Efficient Usage of Debian on Embedded Devices

OSADL Networking Day, 5. June 2019

Matthias Lüscher
lueschem@gmail.com

Initial Position

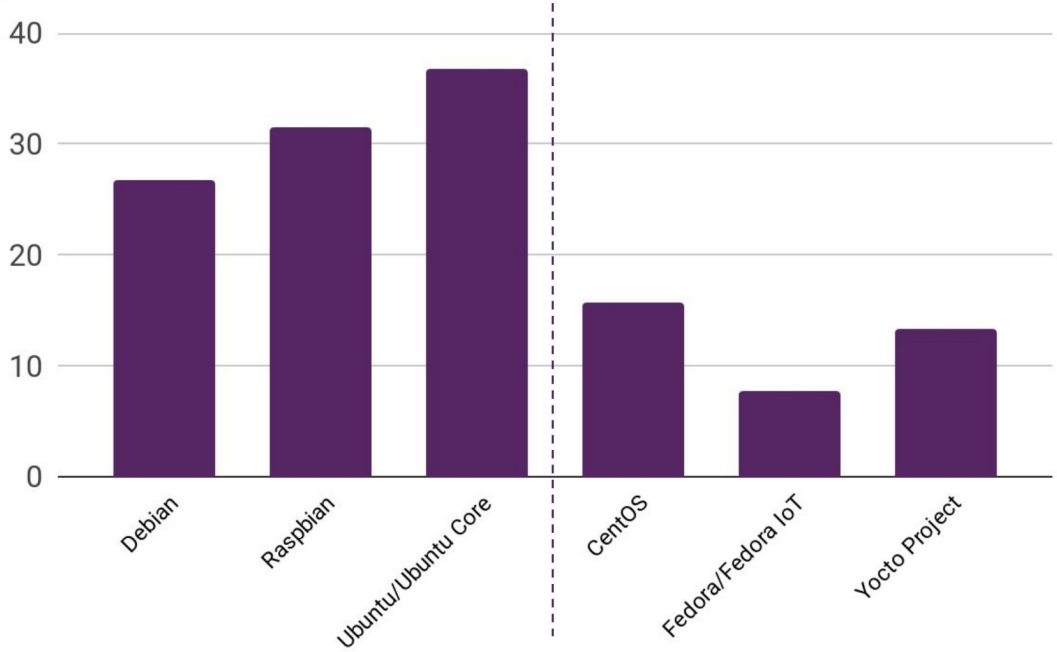
IoT – It’s a Debian world...

Key takeaway 7

2019 IoT SURVEY

Linux distributions

It’s a Debian World...



Debian and derivatives (Raspbian, Ubuntu / Ubuntu Core) were picked by at least **a third** of respondents.

CentOS & Fedora / Fedora IoT *came in second place, with a strong showing by* ***Yocto***

Source: https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf

… dominated by ARM and Intel hardware ...

Key takeaway 9

Hardware architectures used for IoT gateways

70%

Use gateways and edge nodes with **ARM Variants**

42%

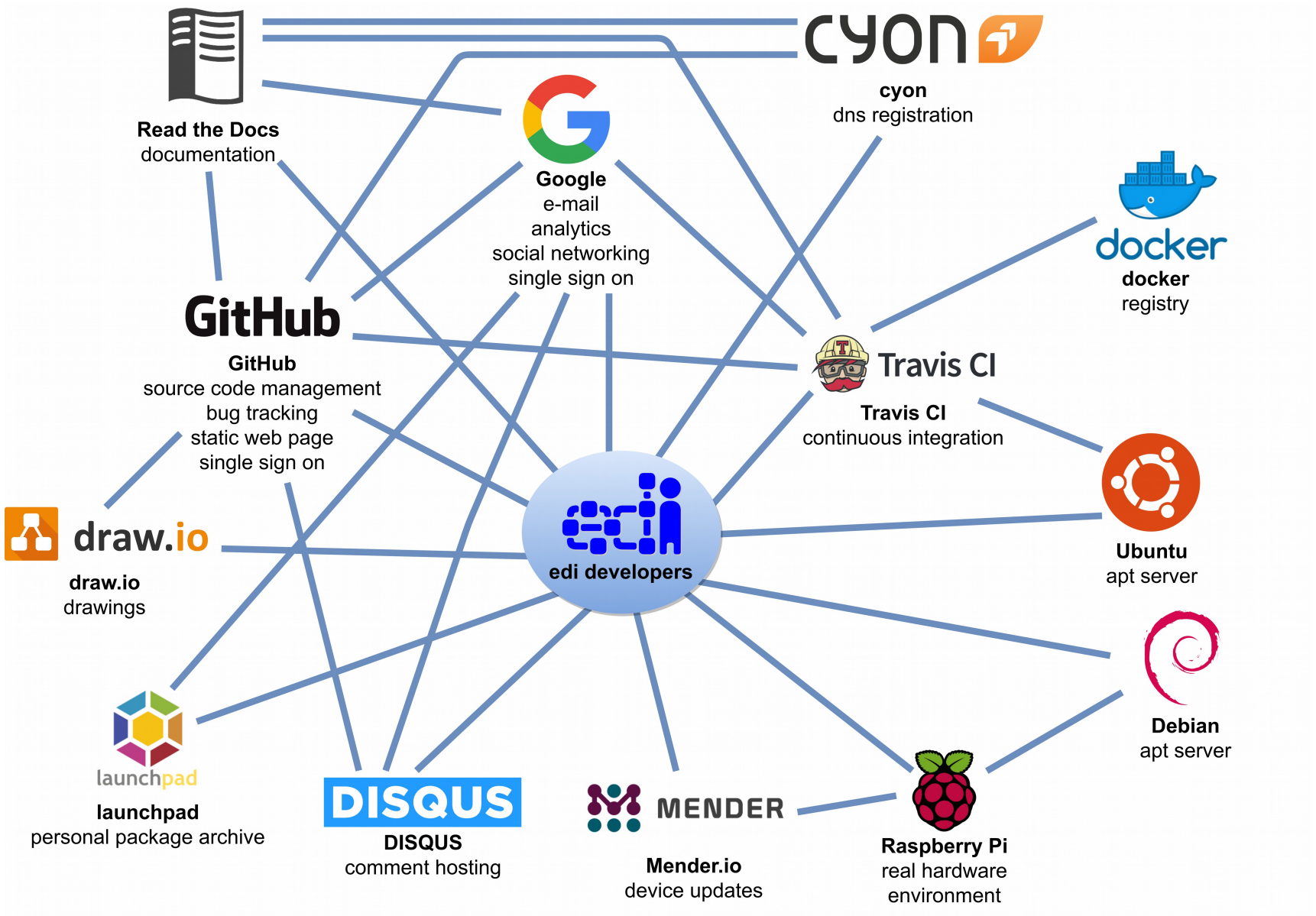
Use gateways and edge nodes with **Intel x86 and x86_64** CPUs

