redhat.

DATABASE CONTAINERS IN ENTERPRISE WORLD

Honza Horak <hhorak@redhat.com>
PGConf.eu, 29th Oct 2015

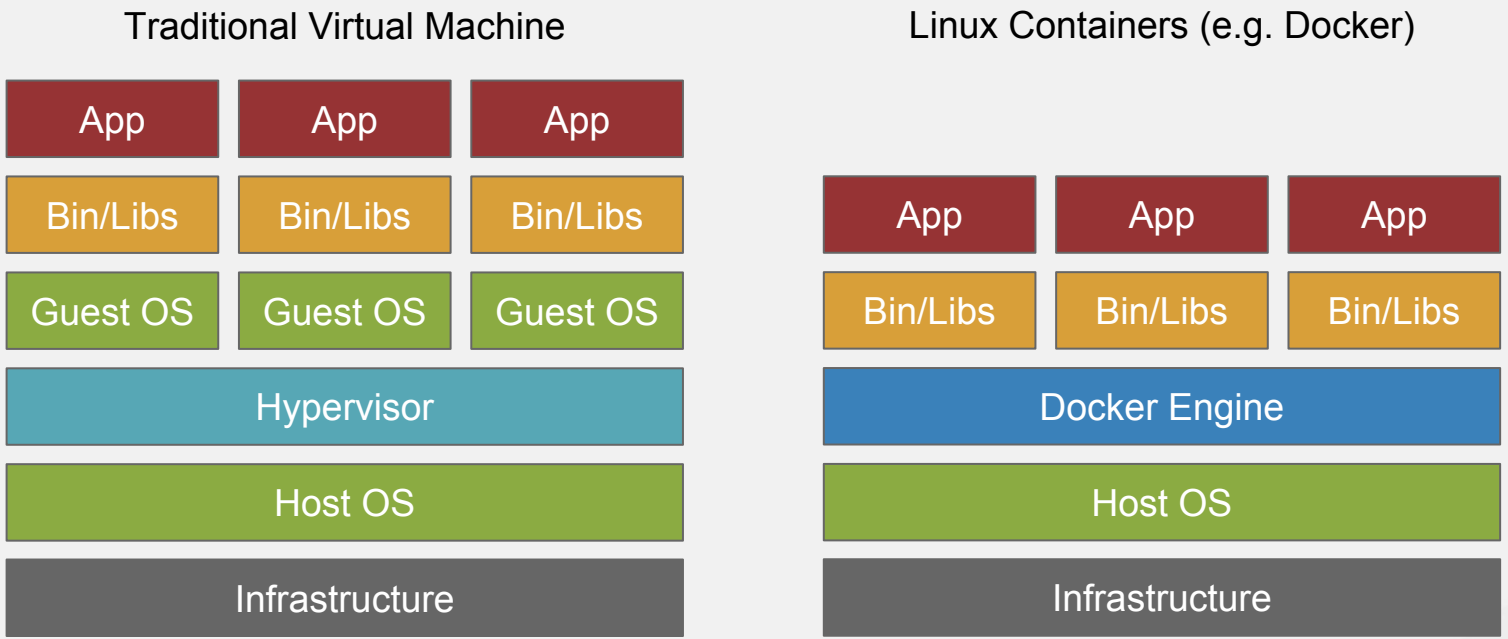
What this talk includes

- alternatives to containers in traditional system
- what we care about when creating containers in Red Hat (12 tips)
- example of container-based application based on PostgreSQL 9.4 and Python 3.4 containers

Containers.


ВАТ "Київспецтранс"
ТЕЛ. 400-10-11

Containers in principle



Example of simplified Dockerfile

```
FROM centos:centos7

RUN yum -y --setopt=tsflags=nodocs install gettext bind-utils rh-
postgresql94 epel-release && \
    mkdir -p /var/lib/pgsql/data && chown postgres.postgres
/var/lib/pgsql/data

ADD run-*.sh /usr/bin/

VOLUME ["/var/lib/pgsql/data"]
EXPOSE 5432
USER 26

CMD ["/usr/bin/run-postgresql.sh"]
```

Example of PostgreSQL 9.4 container

```
# 1) pull image from docker hub (docker.io)
#> docker pull centos:centos7

# 2) build a layered image on top of the centos7 image
#> docker build --no-cache -t postgresql-94 .

# 3) run a container from postgresql-94 image
#> docker run -d \
          -p 5432:5432 \
          -e POSTGRESQL_ADMIN_PASSWORD=secret \
          postgresql-94

# 4) connect to the container from the host
#> psql -h 172.17.0.24 -U guestbook
```

Tip #0:
Use kubernetes for orchestration.

Why do we need containers?

Application development with containers

Because we want to develop applications, no packages.

- flexibility
- grouping
- isolation
- transparency

Traditional Linux distribution = one version.
More versions = more problems.

What is the whole problem of more versions on one system?

Conflicts.

