IPAaaS: turning complex application to “a service”

Jan Pazdziora
Sr. Principal Software Engineer
Identity Management Special Projects, Red Hat
jpazdziora@redhat.com

DEVCONF.cz

28th January 2017

Problem space

- Have favourite application setup / solution
  - Currently maybe on physical host or some VM
  - Multiple services / daemons
  - Additional logic in the setup
- What if we want the solution
  - To be more lightweight
  - Easily deployable elsewhere
  - Potentially offered as a service, with instances started on demand

Application and its setup

- Multiple services / daemons
  - E.g. CMS, wiki, time-tracking system, ...
  - Maybe with some database server
- Some cron jobs for sync and backup and monitoring
  - And email setup for sending warnings out
- With sshd
  - Used to yum upgrade the software or tweak the configuration
- Perhaps with some configuration management setup / agent

The goal

- Containerize the application setup
  - Without necessarily breaking it into multiple containers
  - Yes, it is an antipattern
    - But one matching real needs
    - Organizations do not want to unnecessarily spend resources to break their setups to multiple containers, only to stitch them back together
- Make it easy to deploy in some PaaS
  - For example OpenShift

Containerizing complex application

- Install software in build time
- Set whatever can be pre-set in build time
- Identify and isolate configuration and data
  - This can be iterative process
  - Configuration and data need to end up in persisted volumes
- In run time, do initial “firstboot” configuration
  - Based on small set of configuration parameters
  - Remember, there will be multiple services to set up
- In run time, start and run the services, in proper order

Multiple services in container: execution

- Just like outside of container: init/systemd and targets and services

```
host$ docker run -ti -e container=docker fedora:24 /usr/sbin/init
systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA -APPARMOR +S
Detected virtualization docker.
Detected architecture x86-64.
Welcome to Fedora 24 (Twenty Four)!
Set hostname to <06cd3aff5b2e>.
[  OK  ] Reached target Local File Systems.
[...]
[  OK  ] Started Update UTMP about System Runlevel Changes.
```

- The `container=docker` environment variable needed for status output
  - Except for `fedora:25+` where it does not work (bug 1373780)
- The infamous `-v /sys/fs/cgroup:/sys/fs/cgroup:ro` and `--tmpfs` for `/run` and `/tmp` are no longer needed when `oci-systemd-hook` is installed on the host and executed command looks like init

Execution with systemd in container

- The “firstboot” set can be run before exec-ing systemd
- Or it can be a systemd service, especially when the initialization expects systemd to be running

  `app-configure.service`:

```
[Unit]
Before=app1.service app2.service
```

  and

```
COPY app-configure.service /usr/lib/systemd/system/
RUN systemctl enable app-configure
```

- Minimize the targets
  - `RUN systemctl mask dnf-makecache.timer`
  - Or to workaround bug 1309574:

```
RUN systemctl mask systemd-sysusers.service
```

Things to consider

- Passing parameters to services
  - For example via files in `/run`
- Output from services to container output
  - For debugging and progress tracking
  - `/proc/1/fd/1` no longer works (bug 1413099)

Systemd-based setup in container

- `Dockerfile`:

```
FROM fedora
RUN yum install -y httpd mod_ssl ... && yum clean all
ENV container docker
RUN systemctl enable httpd ...
# Tweak configuration to turn distribution defaults
# to your application's defaults
EXPOSE 80 443
ENTRYPOINT [ "/usr/sbin/init" ]
```

- Build and run the container
- The services use and populate various files, those are the application setup's config and data

Identify configuration and data

- Software installed
  - `/usr`, default content of `/var` and `/etc`
- Services enabled
  - `/etc/systemd/system/multi-user.target.wants`
- Configuration of individual services
  - `/etc`
- Data of individual services
  - `/var`, `/etc`, ... and sometimes other locations

- Querying rpm database only helps so much; `docker diff` helps a lot

docker diff systemd-httpd

```
D /var/lib/rpm/__db.002
[...]
C /var/lib/systemd/catalog/database
[...]
A /var/log/journal/7fb94583cd37a375e6c10909033c247b
A /var/log/journal/7fb94583cd37a375e6c10909033c247b/system.journal
[...]
A /var/log/httpd/error_log
A /var/log/httpd/ssl_access_log
[...]
C /var/cache/ldconfig/aux-cache
[...]
A /var/tmp/systemd-private-50192283a3134194bd3381d9416080f4-httpd.service-s4TWT0
[...]
C /etc/group
C /etc/machine-id
C /etc/passwd
C /etc/mtab
[...]
C /etc/ld.so.cache
A /etc/.updated
```

