

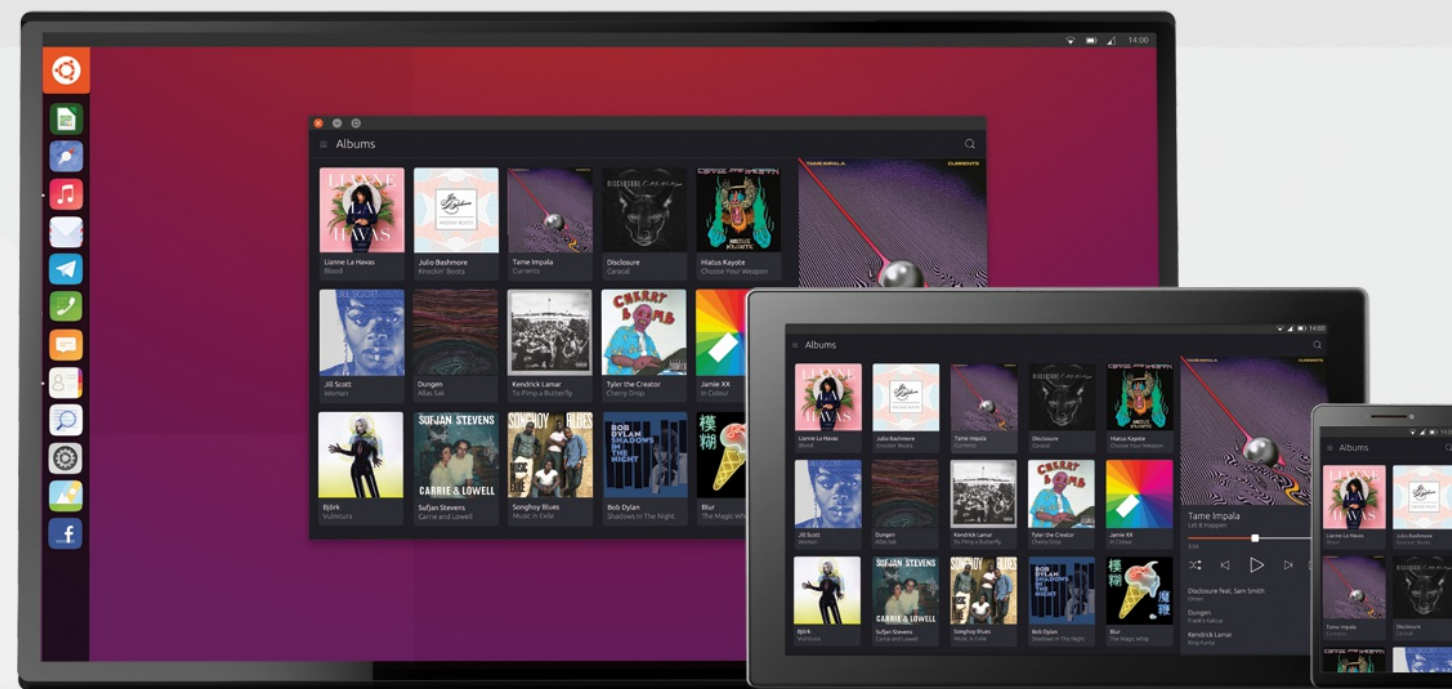
Running Classic Applications in a Confined Ecosystem

Larry Price

Classic Applications?

Confined Ecosystem?

Convergence



Classic Applications

Definition: *An application installed via a traditional package manager or build tool in an unconfined environment.*

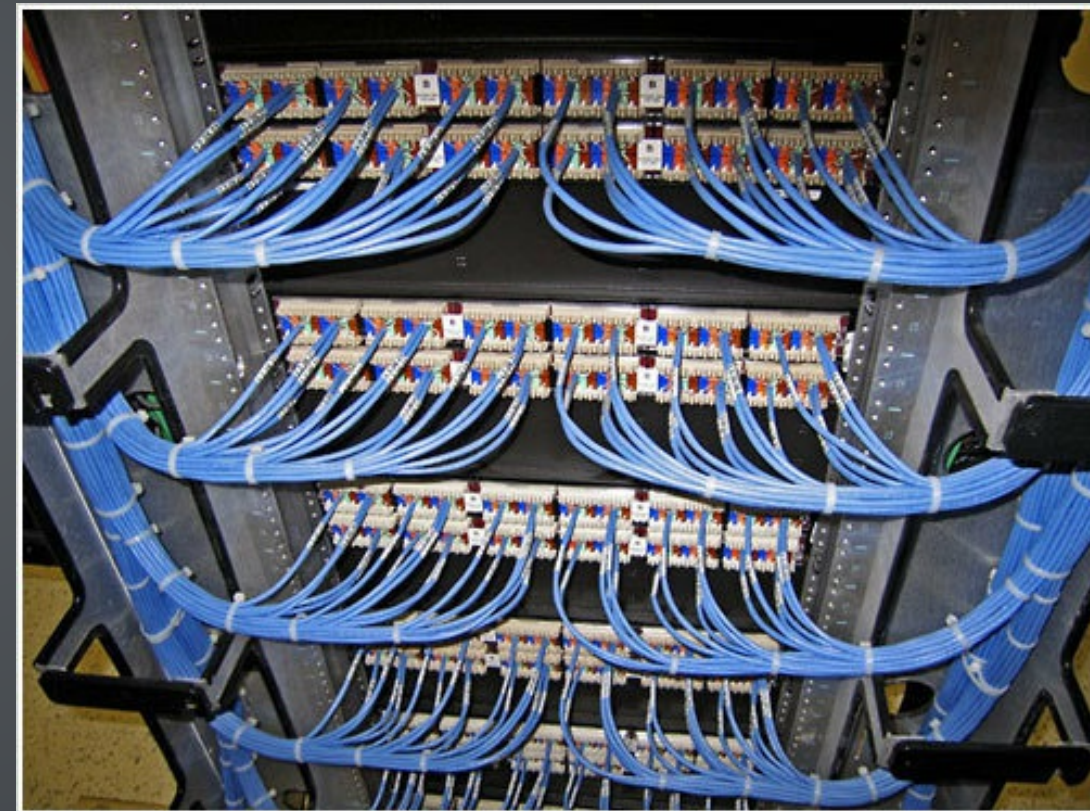
- LibreOffice
- gimp
- HexChat
- Firefox
- Oad
- pingus
- steam
- ...