ARM and Intel Dominate

iot eclipse.org

Source: https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf

… requiring a lot of infrastructure!



See also: https://www.get-edi.io/Infrastructure-at-Your-Fingertips/

Presentation Content

A collection of best practices on how you can efficiently use Debian in such an environment.

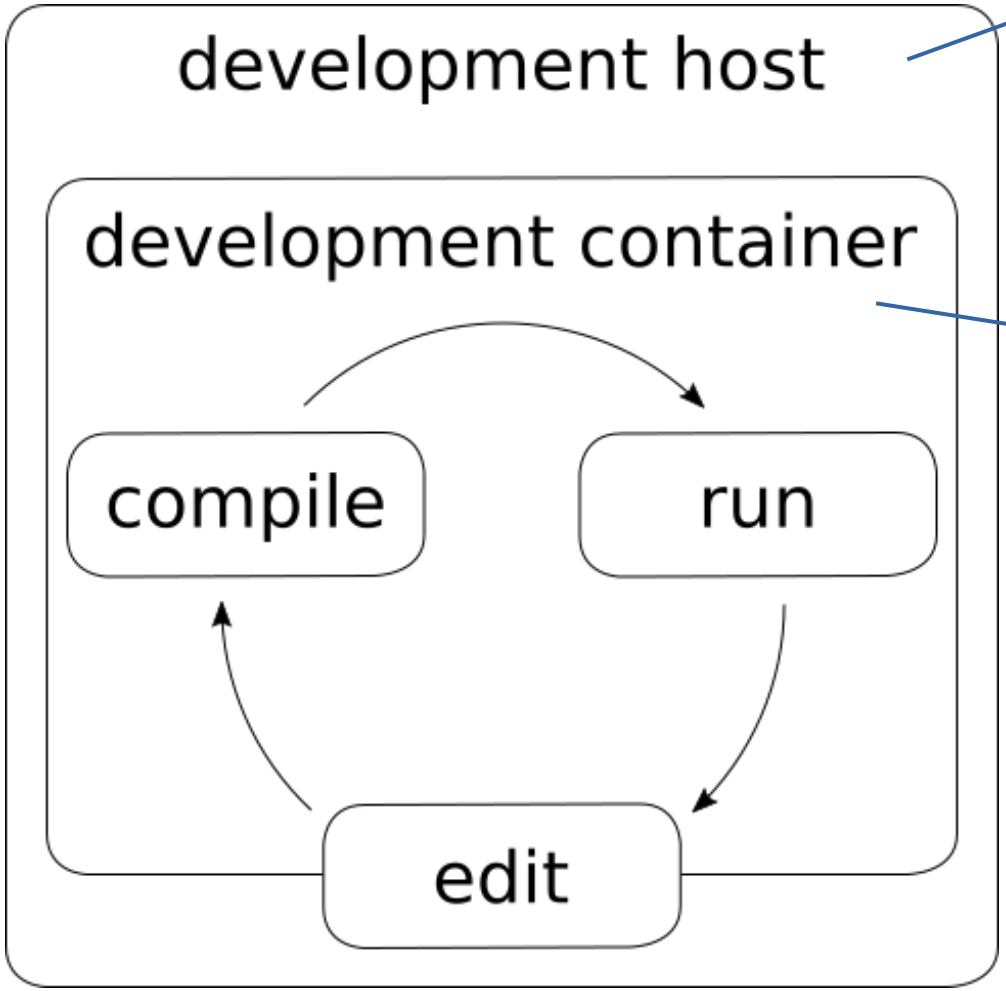
An introduction of the tool edi and some insights on how it will speed up your development process (with a special attention to the build of OS images).

https://www.get-edi.io

Development Setup

Create a digital twin of your target hardware!

(And do 95% of the development without the target hardware.)