Storage of complex setup

- Instance's state is what needs to be persisted
  - All over the `/var` and `/etc` (but not the whole `/var` and `/etc`)
- We need single volume for the application setup
  - Avoid mixups when only parts of the config or data would get mounted
  - Upgrades when new locations need to be included
- Suggested approach: symlinked layout
  - In build time
    - `/data-template` with the initial content
    - symlinks from the image to `/data`
  - In run time populate / update `/data` from `/data-template`

Build time preparation

- `volume-data-list`:

```
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/conf.modules.d
/etc/systemd/system
/var/log/httpd
[...]
```

```
COPY volume-data-list /etc/
```

- Moving initial config and data aside:

```
RUN mkdir /data-template /data
RUN while read i ; do \
    mkdir -p /data-template$( dirname $i ) ; \
    mv $i /data-template$i ; \
    ln -sv /data$i $i ; done < /etc/volume-data-list
VOLUME [ "/data" ]
```

Runtime initialization

- `init`:

```
#!/bin/bash
( cd /data-template && cp -npr --parents -t /data * )
exec /usr/sbin/init
```

```
COPY init /usr/local/sbin/init
RUN chmod +x /usr/local/sbin/init
ENTRYPOINT [ "/usr/local/sbin/init" ]
```

Storage preparation: initial

```
[...]
/etc/hosts.allow
/etc/hosts.deny
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/conf.modules.d
/etc/httpd/logs -> ../../var/log/httpd
/etc/httpd/modules -> ../../usr/lib64/httpd/modules
/etc/httpd/run -> /run/httpd
/etc/init.d -> rc.d/init.d
/etc/inittab
[...]
/etc/system-release -> fedora-release
/etc/systemd/resolved.conf
/etc/systemd/system
/etc/systemd/system.conf
[...]
/var/log/hawkey.log
/var/log/httpd
/var/log/journal
```

Storage preparation: move to template

```
[...]                                                    /data-template:
/etc/hosts.allow
/etc/hosts.deny

                                                         /etc/httpd/conf
                                                         /etc/httpd/conf.d
                                                         /etc/httpd/conf.modules.d
/etc/httpd/logs -> ...
/etc/httpd/modules -> ...
/etc/httpd/run -> /run/httpd
/etc/init.d -> rc.d/init.d
/etc/inittab
[...]
/etc/system-release -> fedora-release
/etc/systemd/resolved.conf
                                                         /etc/systemd/system
/etc/systemd/system.conf
[...]
/var/log/hawkey.log
                                                         /var/log/httpd
/var/log/journal
```

Storage preparation: point to volume

```
[...]                                                  /data-template has content
/etc/hosts.allow
/etc/hosts.deny
/etc/httpd/conf -> /data/etc/httpd/conf
/etc/httpd/conf.d -> /data/etc/httpd/conf.d
/etc/httpd/conf.modules.d -> /data/etc/httpd/conf.modules.d
/etc/httpd/logs -> ...
/etc/httpd/modules -> ...
/etc/httpd/run -> /run/httpd
/etc/init.d -> rc.d/init.d
/etc/inittab
[...]
/etc/system-release -> fedora-release
/etc/systemd/resolved.conf
/etc/systemd/system -> /data/etc/systemd/system
/etc/systemd/system.conf
[...]
/var/log/hawkey.log
/var/log/httpd -> /data/var/log/httpd
/var/log/journal
```

What is special about services

- Single setup interaction, during initialization
  - Careful consideration needed about supported parameters and deployment options
- Status needs to live in volume(s)
  - Typical container is easy to tweak (`docker exec` post setup)
    - Plain container can be the bearer of application's identity
  - In PaaS, containers get removed and recreated
- Hardened environments
  - It might not be allowed to run systemd as root in PaaS
  - UID-namespaces could help
    - But see bugs 1401537, 1401944, 1402264; and 1406684

What is special about services (cont'd)