Trying to install more versions

We see a conflict.

```
#> dnf install community-mysql-server
Error: package community-mysql-server-5.6.26-1.fc23.x86_64 requires
community-mysql-common(x86-64) = 5.6.26-1.fc23, but none of the
providers can be installed
(try to add '--allowerasing' to command line to replace conflicting
packages)
```

Tip #1:
Do not forget about non-container world.

Software Collections are about having all versions of any software on your system using RPM. Together.

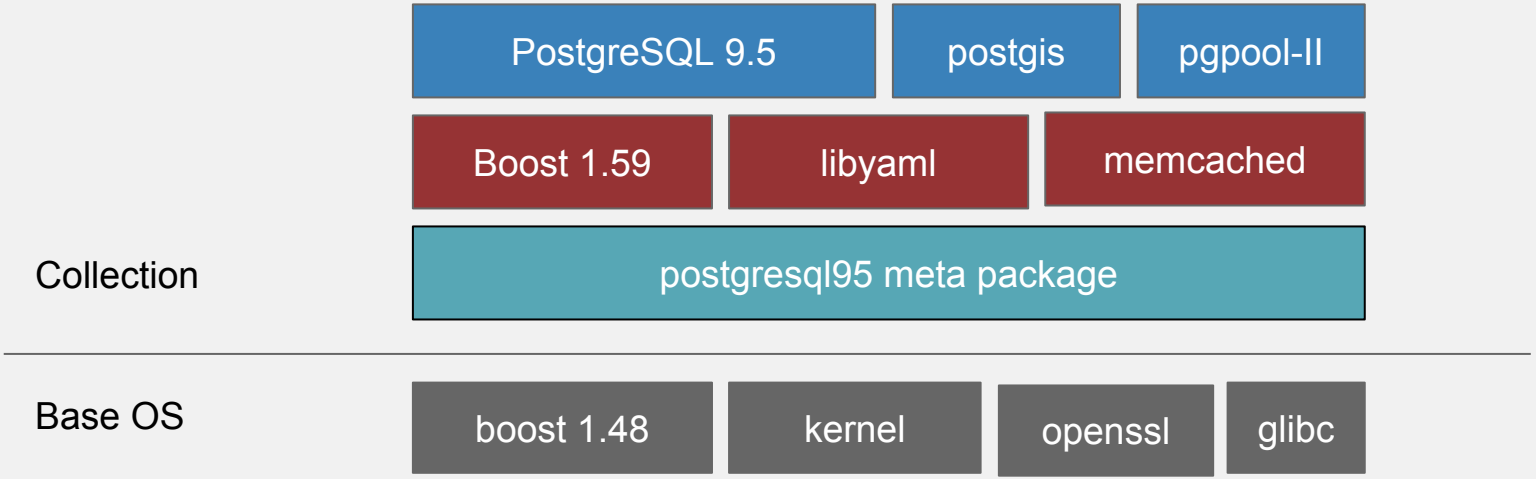
Software Collections principles

softwarecollectoins.org

redhat.

What's in Software Collections

RPM macros-based technology.



Avoiding conflict with base system

and between other collections

- packages name level
- filesystem level
- RPM metadata (provides, requires) level

Avoiding conflict with base system on

packages name level

```
#> 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!
```

Avoiding conflict with base system on

filesystem level

```
#> rpm -ql rh-postgresql94-postgresql-{libs,server,}
/opt/rh/rh-postgresql94/root/usr/lib64/libpq.so.rh-postgresql94-5
/opt/rh/rh-postgresql94/root/usr/lib64/libpq.so.rh-postgresql94-5.7
/opt/rh/rh-postgresql94/root/usr/bin/initdb
/opt/rh/rh-postgresql94/root/usr/bin/pg_ctl
/opt/rh/rh-postgresql94/root/usr/bin/postgres
/opt/rh/rh-postgresql94/root/usr/bin/postgresql-setup
/opt/rh/rh-postgresql94/root/usr/bin/postmaster
/opt/rh/rh-postgresql94/root/usr/lib64/pgsql/ascii_and_mic.so
/opt/rh/rh-postgresql94/root/usr/lib64/pgsql/cyrillic_and_mic.so
/opt/rh/rh-postgresql94/root/usr/share/pgsql/pg_service.conf.sample
/usr/lib/systemd/system/rh-postgresql94-postgresql.service
/usr/lib/systemd/system/rh-postgresql94-postgresql@.service
```

Avoiding conflict with base system on

RPM metadata (provides, requires) level

```
#> rpm -q --provides rh-postgresql94-postgresql-libs
libecpg.so.rh-postgresql94-6()(64bit)
libecpg_compat.so.rh-postgresql94-3()(64bit)
libpgtypes.so.rh-postgresql94-3()(64bit)
libpq.so.rh-postgresql94-5()(64bit)
rh-postgresql94-postgresql-libs = 9.4.4-1.el7
rh-postgresql94-postgresql-libs(x86-64) = 9.4.4-1.el7
scl-package(rh-postgresql94)
```

Example of running SCL

The whole magic is in changing environment variables

```
#> cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.1 (Maipo)

#> psql -V
psql (PostgreSQL) 9.2.10
```