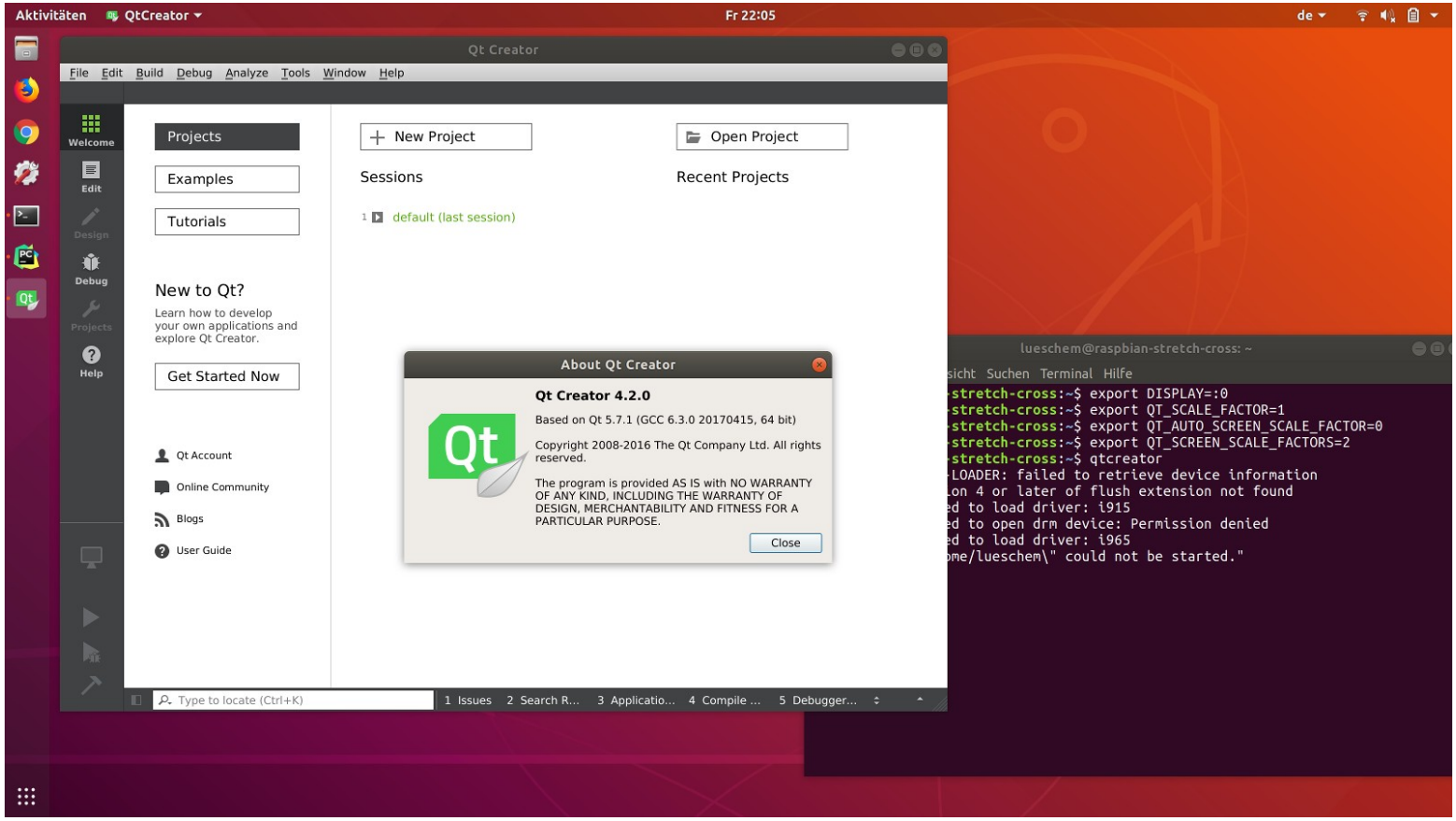
e.g. Ubuntu 18.04 LTS

- running on amd64 hardware
- equipped with LXD

Digital Twin of Target Hardware

- same Debian release as target
- same configuration as target
- same interfaces as target (network, serial, etc.)
- but amd64 architecture
- running as OS container
- ready for cross compilation
- shared folder with host OS
- generated by a single command: `edi -v lxc configure ...`