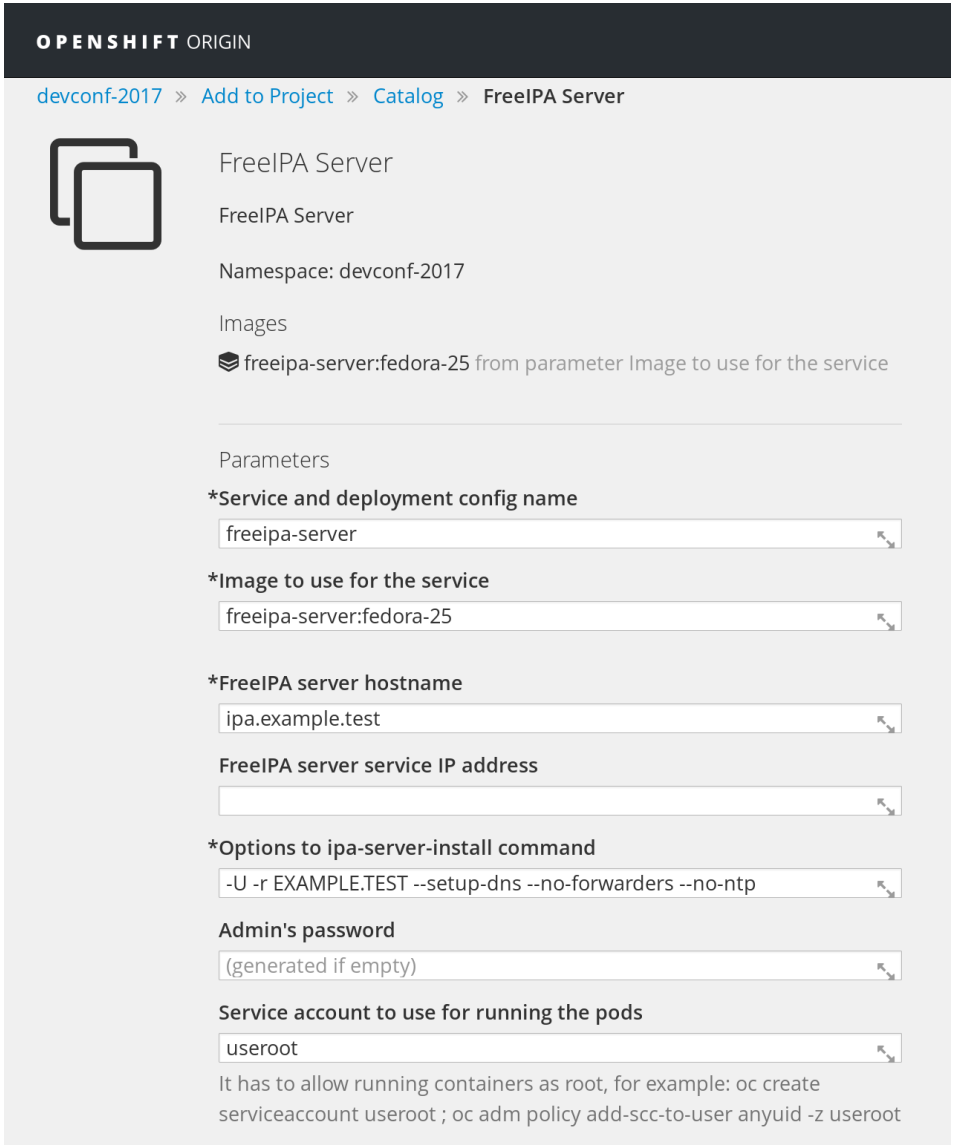
- Hostname handling might be limited
  - It helps if the application can work with identity `!= $( uname -n )`
- Exposure of non-HTTP(S) traffic
  - The PaaS environment might primarily target Web applications
- Application-level replication
  - The PaaS environment might scale containers ... but what if they are stateful?

