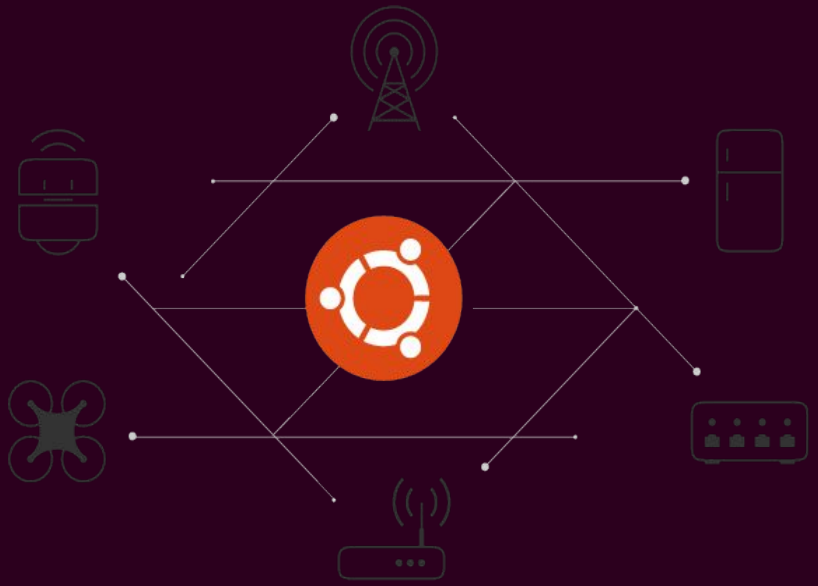
ubuntu

Ubuntu Core

Transactionality, security and app stores



XiaoGuo, Liu
xiaoguo.liu@canonical.com

CANONICAL

Agenda

ubuntu

- Ubuntu Core overview
- Installation for snapcraft and snapd
- Snapcraft: snap development tool
- Ubuntu Core securities
- Debugging snaps
- Ubuntu Store
- Getting involved
- References
- Q&A

Ubuntu Core Overview

What is Ubuntu Core?

A minimal version with the same bits as today's Ubuntu

Ubuntu Core with **transactional updates**

Applications confined by technologies led by Canonical

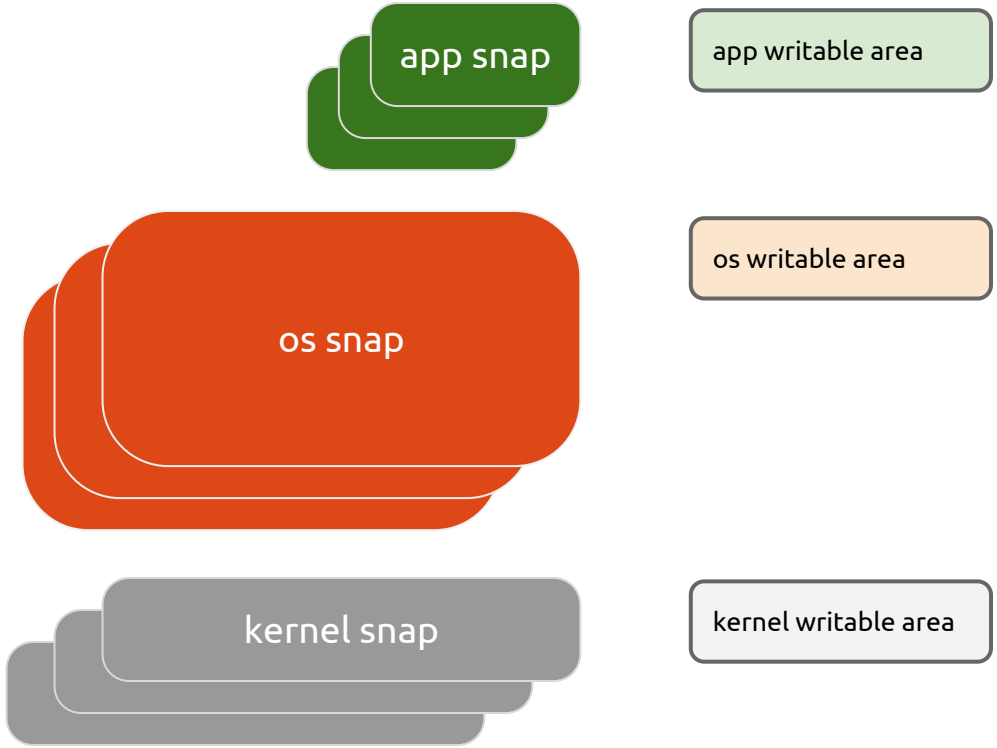
Safe, reliable, worry free **updates** with tests and **rollback**

Amazing developer experience with **snapcraft**

Easily extensible

Easily create **app stores** for all your devices

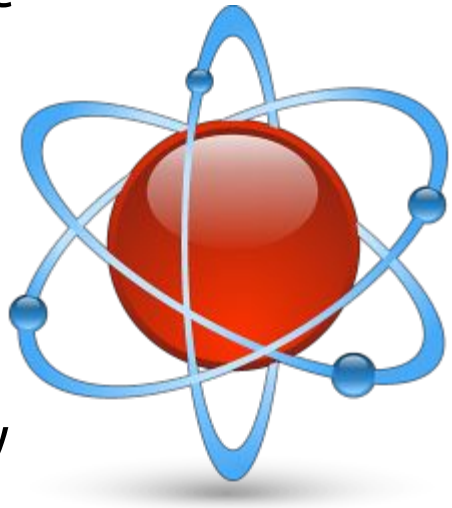
Transactional update and rollback



Transactional updates

Ubuntu Core apps and Ubuntu Core itself can be upgraded atomically and rolled back if needed. Delta change is applied when updating from different versions. Snaps can be easily uninstalled (by deleting the snap package)

A bulletproof approach that is perfect for deployments where predictability and reliability are paramount. It’s called “transactional” or “image-based” systems management

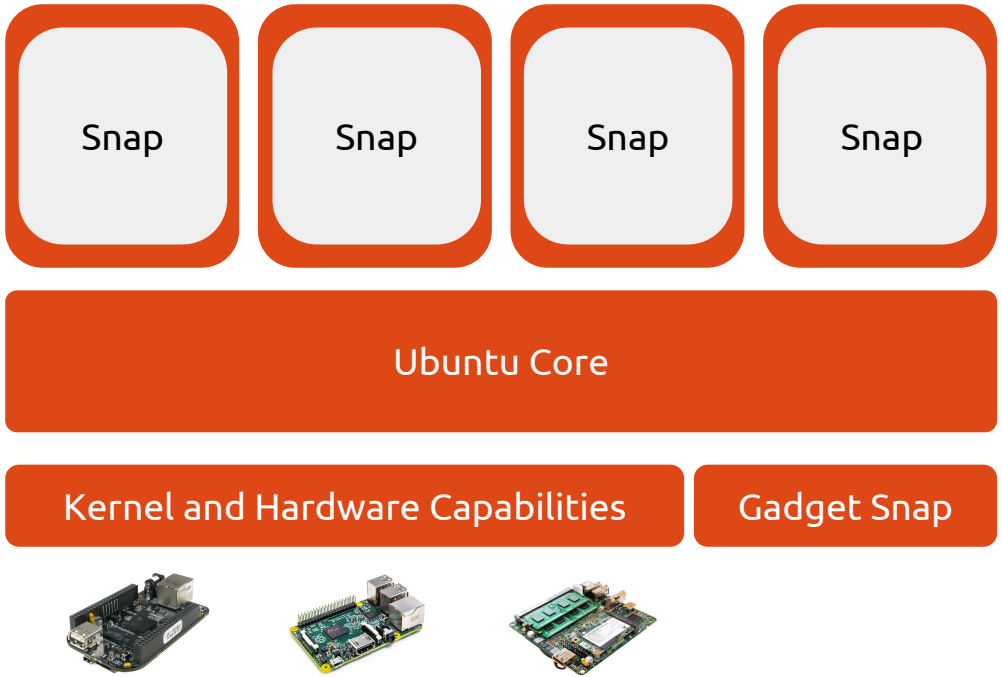


Ubuntu Core architecture

Software Innovator(s)

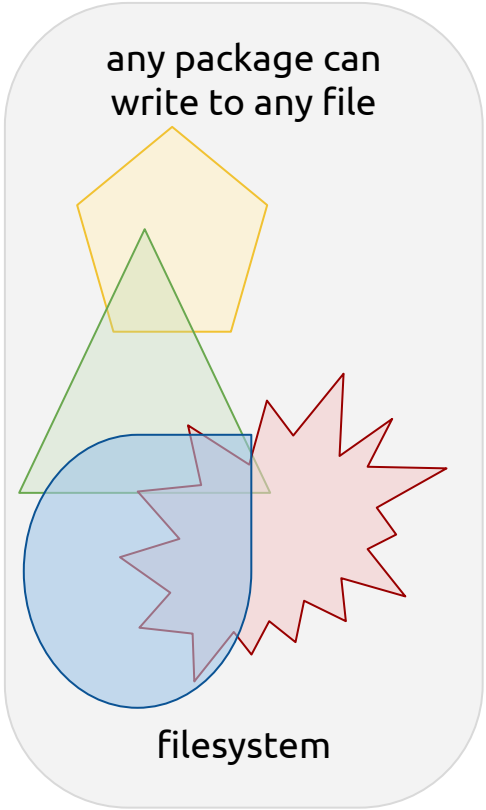
Canonical

Hardware Innovator

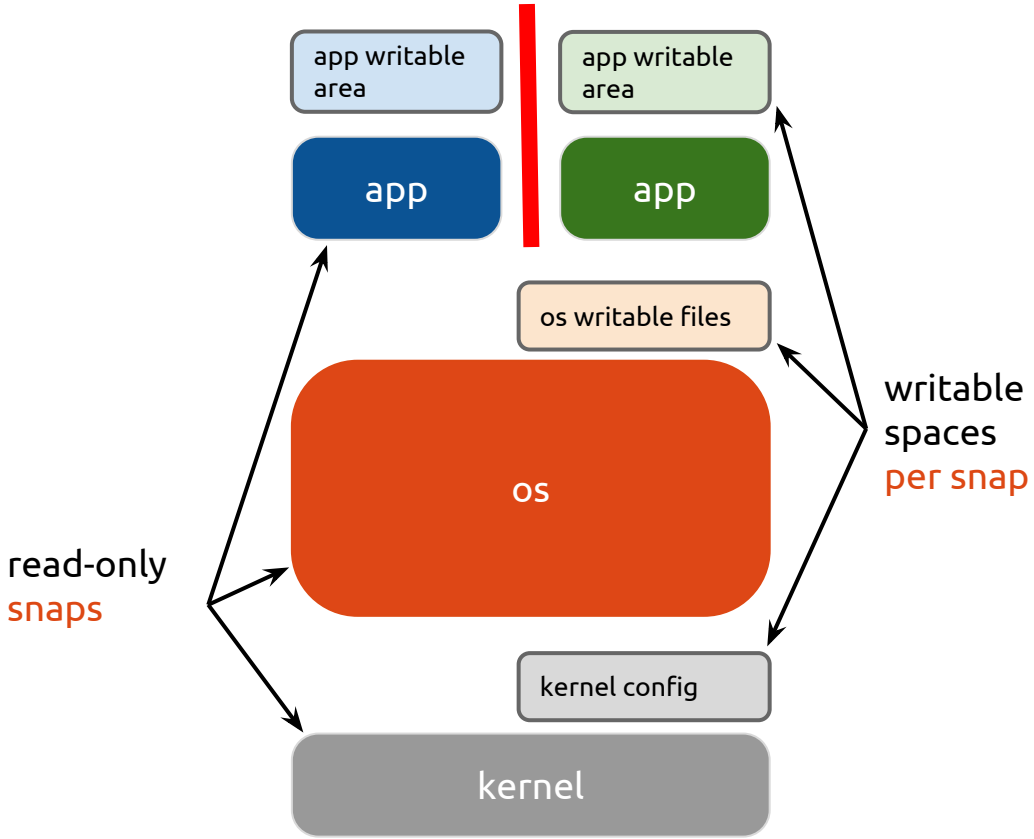


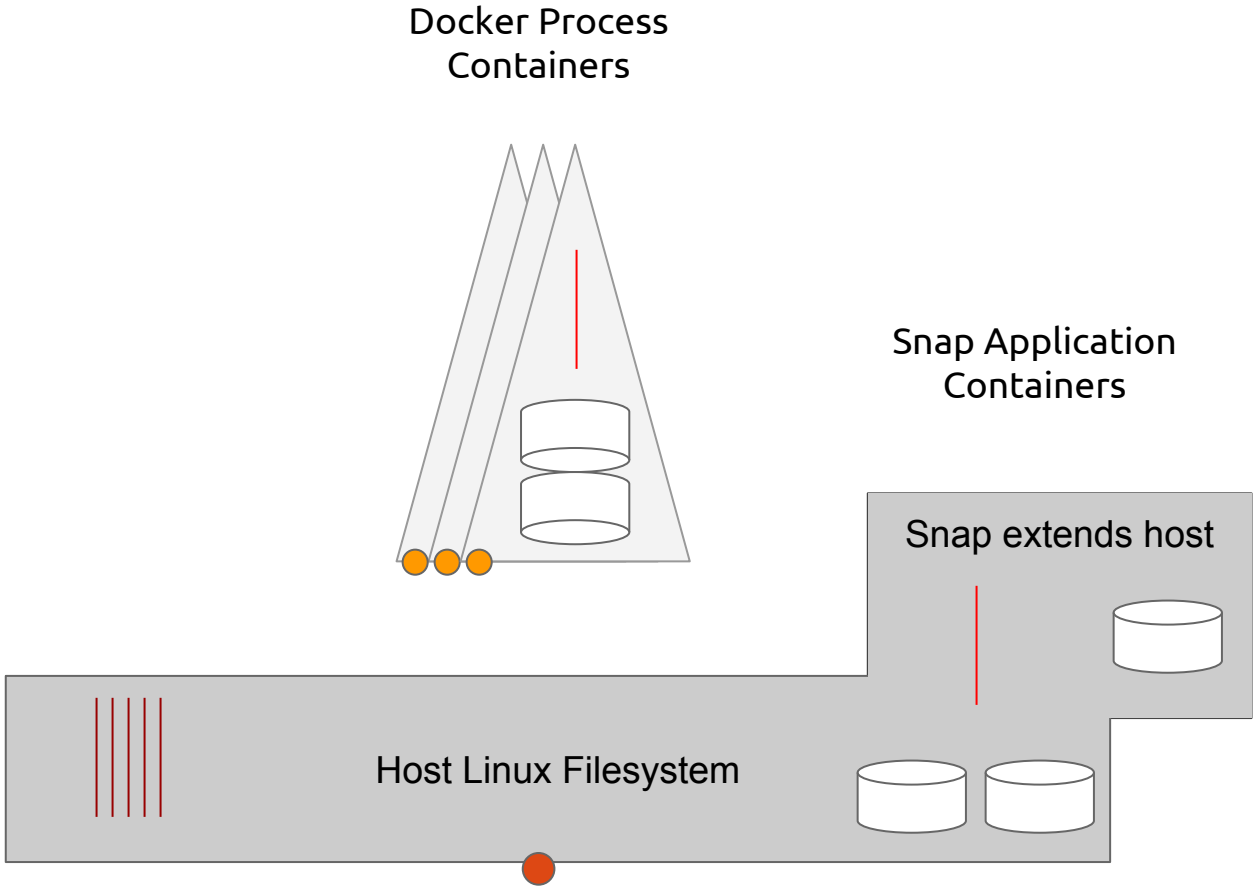
- Automatic updates
- Secure by design
- Automatic backups
- Automatic rollback
- All in snaps
  - Apps
  - Kernel
  - OS
  - Gadget

classic



snappy




Docker Process
Containers
Snap Application
Containers
Snap extends host
Host Linux Filesystem