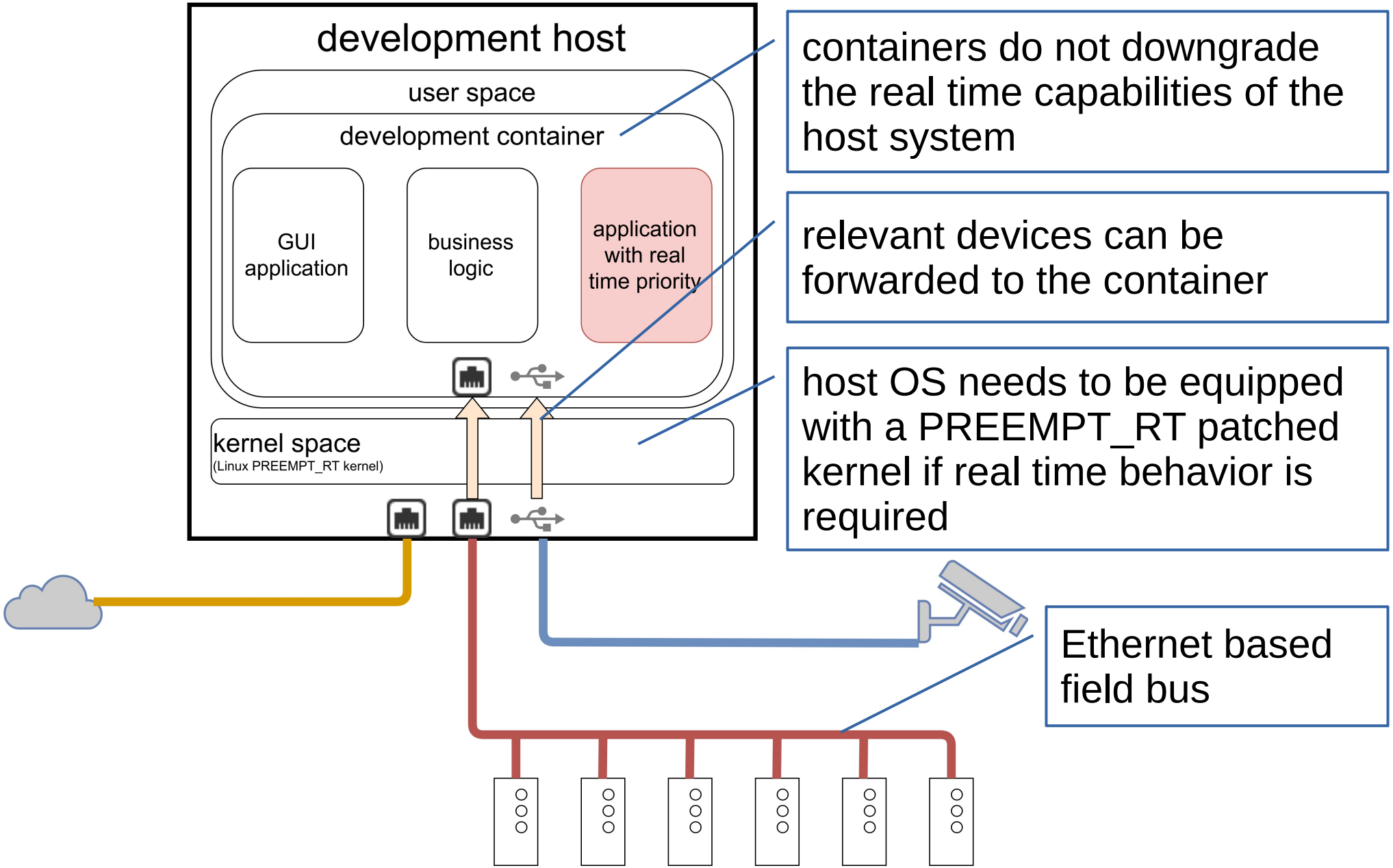
Integrate your favorite IDE!



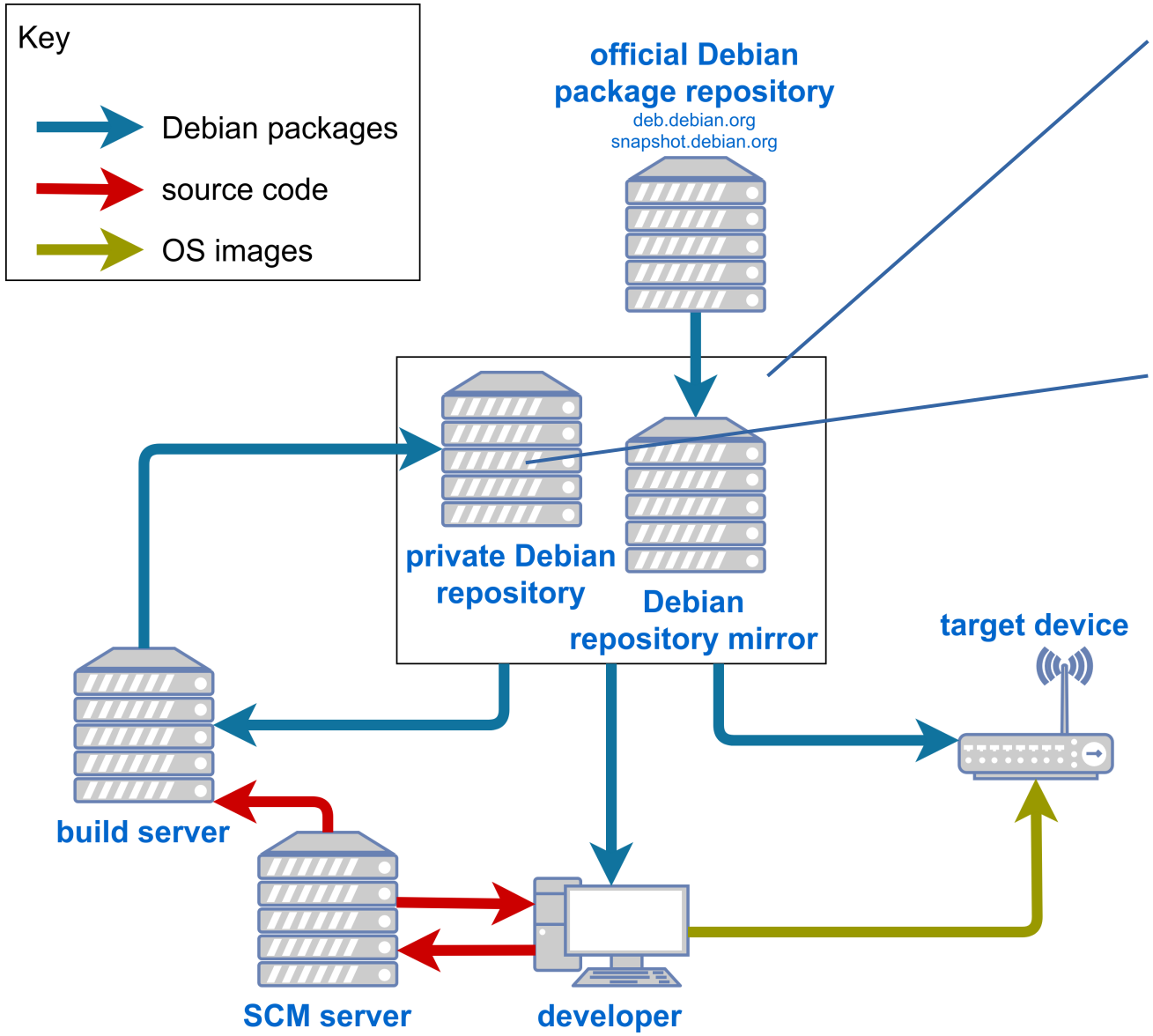
- You can choose whatever IDE you like.
- To improve the overall handling, it is advisable to run the IDE within the development container.

