscs/php-56-rhel7	centos/php-56-centos7

Images based on Software Collections

made available by SCLo SIG in CentOS

Language Collections (2/2)	Red Hat's registry	docker.io
rh-python34	rhsc1/python-34-rhel7	centos/python-34-centos7
rh-ror41	rhsc1/ror-41-rhel7	centos/ror-41-centos7
rh-ruby22	rhsc1/ruby-22-rhel7	centos/ruby-22-centos7
ror40		
ruby193		
ruby20	openshift/ruby-20-rhel7	openshift/ruby-200-centos7
rh-passenger40	rhsc1/passenger-40-rhel7	centos/passenger-40-centos7

Images based on Software Collections

made available by SCLo SIG in CentOS

Others Collections	Red Hat's registry	docker.io
httpd24	rhsc/httpd-24-rhel7	centos/httpd-24-centos7
nginx14		
nginx16	rhsc/nginx-16-rhel7	centos/nginx-16-centos7
devassistant09		
git19		
thermostat1		
maven30, rh-java-common		
devtoolset-4-toolchain	rhel7/devtoolset-4-toolchain	

Tip #16:
Use containers from reliable provider.

Tip #16:
Use containers from reliable provider.
like Red Hat or CentOS :)

Tip #17:
Make sure SCL is enabled in container.

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-entrpoint"]
...

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

Tip #18:
Do not use ENTRYPOINT for anything
else than environment change.

How to enable SCL in container

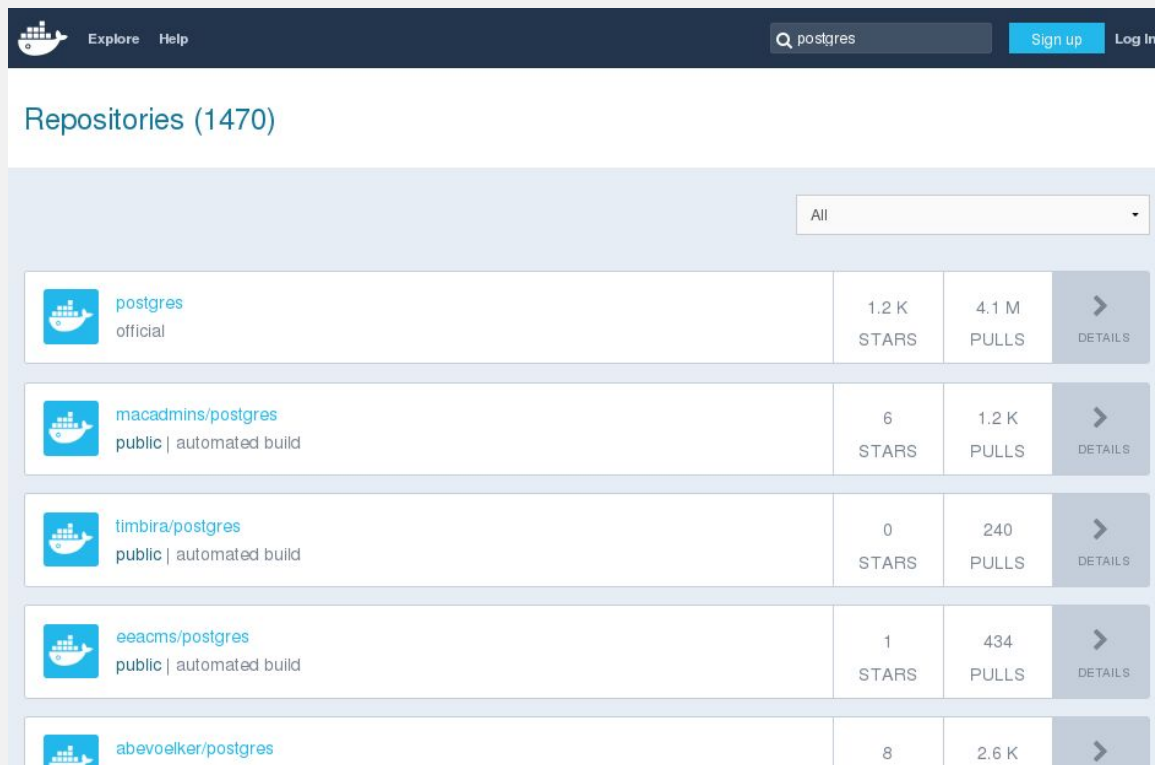
```
#> cat container-entryptpoint
```

```
#!/bin/bash  
set -eu  
cmd="$1"; shift  
exec $cmd "$@"
```






Tip #19:
Hide that we're using SCL underneath.