Example of running SCL

The whole magic is in changing environment variables

```
#> scl enable rh-postgresql94 'psql -V'
psql (PostgreSQL) 9.4.4
```

Example of running SCL

The whole magic is in changing environment variables

```
#> scl enable rh-postgresql94 bash
#> psql -V
psql (PostgreSQL) 9.4.4

#> echo $PATH
/opt/rh/rh-postgresql94/root/usr/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/root/bin
```

Software Collections imperfection

- standard package-based approach needed
  - ansible, puppet
- package application as depended SCL
  - requires substantial RPM and SCL knowledge
- distro-specific only


Containers world is different

Containers culture is different

- common format for all modern Linux distributions
- Project Atomic
  - atomically updated host with only containers used as apps
- kubernetes becoming orchestration standard
  - OpenShift also uses kubernetes
- http://www.opencontainers.org

Tip #2:
Do not be afraid to combine containers & Software Collections.


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

Application Containers in Red Hat Portfolio

Images available now as Beta

- some older images use openshift/ namespace
- newer images
  - use rhscl/ in Red Hat Registry (rhscl_beta namespace now)
  - use centos/ in docker.io
- sources available at https://github.com/sclorg/rhscl-dockerfiles

Images based on Software Collections

usable as standalone services

Databases Collections	Namespace openshift/...	Namespace rhscl/...	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		rhscl/mariadb-100-rhel7	centos/mariadb-100-centos7
rh-mongodb26		rhscl/mongodb-26-rhel7	centos/mongodb-26-centos7
rh-mysql56		rhscl/mysql-55-rhel7	centos/mysql-56-centos7
rh-postgresql94		**rhscl/postgresql-94-rhel7**	**centos/postgresql-94-centos7**

Images based on Software Collections

usable as base image or using source-to-image (s2i)

Language Collections (1/2)	Namespace openshift/...	Namespace rhscl/...	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		rhscl/python-27-rhel7	centos/python-27-centos7
python33	openshift/python-33-rhel7		openshift/python-33-centos7
rh-perl520		rhscl/perl-520-rhel7	centos/perl-520-centos7
rh-php56		rhscl/php-56-rhel7	centos/php-56-centos7

Images based on Software Collections

usable as base image or using source-to-image (s2i)

Language Collections (2/2)	Namespace openshift/...	Namespace rhscl/...	docker.io
rh-python34		**rhscl/python-34-rhel7**	**centos/python-34-centos7**
rh-ror41		rhscl/ror-41-rhel7	centos/ror-41-centos7
rh-ruby22		rhscl/ruby-22-rhel7	centos/ruby-22-centos7
ror40			
ruby193			
ruby20	openshift/ruby-20-rhel7		openshift/ruby-200-centos7
rh-passenger40		rhscl/passenger-40-rhel7	centos/passenger-40-centos7

Images based on Software Collections

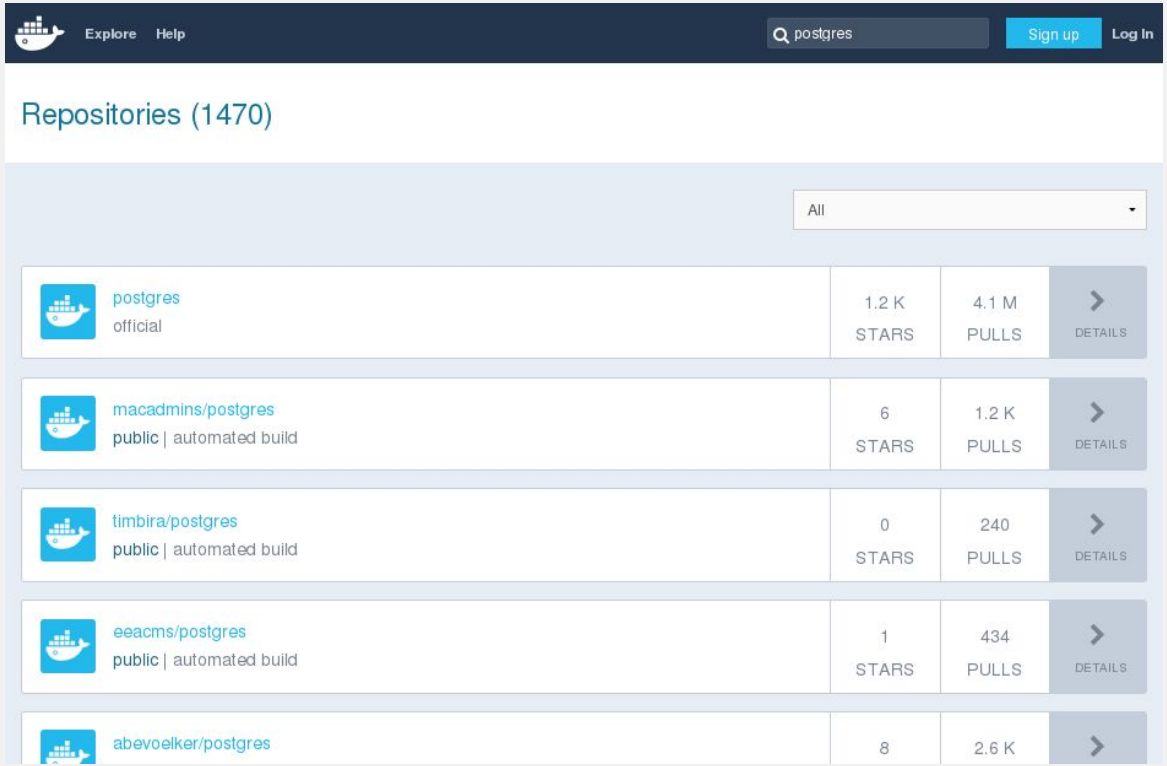
Others Collections	Namespace openshift/...	Namespace rhscl/...	docker.io
httpd24		rhscl/httpd-24-rhel7	centos/httpd-24-centos7
nginx14			
nginx16		rhscl/nginx-16-rhel7	centos/nginx-16-centos7
devassistant09			
git19			
thermostat1			
maven30, rh-java-common			
devtoolset-4-toolchain		rhel7/devtoolset-4-toolchain	