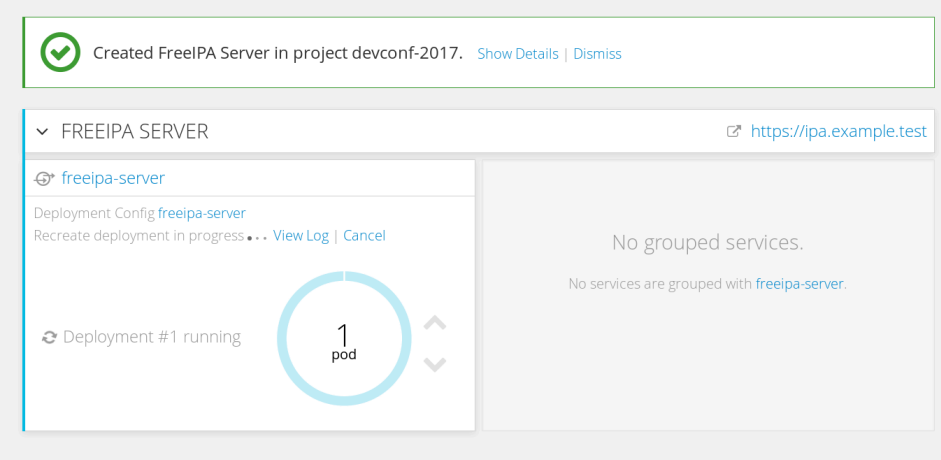
A complex application: FreeIPA server

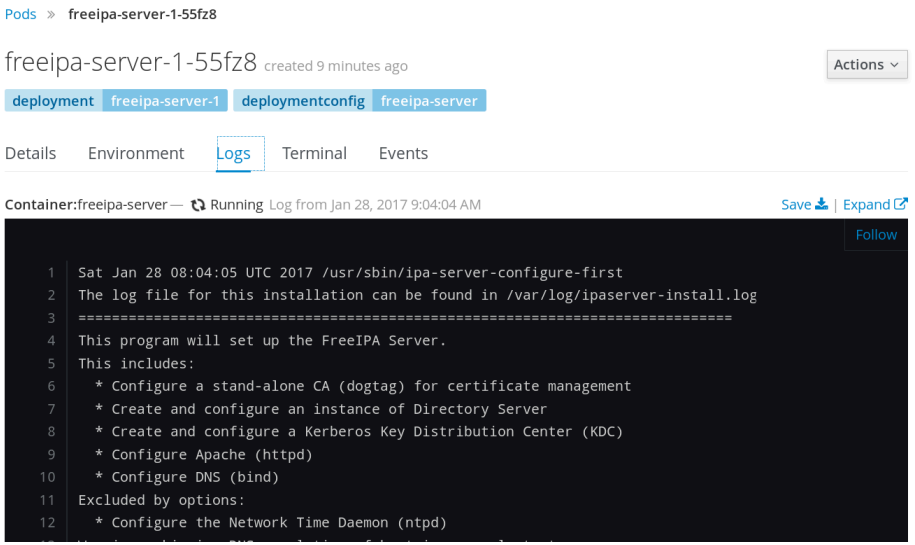
- 389 Directory Server, KDC, CA, HTTP server, DNS server, OTP server, ...
- Many services on many ports
- It can serve as ultimate source of identities, including host identities
  - It can run its own DNS server instance
- Built-in replication
  - Start with one master, set up replicas from it

- github.com/freeipa/freeipa-container
- Containerized (single-container) setup available upstream for a while
  - Available as technology preview with RHEL Atomic 7.3.1

FreeIPA server as a service







FreeIPA in OpenShift

- github.com/freeipa/freeipa-container/blob/master/*.json
- UID-namespacing not yet in OpenShift, coming in 1.6 later this year.

```
# oc login -u system:admin
[...]
Using project "devconf-2017".
# oc create serviceaccount useroot
serviceaccount "useroot" created
# oc adm policy add-scc-to-user anyuid -z useroot
```

- Create `PersistentVolume` — `hostPath` PV example in `freeipa-server-openshift-volume.json`
- New application — via WebUI if the JSON template was imported, or

```
# oc new-app --name freeipa-x -f ./freeipa-server-openshift.json \
    -p IPA_SERVER_SERVICE=freeipa-1 -p IPA_ADMIN_PASSWORD=Secret123 \
    -p IPA_SERVER_HOSTNAME=freeipa.example.test \
    -p IPA_SERVER_IP=172.30.248.14 \
    -p IPA_SERVER_IMAGE=freeipa-server:centos-7
```

Notes about FreeIPA in OpenShift

- Single volume

```
          "volumes": [
            {
              "name": "${IPA_SERVER_SERVICE}-data",
              "persistentVolumeClaim": {
                "claimName": "${IPA_SERVER_SERVICE}"
              }
            },
```

```
    "volumeMounts": [
        {
          "name": "${IPA_SERVER_SERVICE}-data",
          "mountPath": "/data"
        },
```

- Not available in Online yet due to the need of `anyuid` for systemd
- `clusterIP` for `Service`, to expose the non-Web traffic
- Master only for now — neither `PetSet` nor `StatefulSet` seem to start the second pod

Complex applications as service

- Ultimately, the container should run with read-only filesystem
  - Some software is not happy with symlinks
- SELinux separation is lost — the same type for all processes in the container
- Use of environment variable as initial setup parameters
- Native hostname is not what the application wants to use

Conclusion

- Server side works
  - Need to flesh out replication
  - Can be used as an example for other complex setups that are not practical to be broken into smaller containers
- Client side — SSSD containers exist ...
  - Consuming them in other services might be tricky

References

- github.com/freeipa/freeipa-container
  - `*.json` files
- hub.docker.com/r/freeipa/freeipa-server/
- adelton.com/docs/containers/complex-application-in-container