More information: http://www.get-edi.io/Running-GUI-Applications-Within-LXD-Container/

Develop (real time) applications!



Do not forget to setup a package repository!



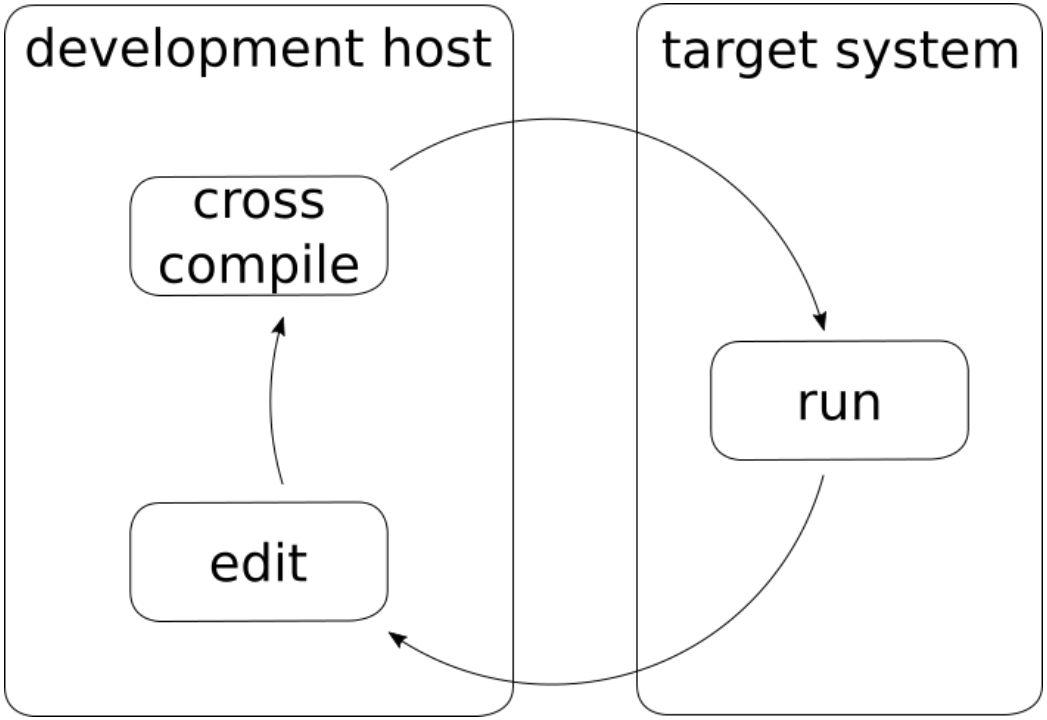
the Debian package repository is an indispensable part of your development infrastructure

do the extra mile and package your software as proper Debian packages

repository hosting solutions:
https://www.aptly.info/
https://jfrog.com/artifactory/
https://packagecloud.io/
...

