Portable Services are Ready to Use

LinuxPiter 2018, St. Petersburg, Russia

November 2018

Portable Services?

Portable Services?

System services + some container features

Portable Services?

System services + some container features

or

Portable Services?

System services + some container features

or

Containers + with some system service features

Containers?

Containers?

Resource Bundles + Isolation + Delivery

Portable Services:

Portable Services:

Resource Bundles + Integration + Sandboxing

Portable Services:

Resource Bundles + Integration + Sandboxing

Modular

Consider Range: *Integrated* $\rightarrow$ *Isolated:*

Consider Range: *Integrated* → *Isolated:*

Classic System Services → *Portable System Services* → Docker-style micro services → Full OS containers á la LXC → VMs á la KVM

Consider Range: *Integrated* → *Isolated:*

Classic System Services → *Portable System Services* → Docker-style micro services → Full OS containers á la LXC → VMs á la KVM

Consider what's shared, not shared: Networking, File System, PID Namespace, Init System, Device Access, Logging

Goal: Leave No Artifacts!

Goal: Leave No Artifacts! (Bind lifecycles!)

Goal: Leave No Artifacts! (Bind lifecycles!)

Goal: Everything in one place!

Goal: Leave No Artifacts! (Bind lifecycles!)

Goal: Everything in one place!

Goal: "Feel" like a native service — because it is one! (Specifically "systemctl" should work for it, like for any native service)

Why?

Why?

Next step for service management

Why?

Next step for service management

Everything already has a systemd service file

Why?

Next step for service management

Everything already has a systemd service file

Admins are used to services already, let's just make them more powerful

Why?

Next step for service management

Everything already has a systemd service file

Admins are used to services already, let's just make them more powerful

"Superprivileged Containers"

Why?

Next step for service management

Everything already has a systemd service file

Admins are used to services already, let's just make them more powerful

"Superprivileged Containers"

Integration is good, not bad (frequently at least)

Three supported service formats: SysV, Native, Portable

Three supported service formats: SysV, Native, Portable

(The building blocks + generators permit more)

Disk Images

Disk Images

Let's avoid defining something new

Disk Images

Let's avoid defining something new (instead: simple directory tree/subvolume, or GPT containing squashfs)

Disk Images

Let's avoid defining something new (instead: simple directory tree/subvolume, or GPT containing squashfs)

Services run directly from it (think: `RootImage=`, similar to `RootDirectory=`)

Disk Images

Let's avoid defining something new (instead: simple directory tree/subvolume, or GPT containing squashfs)

Services run directly from it (think: `RootImage=`, similar to `RootDirectory=`)

Let's fix `chroot()`!

Disk Images

Let's avoid defining something new (instead: simple directory tree/subvolume, or GPT containing squashfs)

Services run directly from it (think: `RootImage=`, similar to `RootDirectory=`)

Let's fix `chroot()`!

(`RootImage=` with Crypto and Verity!)

Disk Images

Let's avoid defining something new (instead: simple directory tree/subvolume, or GPT containing squashfs)

Services run directly from it (think: `RootImage=`, similar to `RootDirectory=`)

Let's fix `chroot()`!

(`RootImage=` with Crypto and Verity!)

Image just needs to carry systemd unit files, and `/usr/lib/os-release`, then it qualifies as portable service image

Sandboxing: `PrivateDevices=`, `PrivateNetwork=`, `DynamicUser=`, `RemoveIPC=`, `PrivateTmp=`, `PrivateUsers=`, `ProtectSystem=`, `ProtectHome=`, `SystemCallFilter=`, `SystemCallArchitectures=` `RestrictAddressFamilies=`, `RuntimeDirectory=`, `StateDirectory=`, `CacheDirectory=`, `LogsDirectory=`, `ConfigurationDirectory=`, `RestrictRealtime=`, `ProtectKernelModules=`, `ProtectKernelTunables=`, `ProtectControlGroups=`, `RestrictNamespaces=`, `PrivateMounts=`, `MemoryDenyWriteExecute=`, `LockPersonality=`, `CapabilityBoundingSet=`, `NoNewPrivileges=`, ...

