WHAT KEEPS CONTAINER MAINTAINERS AWAKE

Honza Horak
Petr Kubat
Red Hat Platform Engineering
January 2019
Who is this container maintainer
Who is this container maintainer

Click to add subtitle

RPM Package maintainer

- Takes source from upstream
- Maintains RPM SPEC
- Drives productization
- Fixes bugs
Who is this container maintainer

Click to add subtitle

RPM Package Container image maintainer

- Takes source from upstream
- Maintains RPM SPEC Dockerfile and other scripts
- Drives productization
- Fixes bugs
- Tips how to write Dockerfile:
  - [http://docs.projectatomic.io/container-best-practices/](http://docs.projectatomic.io/container-best-practices/)
What we do

A team that develops and maintains several container images

- PostgreSQL, MariaDB, Python, Ruby, PHP, Httpd, Nginx, ...
- OpenShift is the main target platform
- RHEL, CentOS, Fedora
- http://github.com/sclorg/*-container
Challenges we meet
Challenges we meet
Click to add subtitle

No upstream source for the container image
Duplication of the code for different platforms and streams
With more streams: dozens of container images for each platform
Container is a static bundle
  - For each CVE or bug fix or behaviour change it needs to be rebuilt
  - ...and tested
  - Inter-project dependencies
Not every SME is a container master
How we solve the challenges

Container is a bundle vs. one size doesn’t fit all

Users have specific use cases
Flexible container is when:
  ● Tweaking to a specific use case is simple
  ● Changing the configuration by k8s templates
  ● New layer just adds new files

Source-to-image strategy in OpenShift
  ● Configuration of a service becomes an app from s2i perspective

A nice write-up by Eliska:
How we solve the challenges

No upstream source for the container image

Upstream sources under [http://github.com/sclorg](http://github.com/sclorg)
Single repo for more streams and platforms
- Stream examples: MariaDB 10.2, MariaDB 10.3, ...
CI testing for each PR (ci.centos.org + internal Jenkins)
How we solve the challenges

Duplication of the code for different platforms and streams

Symlinks approach: e.g. [https://github.com/sclorg/mariadb-container](https://github.com/sclorg/mariadb-container)

- Only README.md, and Dockerfile* are duplicated
- All the differences defined by ENV variables
How we solve the challenges

Deduplication: ENV in Dockerfile and a branch in scripts

```bash
if [ "$MYSQL_VERSION" \> "10.0" ] ; then
    mysql $mysql_flags <<EOSQL
        DROP USER IF EXISTS 'root'@'{}';
        FLUSH PRIVILEGES;
    EOSQL
else
    ...
```

```bash
ENV MYSQL_VERSION=10.2
...
```
How we solve the challenges

Duplication of the code for different platforms and streams

Dist-gen: e.g.
https://github.com/sclorg/postgresql-container

- Requires distgen tool:
  https://github.com/devexp-db/distgen
- Jinja templates
- No IFs in the code
- Every file only once
Productization in scale
Upstream to downstream
Using symlinks to avoid duplication

<table>
<thead>
<tr>
<th>Upstream github</th>
</tr>
</thead>
<tbody>
<tr>
<td>├── 10.2</td>
</tr>
<tr>
<td>│  └── Dockerfile</td>
</tr>
<tr>
<td>│  └── Dockerfile.rhel7</td>
</tr>
<tr>
<td>│  └── Dockerfile.fedora</td>
</tr>
<tr>
<td>│  └── root</td>
</tr>
<tr>
<td>│      └── usr</td>
</tr>
<tr>
<td>│      └── root-common -&gt; ../root</td>
</tr>
<tr>
<td>│      └── s2i-common -&gt; ../s2i</td>
</tr>
<tr>
<td>│  └── test -&gt; ../test</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dist-git</th>
</tr>
</thead>
<tbody>
<tr>
<td>├── Dockerfile</td>
</tr>
<tr>
<td>│  └── root</td>
</tr>
<tr>
<td>│      └── usr</td>
</tr>
<tr>
<td>│      └── root-common</td>
</tr>
<tr>
<td>│      └── s2i-common</td>
</tr>
<tr>
<td>│      └── bin</td>
</tr>
<tr>
<td>│      └── test</td>
</tr>
<tr>
<td>│      └── run.sh</td>
</tr>
</tbody>
</table>
Upstream to downstream

Using Distgen to avoid duplication

<table>
<thead>
<tr>
<th>Upstream github</th>
<th>Dist-git</th>
</tr>
</thead>
<tbody>
<tr>
<td>└── common</td>
<td>├── Dockerfile</td>
</tr>
<tr>
<td></td>
<td>│   └── root</td>
</tr>
<tr>
<td></td>
<td>│       └── usr</td>
</tr>
<tr>
<td></td>
<td>└── root-common</td>
</tr>
<tr>
<td>└── LICENSE</td>
<td>└── s2i-common</td>
</tr>
<tr>
<td>└── Makefile</td>
<td>└── bin</td>
</tr>
<tr>
<td>└── manifest.sh</td>
<td>└── test</td>
</tr>
<tr>
<td>└── specs</td>
<td>└── run.sh</td>
</tr>
<tr>
<td>└── src</td>
<td></td>
</tr>
<tr>
<td>└── test</td>
<td></td>
</tr>
</tbody>
</table>

... 10.2?

10.2
Productization in scale
Keeping dozens of images for each platform fresh and updated

RHEL:
- 38 images in rhel7
- No code changes for CVE rebuilds
- Source code changes when rebuilding for new base image
  - ~every 6 weeks

Fedora:
- ~15 images per Fedora release
- Rebuild every 2 weeks (not currently)

Automated and reliable testing is a must
Productization in scale
Automation within infrastructure

Freshmaker
- No changes to sources
- Refresh content that delivered by RPM

Chain auto-rebuilds in OSBS
- Allows to pick git HEAD, so latest changes in git may be taken

Github to dist-git
- Requires transformation
- Cyborg colleagues (Betka, ...)

No changes to sources
Refresh content that delivered by RPM
CWT

https://github.com/sclorg/container-workflow-tool

Bulk actions

Git content synchronisation
  - Incl. necessary layout transformations

Handling image builds
  - Latest build status
  - Triggering new builds

Future Feature: Bodhi Updates
Questions?

1. Container maintainer:
   - De-duplicated github source code
   - Distgen
   - Generated github source code
   - cwt

2. User:
   - Registry
   - Bodhi
   - Container image
   - OSBS
   - Freshmaker
   - koji

   (Chain auto-rebuilds)