Apps confinement: Trust model

By default all **application snaps** are untrusted by the OS and they:

- Cannot access other applications' data
- Cannot access non-app-specific user data
- Cannot access privileged portions of the OS

Trusted by the OS **VS** Untrusted by the OS

Apps confinement: Technologies

Several technologies are used by Ubuntu Core to:

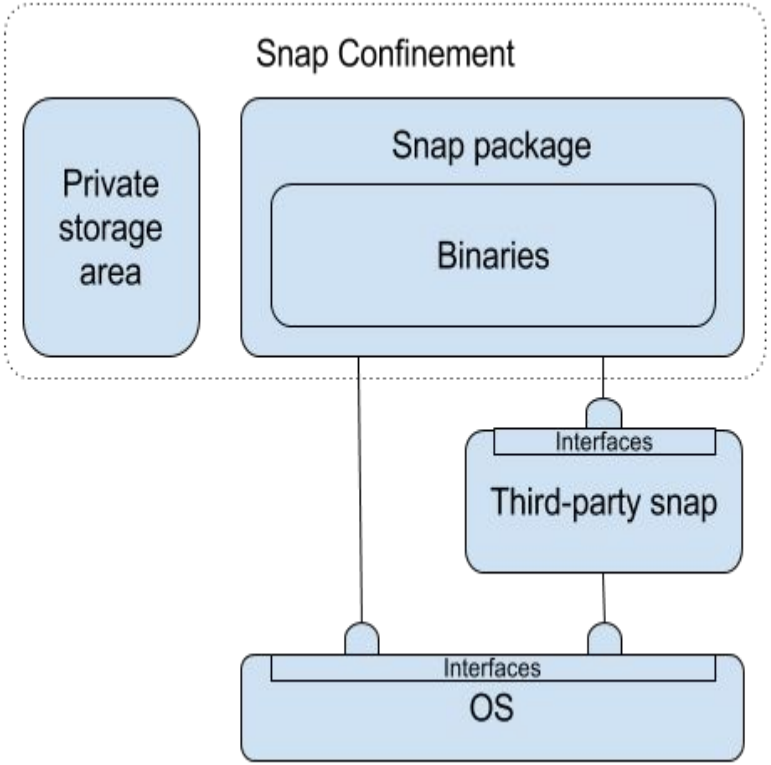
- Implement the security sandboxing
- Implement the application isolation

These technologies are mainly:

- **AppArmor**: A **Mandatory Access Control** system to confine programs and processes to a limited set of resources (Application Isolation)
- **Seccomp**: A **secure computing mode** that provides an application sandboxing mechanism
- **Device cgroups:** are a kernel mechanism for grouping, tracking, and limiting the resource usage of tasks

The snapd system

- **snapd**, a management environment that handles installing and updating snaps using the transactional system, as well as garbage collection of old versions of snaps
- **snapd-confine**, an execution environment for the applications and services delivered in snap packages
- **Interface**, snaps interact with each other using interface



What is a snap?

- Is a **squashFS filesystem** containing your app code and a snap.yaml file containing specific metadata. It has a read-only filesystem and, once installed, a writable area
- **Is self-contained**. It bundles most of the libraries and runtimes it needs and can be updated and reverted including its data without affecting the rest of the system
- Is **confined** from the OS and other apps through security mechanisms, but can exchange content and functions with other snaps according to fine-grained policies controlled by the user and the OS defaults
- In a snappy sys., all software beyond the bootloader is distributed as a snap

Snap directory architecture

Directory	Purpose	Content
`root`	Where your code is shipped, along wrapper files generated by Snapcraft.	App source code, `*.wrapper` files
`bin`	Contains all binaries shipped by the app.	Binaries
`meta`	Contains details that describe the snap.	`snap.yaml`, licence.txt
`meta/gui`	Contains GUI related files for the snap.	Desktop files, icons

```
liuxg@liuxg:/snap/hello/current$ ls
bin                                command-env.wrapper          command-hello.wrapper  share
command-createfiletohome.wrapper   command-evil.wrapper        command-sh.wrapper
command-createfile.wrapper         command-hello-world.wrapper  meta
```

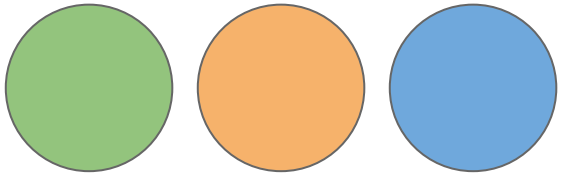
Snap - a universal Linux format package

Snaps aim to work on any distribution or device, from IoT devices to servers, desktops to mobile devices

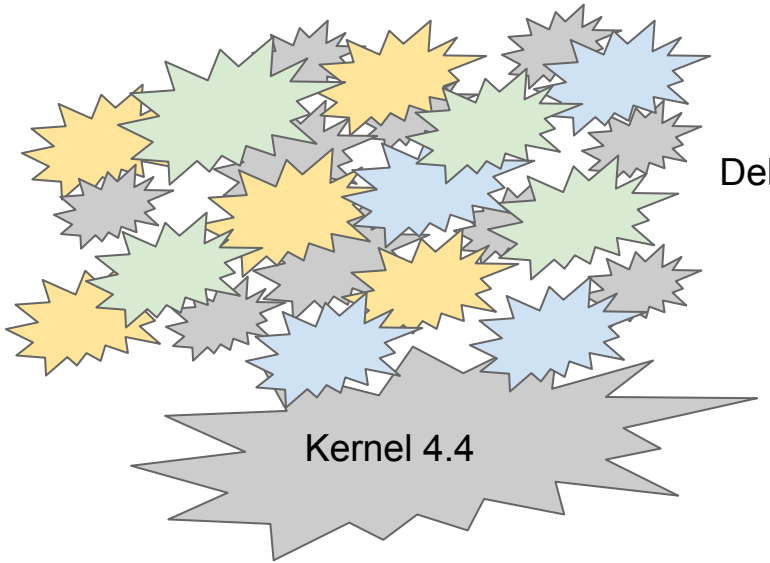
http://snapcraft.io/docs/core/install

Ubuntu 16.04 LTS supports snaps by default

Classic Ubuntu 16.04

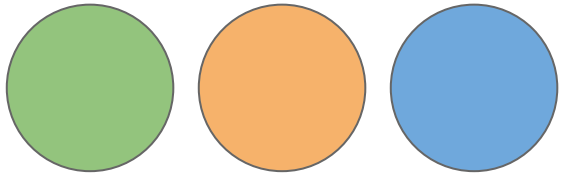


App Snaps

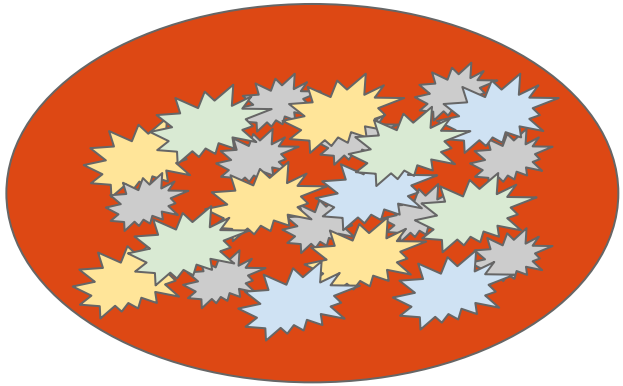


Debs

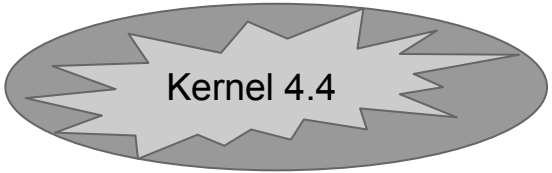
All-Snap Ubuntu Core



App Snaps

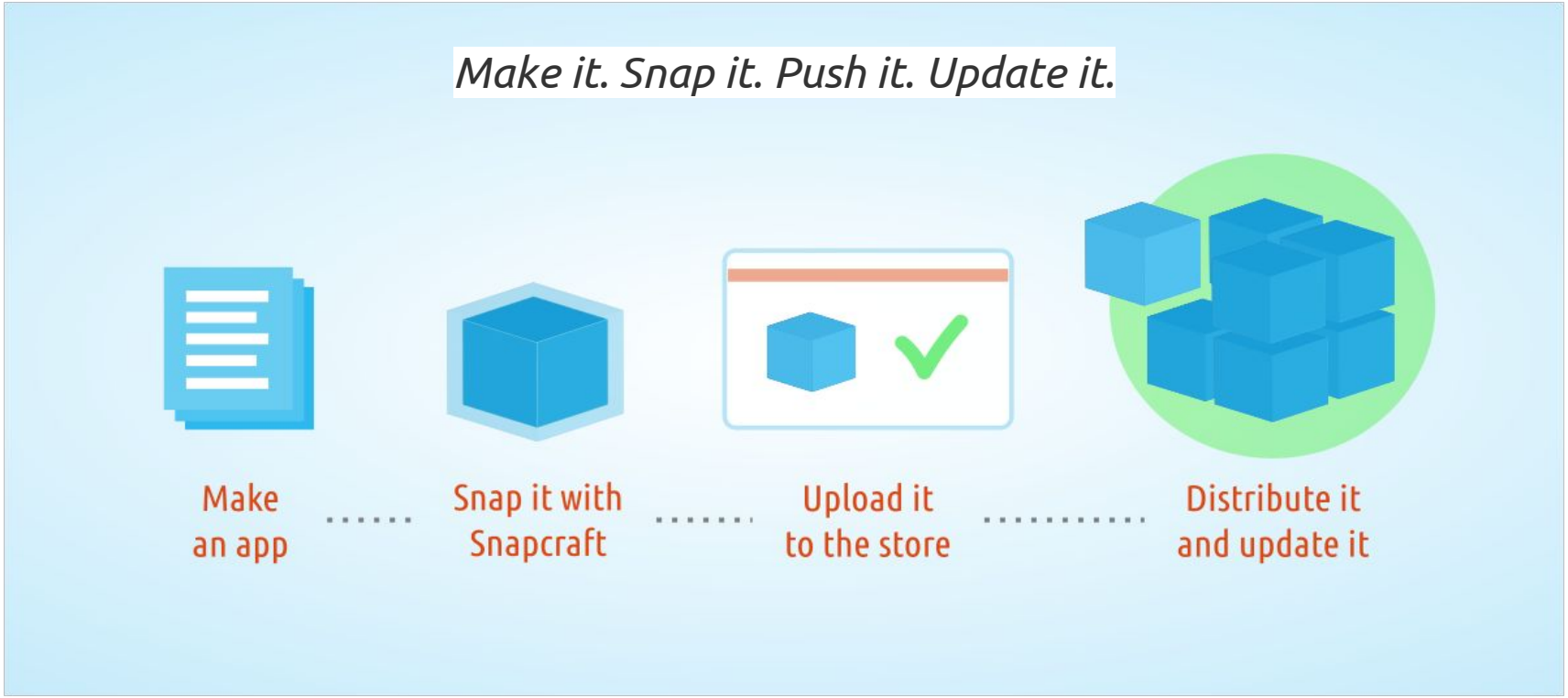


OS Snap



Kernel Snap

A new way to get your app on Ubuntu

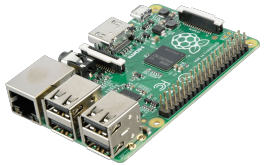


http://snapcraft.io/

Ubuntu Core 16 Requirements

ubuntu core

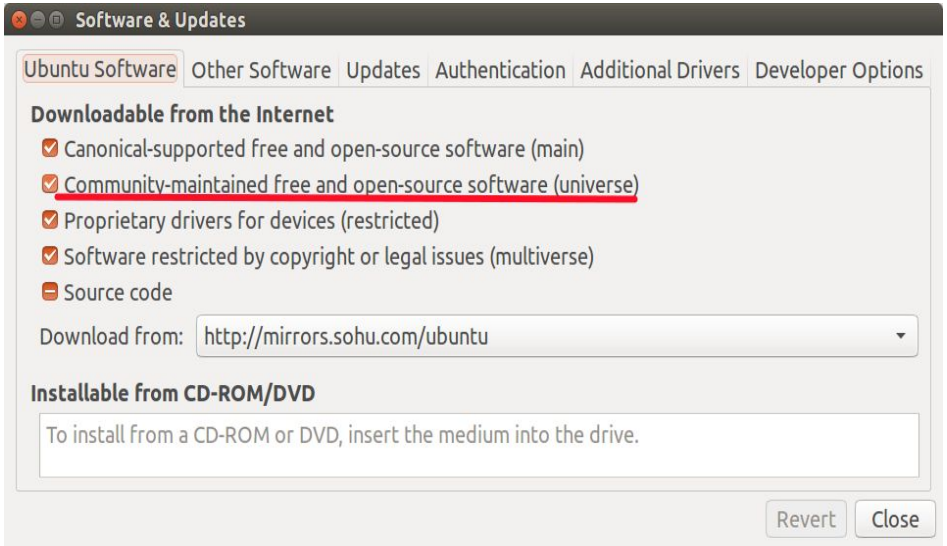
Architecture	x86: 32 & 64 bits ARM: 32 & 64 bits
Memory	**256 MB**
Storage	**512 MB**
Connectivity	**WiFi, Ethernet, USB, BT 4.0...**



https://developer.ubuntu.com/en/snappy/start/gadget-snaps/

Installation for snapcraft and snapd

Install snapcraft and snapd



Or use $ sudo add-apt-repository universe

$ sudo apt update
$ sudo apt install snapd
$ sudo apt install snapcraft build-essential