“Running a container from Docker Hub is
same as running `curl ... | sudo bash`”
(unknown developer).

Which one?



The screenshot shows the Quay.io search results for 'postgres'. The header includes a search bar with 'postgres' entered, and links for 'Explore', 'Help', 'Sign up', and 'Log in'. Below the header, the title 'Repositories (1470)' is displayed. A dropdown menu is set to 'All'. The results are shown in a table with columns for repository name, stars, pulls, and a details link.

Repository	Stars	Pulls	Details
 postgres official	1.2 K	4.1 M	DETAILS
 macadmins/postgres public automated build	6	1.2 K	DETAILS
 timbira/postgres public automated build	0	240	DETAILS
 eeacms/postgres public automated build	1	434	DETAILS
 abevoelker/postgres	8	2.6 K	DETAILS

Tip #20:
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 #21:
Consider what is part of image's API.

Paths

- /usr rather than /usr/local
- hide the /opt (Software Collections specifics)
- expected paths for volumes /var/lib/..., configuration

Tip #22:
Set metadata to containers.

OpenShift and Kubernetes labels

```
LABEL io.k8s.description="MySQL database server" \  
      io.k8s.display-name="MySQL 5.6" \  
      io.openshift.expose-services="3306:mysql" \  
      io.openshift.tags="database,mysql,mysql56,rh-mysql56"
```

Tip #23:
Take security seriously.

Security

“Containers do not contain”

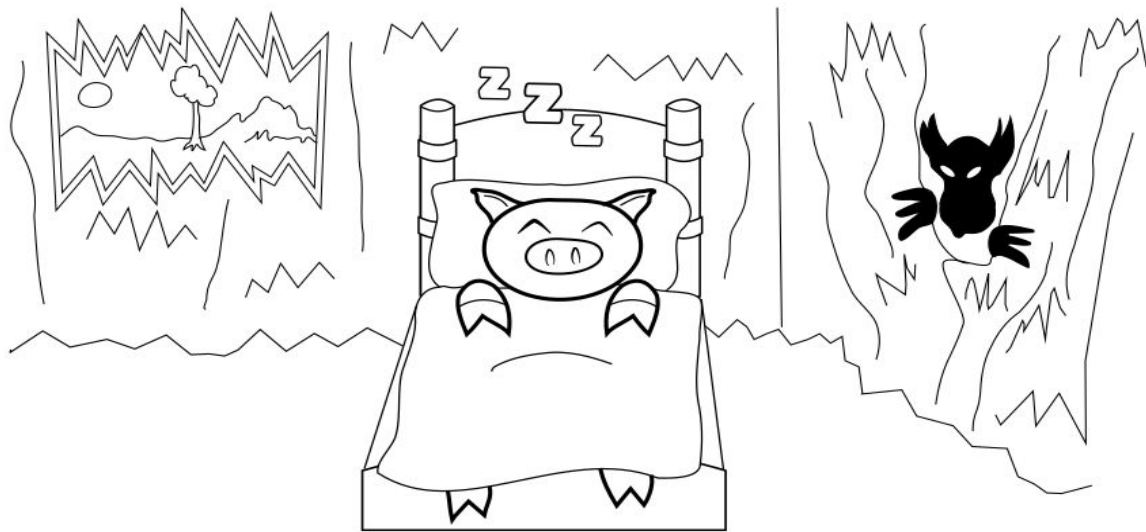
- colouring book by Dan Walsh

<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...

What is the thing that matters?

What about some complex apps?