Unconfined



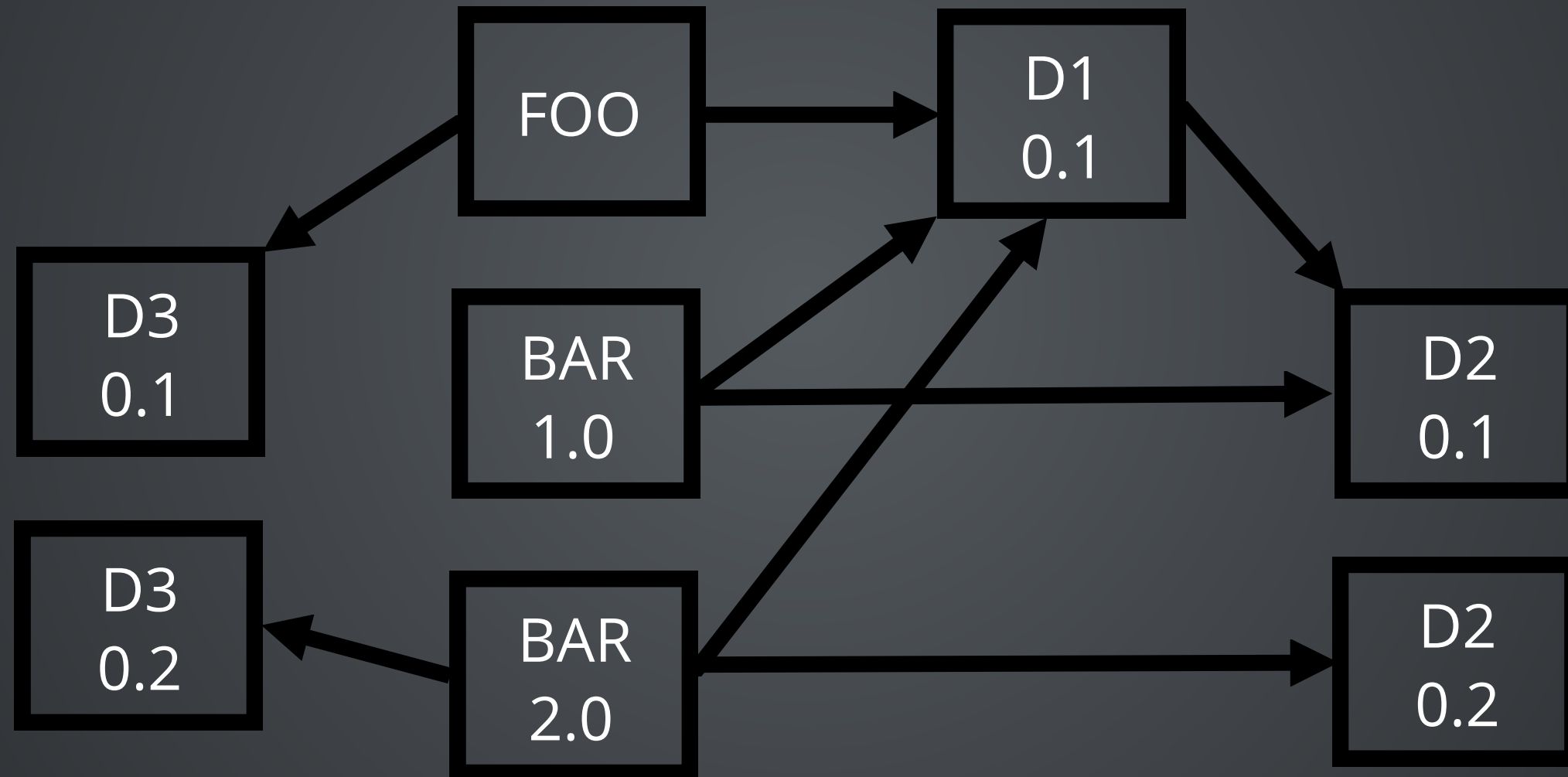
Courtesy <http://www.darkroastedblend.com/2009/05/cable-blues-tangled-crazy-wiring-part-6.html>

Confined

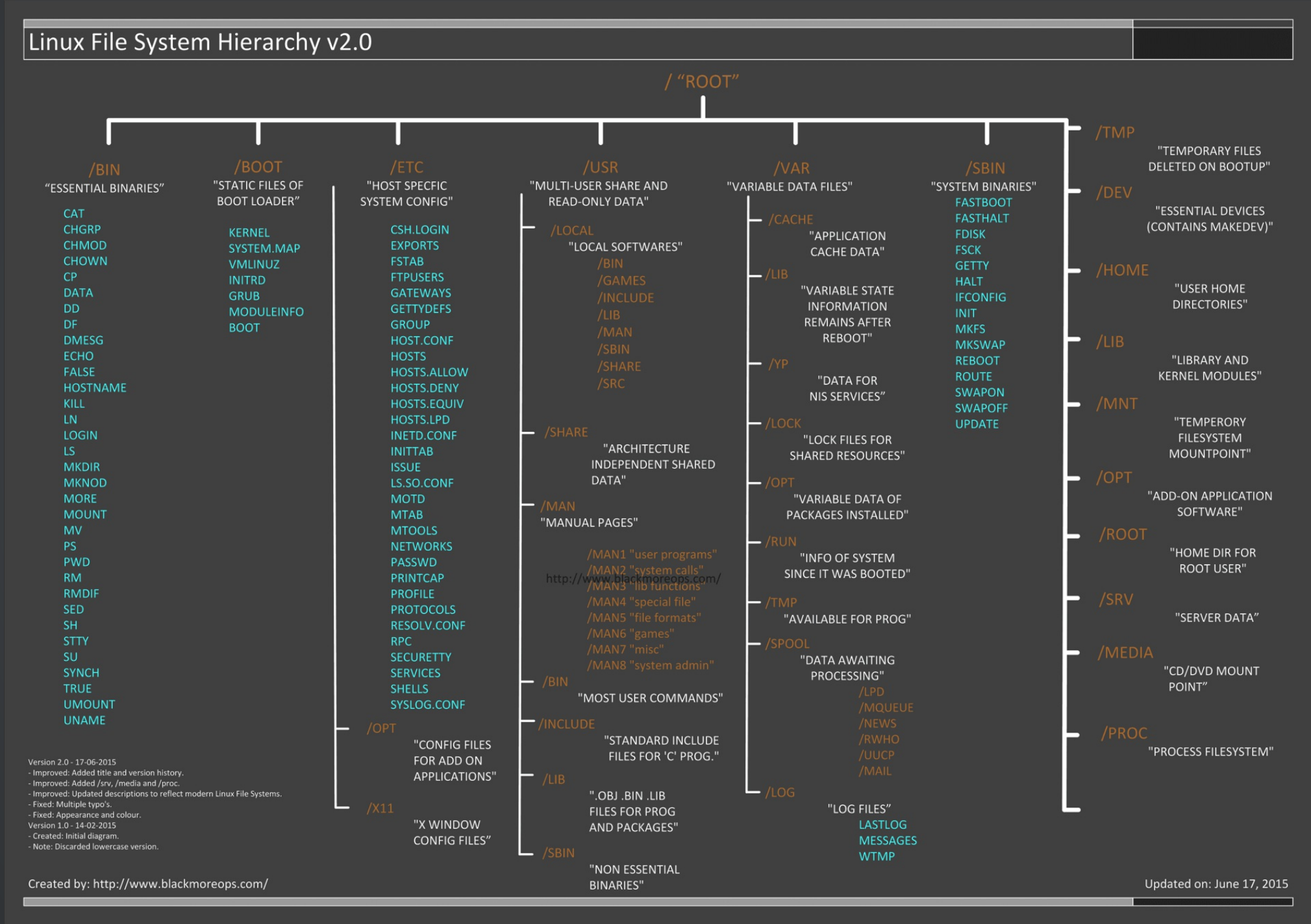


Courtesy <http://royal.pingdom.com/2008/01/24/when-data-center-cabling-becomes-art/>

Unconfined Dependency Management



Unconfined Filesystem Pollution



Unconfined Filesystem Pollution

echo \$PATH

- /bin
- /sbin
- /usr/bin
- /usr/sbin
- /usr/games
- /usr/local/bin
- /usr/local/sbin
- /usr/local/games
- /home/\$USER/.rvm/bin
- /home/\$USER/.rvm/bin
- /home/\$USER/.rvm/bin
- /home/\$USER/.rvm/gems/ruby-2.2.0/bin
- /home/\$USER/.rvm/gems/ruby-2.2.0@global/bin
- /home/\$USER/.rvm/rubies/ruby-2.2.0/bin
- /usr/local/heroku/bin
- /home/\$USER/.nvm/versions/node/v6.5.0/bin
- /home/\$USER/.local/bin

Unconfined Filesystem Pollution

ldconfig -v

- /lib
- /lib/x86_64-linux-gnu
- /lib/i386-linux-gnu
- /usr/lib
- /usr/lib/x86_64-linux-gnu
- /usr/lib/i386-linux-gnu
- /usr/local/lib
- /usr/lib/x86_64-linux-gnu/fakechroot
- /usr/lib/x86_64-linux-gnu/libfakeroot
- /usr/lib/x86_64-linux-gnu/mesa-egl
- /usr/lib/x86_64-linux-gnu/mesa