$ snap --version
snap 2.16ubuntu3
snapd 2.16ubuntu3
series 16
ubuntu 16.04

Install & remove snaps

$ sudo snap install ss-qt

$ ss-qt

```
liuxg@liuxg:~$ snap list
Name                  Version                 Rev
ubuntu-calculator-app 2.1+snap3               5
ubuntu-core           16.04+20160531.11-56    122
webcam-webui          1.0                     x1
```

Apps from store have numeric revision

$ sudo snap remove ss-qt

```
liuxg@liuxg:~$ snap find calculator
Name                  Version    Developer
ubuntu-calculator-app 2.1+snap3  ubuntucoredev
```

$ sudo snap try prime/

$ sudo snap install hello --beta

$ sudo snap install hello_1.0_all.snap --dangerous

```
$ snap install flubber --beta
Error: this version of foo requires devmode.
$ snap install flubber --beta --devmode
WARNING: snaps installed in devmode are not confined. You are trusting
all the private data on this system to the developer "frankie".

Do you still want to install ‘flubber’? [y/N] N
```

--devmode enables you to run the snap outside the confinement sandbox and get unrestricted access to system resources

devmode and jailmode

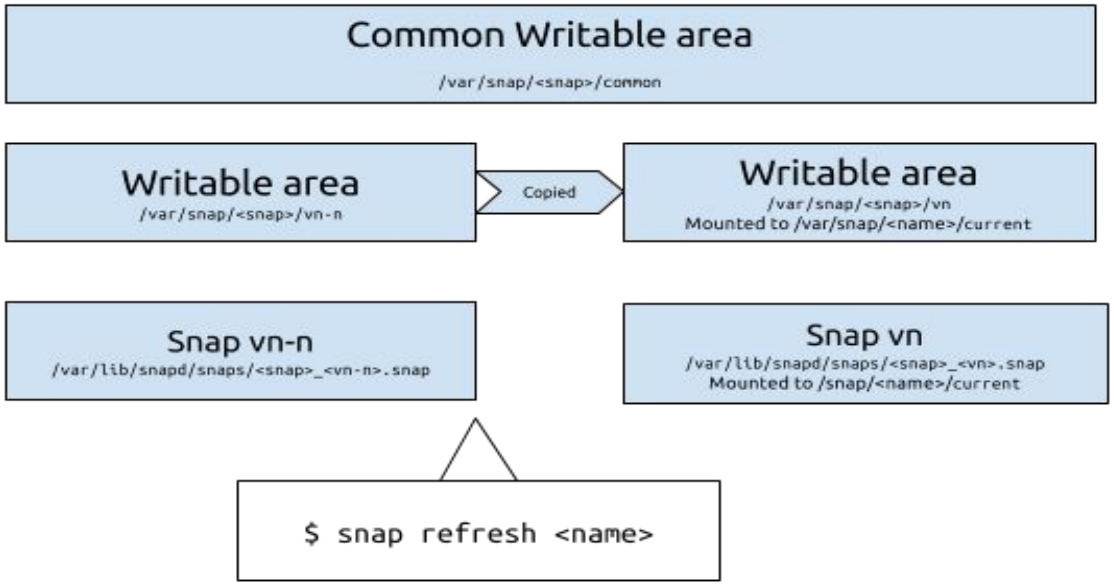
- devmode
  - Sometimes it is helpful when developing a snap to not have to worry about the security sandbox in order to focus on developing the snap. To support this, snappy allows installing the snap in developer mode which puts the security policy in complain mode (where violations against security policy are logged, but permitted).
  - **$ sudo snap install --devmode <snap>**

- Jailmode
  - Even if a developer has specified a package to be only installable in devmode, you can force a strict confinement by using the *--jailmode* flag
  - **$ sudo snap install --jailmode <snap>**

Snap update and revert

$ snap refresh **<snap** name**>**
$ snap revert **<snap** name**>**

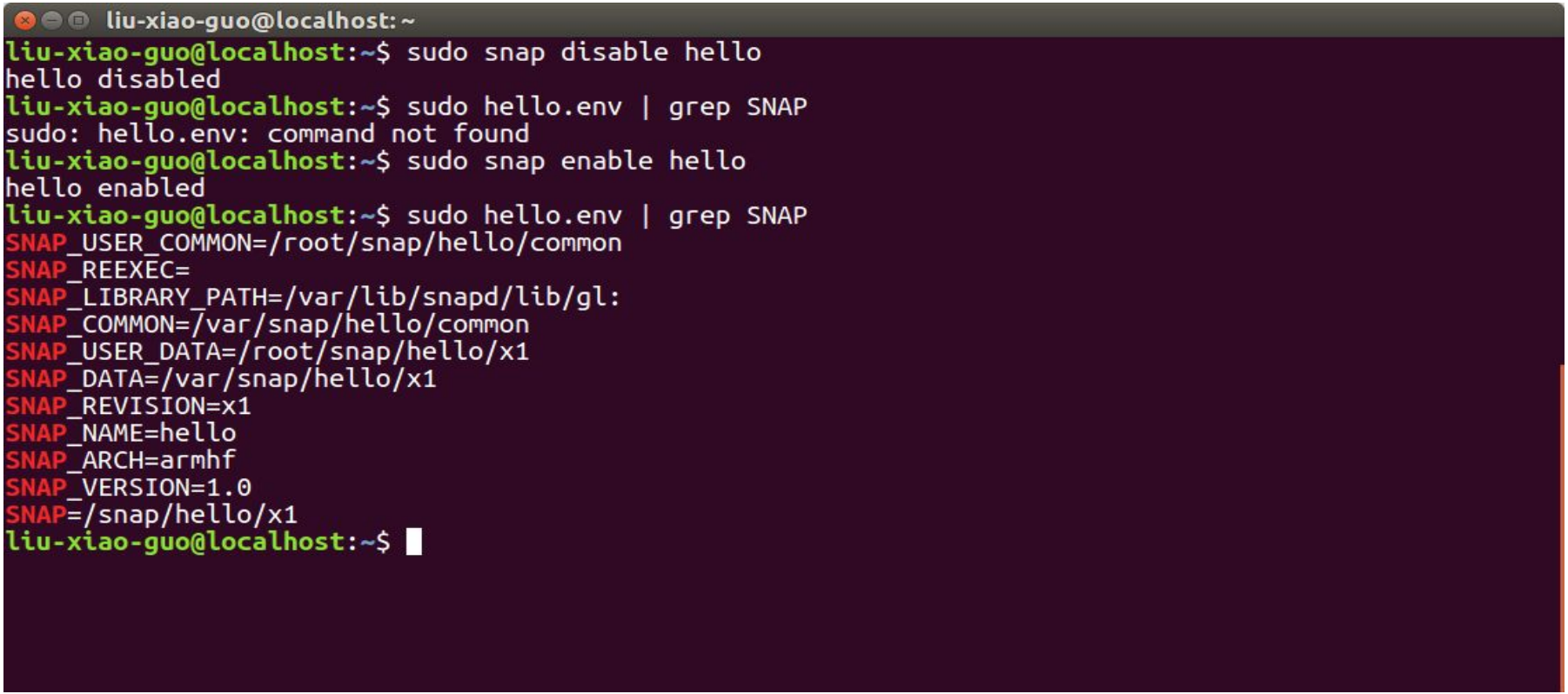
liuxg@liuxg:~$ mount | grep calculator
/var/lib/snapd/snaps/ubuntu-calculator-app_5.snap on
/snap/ubuntu-calculator-app/5 type squashfs (ro,relatime)




```
liuxg@liuxg:/snap/ubuntu-calculator-app/current$ tree -L 2
.
├── bin
│   └── calculator
├── build
│   └── ubuntu-calculator-app
├── command-calculator.wrapper
├── etc
│   ├── apparmor.d
│   ├── dbus-1
│   ├── default
│   ├── drirc
│   ├── fonts
│   ├── gps.conf
│   ├── gss
│   ├── init
│   ├── init.d
│   ├── ldap
│   ├── pki
│   ├── pulse
│   ├── ucf.conf
│   ├── X11
│   └── xdg
├── lib
│   ├── systemd
│   └── x86_64-linux-gnu
├── meta
│   ├── gui
│   └── snap.yaml
├── usr
│   ├── bin
│   ├── lib
│   └── share
└── var
    └── lib
```

Enable/disable an installed snap

If you have an installed snap app, and you do not want it to be effective without uninstalling it, or you want to restart your daemon apps, you may do it as follows:



How to view/extract the files in a snap package

$ snap download <snap-name>

$ unsquashfs -l ubuntu-calculator-app_2.1+snap3_amd64.snap | less

We can use the following command to get all of the files in the snap:

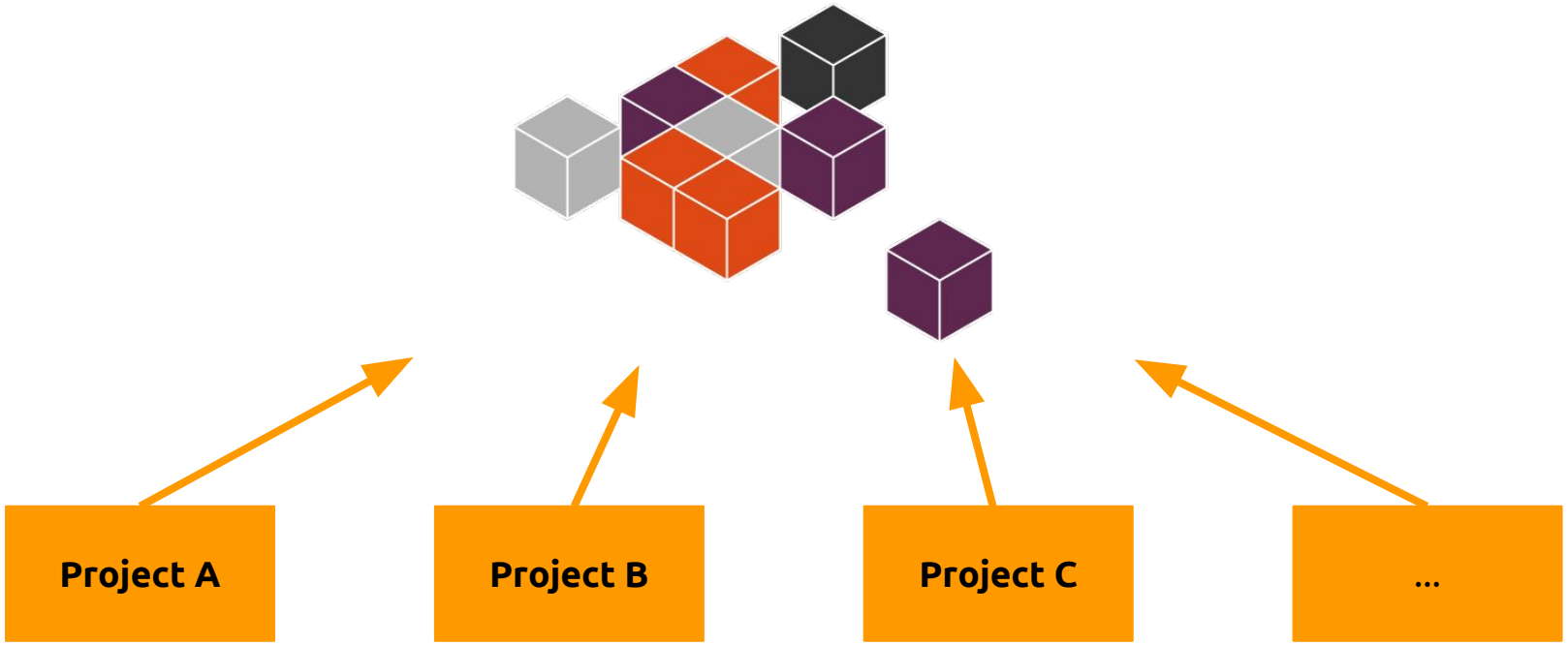
$ unsquashfs ubuntu-calculator-app_2.1+snap3_amd64.snap
$ cd squashfs-root
 # Hack hack hack and go back to the previous directory
$ snapcraft snap squashfs-root