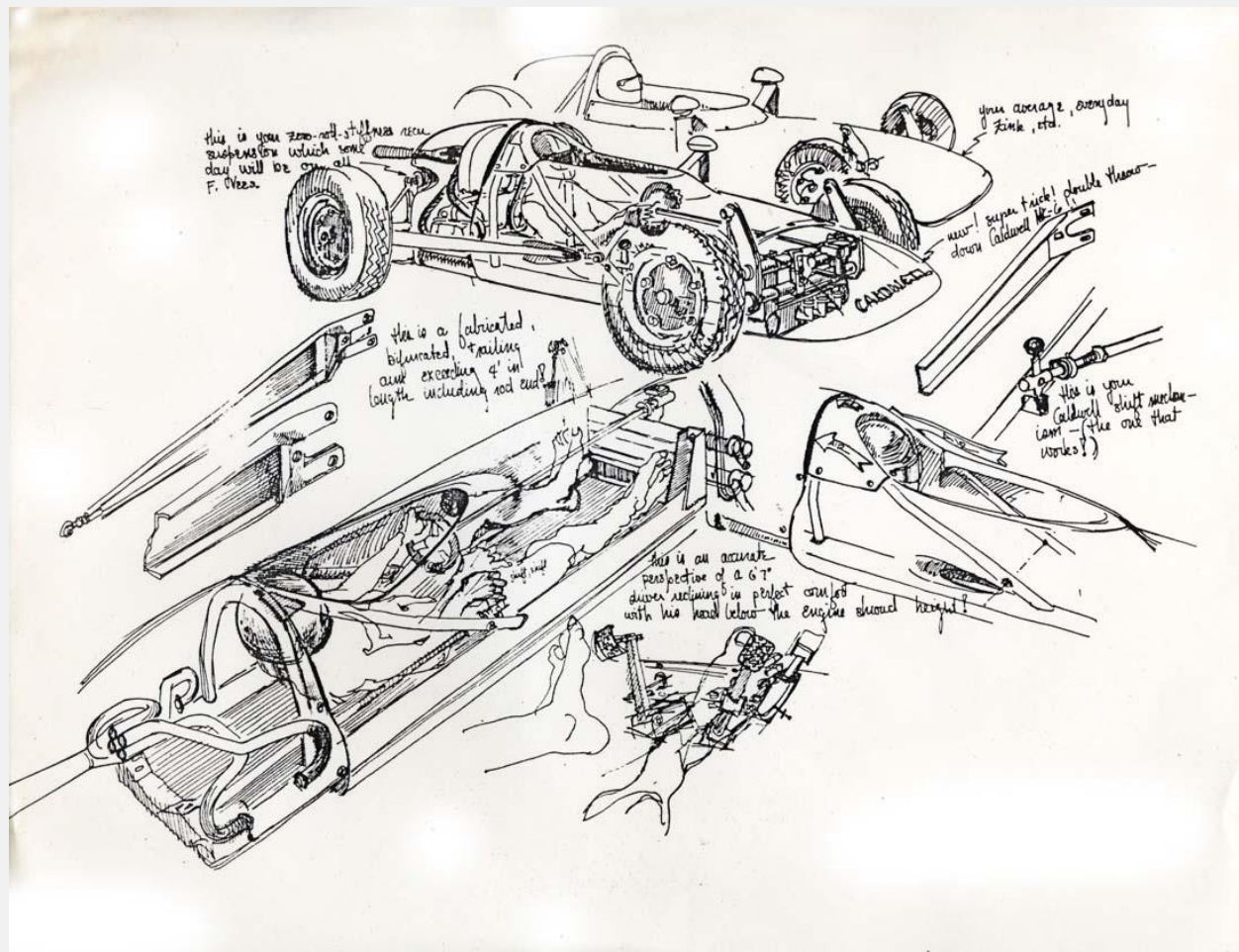
6. DISTRIBUTION OF CONTAINER APPS

Application development with containers

Because we want to develop applications, no packages.

- flexibility
- grouping
- isolation
- transparency





How to distribute...

- “how to run” instructions (readme, bash script)
- 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 #24:
Use Nucleule to deliver artefacts to run
container applications.

Nulecule concept

- description of the parameters is done by image author **once**
- Nulecule specification is distributed as container
- user provides only specific missing values
- plug-able providers architecture

Nulecule specification for PostgreSQL

Basic info about application

```
id: postgresql-atomicapp
metadata:
  name: PostgreSQL Atomic App
  description: PostgreSQL database available as Atomic App
graph:
  - name: postgresql-atomicapp
    params:
      - name: db_user
        description: Database User
      - name: db_pass
        description: Database Password
      - name: db_name
        description: Database Name
    artifacts:
      docker:
        - file://artifacts/docker/postgresql-app-run
```

Nulecule specification for PostgreSQL

Specification for docker parameters

```
id: postgresql-atomicapp
metadata:
  name: PostgreSQL Atomic App
  description: PostgreSQL database available as Atomic App
graph:
  - name: postgresql-atomicapp
    params:
      - name: db_user
        description: Database User
      - name: db_pass
        description: Database Password
      - name: db_name
        description: Database Name
    artifacts:
      docker:
        - file://artifacts/docker/postgresql-app-run
```

Nulecule specification for PostgreSQL

Specification for docker provider

```
id: postgresql-atomicapp
metadata:
  name: PostgreSQL Atomic App
  description: PostgreSQL database available as Atomic App
graph:
  - name: postgresql-atomicapp
    params:
      - name: db_user
        description: Database User
      - name: db_pass
        description: Database Password
      - name: db_name
        description: Database Name
  artifacts:
    docker:
      - file://artifacts/docker/postgresql-app-run
```