Tip #3:
Consider more run-time environments.

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)

Which one?



Tip #4:
Content matters.

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

Tip #5:
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 #6:
Choose only few parameters to configure container.

Parametrization

- docker run -e IMAGENAME_VARIABLE=value ...
- Example:
  - docker run -e POSTGRESQL_ADMIN_PASSWORD=secret_passwd ...
- variables visible in docker inspect
  - security issue in case of passwords
- limited set of options

PostgreSQL 9.4 container

(alternatively use centos/postgresql-94-centos7 from docker.io)

```
#> docker run --rm rhscl/postgresql-94-rhel7
You must either specify the following environment variables:
  POSTGRESQL_USER (regex: '^[a-zA-Z_][a-zA-Z0-9_]*$')
  POSTGRESQL_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$')
  POSTGRESQL_DATABASE (regex: '^[a-zA-Z_][a-zA-Z0-9_]*$')
Or the following environment variable:
  POSTGRESQL_ADMIN_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,...
Or both.
Optional settings:
  POSTGRESQL_MAX_CONNECTIONS (default: 100)
  POSTGRESQL_SHARED_BUFFERS (default: 32MB)
```

Tip #7:
Consider use cases the image supports.

Use cases

- focus only on most common and basic use cases
- what we've struggled with:
  - master/slave replication ?
  - advanced database tuning ?
  - source-to-image ?
- general approach:
  - Need more? Create a layered image on top of ours.

Tip #8:
Take security seriously.

Security

“Containers do not contain”

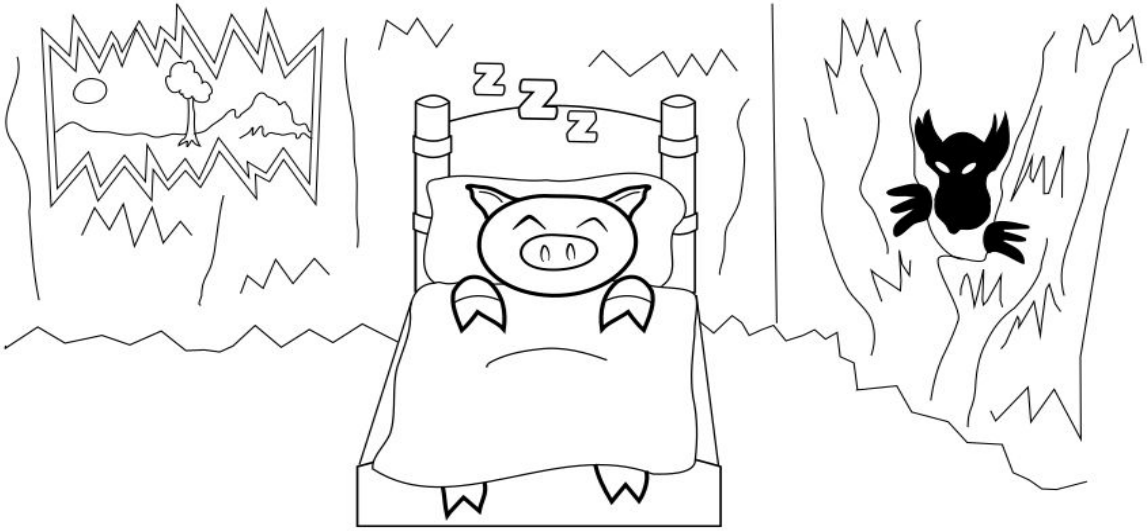
- non-root user by default
- allow to run as any other user
  - docker run -u 8367 ...
  - important in OpenShift
- colouring book by Dan Walsh

  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...

Tip #9:
Content matters.

Tip #10:
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 #11:
Choose a standard way to get user's source to an image.