http://blog.csdn.net/ubuntutouch/article/details/53760045

Snapcraft: snap development tool

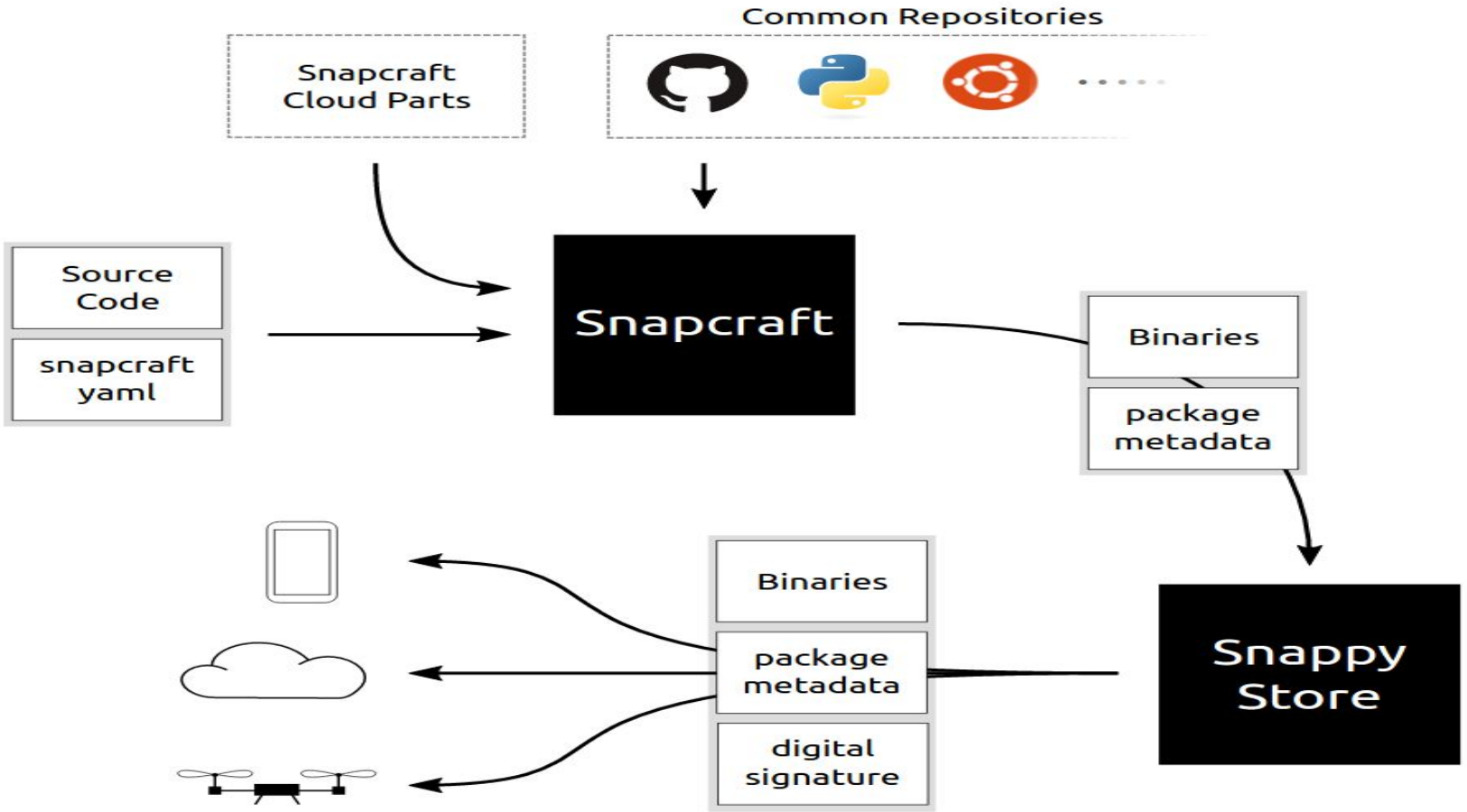
Developer tools: Snapcraft

Snapcraft lets developers assemble their snap from existing projects, leveraging different technologies

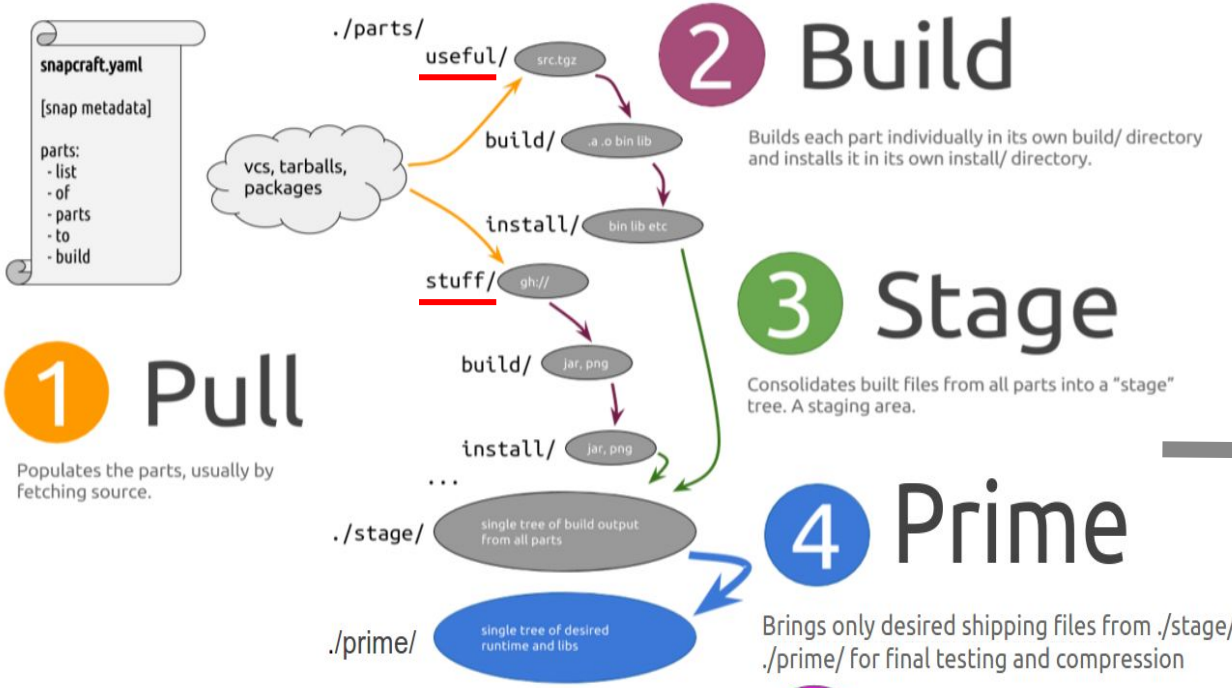


https://github.com/snapcore/snapcraft

Snap development process



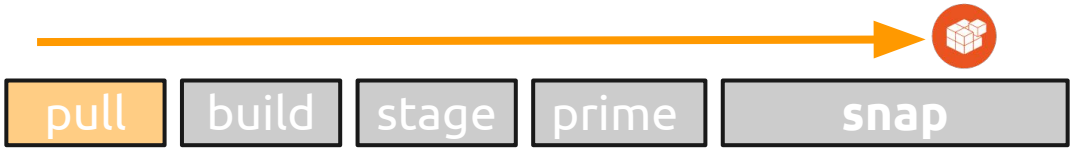
Build snaps on classic, deploy on Ubuntu Core



http://snapcraft.io/create/

Developer tools: Snapcraft

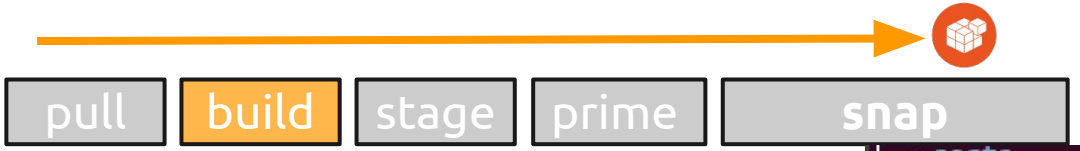
Snapcraft 2.x



The pull phase takes care of the downloading / cloning of the remote files needed for this part or **copy** from local source

Developer tools: Snapcraft

Snapcraft 2.x



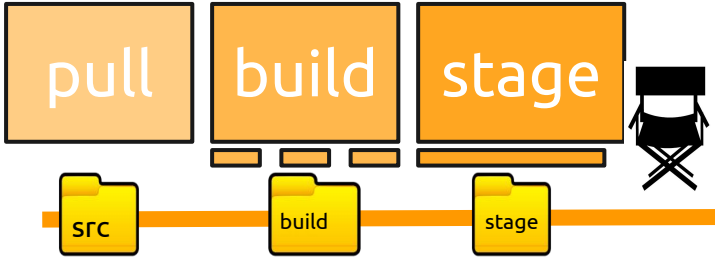
The build phase builds the parts of the downloaded code.

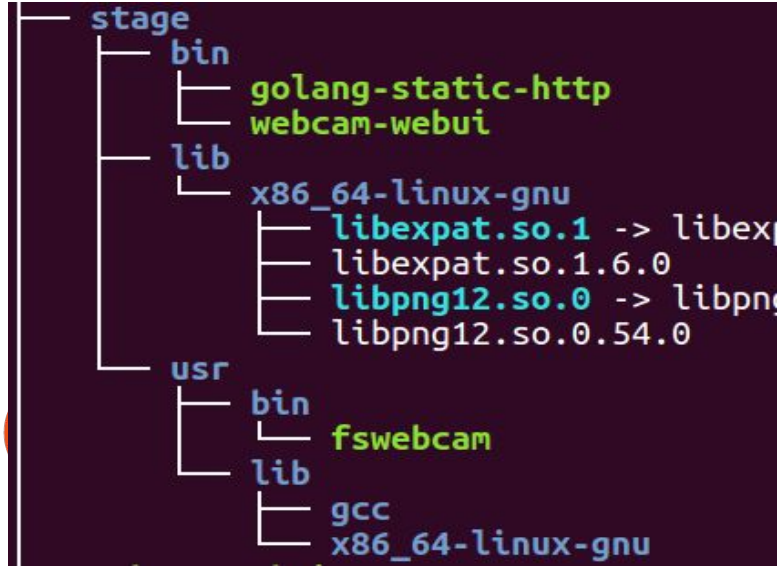

```
parts
├── cam
│   ├── build
│   ├── go
│   │   ├── bin
│   │   └── src
│   ├── install
│   │   ├── bin
│   │   ├── etc
│   │   ├── lib
│   │   ├── usr
│   │   └── var
│   ├── src
│   ├── state
│   │   ├── build
│   │   ├── prime
│   │   ├── pull
│   │   └── stage
│   └── ubuntu
│       ├── download
│       ├── etc
│       └── var
└── glue
    ├── build
    │   └── webcam-webui
    ├── install
    │   └── bin
```

Developer tools: Snapcraft

Snapcraft 2.x

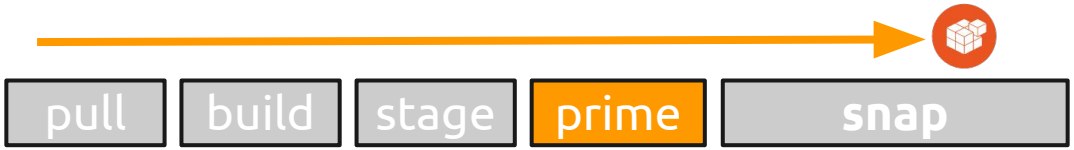
The stage phase copies the installed files into a user-visible stage/folder. All parts share the same file layout



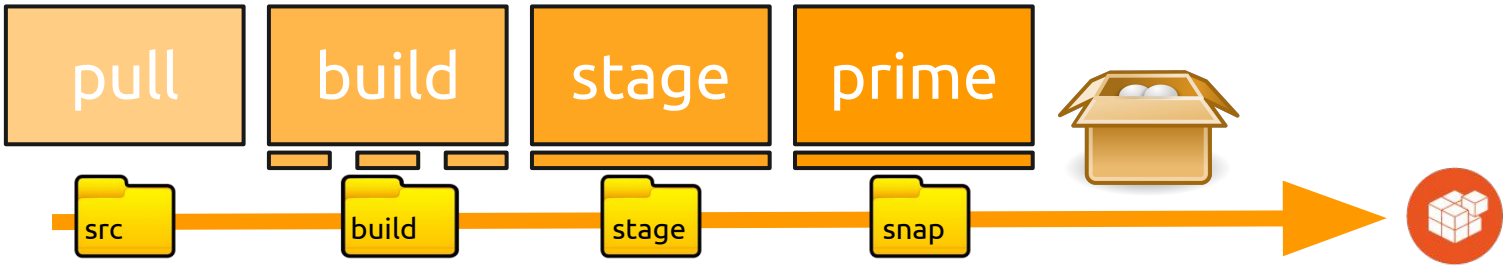


Developer tools: Snapcraft

Snapcraft 2.x

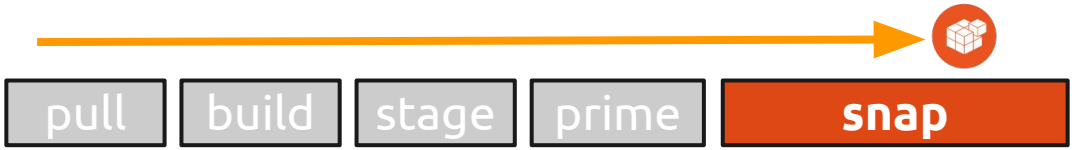


The prime phase performs the final copy and preparation for the snap, removing the parts that are not needed

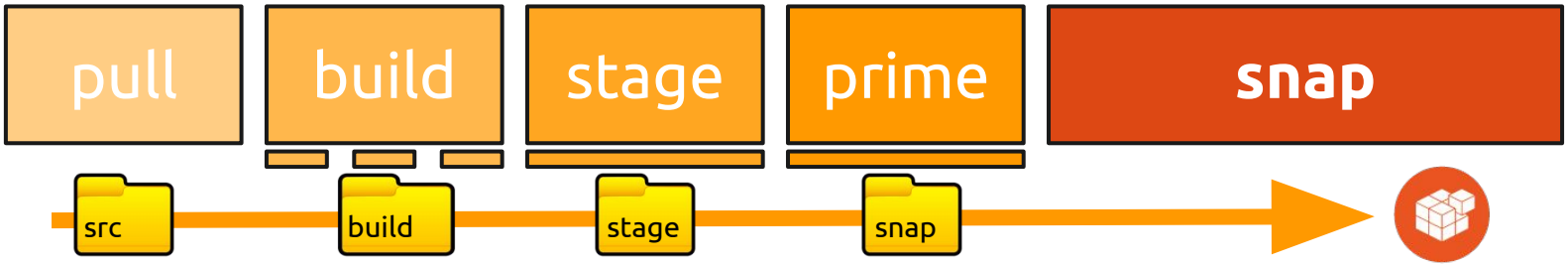


Developer tools: Snapcraft

Snapcraft 2.x



The snap phase finalises the snapcraft lifecycle creating the snap that can now be shared with your devices or made available to the world



Developer tools: Snapcraft plugins

Snapcraft

Snapcraft supports numerous technologies through an ecosystem of plugins

Snapcraft is extensible and new plugins to leverage existing technologies can be developed

Java, Python, ROS, Go, Maven, QMake, NodeJS, make, kernel are just a few examples of the languages and technologies that can be used.