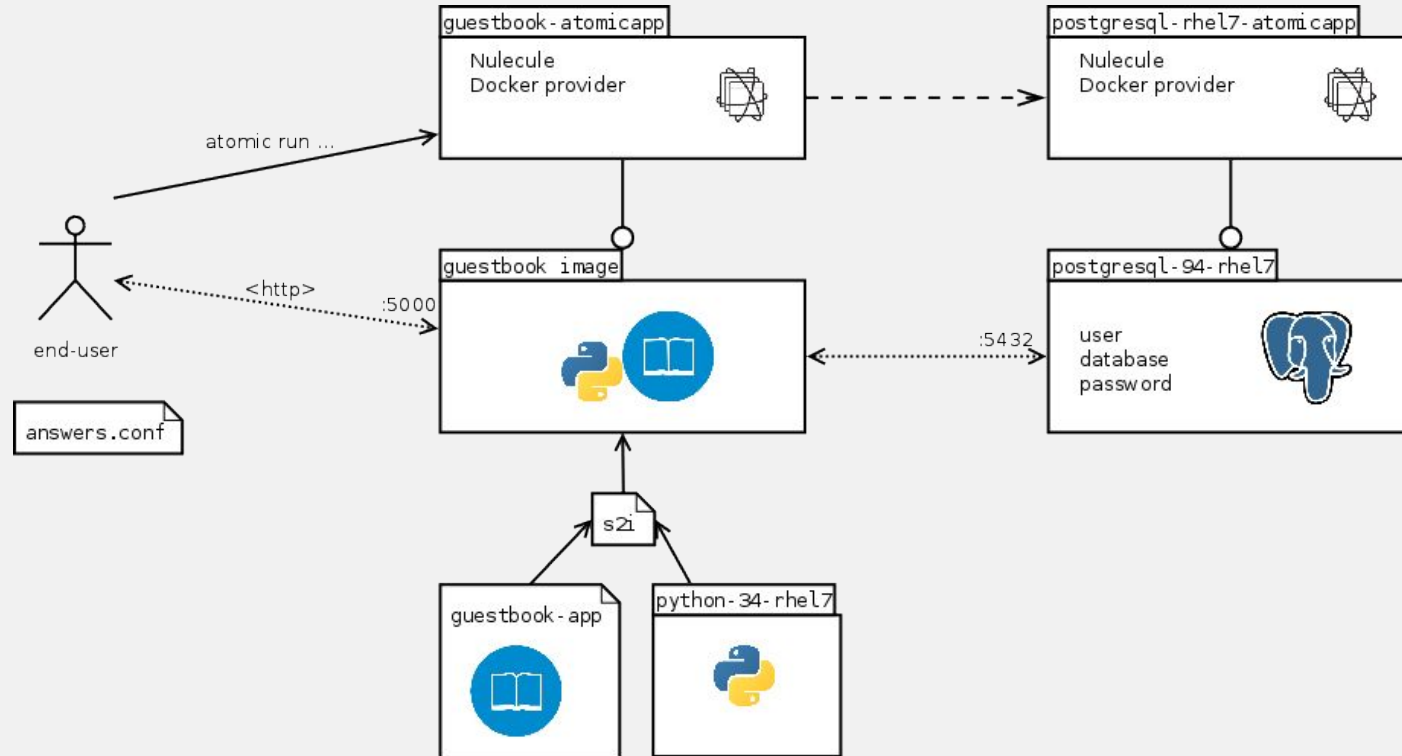
Nulecule specification for PostgreSQL

Building and running the image with Nulecule specification

```
#> cd postgresql-rhel7-atomicapp  
#> docker build -t projectatomic/postgresql-rhel7-atomicapp:latest .
```

```
#> atomic run projectatomic/postgresql-rhel7-atomicapp
```

Complete schema of Nuleculized app





Thanks.

Software Collections home: <https://www.softwarecollections.org/en/docs/guide/>

Nulecule home: <https://github.com/projectatomic/nulecule>

Sources of Docker images: <https://github.com/sclorg/rhsc-dockerfiles>

Example of Nulecule app: <https://github.com/hhorak/guestbook-pgsql>

Mailing list about Software Collections: <sclorg@redhat.com>

Honza Horak <hhorak@redhat.com>

@HonzaHorak

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



**Do not forget,
content does matter.**



Honza Horak <hhorak@redhat.com>
@HonzaHorak

https://hhorak.fedorapeople.org/2016/160128_Path_from_Software_Collections_to_Containers_for_OpenShift.pdf