Confinement Goals

- Prevent unwanted app communication
- Controlled filesystem manipulation
- Individualized dependency management

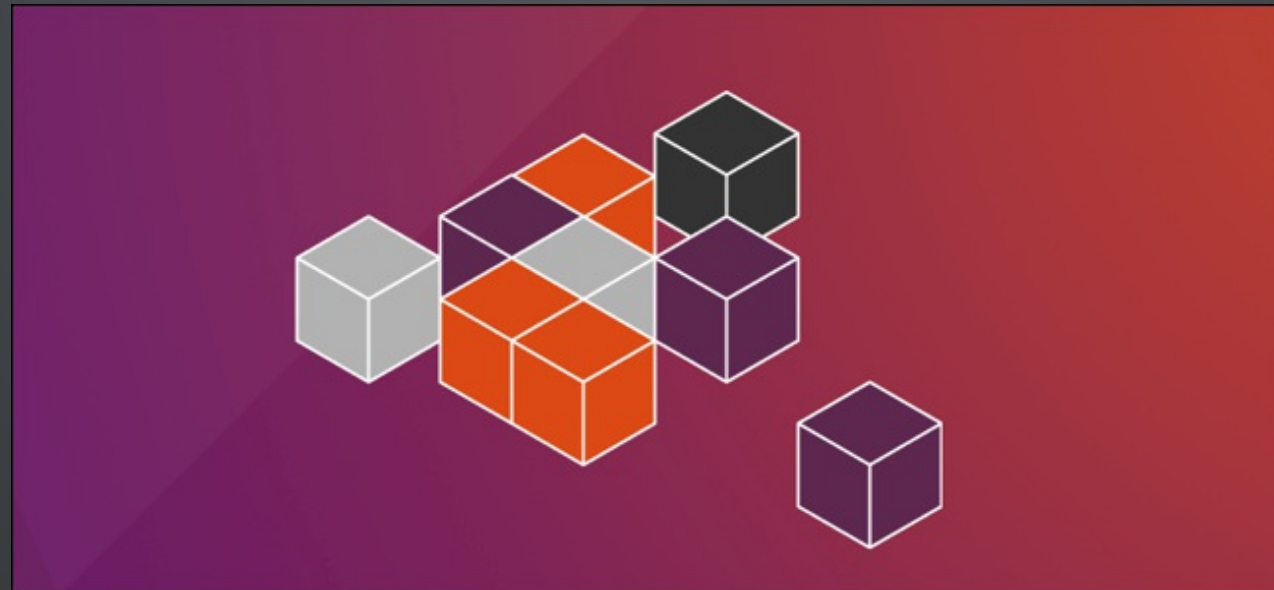
Clicks

- Easily create packages for Ubuntu Touch
- Bundle dependencies
- Utilize Ubuntu "Frameworks" for common packages
- "Easier" than packaging as a deb; automatic through Ubuntu SDK
- Confine apps with apparmor



Snap

- Read-only SquashFS filesystem
- Self-contained application with most dependencies bundled inside the snap
- Confined from the OS and other apps through apparmor
- Exchange content and functionality through interface definitions



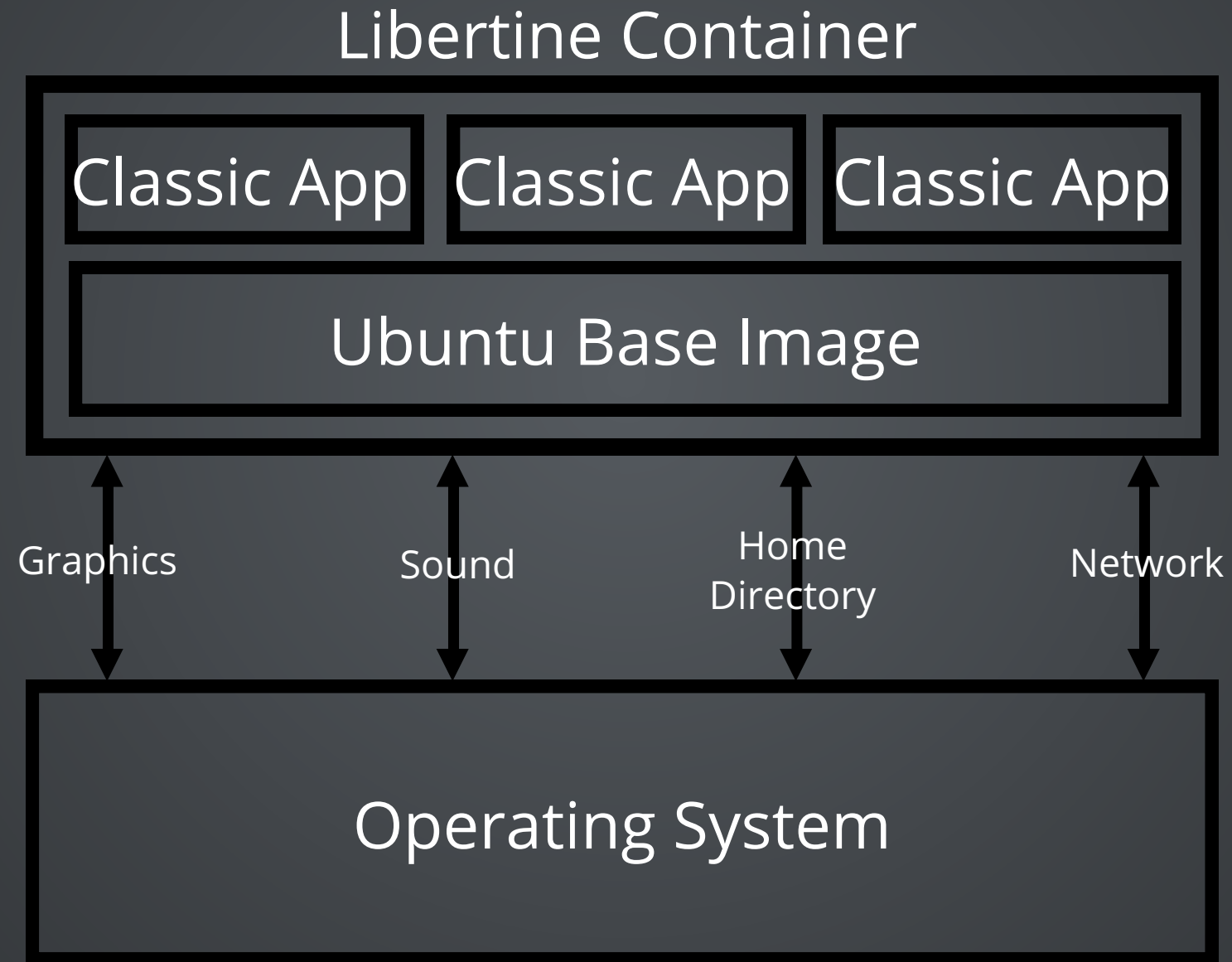
Ubuntu Personal

- Desktop Ubuntu built entirely from snaps
- Secure environment for mobile devices
- Needs to run classic applications