We can reuse deb packages from ubuntu

Developer tools: Snapcraft

Snapcraft allows you to **init** your project creating a template:

```
liuxg@liuxg:~/snappy/demo
liuxg@liuxg:~/snappy/demo$ snapcraft --version
2.22.1
liuxg@liuxg:~/snappy/demo$ snapcraft init
Created snapcraft.yaml.
Edit the file to your liking or run `snapcraft` to get started
liuxg@liuxg:~/snappy/demo$ ls
snapcraft.yaml
liuxg@liuxg:~/snappy/demo$ cat snapcraft.yaml
name: my-snap-name # you probably want to 'snapcraft register <name>'
version: '0.1' # just for humans, typically '1.2+git' or '1.3.2'
summary: Single-line elevator pitch for your amazing snap # 79 char long summary
description: |
  This is my-snap's description. You have a paragraph or two to tell the
  most important story about your snap. Keep it under 100 words though,
  we live in tweetspace and your description wants to look good in the snap
  store.

grade: devel # must be 'stable' to release into candidate/stable channels
confinement: devmode # use 'strict' once you have the right plugs and slots

parts:
  my-part:
    # See 'snapcraft plugins'
    plugin: nilliuxg@liuxg:~/snappy/demo$
```

Snapcraft will look for **snapcraft.yaml** or **.snapcraft.yaml**, and will use either one, but it will error out if you have files both there

snapcraft example

```
name: hello
version: "1.0"
summary: The 'hello-world' of snaps
description: |
    This is a simple snap example that includes a few interesting binaries
    to demonstrate snaps and their confinement.

grade: stable
confinement: strict
type: app  # or gadget| kernel | core

apps:
 env:
   command: bin/env
 evil:
   command: bin/evil
 hello:
   command: hello

parts:
 hello:
  plugin: copy
  files:
    ./bin: bin
 gnu-hello:
  source: http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  plugin: autotools
```

If confined is defined as “**devmode**”, a snap cannot be uploaded to the “stable” channel, and you have to use the option “--devmode” to install it

Just run the app <package_name>.<app_name>, i.e, hello.evil

If the package name and the app name are the same, just run it by <package name>, i.e, **hello**

https://github.com/liu-xiao-guo/helloworld-demo

Snapcraft syntax

Creating aliases for snap apps

```
name: my-alias
version: '0.1'
summary: command alias example
description: |
  Command alias example
grade: devel
confinement: devmode

apps:
  hello:
    command: hello.sh
    aliases: [hi.sh, howdy.sh]

parts:
  aliases:
    plugin: dump
    source: .
```

```
$ sudo snap alias my-alias hi.sh

$ sudo snap alias --reset my-alias hi.sh

sudo snap unalias my-alias hi.sh
```

grade and confinement

	confinement: strict	confinement: devmode
grade: stable	*all* channels	beta and edge only
grade: devel	beta and edge only	beta and edge only

It's worth noting that the user of your snaps will have to use **--devmode** to install a snap using confinement: **devmode**. This means that they have to willingly accept that the snap is breaking out of confinement.

snapcraft.yaml (1/2)

Snapcraft part:

A central aspect of a snapcraft recipe is a "**part**". A part is a piece of software or data that the snap package requires to work or to build other parts. Parts' name can be anything you like.

Each part is managed by a snapcraft plugin and parts are usually independent of each other

```
parts:
  cam:
    source: git://github.com/mikix/golang-static-http
    plugin: go
    stage-packages:
      - fswebcam
  glue:
    plugin: copy
    files:
      webcam-webui: bin/webcam-webui
```

snapcraft.yaml (2/2)

Snapcraft plugin:

Snapcraft plugins are written in Python and have a yaml description.

A lot of default plugins are included, and they can be used for building different parts in the project. Custom plugins can be included in the tree alongside the snapcraft.yaml as well. It is also possible to simply download binary content as part of the snapcraft recipe.

```
parts:
  cam:
    source: git://github.com/mikix/golang-static-http
    plugin: go
    stage-packages:
      - fswebcam
  glue:
    plugin: copy
    files:
      webcam-webui: bin/webcam-webui
```

Snapcraft plugins

$ snapcraft list-plugins

ant	cmake	gradle	kbuild	maven	plainbox-provider	qmake
autotools	copy	gulp	kernel	nil	python2	scons
catkin	go	jdk	make	nodejs	python3	tar-content

Write your own plugins:

- https://developer.ubuntu.com/en/snappy/build-apps/plugins/
- http://blog.csdn.net/ubuntutouch/article/details/53157531

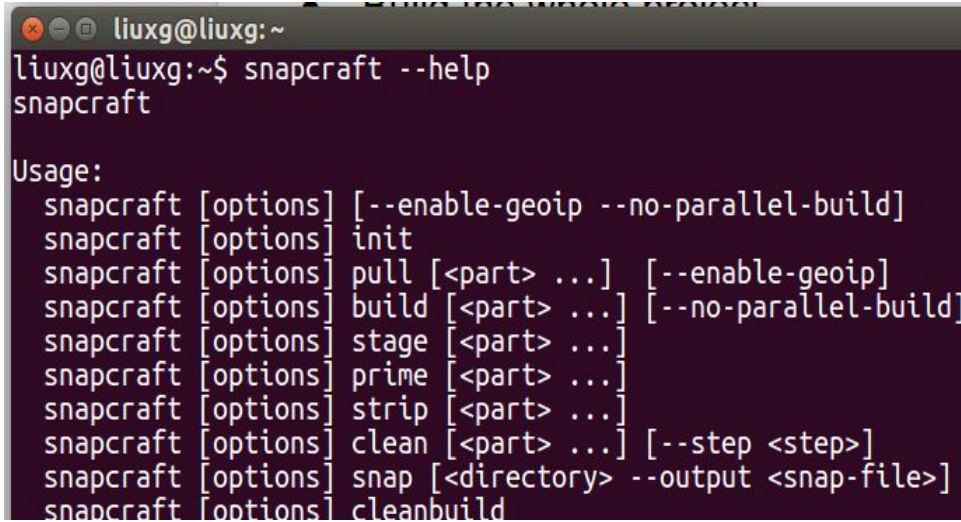
Custom plugin examples:

- https://github.com/ubuntu/snappy-playpen

Snapcraft commands

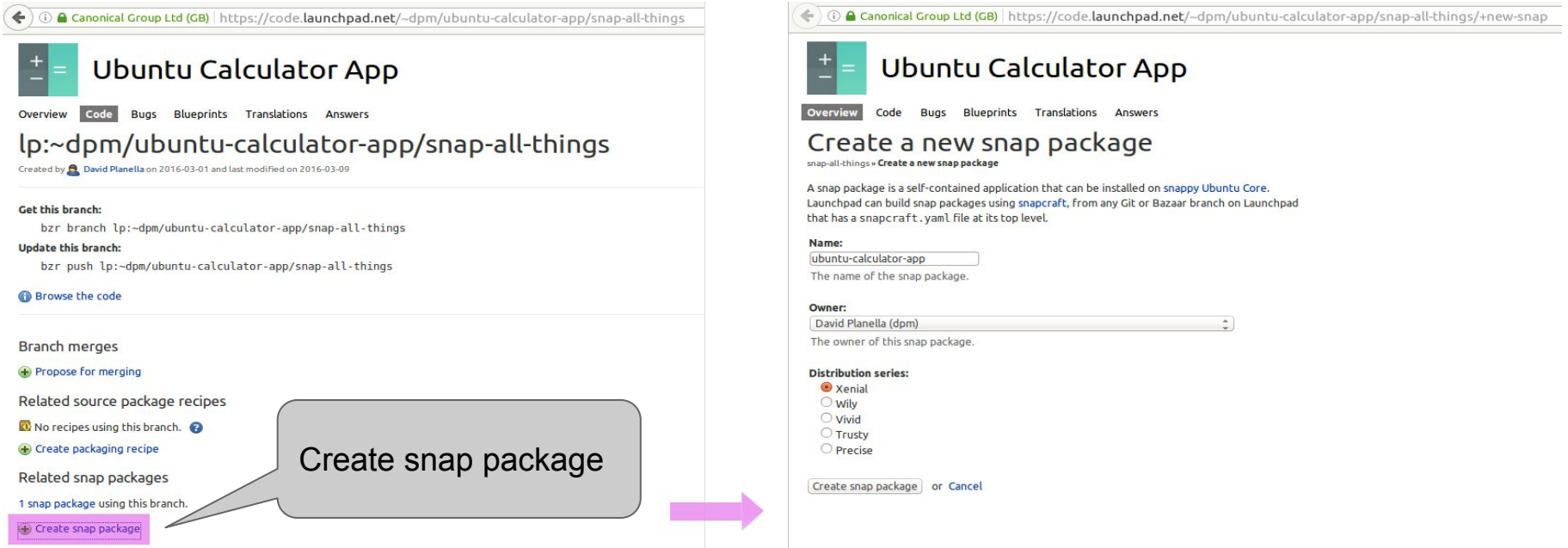
- Build the whole project
  - snapcraft
  - snapcraft clean build
- Clean the whole project
  - snacraft clean
  - Option **-s** <step> or **--step** <step>
  - snapcraft clean gnu-bash -s build
- Build each step in the life cycle
  - snapcraft pull
  - snapcraft build
  - snapcraft stage
  - snapcraft prime
  - snapcraft snap prime/
  - Each step depends on the prev.

- More commands
  - snapcraft --help
  - snapcraft help **python3**
  - snapcraft help **sources**



How to build your app for all architectures?

- Develop your application for one architecture and test it successfully, let's say amd64
- Create a project on Launchpad and make use of the services there
  - https://kyrofa.com/posts/building-your-snap-on-device-there-s-a-better-way
  - Click on the "**Create snap package**" button



Build for ARM architecture

- Log in to ARM boards
  - $ ssh

    ubuntu_sso_account@your_arm_board_ip_address
- Install “**classic**” snap application
  - $ sudo snap install classic --devmode --beta
- Create classic environment
  - $ sudo classic.create
  - $ sudo classic
  - $ sudo classic.reset
- Install build environment
  - $ sudo apt-get update
  - $ sudo apt install snapcraft git-core
- Build your snap project
  - Enter your project root dir, and type “**snapcraft**”

http://blog.csdn.net/ubuntutouch/article/details/52312246

Application types

```
name: webcam-webui
version: "1.0"
summary: Webcam web UI
description: Exposes your webcam over a web UI
confinement: strict
apps:
  webcam-webui:
    command: bin/webcam-webui
    daemon: simple
    plugs: [camera,network-bind]
```

https://github.com/liu-xiao-guo/webcam-snap

- If daemon is not present, it is simply an application
  - Run by <**package-name**>.<**application-name**>
  - If package name and app name are the same, just run it by <**package-name**> only
- If daemon is present:
  - “**simple**” if the “command” configured is the main process
  - “**forking**” if the configured “command” calls fork() as part of its start-up resulting in a different PID to track
  - Started when installed or refreshed. A daemon has “root” rights, but confinement still applies

How to check the status of a daemon?

- Check the status of a daemon

  $ systemctl status -l snap.webcam-webui.webcam-webui

- Stop a daemon

  $ sudo systemctl stop snap.webcam-webui.webcam-webui

  $ sudo service snap.<name>.<command> stop

- Start a daemon

  $ sudo systemctl start snap.webcam-webui.webcam-webui

  $ sudo service snap.<name>.<command> start

- Check the log

  $ journalctl -u snap.webcam-webui.webcam-webui

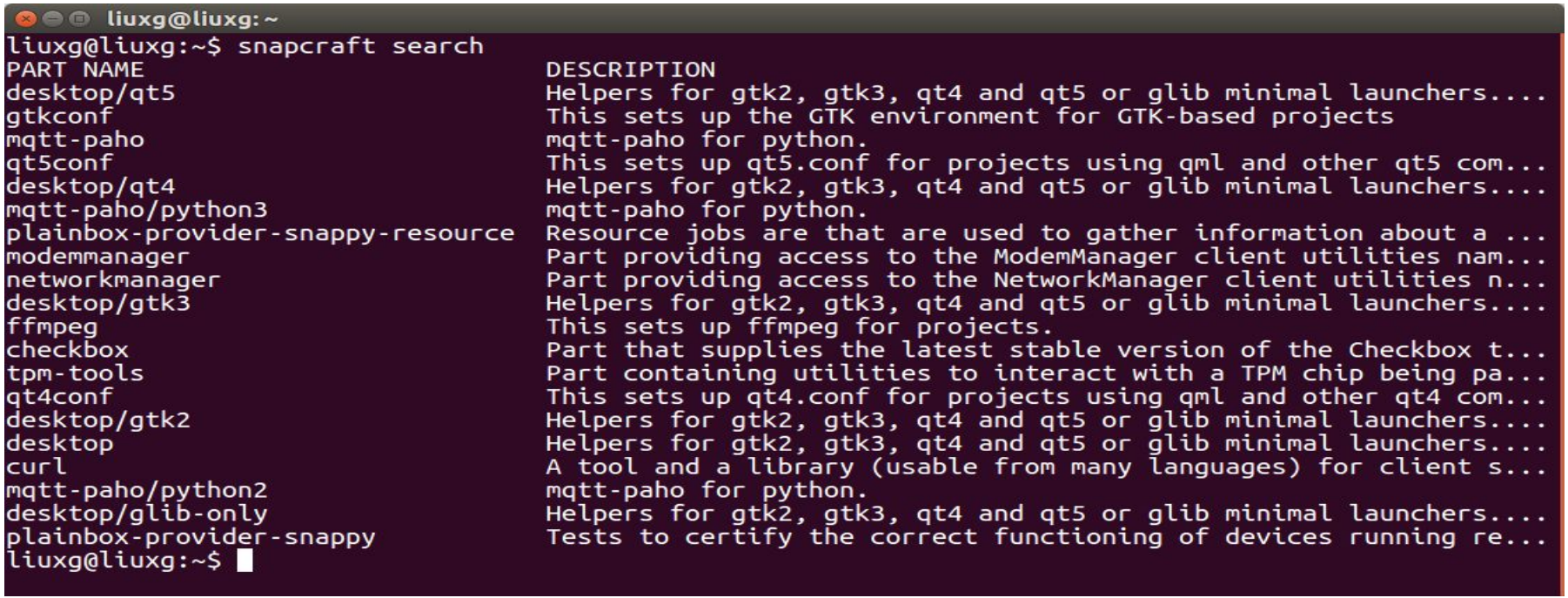
- Show the boot log

  $ journalctl -b

Tips: for daemon app, it would be good to make it a normal app first, then run it like: sudo <snap_app>, so we can see the output directly on the terminal

