MULTI-IMAGE SINGLE CONTAINER

AIDEN MCCLELLAND
CTO, START9

What is StartOS?

- A Linux distribution designed to enable non-technical users to self-host open source software
- MIT licensed
- Rust Monolith, with Angular/Typescript user interface (webui)
- Sits on top of Debian, but doesn’t use Debian packages

One Container = One Service/App

Reduces complexity

One VLAN IP address

Fewer virtual interfaces

One set of resource limits

One in-container service manager

Why multiple containers?

1. Use pre-packaged docker images ie. Nginx, Postgresql, Redis, etc.
2. Don’t worry about distro compatibility
3. Isolate application sub-components from each other
4. Create resource limits on individual application subcomponents
5. ???

Why multiple containers?

1. Use pre-packaged docker images ie. Nginx, Postgresql, Redis, etc.
2. Don’t worry about distro compatibility
3. Isolate application sub-components from each other
4. Create resource limits on individual application subcomponents

Why LXC?

Easily manipulate container rootfs from host at runtime

— Requires rshared mount propagation

Perform chroot and mount --bind inside unprivileged container

Creating a Service

Single rootfs image, mounted with overlayfs
Runs Alpine Linux with NodeJS

Custom Javascript service manager:

- Loads package maintainer scripts
- Serves JSONRPC API over unix domain socket
- Connects to host (StartOS Daemon), also over JSONRPC unix domain socket
- Launches binaries in chroots

Our Strategy

StartOS Daemon Host API

Connection

- Attach overlayed package images to container rootfs
- mount -t overlay not possible in unprivileged LXC container

Dependency Integration

- Interact with other services on host

Data

- Export information to end user

Hassle free networking

- Tor
- SSL
  - signed by host root CA,
- Bind to host port for LAN access
- Listen on clearnet
  - dynamic DNS
  - automatic letsencrypt ssl certs
  - share ports with SNI based SSL proxy

Our Strategy

Service API

Administration

- init
- start
- stop
- exit

Respond to user-initiated actions

- edit config
- install / update / uninstall hooks
- perform backup
- etc...

Launching Binaries

Package maintainer script defines:

— What binaries to launch

— Which image to launch each binary in

— Where to mount persistence volumes

— Environment variables & arguments

For each command, in-container service manager:

— Calls host api to mount overlayed image to container

— Bind mounts /proc, /sys, /dev, and /run inside the overlayed image

— Bind mounts persistence volumes at requested paths (provided by host at /media/startos/volumes)

— Runs chroot <overlay path> <command> <args>

An Example

Package Maintainer Script

An Example

```typescript
import { sdk } from '../sdk'
import { ExpectedExports } from '@start9labs/start-sdk/lib/types'
import { HealthReceipt } from '@start9labs/start-sdk/lib/health/HealthReceipt'
import { Daemons } from '@start9labs/start-sdk/lib/mainFn/Daemons'
import { uiPort } from './interfaces'

export const main: ExpectedExports.main = sdk.setupMain(
  async ({ effects, utils, started }) => {
    /**
     * ======================== Setup ========================
     *
     * In this section, you will fetch any resources or run any commands necessary to run the service
     */
    console.info('Starting Hello World!')

    /**
     * ======================== Additional Health Checks (optional) ========================
     *
     * In this section, you will define *additional* health checks beyond those associated with daemons
     */
    const healthReceipts: HealthReceipt[] = []

    /**
     * ======================== Daemons ========================
     *
     * In this section, you will create one or more daemons that define the service runtime
     *
     * Each daemon defines its own health check, which can optionally be exposed to the user
     */
    return Daemons.of({
      effects,
      started,
      healthReceipts, // Provide the healthReceipts or [] to prove they were at least considered
    })
      .addDaemon('db', {
        imageId: 'postgres',
        command: 'docker-entrypoint.sh', // The command to start the daemon
        ready: {
          // If display is null, it will not be displayed to the user in the UI
          display: 'Database',
          // The function to run to determine the health status of the daemon
          fn: () =>
            sdk.healthCheck.checkPortListening(effects, 5432, {
              successMessage: 'The database is ready',
              errorMessage: 'The database is not ready',
            }),
        },
        requires: [],
      })
      .addDaemon('webui', {
        imageId: 'main',
        command: 'hello-world', // The command to start the daemon
        ready: {
          // If display is null, it will not be displayed to the user in the UI
          display: 'Web Interface',
          // The function to run to determine the health status of the daemon
          fn: () =>
            sdk.healthCheck.checkPortListening(effects, uiPort, {
              successMessage: 'The web interface is ready',
              errorMessage: 'The web interface is not ready',
            }),
        },
        requires: ['db'],
      })
  },
)
```

An Example

Demo

An Example

Q&A

Relevant Links

Company	https://start9.com
Me	https://github.com/dr-bonez
StartOS	https://github.com/Start9Labs/start-os/tree/feature/lxc-container-runtime

THANK YOU

@DRBONEZ:MATRIX.START9LABS.COM