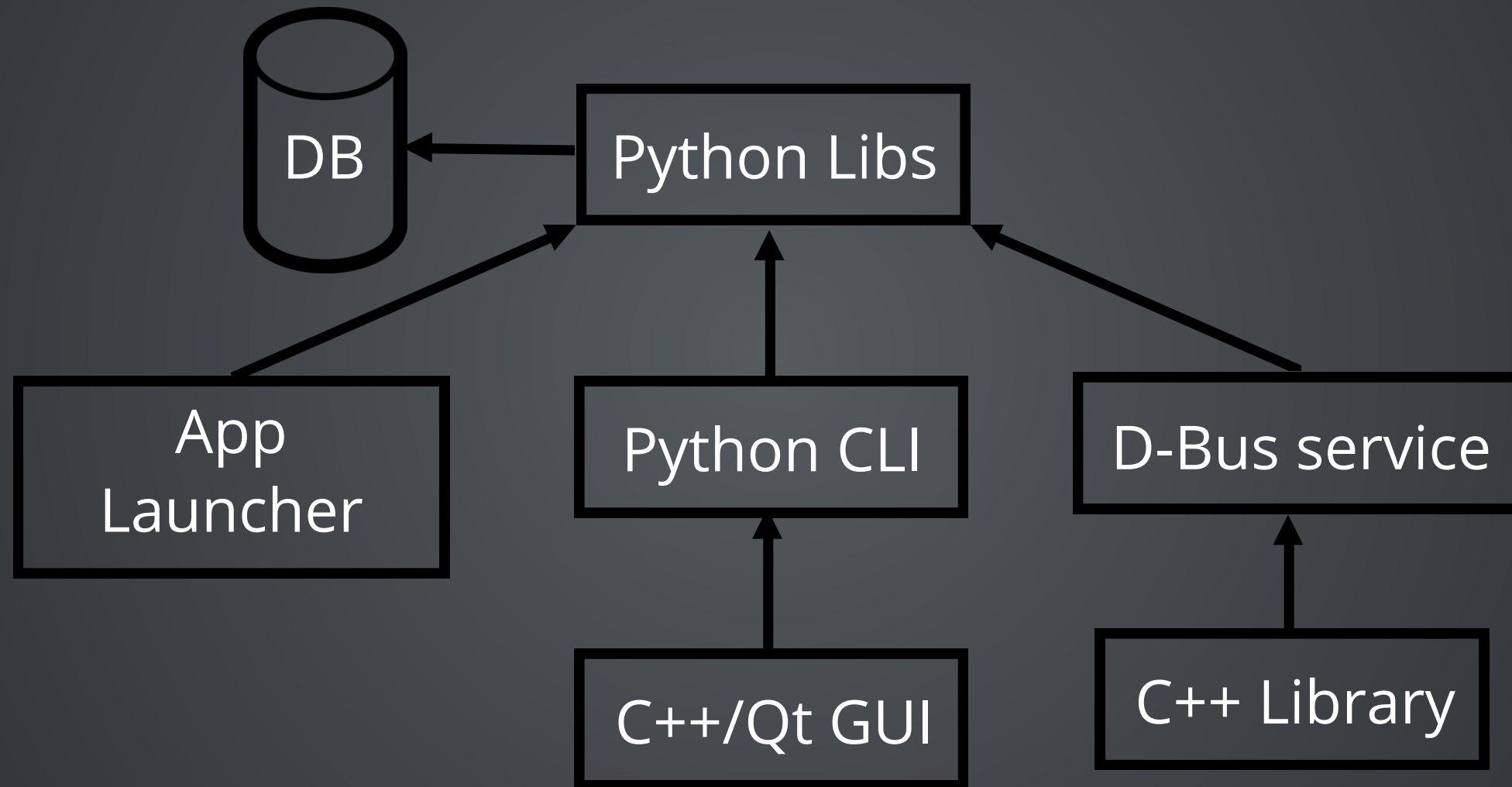
lib•er•tine (lĭb'ər-tēn,)

- n. One who acts without moral restraint; a dissolute person.
- n. One who defies established religious precepts; a freethinker.
- adj. Morally unrestrained; dissolute.

High-Level Libertine



Libertine Breakdown



Libertine Database

```
1 {
2   "_warning": "This file is automatically generated by Libertine and should not be manually edited.",
3   "containerList": [
4     {
5       "bindMounts": [
6         "/media/lrp"
7       ],
8       "distro": "zesty",
9       "extraArchives": [
10        "ppa:nilarimogard/webupd8"
11      ],
12      "id": "zesty",
13      "installStatus": "ready",
14      "installedApps": [
15        {
16          "appStatus": "installed",
17          "packageName": "supertuxkart"
18        },
19        {
20          "appStatus": "installed",
21          "packageName": "steam"
22        },
23        {
24          "appStatus": "installed",
25          "packageName": "atom"
26        },
27        {
28          "appStatus": "installed",
29          "packageName": "leafpad"
30        }
31      ],
32      "multiarch": "enabled",
33      "name": "User Applications",
34      "runningApps": [],
35      "type": "lxd"
36    }
37  ],
38  "defaultContainer": "zesty"
39 }
```




Python Library

- Database r/w access
- Container implementations
- Utility functions
- Container lifecycle management
- Application launchers
- D-bus backend

Supported Containers

- chroot
- lxc
- lxd

chroot

We create chroots using fakeroot to build a fakechroot with debootstrap

```
def create_libertine_container(self, password=None, multiarch=False):
    # Create the actual chroot
    command_line = "{} fakeroot debootstrap --verbose --variant=fakechroot {} {}".format(
        self._build_fakechroot_command(), self.installed_release, self.root_path)
    args = shlex.split(command_line)
    cmd = subprocess.Popen(args)
    cmd.wait()

    # ...
```

chroot

We execute commands with the chroot using proot, adding bind-mounts to user-accessible directories

```
def _build_proot_command(self):
    proot_cmd = shutil.which('proot')
    if not proot_cmd:
        raise RuntimeError('executable proot not found')

    proot_cmd += ' -R {} -b /usr/lib/locale -b /var/lib/extrausers {}'.format(self.root_path, self._get_bind_mounts())

    user_dconf_path = os.path.join(home_path, '.config', 'dconf')
    if os.path.exists(user_dconf_path):
        proot_cmd += " -b %s" % user_dconf_path
```


lxc

The lxc backend will create a rootfs in the user's local cache directory and bind-mount the container's home directory to the user's local share.

Mapping the user allows us to run all commands from within the container as the user, helping solve permissions issues.

```
def create_libertine_container(self, password=None, multiarch=False):
    # ...

    self.container.load_config(config_file)

    utils.create_libertine_user_data_dir(self.container_id)

    # Create container
    with EnvLxcSettings():
        lxc_logfile = get_logfile(self.container)
        if not self.container.create("download", 0,
                                     {"dist": "ubuntu",
                                      "release": self.installed_release,
                                      "arch": self.architecture}):
            utils.get_logger().error("Failed to create container")
            _dump_lxc_log(lxc_logfile)
            return False

    # ...

    # Create host user in container
    self.run_in_container("userdel -r ubuntu")
    self.run_in_container("useradd -u {} -p {} -G sudo {}".format(
        str(user_id), crypt.crypt(password), str(username)))

    if multiarch and self.architecture == 'amd64':
        utils.get_logger().info("Adding i386 multiarch support...")
        self.run_in_container("dpkg --add-architecture i386")

    utils.get_logger().info("Updating the contents of the container after creation...")
    self.update_packages()

    # ...
```

lxc

We use the python-lxc API to manage configuration for the container and run commands.

```
def run_in_container(self, command_string):  
    cmd_args = shlex.split(command_string)  
    return self.container.attach_wait(lxc.attach_run_command, cmd_args)
```

lxd

- New interface written on top of lxc
- Clients connect through REST API
- Unprivileged by default
- Entirely image-based
- Enables connecting over network

lxd

Adding a temporary bind-mount for /run/user/\$USER_ID

```
run_user = os.environ['XDG_RUNTIME_DIR']
container.devices[run_user] = {'source': run_user, 'path': '/var/tmp{}'.format(run_user), 'type': 'disk'}
```