Use of remote parts

$ snapcraft update
$ snapcraft search



https://wiki.ubuntu.com/snapcraft/parts

Example of using the remote part

```
name: mosquitto
version: 0.1
summary: mosquitto server and client
description: MQTT example with a server, a publisher and a subscriber.
confinement: strict

apps:
  mosquitto:
    command: usr/sbin/mosquitto -c $SNAP/mosquitto.conf
    daemon: simple
    plugs: [network, network-bind]
  subscribe:
    command: bin/subscribe
    plugs: [network, network-bind]
  publish:
    command: bin/publish
    plugs: [network, network-bind]

parts:
  mosquitto:
    plugin: copy
    stage-packages: [mosquitto]
    files:
      mosquitto.conf: mosquitto.conf
  mqtt-client:
    plugin: copy
    files:
      subscribe.py: bin/subscribe
      publish.py: bin/publish
    after: [mqtt-paho-python3]
```

Prerequisites during build

```
name: downloader
version: 1.0
summary: curl based downloader
description: this is an example package
confinement: strict
apps:
   test:
       command: bin/test
       plugs: [network]

parts:
   main:
       plugin: make
       source: .
       after:
           - curl
       build-packages: [libssl-dev]
```

The above will install the **`libssl-dev`** package from the Ubuntu archive before an attempted build. Also note that the above will not define which libraries are shipped with the app. It merely makes sure you have all the relevant build tools installed

How to install Debian packages/pip packages?

snapcraft.yaml

```
...

parts:
  django:
    plugin: python2
    source: .
    stage-packages:
      - python-django
      - python-djangorestframework
      - python
      - libpython2.7-stdlib
```

```
$ snapcraft help python
```

snapcraft.yaml

```
...

parts:
  django:
    plugin: python2
    source: .
    requirements: ./requirements.txt
```

requirements.txt

```
Django==1.9
argparse==1.2.1
djangorestframework==3.3.1
wsgiref==0.1.2
```

How to specify build orders?

- **After**
  - Specifies any parts that should be built before this part is. This is mostly useful when a part needs a library or build tool built by another part

```
name: downloader
version: “1.0”
summary: curl based downloader
description: this is an example package
apps:
    test:
        command: bin/test
        plugs: [network]

parts:
    main:
        plugin: make
        source: .
        after:
            - curl
```

Snap configure

```
name: hello
version: "1.0"
summary: The 'hello-world' of snaps
description: |
    This is a simple snap example
type: app  #it can be gadget or framework

apps:
 env:
   command: bin/env

parts:
 config:
  plugin: dump
  source: .
  organize:
    configure: meta/hooks/configure
```

```
$ snap set <snap-name> username=foo password=bar
$ snap get <snap-name> username
```

Limiting number of installed files

To check the list of files included in your snap, you can use `unsquashfs -l` on the resulting `.snap` file. If you find that certain files should not be shipped to the user (download size being just one factor), you can explicitly tell `snapcraft` which files to snap:

```
cam:
  plugin: go
    go-packages:
      - github.com/mikix/golang-static-http
    stage-packages:
      - fswebcam
    filesets:
      fswebcam:
        - usr/bin/fswebcam
        - lib
        - usr/lib
      go-server:
        - bin/golang-*
    stage:
      - $fswebcam
      - $go-server
    snap:
      - $fswebcam
      - $go-server
      - -usr/share/doc
```

This is is useful to resolve a conflict when more than one part ships the same file to the same location in the snap. We can use **snap** and **stage** to ship only one from one of the parts. An example at: https://github.com/liu-xiao-guo/rssreader_snap/

This file will not be installed due to the second "**-**"

Create an application icon

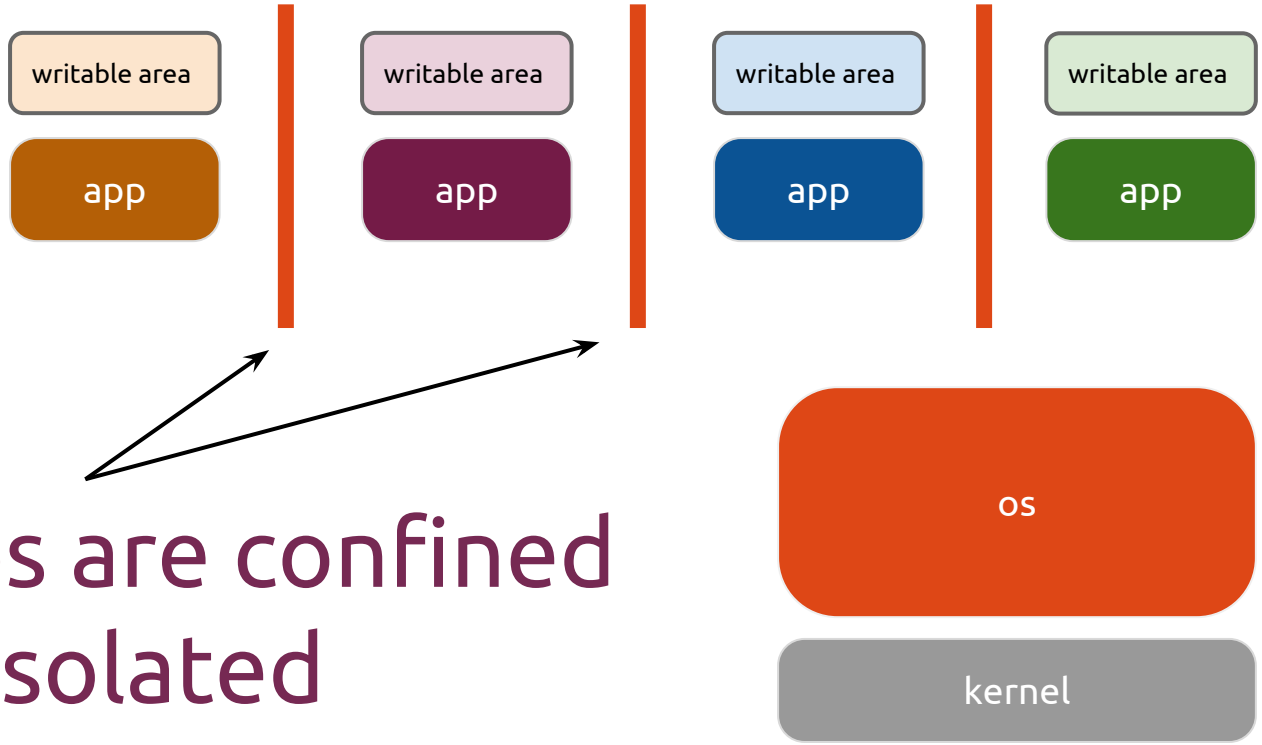
```
liuxg@liuxg: ~/snappy/desktop/rssreader

liuxg@liuxg:~/snappy/desktop/rssreader$ tree -L 3
.
├── setup
│   └── gui
│       ├── rssreader.desktop
│       └── rssreader.png
├── snapcraft.yaml
└── src
    ├── manifest.json.in
    ├── po
    │   └── rssreader.liu-xiao-guo.pot
    ├── rssreader
    │   ├── components
    │   ├── main.cpp
    │   ├── Main.qml
    │   ├── rssreader.apparmor
    │   ├── rssreader.desktop
    │   ├── rssreader.png
    │   ├── rssreader.pro
    │   ├── rssreader.qrc
    │   └── tests
    └── rssreader.pro

7 directories, 13 files
liuxg@liuxg:~/snappy/desktop/rssreader$
```

```
[Desktop Entry]
Type=Application
Name=RSS Reader
GenericName=RSS Reader
Comment=A RSS reader for Ubuntu Desktop
Keywords=rss;reader;
Exec=rssreader-app.rssreader
Icon=${SNAP}/meta/gui/rssreader.png
Terminal=false
X-Ubuntu-Touch=true
X-Ubuntu-Default-Department-ID=accessories
X-Ubuntu-Splash-Color=#F5F5F5
StartupNotify=true
```

Ubuntu Core Security



Snaps are confined and isolated

App specific writable directories

$ hello.env | grep SNAP

$ snap run --shell <snap>

Series 16	Description
SNAP (/snap/hello/x1)	The directory where the snap is mounted
SNAP_COMMON (/home/snap/hello/common)	A directory to store data common to **all revisions** of the snap. Write access requires **root** privileges
SNAP_DATA (/var/snap/hello/**x1**)	A directory to store data or configuration needed whenever the an app from the snap is run. Write access would require **root** privileges.
SNAP_NAME (hello)	The package name
SNAP_VERSION (1.0)	The package version
SNAP_ARCH (amd64)	The target device CPU architecture
SNAP_USER_COMMON (/home/<user_name>/snap/hello/common)	A directory to store data or configuration specific to the user. Always writeable.
SNAP_REVISION (x1)	the revision of the snap indicated by the store
SNAP_LIBRARY_PATH (/var/lib/snapd/lib/gl:)	directories that will be added to the LD_LIBRARY_PATH
SNAP_USER_DATA (/home/<user_name>/snap/hello/**x1**)	A directory to store data specific to the user for a specific version of the snap. Always writeable

Data directories

Two directories which the snap can write to **independent of the user.** Typically, configuration is stored in one of these, along with system-wide data for the snap:

```
/var/snap/<app_name>/current/    ← $SNAP_DATA is the versioned snap data directory
/var/snap/<app_name>/common/  ← $SNAP_COMMON will not be versioned on upgrades
```

There are also an equivalent two writable directories for each snap in the user home, which can be used to store snap data that is **specific to one user or another**, separately:

```
/home/<user_name>/snap/<app_name>/current/      ← $SNAP_USER_DATA that can be rolled back
/home/<user_name>/snap/<app_name>/common/     ← $SNAP_USER_COMMON unversioned
user-specific data
```

Ubuntu Core Interfaces

Used to grant access to resources, e.g. to give permission to do something or to control something

Snaps can either offer (implement) or use (consume) interfaces

Interfaces are a key component in secure and flexible access to features implemented in a system or third party snaps

- **Resource interfaces**: physical devices or similar.
  `front-camera, green-led...`
- **Behaviour interfaces**: associated with the demeanour of the snap
  `network-listening, timezone-managing...`

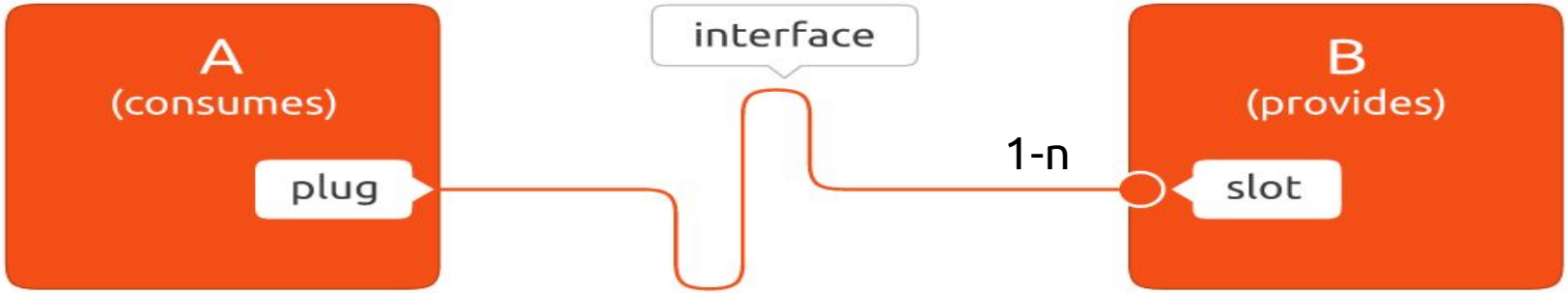
http://www.zygoon.pl/2016/08/creating-your-first-snappy-interface.html

snapcraft.yaml

```
name: webcam-webui
version: "1.0"
summary: Webcam web UI
description: Exposes your webcam
over a web UI
confinement: strict

apps:
  webcam-webui:
    command: bin/webcam-webui
    daemon: simple
    plugs: [camera,network-bind]
```

Snap integration with interfaces: plugs and slots



These two snaps can be connected because they have a plug and a slot with a matching interface.

Specifying a plug essentially pokes a hole in the otherwise-complete confinement

$ snap connect/disconnect <snap>:<plug interface> <snap>:<slot interface>
$ sudo snap connect webcam-webui:camera ubuntu-core:camera

$ snap interfaces <snap> # to find the slots offered and plugs used by the specified snap
$ snap interfaces <snap>:<slot or plug> # for details of only the specified slot or plug
$ snap interfaces -i=<interface> [<snap>] # to get a filtered list of plugs and/or slots

Ubuntu Core Interfaces

Check the interfaces on your Ubuntu Core system

```
liuxg@liuxg:~
liuxg@liuxg:~$ snap interfaces
Slot                  Plug
:camera               webcam-webui
:cups-control         -
:firewall-control     -
:gsettings            -
:home                 hello
:locale-control       -
:log-observe          -
:modem-manager        -
:mount-observe        -
:network              -
:network-bind         webcam-webui
:network-control      -
:network-manager      -
:network-observe      -
:opengl               hello,ubuntu-calculator-app
:optical-drive        -
:ppp                  -
:pulseaudio           -
:snapd-control        -
:system-observe       -
:timeserver-control   -
:timezone-control     -
:unity7               hello,ubuntu-calculator-app
:x11                  -
liuxg@liuxg:~$
```

Debugging snaps

Policy violations

To check whether there is a policy violation, supposing a snap called “**audit**”
$ **sudo grep audit /var/log/syslog**

An **AppArmor** violation looks like:

```
audit: type=1400 audit(1431384420.408:319): apparmor="DENIED" operation="mkdir"
profile="snap.foo.bar" name="/var/lib/foo" pid=637 comm="bar" requested_mask="c"
denied_mask="c" fsuid=0 ouid=0
```

A **seccomp** violation will look something like:

```
audit: type=1326 audit(1430766107.122:16): auid=1000 uid=1000 gid=1000 ses=15 pid=1491
comm="env" exe="/bin/bash" sig=31 arch=40000028 syscall=983045 compat=0 ip=0xb6fb0bd6
code=0x0
```

The syscall=983045 can be resolved with the scmp_sys_resolver command

```
$ scmp_sys_resolver 983045
set_tls
```

How to debug a snap?