Source to image

- source-to-image (s2i) tool and concept
- standard way of building container images with adding an application on top of base image

```
#> yum -y install source-to-image
#> s2i build /path/to/guestbook rhscl/python-34-rhel7 guestbook
```

Other best practices

- no or only simple entrypoint (just for set proper environment)
- tricks to enable Software Collections in containers
- logging to stdout
- atomic run support

```
#> atomic run centos/postgresql-94-centos7
```

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"
```

Example of PostgreSQL 9.4 container

Working with PostgreSQL 9.4 container

(alternatively use centos/postgresql-94-centos7 from docker.io)

```
#> docker pull rhscl/postgresql-94-rhel7
#> mkdir /var/lib/pgcont
#> chown postgres.postgres /var/lib/pgcont
```

Working with PostgreSQL 9.4 container

(alternatively use centos/postgresql-94-centos7 from docker.io)

```
#> docker run --rm rhscl/postgresql-94-rhel7
```

Working with PostgreSQL 9.4 container

(alternatively use centos/postgresql-94-centos7 from docker.io)

```
#> docker run --rm rhscl/postgresql-94-rhel7
You must either specify the following environment variables:
  POSTGRESQL_USER (regex: '^[a-zA-Z_][a-zA-Z0-9_]*$')
  POSTGRESQL_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$')
  POSTGRESQL_DATABASE (regex: '^[a-zA-Z_][a-zA-Z0-9_]*$')
Or the following environment variable:
  POSTGRESQL_ADMIN_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,...
Or both.
Optional settings:
  POSTGRESQL_MAX_CONNECTIONS (default: 100)
  POSTGRESQL_SHARED_BUFFERS (default: 32MB)
```

Working with PostgreSQL 9.4 container

running with parameters specified as environment variables

```
#> docker run -d -v /var/lib/pgcont:/var/lib/pgsql/data:Z \
   -p 5432:5432 \
   --name=postgresql \
   -e POSTGRESQL_USER=guestbook \
   -e POSTGRESQL_PASSWORD=secret \
   -e POSTGRESQL_DATABASE=guestbook \
   rhscl/postgresql-94-rhel7
```

Working with PostgreSQL 9.4 container

connecting to running server from host

```
#> psql -h 172.17.0.24 -U guestbook
Password for user guestbook:
psql (9.4.4)
Type "help" for help.

guestbook=>
```

Working with PostgreSQL 9.4 container

connecting to running server from another container

```
#> docker run -ti --link postgresql:db rhscl/postgresql-94-rhel7 bash
bash-4.2$ psql -h $DB_PORT_5432_TCP_ADDR -U guestbook
Password for user guestbook:
psql (9.4.4)
Type "help" for help.

guestbook=>
```

Example of Python 3.4 container

Working with Python 3.4 container

pulling image and just running python interpreter

```
#> docker pull rhscl/python-34-rhel7

#> docker run --rm -ti rhscl/python-34-rhel7 python
Python 3.4.2 (default, Mar 25 2015, 04:25:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Working with Python 3.4 container

creating image with application

```
#> cat Dockerfile
FROM rhscl/python-34-rhel7
...
```

```
#> yum -y install source-to-image
#> s2i build /path/to/guestbook rhscl/python-34-rhel7 guestbook
```