Building a custom binary to fix user mounts

```
def _setup_bind_mount_service(container, uid, username):
    utils.get_logger().info("Creating mount update shell script")
    script = '''
#!/bin/sh

mkdir -p /run/user/{uid}
chown {username}:{username} /run/user/{uid}
chmod 755 /run/user/{uid}
mount -o bind /var/tmp/run/user/{uid} /run/user/{uid}

chgrp audio /dev/snd/*
chgrp video /dev/dri/*
[ -n /dev/video0 ] && chgrp video /dev/video0
'''[1:-1]
    container.files.put('/usr/bin/libertine-lxd-mount-update', script.format(uid=uid, username=username).encode('utf-8'))
    container.execute(shlex.split('chmod 755 /usr/bin/libertine-lxd-mount-update'))
```


lxd

Using python-pylxd, create or update the libertine profile with available host devices

```
def update_libertine_profile(client):
    try:
        profile = client.profiles.get('libertine')

        utils.get_logger().info('Updating existing lxd profile.')
        profile.devices = _get_devices_map()
        profile.config['raw.idmap'] = 'both 1000 1000'

        _lxd_save(profile, 'Saving libertine lxd profile raised:')
    except pylxd.exceptions.LXDAPIException:
        utils.get_logger().info('Creating libertine lxd profile.')
        client.profiles.create('libertine',
                               config={'raw.idmap': 'both 1000 1000'},
                               devices=_get_devices_map())
```

```
def _get_devices_map():
    devices = {
        '/dev/tty0': {'path': '/dev/tty0', 'type': 'unix-char'},
        '/dev/tty7': {'path': '/dev/tty7', 'type': 'unix-char'},
        '/dev/tty8': {'path': '/dev/tty8', 'type': 'unix-char'},
        '/dev/fb0': {'path': '/dev/fb0', 'type': 'unix-char'},
        'x11-socket': {'source': '/tmp/.X11-unix', 'path':
                       '/tmp/.X11-unix', 'type': 'disk', 'optional': 'true'},
    }

    if os.path.exists('/dev/video0'):
        devices['/dev/video0'] = {'path': '/dev/video0', 'type':
                                   'unix-char'}

    # every regular file in /dev/snd
    for f in ['/dev/snd/{}'.format(f) for f in os.listdir('/dev/snd')]:
        if not os.path.isdir('/dev/snd/{}'.format(f)):
            devices[f] = {'path': f, 'type': 'unix-char'}

    # every regular file in /dev/dri
    for f in ['/dev/dri/{}'.format(f) for f in os.listdir('/dev/dri')]:
        if not os.path.isdir('/dev/dri/{}'.format(f)):
            devices[f] = {'path': f, 'type': 'unix-char'}

    # Some devices require special mappings for snap
    if utils.is_snap_environment():
        devices['x11-socket']['source'] =
            '/var/lib/snapd/hostfs/tmp/.X11-unix'

    return devices
```

lxd

Using psutil.Popen to launch an application by calling the lxc command line tool directly

```
def start_application(self, app_exec_line, environ):
    # ...

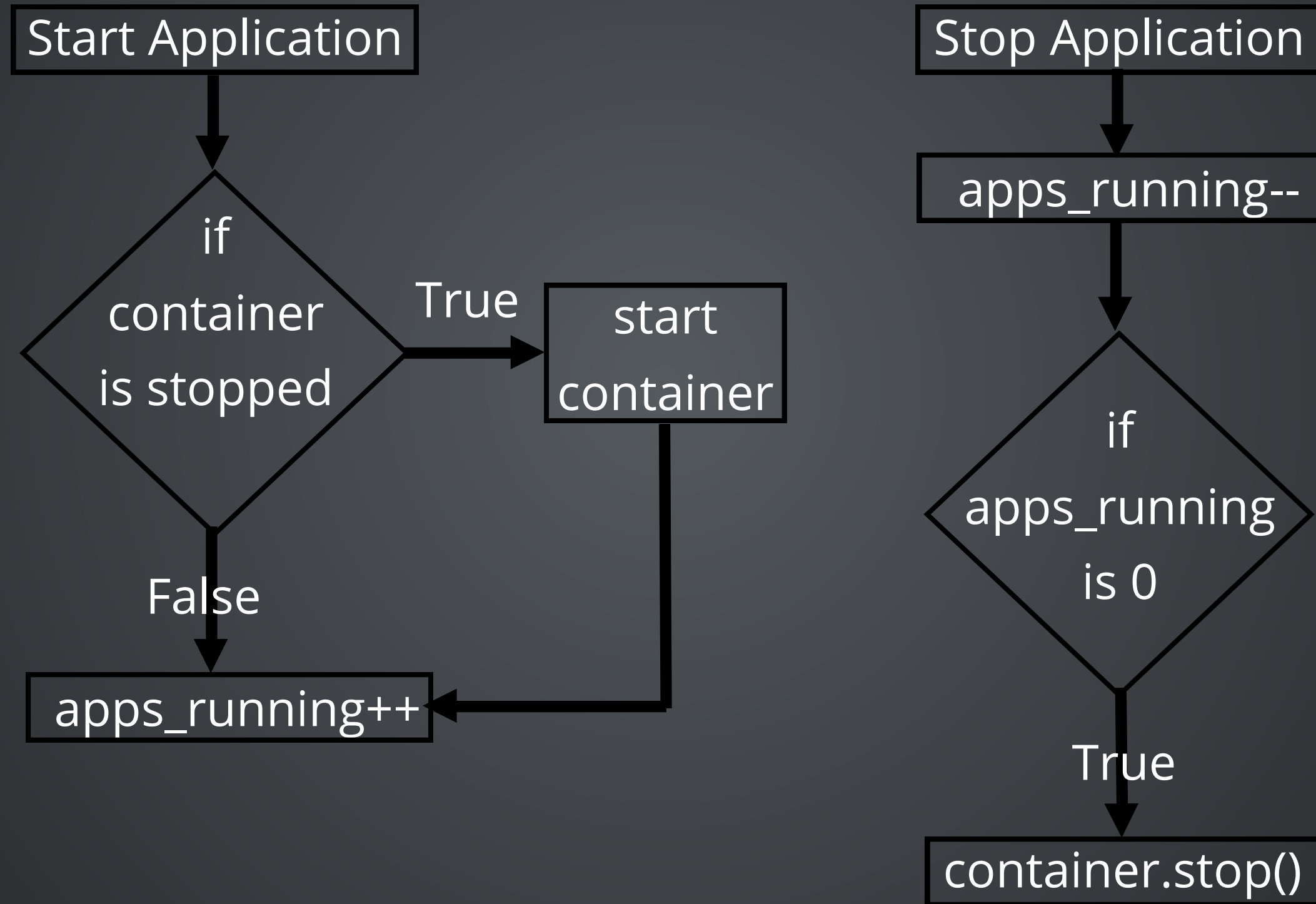
    # Format environment for consumption by lxc command line tool
    env_as_args = ' '
    for k, v in environ.items():
        env_as_args += '--env {}={} ".format(k, v)

    args = shlex.split('lxc exec {id}{env}-- '
                        'sudo -E -u {user} env PATH={path}'.format(id=self._id,
                                                                    path=environ['PATH'],
                                                                    user=environ['USER'],
                                                                    env=env_as_args))

    self._start_window_manager(args.copy())

    # Add the pre-split application exec line to args
    args.extend(app_exec_line)
    return psutil.Popen(args)
```

lxc-manager, lxd-manager



libertine-container-manager

Command-line tool to allow users to manage containers

- Create/destroy/update containers
- Install/remove packages
- Enable/Disable multiarch support
- Add/remove bind-mounts

libertine-container-manager

```
$ libertine-container-manager create  
  --id      my-container  
  --name "My Container"  
  --distro xenial  
  --type   lxc
```

Use lxc to
build
container

```
{  
  "id": "my-container",  
  "name": "My Container",  
  "distro": "xenial",  
  "type": "lxc",  
  "installedApps": []  
}
```

Filesystem:

\$HOME/.cache/libertine-container/my-container/rootfs/...