$ sudo snap install snappy-debug

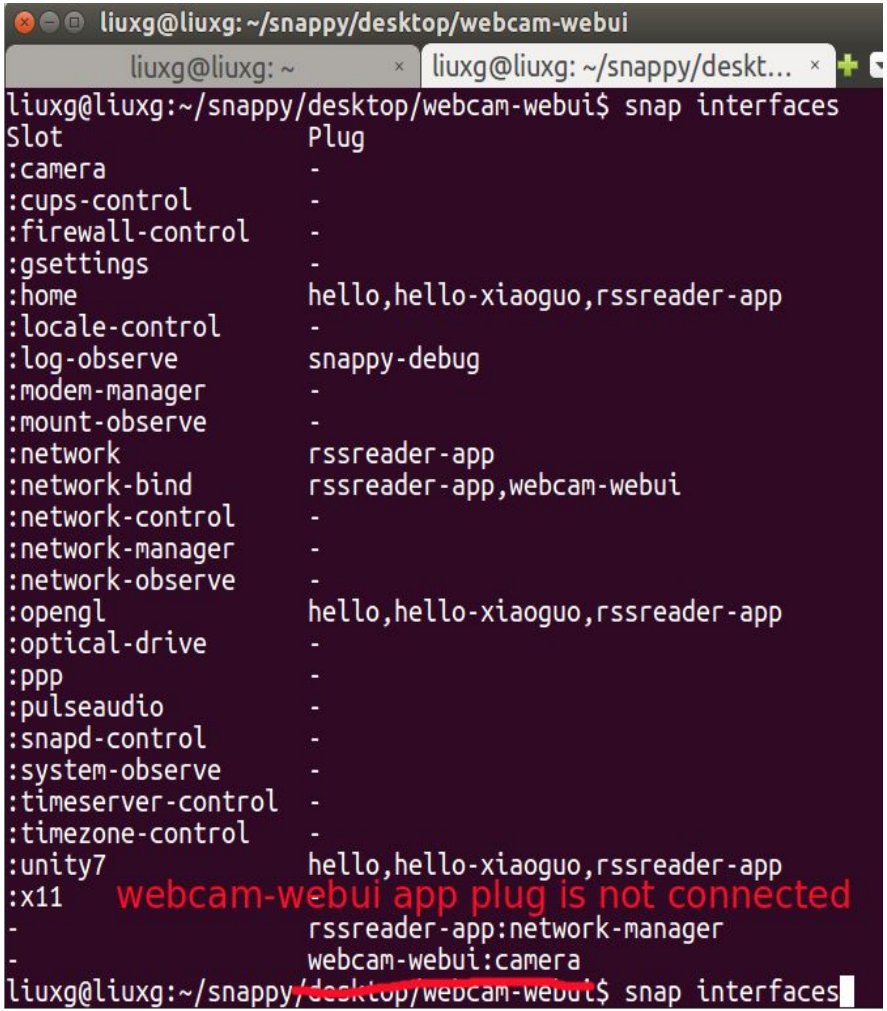
$ sudo snap connect snappy-debug:log-observe ubuntu-core:log-observe

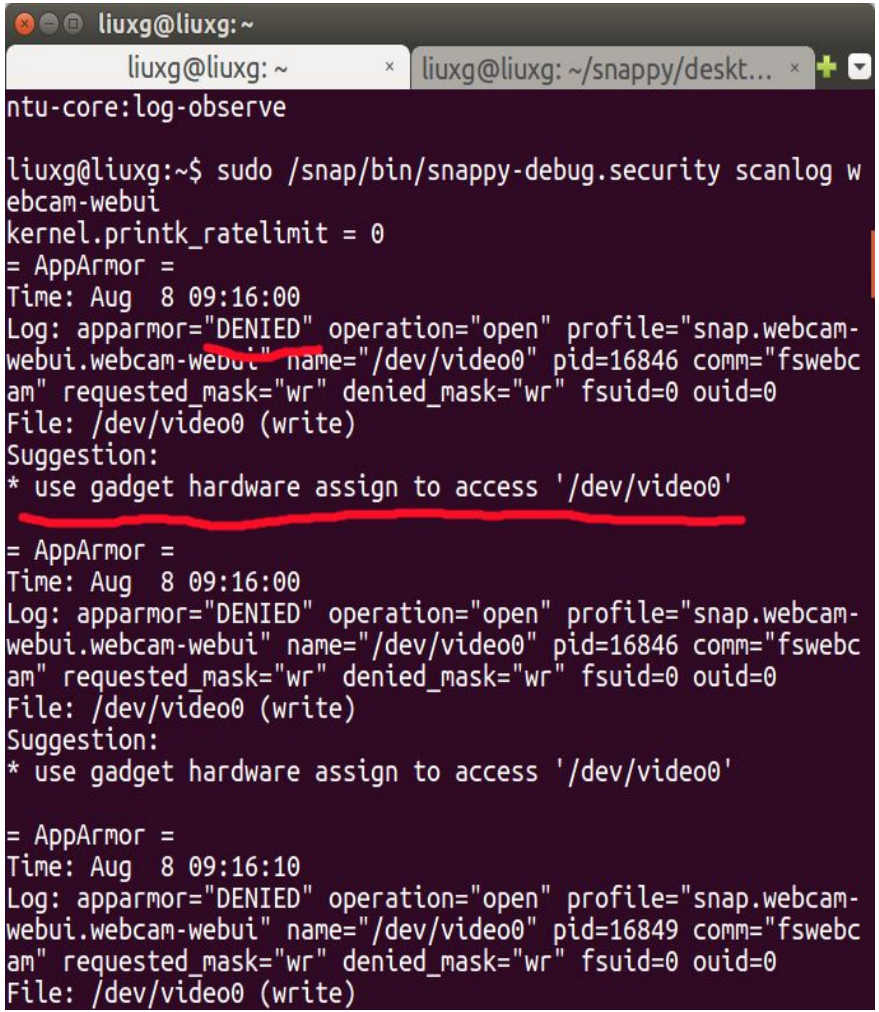
Before running your snap, in a terminal, run

$ sudo /snap/bin/snappy-debug.security scanlog foo

This will:

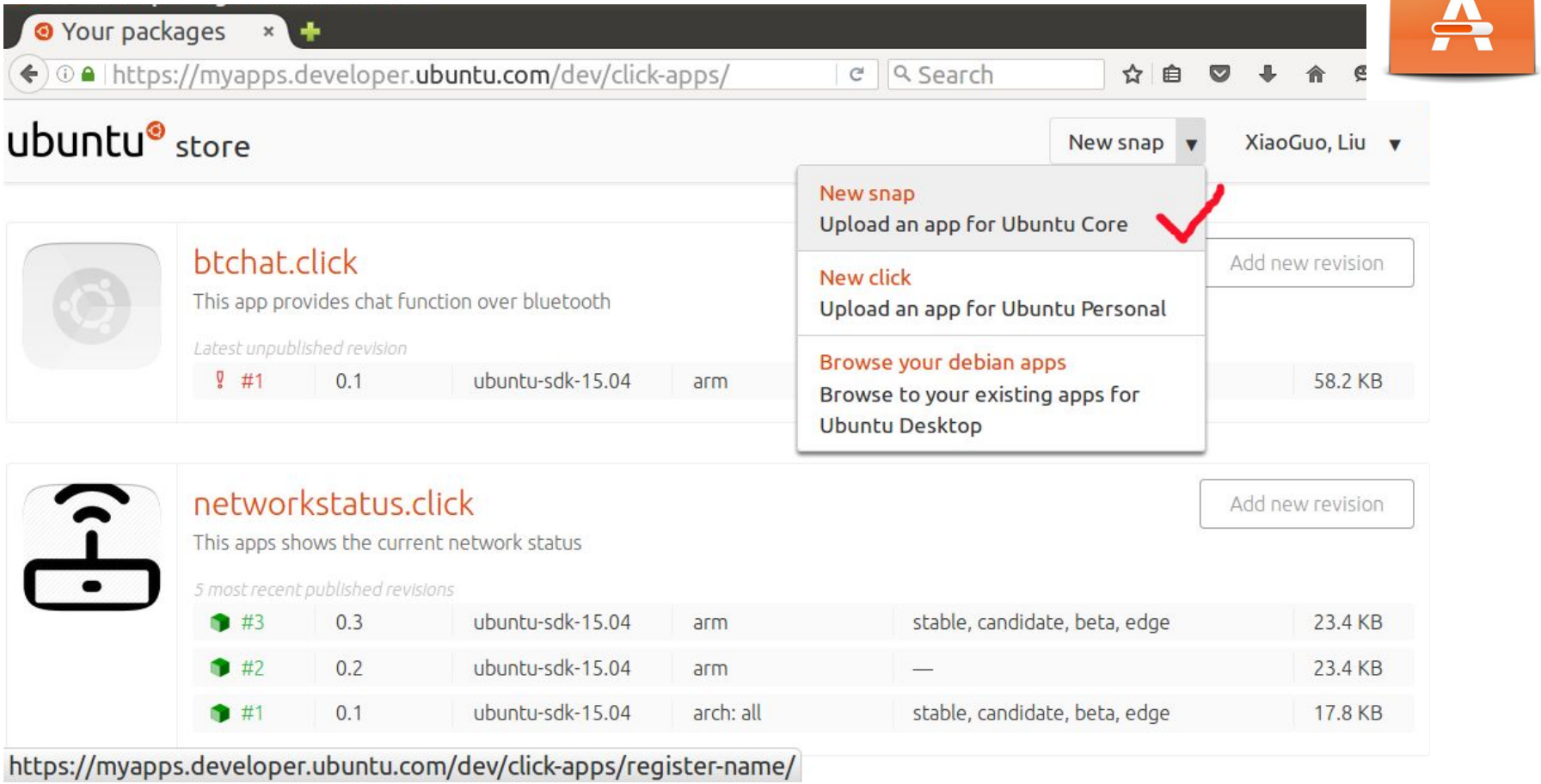
- Adjust kernel log rate limiting
- Follow **/var/log/syslog** looking for policy violations for foo
- Resolve syscall names
- Make recommendations on how to fix violations


liuxg@liuxg: ~/snappy/desktop/webcam-webui
liuxg@liuxg: ~
liuxg@liuxg: ~/snappy/deskt...
liuxg@liuxg:~/snappy/desktop/webcam-webui$ snap interfaces
Slot Plug
:camera -
:cups-control -
:firewall-control -
:gsettings -
:home hello,hello-xiaoguo,rssreader-app
:locale-control -
:log-observe snappy-debug
:modem-manager -
:mount-observe -
:network rssreader-app
:network-bind rssreader-app,webcam-webui
:network-control -
:network-manager -
:network-observe -
:opengl hello,hello-xiaoguo,rssreader-app
:optical-drive -
:ppp -
:pulseaudio -
:snapd-control -
:system-observe -
:timeserver-control -
:timezone-control -
:unity7 hello,hello-xiaoguo,rssreader-app
:x11
webcam-webui app plug is not connected
- rssreader-app:network-manager
- webcam-webui:camera
liuxg@liuxg:~/snappy/desktop/webcam-webui$ snap interfaces