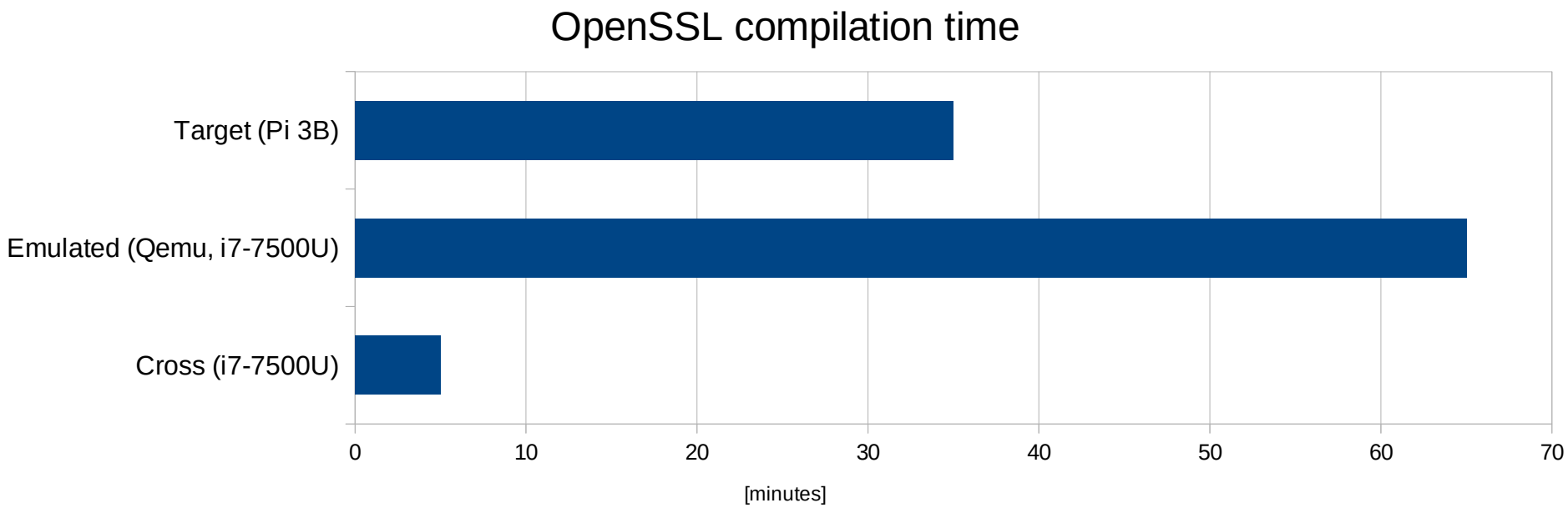
Cross Compilation

What is cross compilation?



- Wikipedia: A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.
- Easy: Kernel (no dependencies, well prepared for cross compilation)
- Less easy: Libraries and executables (dependencies, maybe not cross compilation aware)

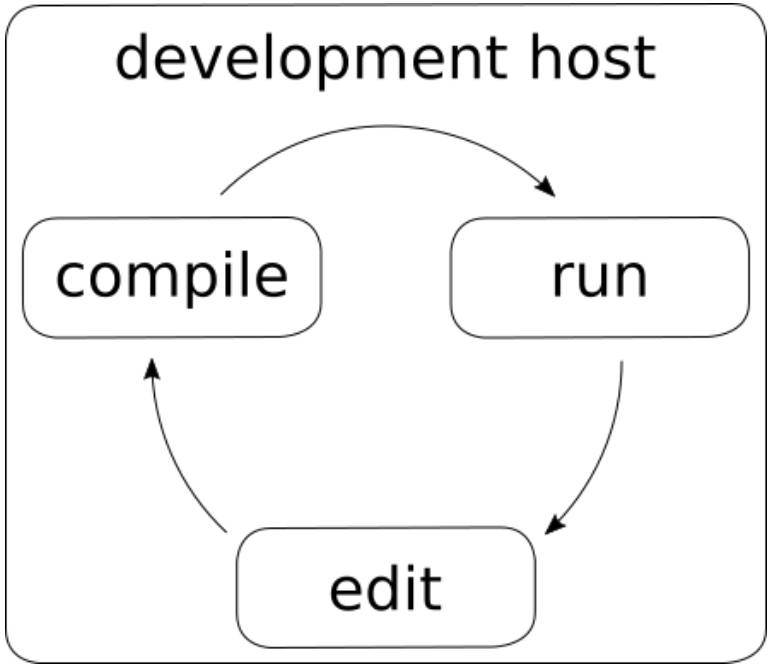
Why cross compile?

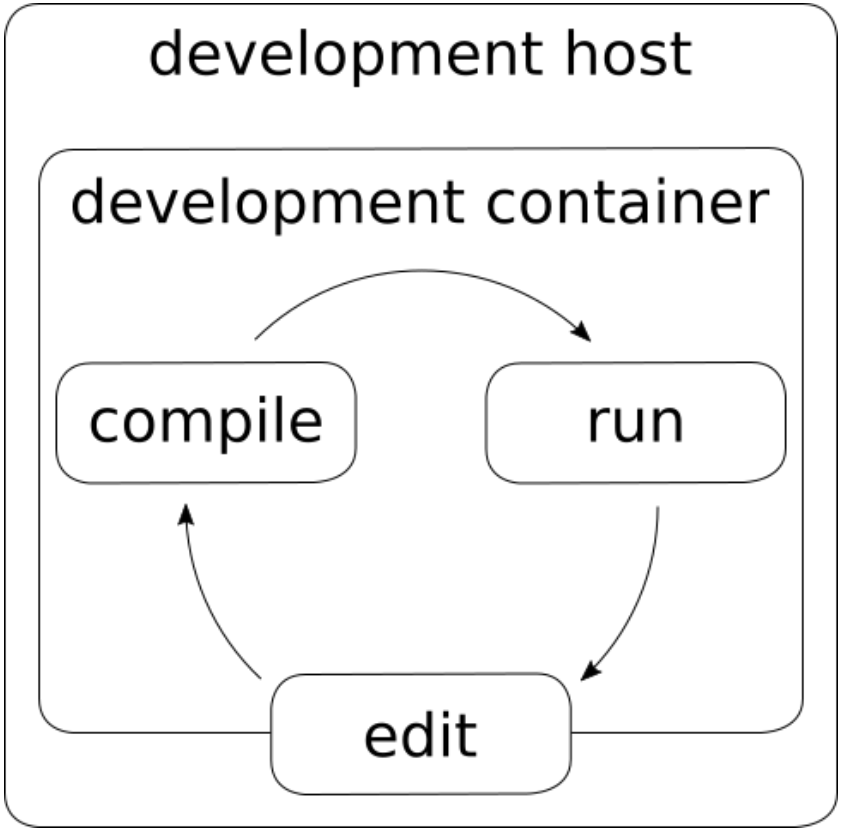


- Speed: Cross compilation is a lot faster!
- Flash: The target system might not have enough flash for compilation.
- Memory: The target system might run out of memory during compilation.

More information: http://www.get-edi.io/Compiling-for-Embedded-Debian-Target-Systems/

How should I cross compile with Debian?