Working with Python 3.4 container

environment variables used in the application

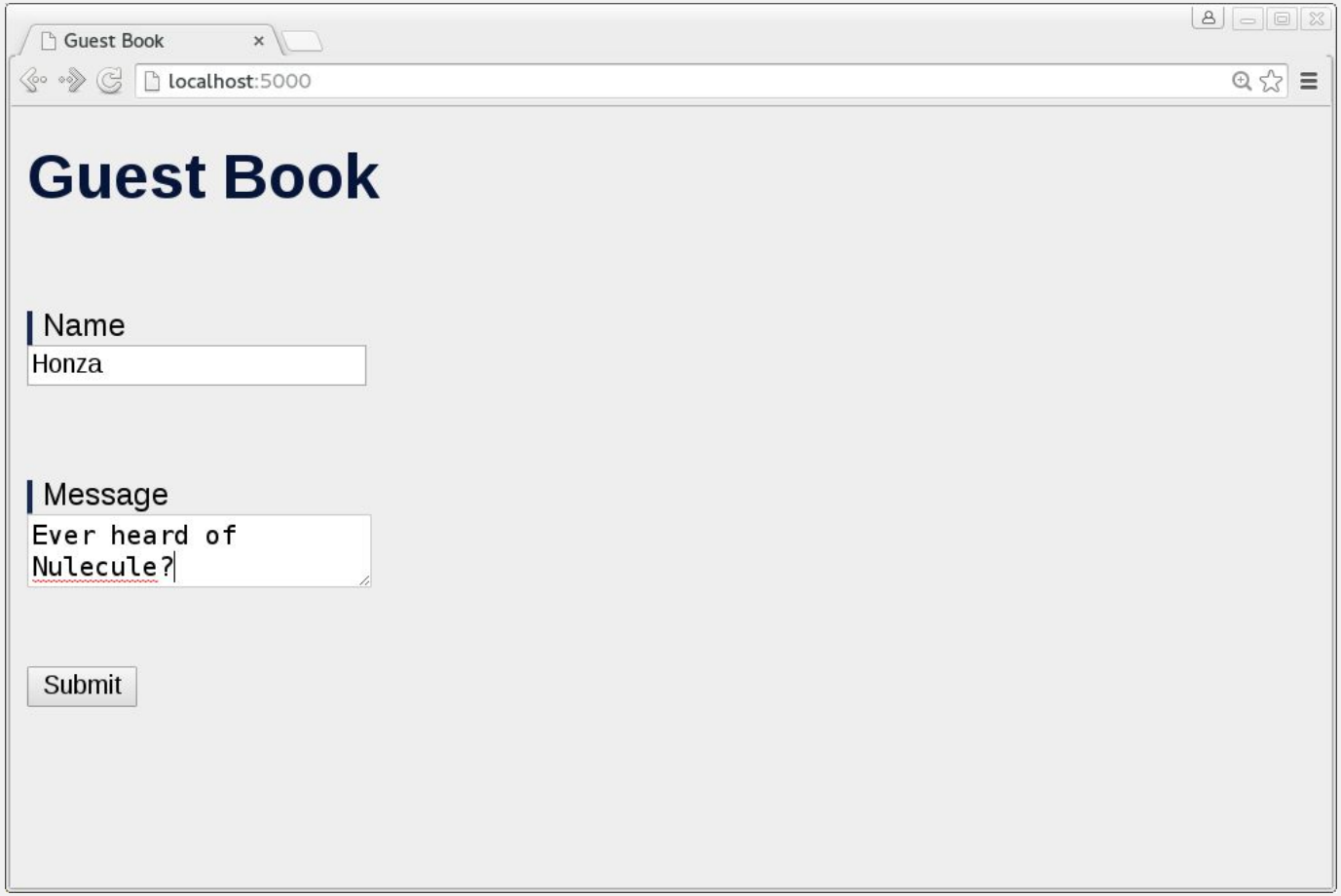
```
$DB_PORT_5432_TCP_ADDR -- IP address of the PostgreSQL database
$DB_PORT_5432_TCP_PORT -- port of the PostgreSQL database
$DB_ENV_POSTGRESQL_USER -- database user used for application
$DB_ENV_POSTGRESQL_PASSWORD -- database password
$DB_ENV_POSTGRESQL_DATABASE -- database name
```

Running application with database

Running both containers

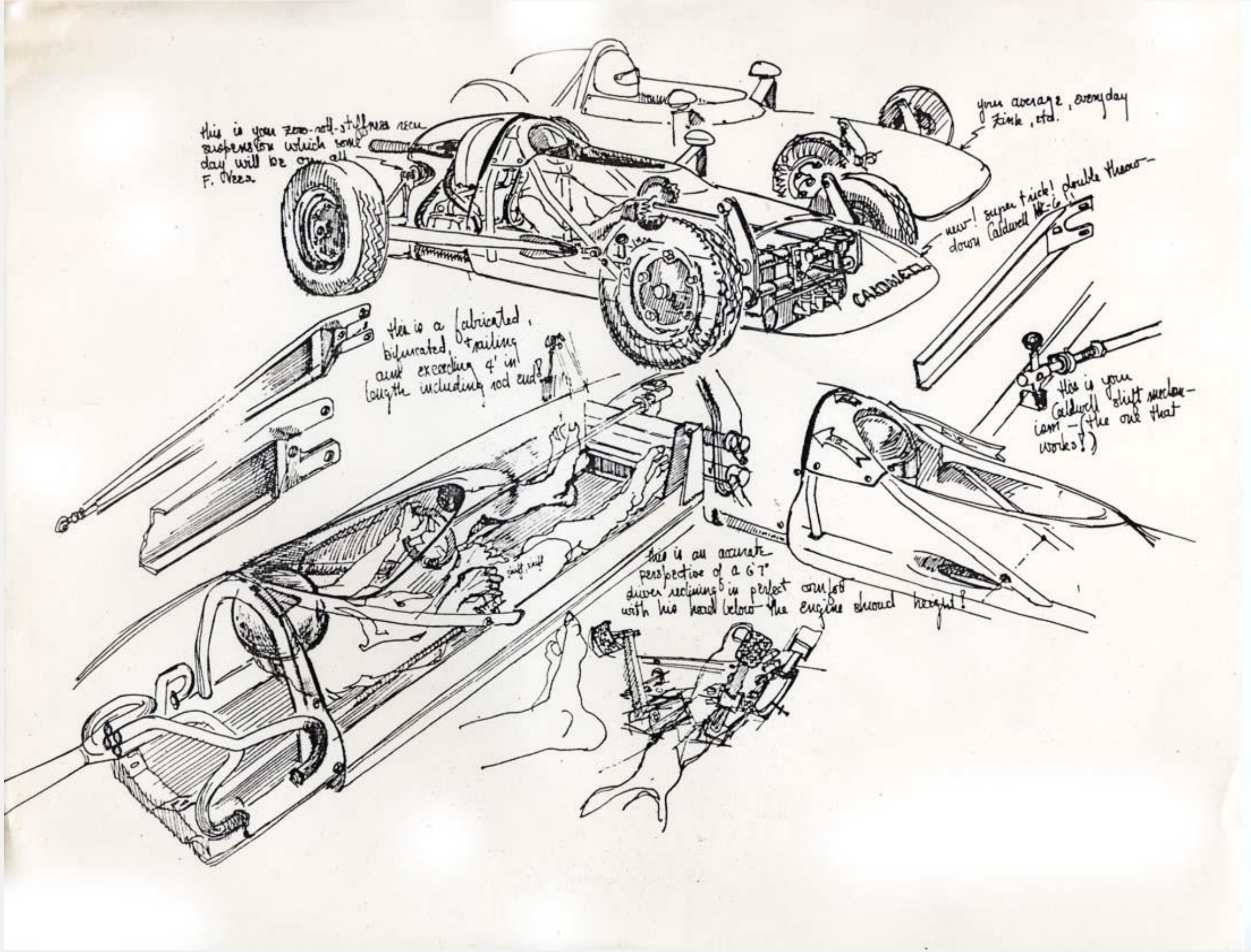
```
#> docker run -d -v /var/lib/pgcont:/var/lib/pgsql/data:Z \
   -p 5432:5432 \
   --name=postgresql \
   -e POSTGRESQL_USER=guestbook \
   -e POSTGRESQL_PASSWORD=secretpassword \
   -e POSTGRESQL_DATABASE=guestbook \
   rhscl/postgresql-94-rhel7

#> docker run --rm -p 5000:5000 --link postgresql:db guestbook
```