Sandboxing: `PrivateDevices=`, `PrivateNetwork=`, `DynamicUser=`, `RemoveIPC=`, `PrivateTmp=`, `PrivateUsers=`, `ProtectSystem=`, `ProtectHome=`, `SystemCallFilter=`, `SystemCallArchitectures=` `RestrictAddressFamilies=`, `RuntimeDirectory=`, `StateDirectory=`, `CacheDirectory=`, `LogsDirectory=`, `ConfigurationDirectory=`, `RestrictRealtime=`, `ProtectKernelModules=`, `ProtectKernelTunables=`, `ProtectControlGroups=`, `RestrictNamespaces=`, `PrivateMounts=`, `MemoryDenyWriteExecute=`, `LockPersonality=`, `CapabilityBoundingSet=`, `NoNewPrivileges=`, ...

More to come: `ProtectKernelLogs=`, `ProtectClock=`, `ProtectTracing=`, `ProtectKeyRing=`, ...

Sandboxing: `PrivateDevices=`, `PrivateNetwork=`, `DynamicUser=`, `RemoveIPC=`, `PrivateTmp=`, `PrivateUsers=`, `ProtectSystem=`, `ProtectHome=`, `SystemCallFilter=`, `SystemCallArchitectures=` `RestrictAddressFamilies=`, `RuntimeDirectory=`, `StateDirectory=`, `CacheDirectory=`, `LogsDirectory=`, `ConfigurationDirectory=`, `RestrictRealtime=`, `ProtectKernelModules=`, `ProtectKernelTunables=`, `ProtectControlGroups=`, `RestrictNamespaces=`, `PrivateMounts=`, `MemoryDenyWriteExecute=`, `LockPersonality=`, `CapabilityBoundingSet=`, `NoNewPrivileges=`, ...

More to come: `ProtectKernelLogs=`, `ProtectClock=`, `ProtectTracing=`, `ProtectKeyRing=`, ...

Per-Service Firewalling and Accounting

Sandboxing: `PrivateDevices=`, `PrivateNetwork=`, `DynamicUser=`, `RemoveIPC=`, `PrivateTmp=`, `PrivateUsers=`, `ProtectSystem=`, `ProtectHome=`, `SystemCallFilter=`, `SystemCallArchitectures=` `RestrictAddressFamilies=`, `RuntimeDirectory=`, `StateDirectory=`, `CacheDirectory=`, `LogsDirectory=`, `ConfigurationDirectory=`, `RestrictRealtime=`, `ProtectKernelModules=`, `ProtectKernelTunables=`, `ProtectControlGroups=`, `RestrictNamespaces=`, `PrivateMounts=`, `MemoryDenyWriteExecute=`, `LockPersonality=`, `CapabilityBoundingSet=`, `NoNewPrivileges=`, ...

More to come: `ProtectKernelLogs=`, `ProtectClock=`, `ProtectTracing=`, `ProtectKeyRing=`, ...

Per-Service Firewalling and Accounting

For portable services (unlike for native and SysV): Sandboxing is opt-out, not opt-in!

Hard problems:

Hard problems:

Dynamic Users

Hard problems:

Dynamic Users

User Database mismatch

Hard problems:

Dynamic Users

User Database mismatch

D-Bus, ...

In scope: Simple delivery, Verification, Simple building, Versioning, Socket activation, ...

In scope: Simple delivery, Verification, Simple building, Versioning, Socket activation, ...

Out of Scope: Load distribution/migration á la fleetd, Cluster deployment, claim we'd define a universal API, server side functionality, desktop stuff

Mode of operation:

Mode of operation:

```
portablectl attach foobar.raw
```

Mode of operation:

```
portablectl attach foobar.raw
```

```
portablectl detach foobar.raw
```

No new metadata!

No new metadata!

Socket, Target, Path, Timer units, too!

No new metadata!

Socket, Target, Path, Timer units, too!

Unit files matched by image name prefix: `foobar-4711.raw` means `foobar*.service|socket|path|timertimer`

No new metadata!

Socket, Target, Path, Timer units, too!

Unit files matched by image name prefix: `foobar-4711.raw` means `foobar*.service|socket|path|timertimer`

Triple use images

Profiles:

Profiles:

`default`, `strict`, `trusted`, `nonetwork`

Build tool: `mkosi`, ...

http://0pointer.net/blog/walkthrough-for-portable-services.html

That's all, folks!