- Environment is “self contained”: For Debian stretch you build within Debian stretch.
- The Debian project does build ARM packages on ARM hardware.
- For a long time Debian was not well suited for cross compilation.

What has changed recently?

- Debian got broadly adopted for embedded devices.
- Multiarch and multilib got introduced in Debian wheezy: You can add a foreign architecture and install libraries and foreign libraries side by side:

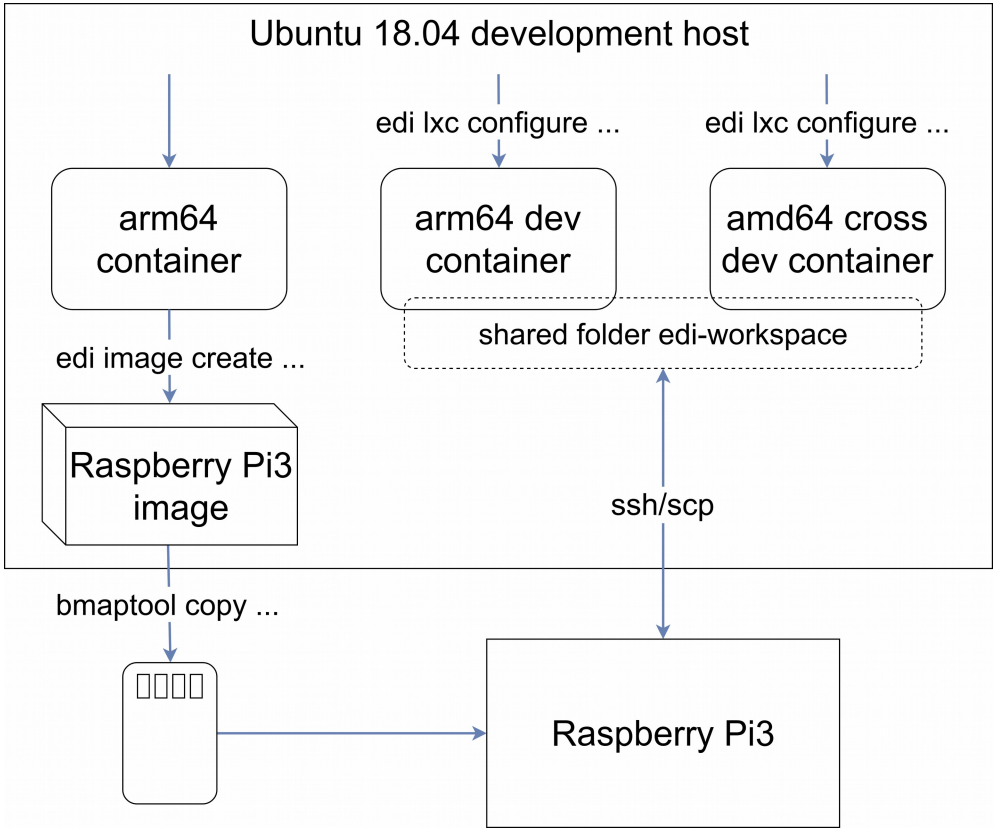
```
sudo dpkg --add-architecture armhf
sudo apt update
sudo apt install <library>:armhf
```

- With Debian stretch the cross compilers became part of the main Debian repository:

```
sudo apt install crossbuild-essential-armhf
```

Building Operating System Images

How can I build images for my target system?



- The “digital twin” cross development container gets built using a command like this:

```
sudo edi -v lxc configure edi-pi-cross-dev pi3-stretch-arm64-cross-dev.yml
```

- Building a full OS image can be achieved with such a command:

```
sudo edi -v image create pi3-stretch-arm64.yml
```

More information: http://www.get-edi.io/A-new-Approach-to-Operating-System-Image-Generation/

What happens behind the scene?

It starts from scratch...

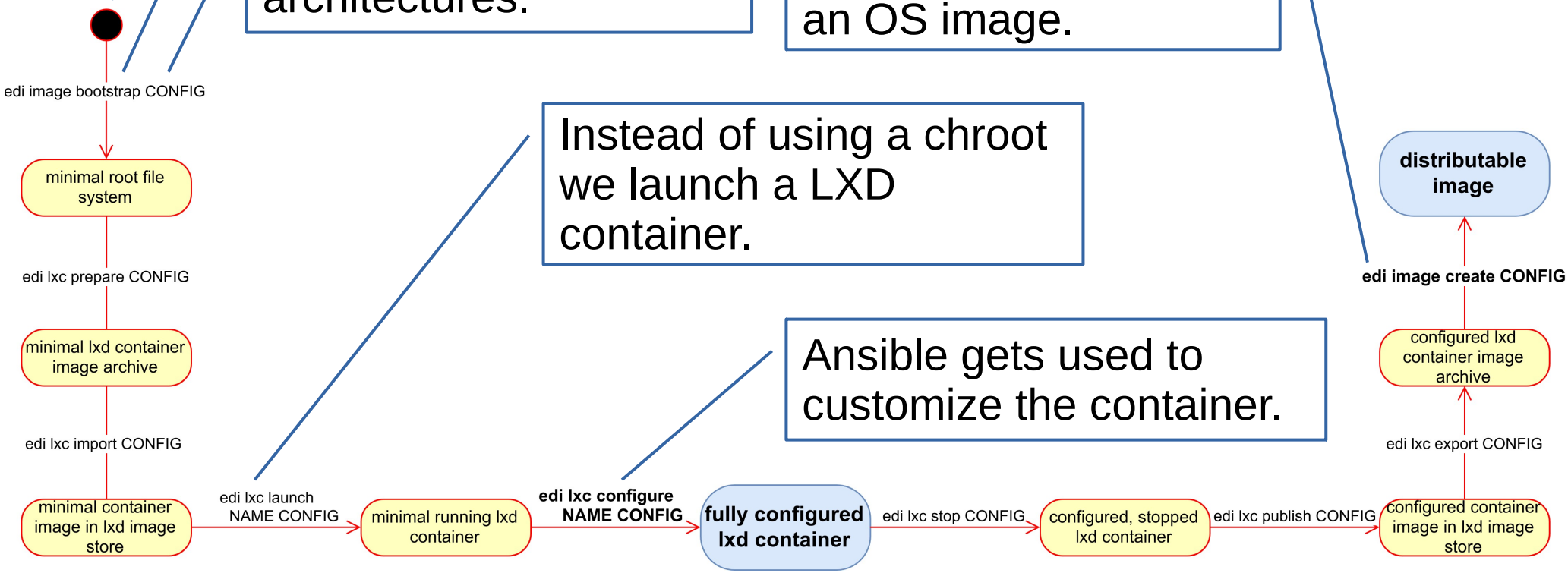
… using debootstrap!