Guest Book
localhost:5000
Guest Book
Name
Honza
Message
Ever heard of
Nulecule?
Submit


Is that really what we want to ship?


STRONGBO
GUINNESS


this is your zero-roll-stiffness rear suspension which some day will be on all F. Vees
your average, everyday Zink, etc.
new! super trick! double throw-down Caldwell Mk-6!
CALDWELL
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!

Tip #12:
Use Nulecule to deliver artefacts to run container applications.

“**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**”

How to distribute...

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

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

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

Specification for docker provider

```
#> cat artifacts/docker/postgresql-app-run

docker run -d --name postgresql-atomicapp-app \
       -e POSTGRESQL_USER=$db_user \
       -e POSTGRESQL_PASSWORD=$db_pass \
       -e POSTGRESQL_DATABASE=$db_name \
       rhscl/postgresql-94-rhel7
```

Nulecule specification for PostgreSQL

Dockerfiles for packaging the specification

```
#> cat Dockerfile
FROM projectatomic/atomicapp:0.1.11

LABEL io.projectatomic.nulecule.specversion="0.0.2" \
      io.projectatomic.nulecule.providers="docker"

ADD /Nulecule LICENSE /application-entity/
ADD /artifacts /application-entity/artifacts
```

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
```

Wait, where the values came from?
(user, pass, dbname)

Nulecule specification for PostgreSQL

answers.conf file for avoiding interactivity

```
#> cat answers.conf

[general]
namespace = default
provider = docker

[postgresql-atomicapp]
db_user = guestbook
db_pass = secretpassword
db_name = guestbook
```

Nulecule for guestbook

Nulecule for guestbook application

Basic info about guestbook application

```
id: guestbook-app
metadata:
  name: Guestbook Application
  appversion: 0.0.1
  description: Atomic app for deploying the guestbook Python app
graph:
  - name: guestbookfront-app
    artifacts:
      docker:
        - file://artifacts/docker/guestbook-app-run
  - name: postgresql-rhel7
     source: docker://projectatomic/postgresql-rhel7-atomicapp
```

Nulecule for guestbook application

Specification of the guestbook application

```
id: guestbook-app
metadata:
  name: Guestbook Application
  appversion: 0.0.1
  description: Atomic app for deploying the guestbook Python app
graph:
  - name: guestbookfront-app
    artifacts:
      docker:
        - file://artifacts/docker/guestbook-app-run
  - name: postgresql-rhel7
     source: docker://projectatomic/postgresql-rhel7-atomicapp
```

Nulecule for guestbook application

answers.conf file for avoiding interactivity

```
id: guestbook-app
metadata:
  name: Guestbook Application
  appversion: 0.0.1
  description: Atomic app for deploying the guestbook Python app
graph:
  - name: guestbookfront-app
    artifacts:
      docker:
        - file://artifacts/docker/guestbook-app-run
  - name: postgresql-rhel7
    source: docker://projectatomic/postgresql-rhel7-atomicapp
```

Nulecule for guestbook application

Docker provider artifact

```
#> cat artifacts/docker/guestbook-app-run
docker run -d -p 5000:5000 --link postgresql-atomicapp-app:db guestbook
```

Nulecule for guestbook application

Dockerfile for packaging everything

```
#> cat Dockerfile
FROM projectatomic/atomicapp:0.1.11

LABEL io.projectatomic.nulecule.specversion="0.0.2" \
      io.projectatomic.nulecule.providers="docker"