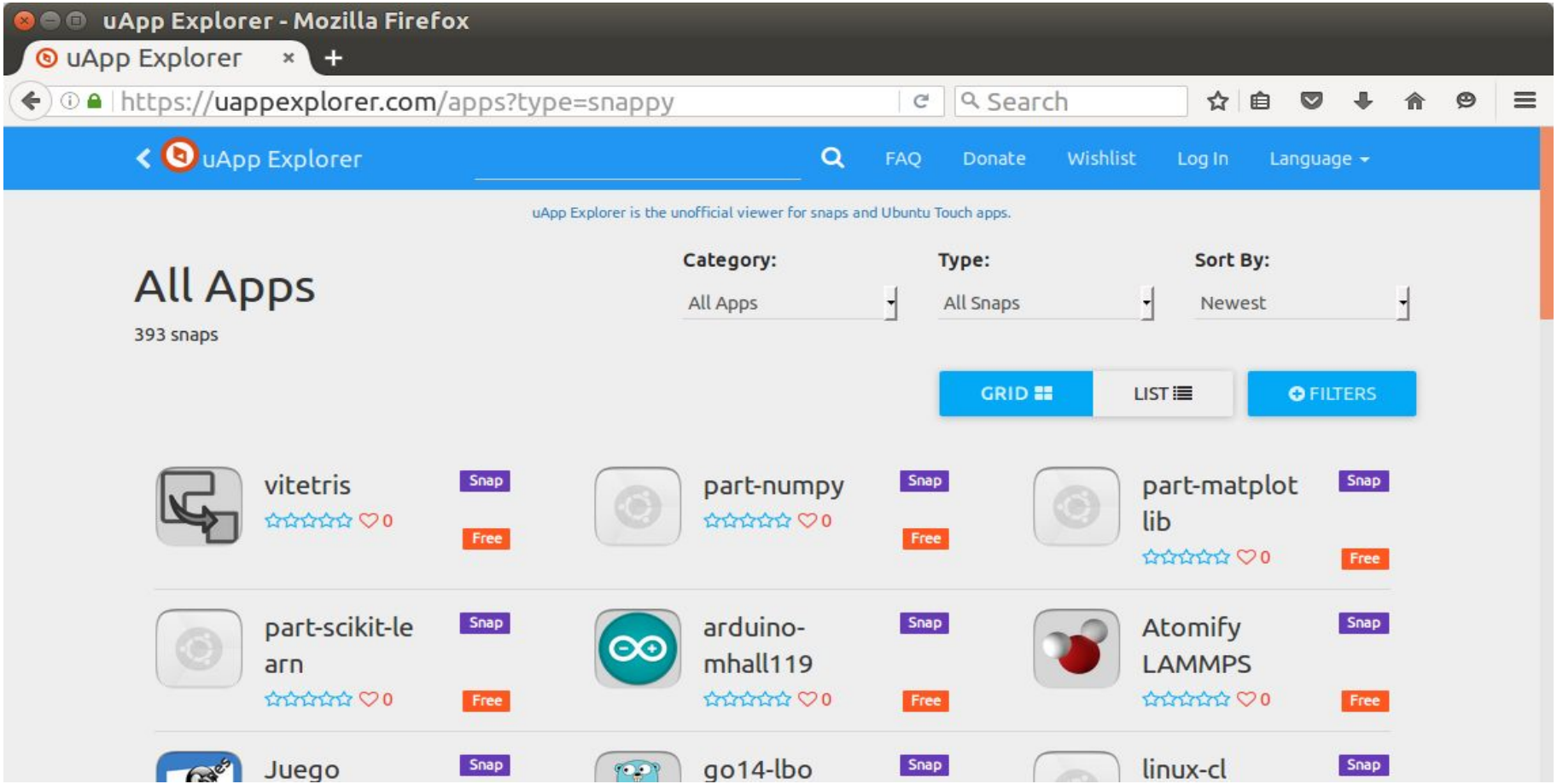

liuxg@liuxg: ~
liuxg@liuxg: ~
liuxg@liuxg: ~/snappy/deskt...
ntu-core:log-observe
liuxg@liuxg:~$ sudo /snap/bin/snappy-debug.security scanlog webcam-webui
kernel.printk_ratelimit = 0
= AppArmor =
Time: Aug 8 09:16:00
Log: apparmor="DENIED" operation="open" profile="snap.webcam-webui.webcam-webui" name="/dev/video0" pid=16846 comm="fswebcam" requested_mask="wr" denied_mask="wr" fsuid=0 ouid=0
File: /dev/video0 (write)
Suggestion:
* use gadget hardware assign to access '/dev/video0'
= AppArmor =
Time: Aug 8 09:16:00
Log: apparmor="DENIED" operation="open" profile="snap.webcam-webui.webcam-webui" name="/dev/video0" pid=16846 comm="fswebcam" requested_mask="wr" denied_mask="wr" fsuid=0 ouid=0
File: /dev/video0 (write)
Suggestion:
* use gadget hardware assign to access '/dev/video0'
= AppArmor =
Time: Aug 8 09:16:10
Log: apparmor="DENIED" operation="open" profile="snap.webcam-webui.webcam-webui" name="/dev/video0" pid=16849 comm="fswebcam" requested_mask="wr" denied_mask="wr" fsuid=0 ouid=0
File: /dev/video0 (write)

Ubuntu Store

Ubuntu Store

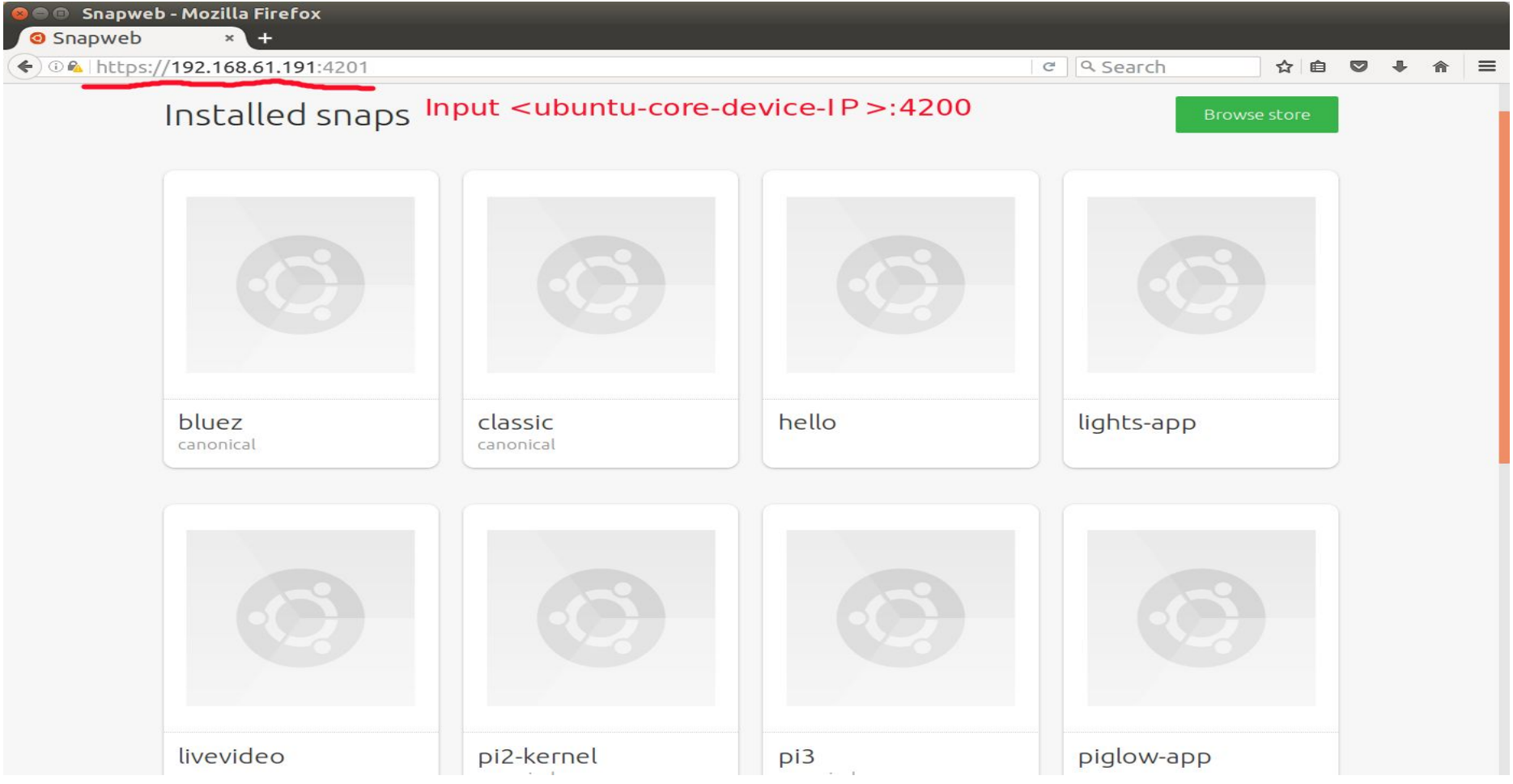


Where to find the published snaps?

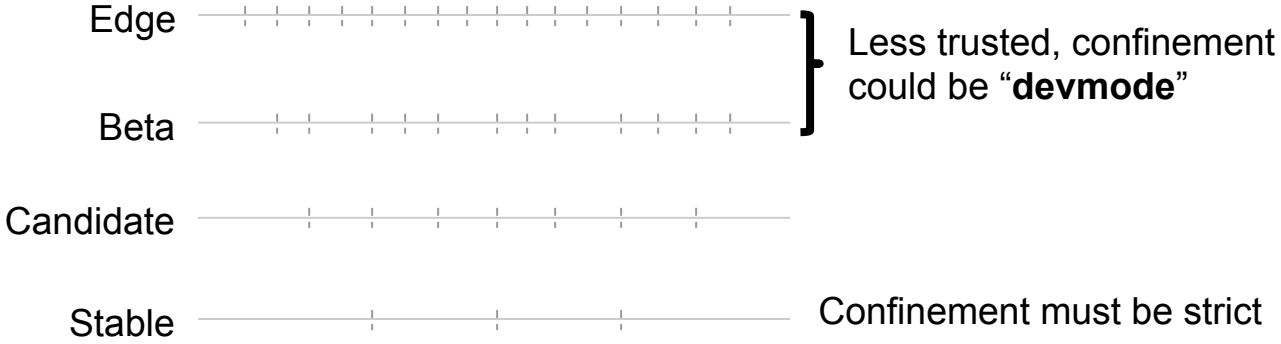


Snapweb

$ sudo snap install snapweb



Store and channels



Release	'edge' channel	'beta' channel	'candidate' channel	'stable' channel
16	Only suitable for developers tracking the latest development	Beta milestones	Release Candidate milestones	Rock-solid, to be shipped to device customers

Publish your apps to the store

Publish apps via commands:
myapps.developer.ubuntu.com
namespace

$ snapcraft login
$ snapcraft register <app_name>
$ snapcraft push <your_snap_file>
$ snapcraft release <snap> 1 stable
$ snapcraft logout

```
~/hello$ snapcraft push hello-didrocks_2.10_amd64.snap --release=beta
Uploading hello-didrocks_2.10_amd64.snap.
Uploading hello-didrocks_2.10_amd64.snap [==============================] 100%
```

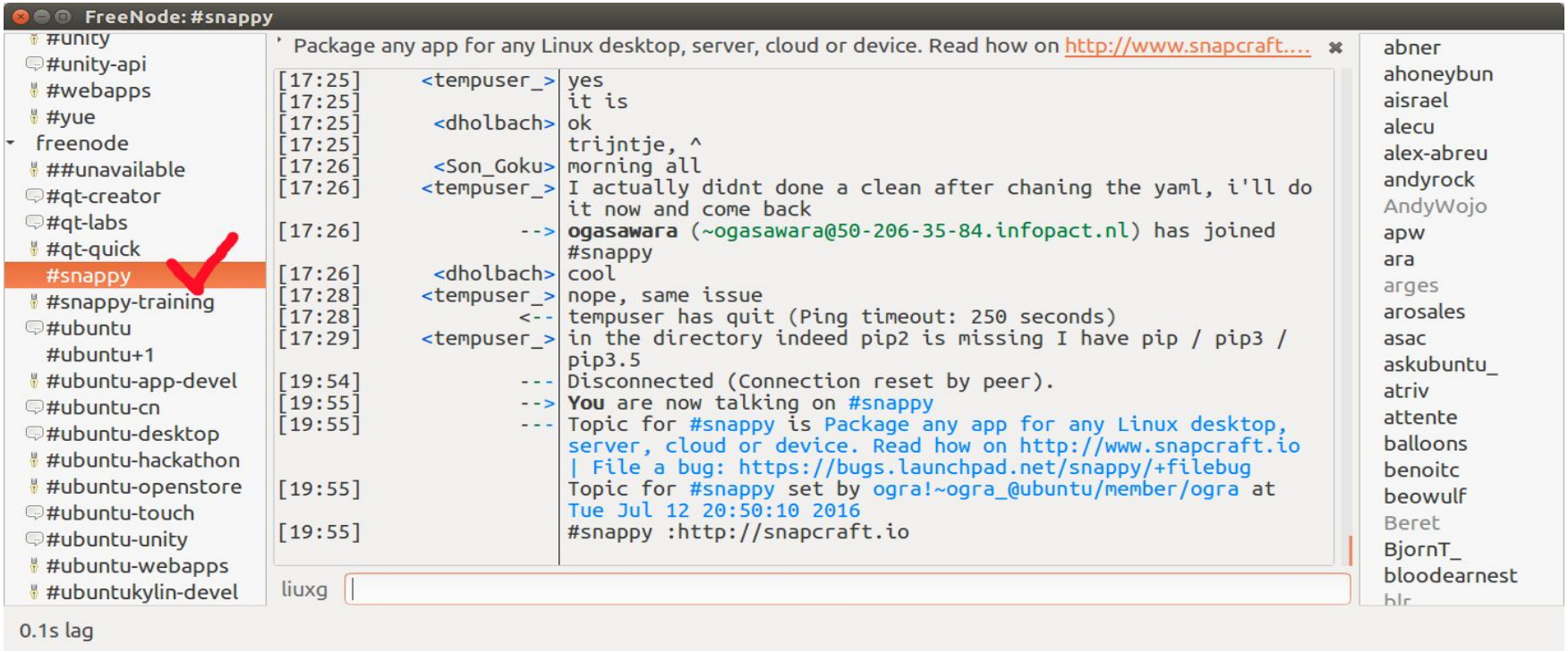
http://blog.csdn.net/ubuntutouch/article/details/53007440

Getting involved

Ubuntu Core: Getting involved

- **Ask a question on Ask Ubuntu**
  - If you're stuck on a problem, someone else has probably encountered it too and they can help you. Take a look at the "ubuntu-core" tag on Ask Ubuntu or ask a question.
- **Join our real time chat** (#snappy on freenode.net)
  - Share your projects and ask other developers for support. This high-bandwidth IRC channel is a good place when you are looking for a quick answer to a single question.
- **For app developers**
  - Reach out to other snap developers by using the"snapcraft" tag on Ask Ubuntu, join the snapcraft mailing list and make sure to join the Ubuntu App Developers Google+ community.
- **Ubuntu Core Clinics**
  - Our Ubuntu Core Clinics are video sessions where we want to inform you about what's new, show some demos, answer questions and where you can bring a problem and we are going to look at it together. Watch the sessions which already happened and check out which new ones are planned.
- **Ubuntu Core play-pen**
  - https://gitter.im/ubuntu/snappy-playpen

Ubuntu Core technical support



Support mailinglist: snappy-app-devel@lists.ubuntu.com, snapcraft@lists.snapcraft.io
Community: http://snapcraft.io/community/

References

References

- https://github.com/snapcore/snapd
- https://github.com/snapcore/snapcraft
- http://snapcraft.io/
- http://www.ubuntu.com/desktop/snappy
- https://developer.ubuntu.com/en/snappy/
- http://www.ubuntu.com/desktop/snappy
- http://blog.csdn.net/ubuntutouch -- my blog
- https://github.com/liu-xiao-guo?tab=repositories
- File a bug at https://bugs.launchpad.net/snapcraft/+filebug
- snappy mailing list: snappy-app-devel@lists.ubuntu.com
- http://www.ubuntu.com/internet-of-things
- http://www.ubuntu.com/desktop/snappy
- https://developer.ubuntu.com/en/desktop/examples/#snap-qt
- https://github.com/ubuntu/snappy-playpen
- http://snapcraft.io/create/#tour
- https://github.com/ubuntu/codelabs $sudo snap install snap-codelabs
- http://docs.ubuntu.com/core/en/

Ubuntu Core

Q&A

http://snapcraft.io/