\$HOME/.local/libertine-container/user-data/my-container/...

libertine-container-manager

```
$ libertine-container-manager install-package  
  --id      my-container  
  --package libreoffice
```

Install
package
within lxc

```
{  
  "id": "my-container",  
  "name": "My Container",  
  "distro": "xenial",  
  "type": "lxc",  
  "installedApps": [  
    "libreoffice"  
  ]  
}
```

libertined

- Asynchronous libertine-container-manager
- Creates locks per container to prevent calling apt simultaneously
- `_run` implementation mirrors libertine-container-manager subcommands
- Sends progress signals over dbus

```
def start(self):
    self._progress = libertine.service.progress.Progress(self._connection)
    thread = threading.Thread(target=self.run)
    thread.start()
    return thread

def run(self):
    if not self._before():
        self._progress.finished(self.container)
        self._delayed_callback()
        return

    if self._lock is not None:
        with self._lock:
            self._run()
    else:
        self._run()

    if self.running:
        self._progress.finished(self.container)
        utils.refresh_libertine_scope()

    self._delayed_callback()
```

liblibertine

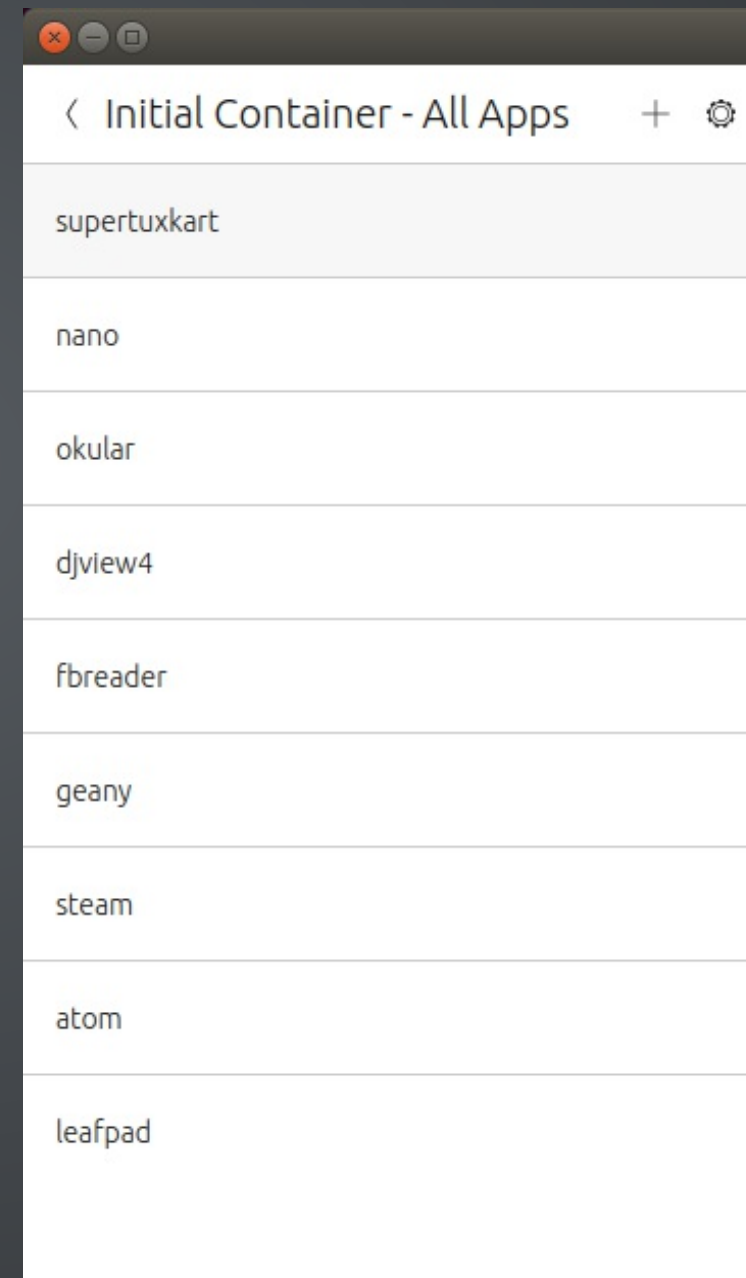
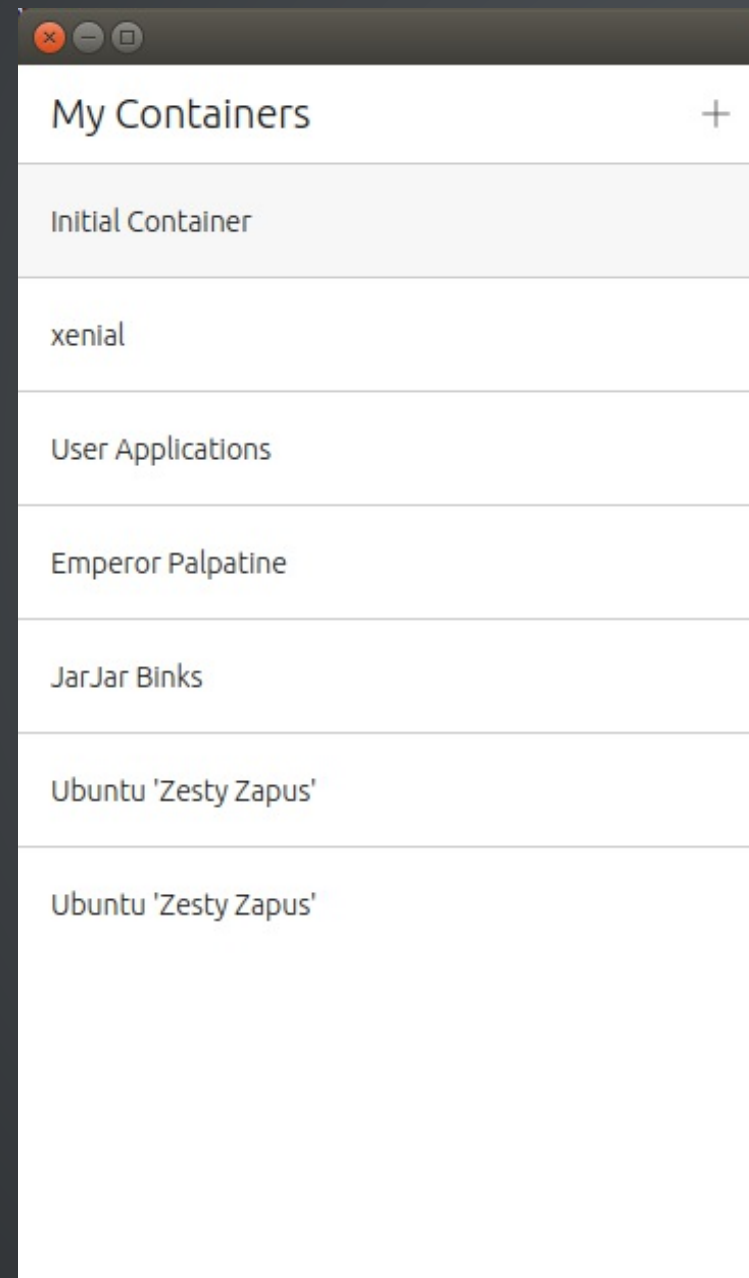
C++ library for reporting container information from the d-bus service to lower-level external libraries

```
static QVariantList
dbusCall(QDBusConnection const& bus, QString const& iface, QString const& path,
        QString const& method, QVariantList const& args = QVariantList())
{
    auto message = QDBusMessage::createMethodCall(SERVICE_INTERFACE, path, iface, method);
    message.setArguments(args);
    auto response = bus.call(message);
    if (response.type() == QDBusMessage::ErrorMessage)
    {
        qWarning() << "error calling result" << response.errorMessage();
        return QVariantList();
    }

    return response.arguments();
}
```


libertine-manager-app

C++/Qt GUI to wrap libertine-container-manager



libertine-launch

python tool to launch libertine applications isolated from the host environment through a session providing:

- A cleansed and remapped set of environment variables
- A collection of redirected sockets
- Zero or more running auxiliary programs
- An executable command line forwarded to the container

pasted

C++ binary to
monitor clipboard
events from the
Ubuntu ContentHub
and forward these
events to libertine
applications using
Qt signals/slots.

```
QString surfaceId = getPersistentSurfaceId();

QDBusReply<bool> isFocused = unityFocus->call(UNITY_FOCUSINFO_METHOD, surfaceId);

if (isFocused == true)
{
    focusChanged();
    hasFocus = true;
}

Display *dpy = XOpenDisplay(NULL);
XSelectInput(dpy, XDefaultRootWindow(dpy), FocusChangeMask);

while (1)
{
    XNextEvent(dpy, &event);

    isFocused = unityFocus->call(UNITY_FOCUSINFO_METHOD, surfaceId);

    if (hasFocus == false && isFocused == true)
    {
        qDebug() << "Surface is focused";
        focusChanged();
        hasFocus = true;
    }
    else if (hasFocus == true && isFocused == false)
    {
        qDebug() << "Surface lost focus";
        hasFocus = false;
    }
}
```

libertine-launch flow

System discovers libertine apps with liblibertine



User clicks
application icon

libertine-launch --id my-container gimp

Setup
Session
(sockets +
environ)

Start
utilities
(pasted,
xmir)

Container
instance

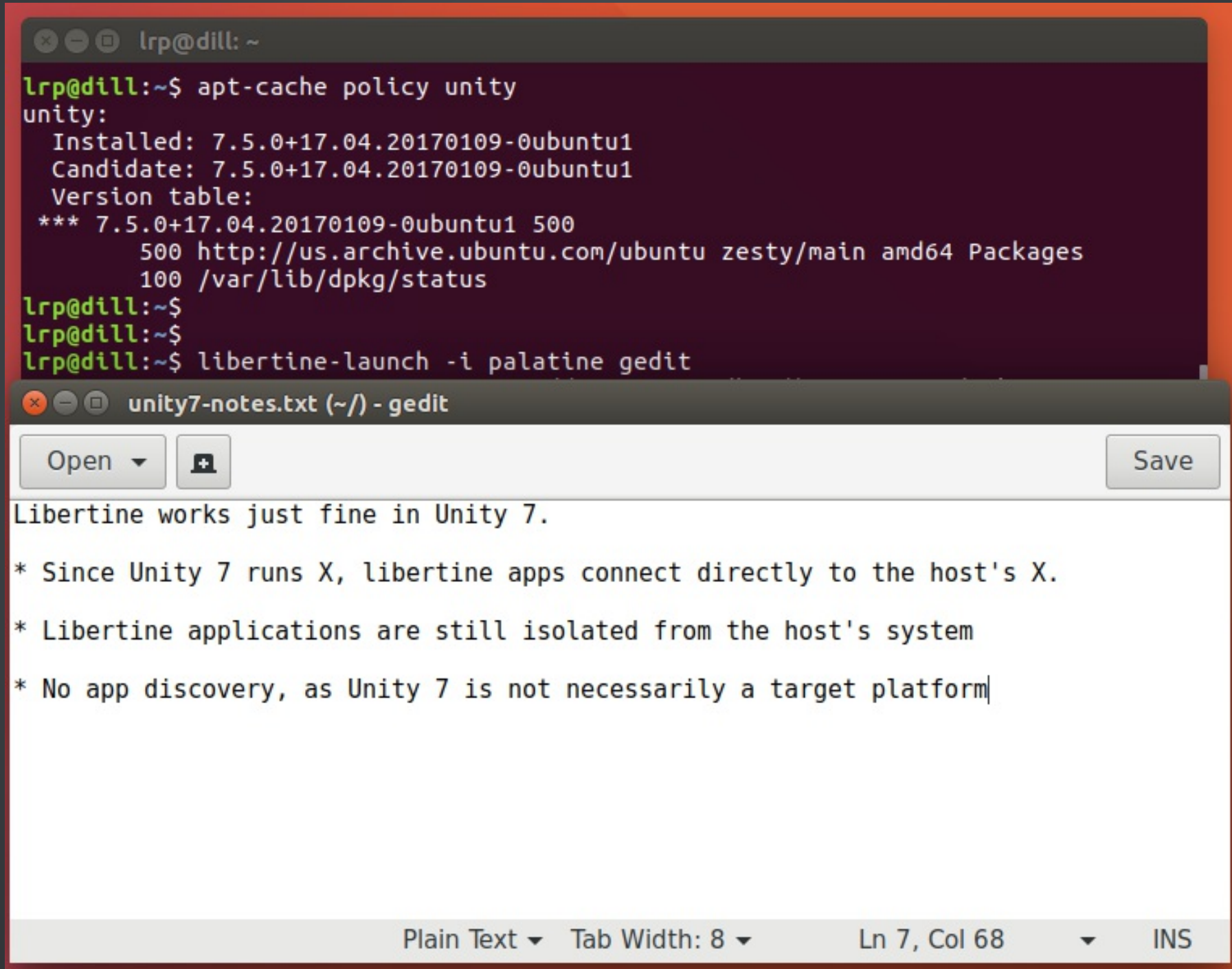
- Update bind-mounts
- Launch window manager
- Launch app!

Libertine Review

- Create containers as a safe place for classic apps to live isolated from the host system
- Install packages using dpkg (even if dpkg unavailable on host)
- Create connections between containers and host system for graphics, sound, network, etc.
- Start window manager within container given a known \$DISPLAY connected to the host
- Launch applications with a sanitized version of host's environment

So... Does it work?

Unity 7



The screenshot shows a terminal window and a text editor window. The terminal window displays the output of the command `apt-cache policy unity`, which shows that the installed and candidate versions are `7.5.0+17.04.20170109-0ubuntu1`. The version table lists the source as `http://us.archive.ubuntu.com/ubuntu zesty/main amd64 Packages` and the package status as `/var/lib/dpkg/status`. The text editor window, titled `unity7-notes.txt (~/) - gedit`, contains the following text:

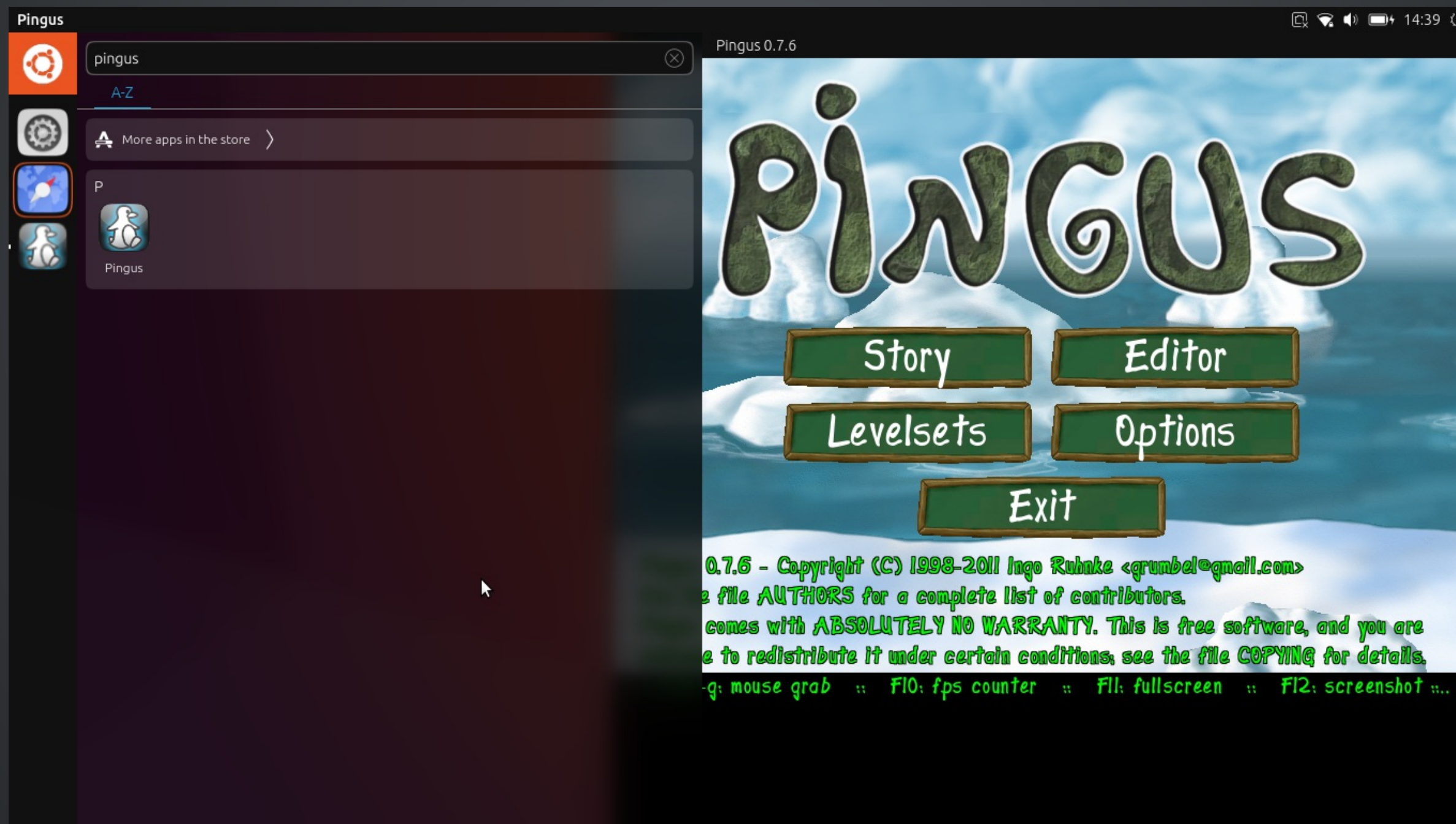
```
Libertine works just fine in Unity 7.
```

- * Since Unity 7 runs X, libertine apps connect directly to the host's X.
- * Libertine applications are still isolated from the host's system
- * No app discovery, as Unity 7 is not necessarily a target platform

The text editor window has a status bar at the bottom showing `Plain Text`, `Tab Width: 8`, `Ln 7, Col 68`, and `INS`.

Unity 8 + Mir

Unity8 uses Mir instead of X, requiring a compatibility layer (xmir) to run X11-based applications.



Ubuntu Touch (Clicks)

- Same setup as Unity 8 on Desktop
- Different screen sizes of mobile devices caused new issues
- Needed on-screen keyboard for mobile devices
- Required chroot-based libertine container due to older Linux kernel

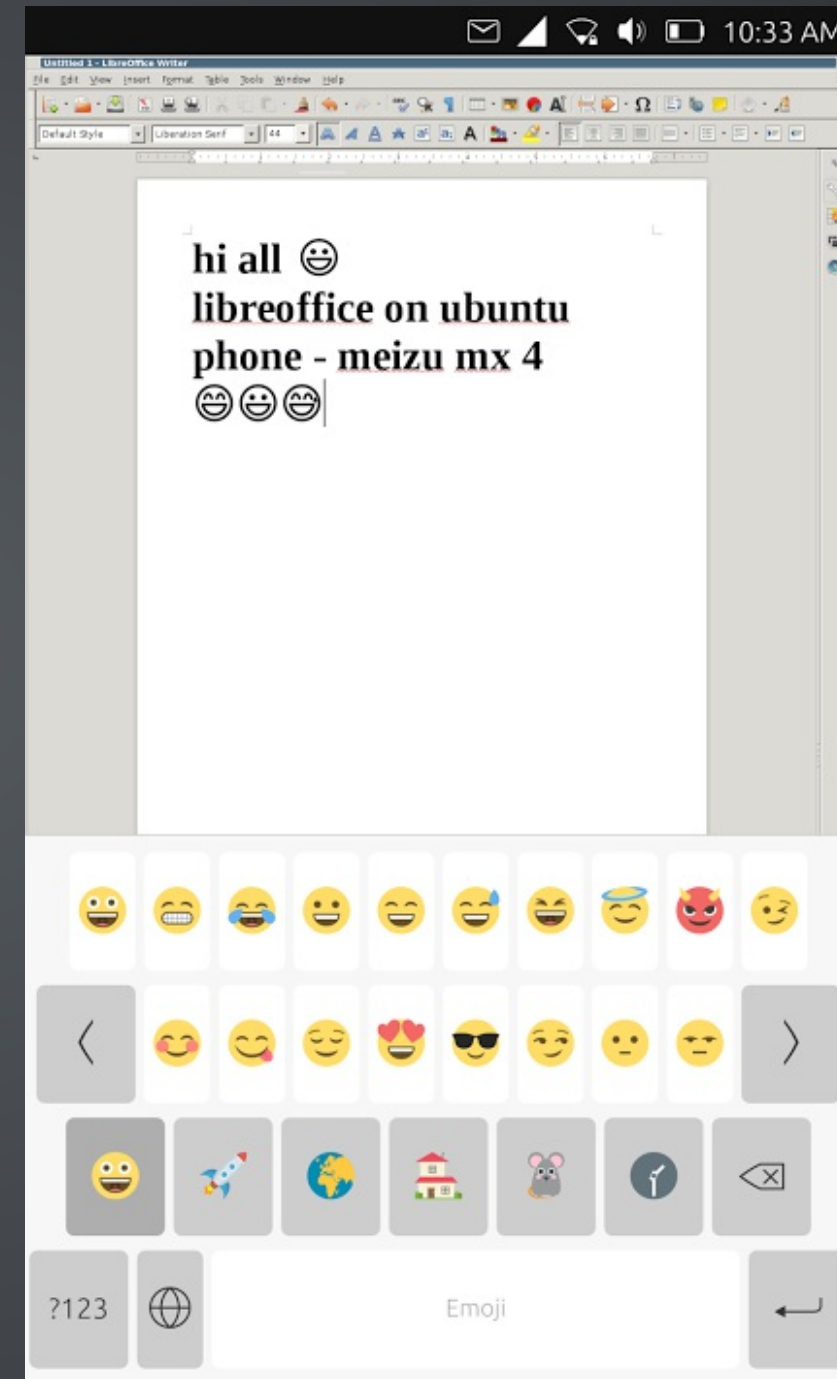


Image courtesy

https://www.reddit.com/r/Ubuntu/comments/4sb03p/libreoffice_on_ubuntu_phone/

Ubuntu Personal (All-Snaps)

- Updated path and environment checks throughout libertine python library
- Loosened hard failures during creation to account for unavailable d-bus services or initialization binaries
- Availability of a lxd snap hurried production of a libertine lxd backend

Snap Interfaces

- lxd
- network
- network-bind
- home
- mir
- x11
- unity7
- opengl
- pulseaudio
- alsa

Live Demo!

[Alternative video link](#)

Goal: To run classic applications in a confined environment using a new package management system and display server

Solution: A container-based software suite for installing and launching classic applications without the knowledge of package management or container syntax called Libertine.

What We're Still Working On...

- lxd start/stop time can be slow, causing sluggish application start
- "Classic Application Store" integrated with the Desktop Snap Store
- D-bus accessibility difficulties
- Our snap is very large

Thanks for listening!

My Blog: [**https://larry-price.com**](https://larry-price.com)

Twitter: [**@larryrprice**](https://twitter.com/larryrprice)

Freenode: [**larryprice**](https://freenode.net/larryprice)

Github: [**https://github.com/larryprice**](https://github.com/larryprice)

Launchpad: [**https://launchpad.net/~larryprice**](https://launchpad.net/~larryprice)

Libertine code can be found at
[**https://code.launchpad.net/libertine**](https://code.launchpad.net/libertine)