ADD /Nulecule LICENSE /application-entity/
ADD /artifacts /application-entity/artifacts
```

Nulecule for guestbook application

Building and running the Nulecule application (same answers file)

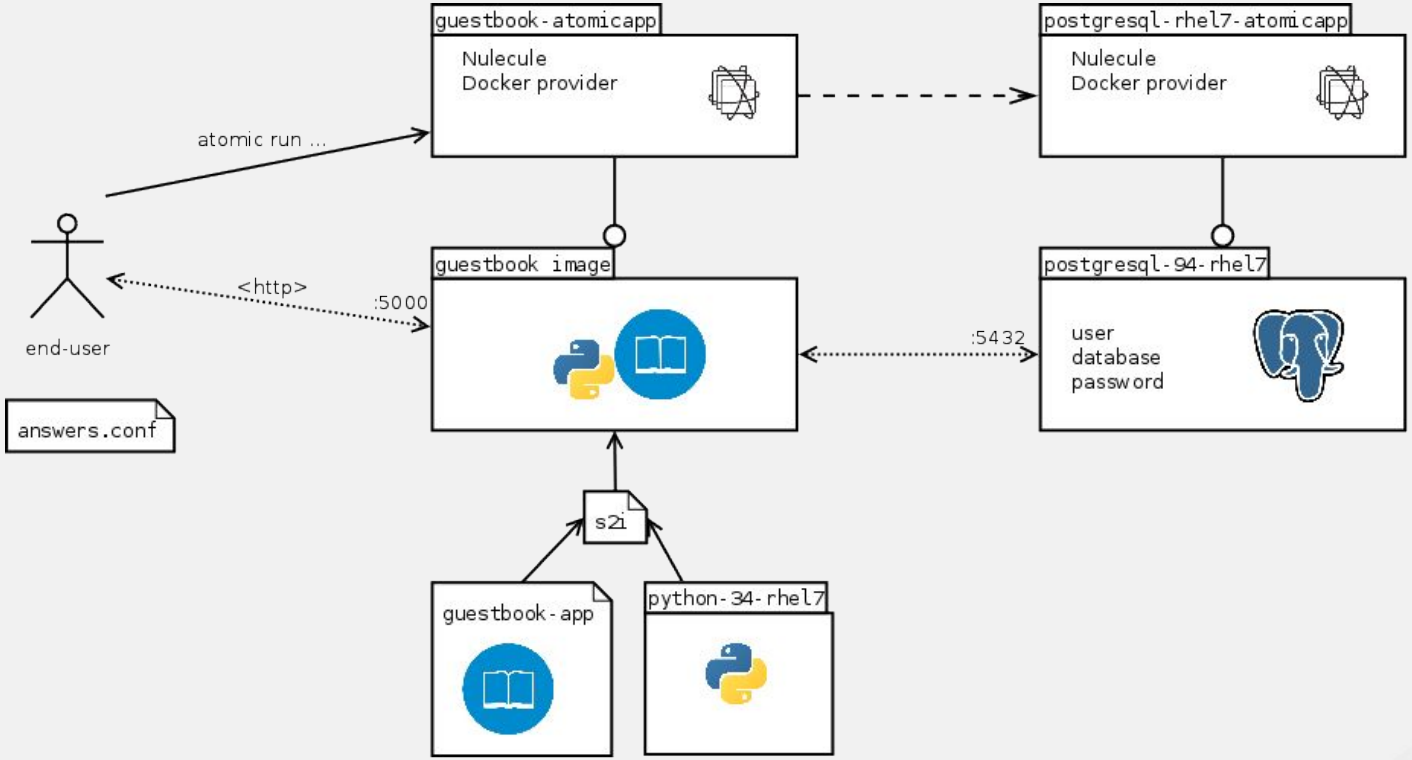
```
#> docker build -t projectatomic/guestbook-atomicapp:latest .

#> atomic run projectatomic/guestbook-atomicapp

#> cat answers.conf
[general]
namespace = default
provider = docker

[postgresql-atomicapp]
db_user = guestbook
db_pass = secretpassword
db_name = guestbook
```

Complete schema of Guest Book app



Tips recap

0. Use kubernetes for orchestration.
1. Do not forget about non-container world.
2. Do not be afraid to combine containers & Software Collections.
3. Consider more run-time environments.
4. Content matters.
5. Think about what the name promises.
6. Choose only few parameters to configure container.

Tips recap

7. Consider use cases the image supports.
8. Take security seriously.
9. Content matters.
10. Consider what is part of image's API.
11. Choose a standard way to get user's source to an image.
12. Use Nulecule to deliver artefacts to run container applications.

redhat.

Thanks.

Software Collections home: https://www.softwarecollections.org/en/docs/guide/
Info about RHEL based images: https://access.redhat.com/articles/1752723
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
Mailing list about Software Collections: <sclorg@redhat.com>

Honza Horak <hhorak@redhat.com>
@HonzaHorak

Do not forget, content does matter.

Honza Horak <hhorak@redhat.com>
@HonzaHorak

https://hhorak.fedorapeople.org/2015/2015-database-containers-in-enterprise-world.pdf