QEMU emulates foreign architectures.

Custom commands turn the root file system into an OS image.

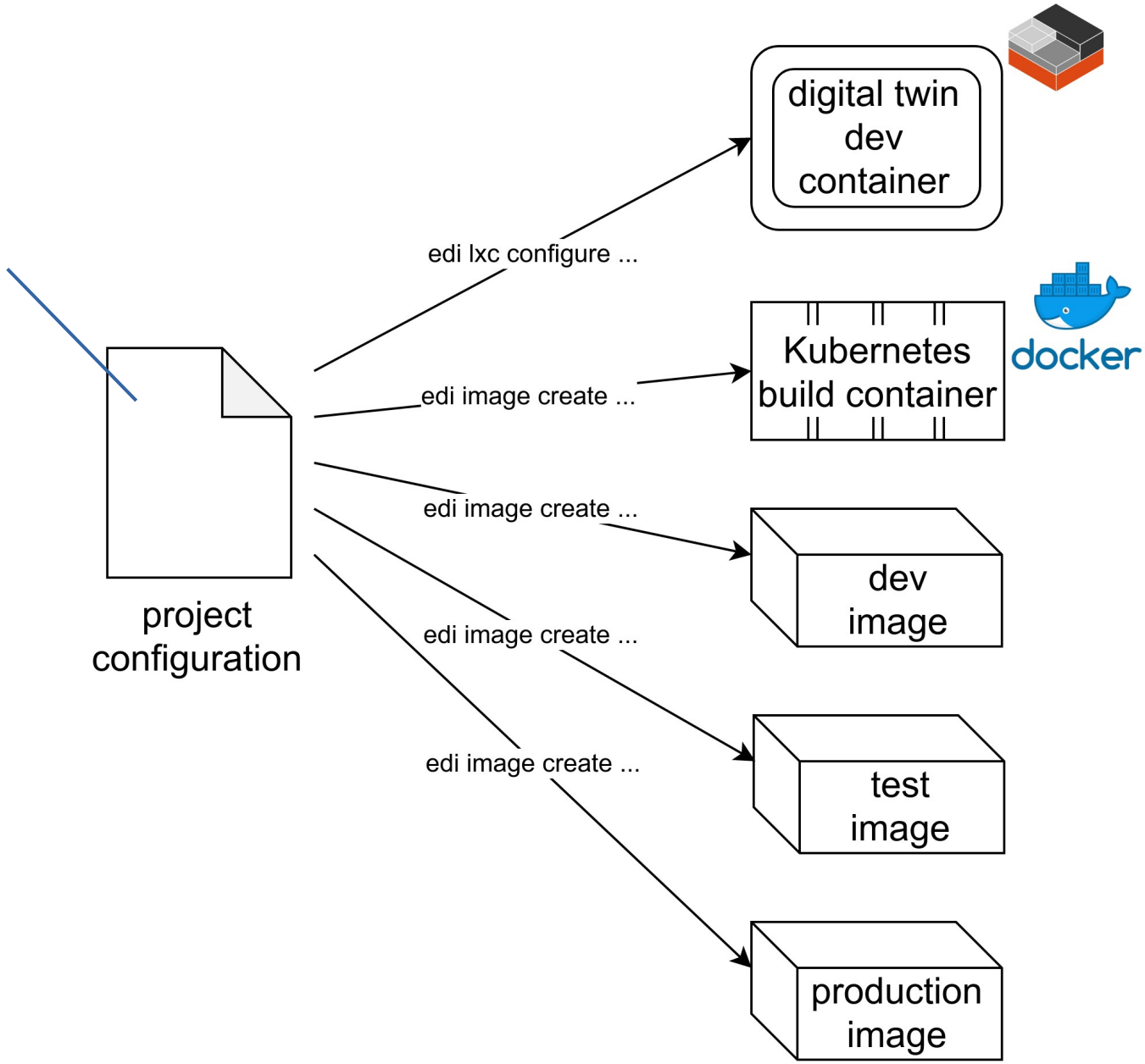
Instead of using a chroot we launch a LXD container.

Ansible gets used to customize the container.



What about different use cases?

The edi project configuration is built in a way that makes it easy to support multiple use cases without duplicating the setup.

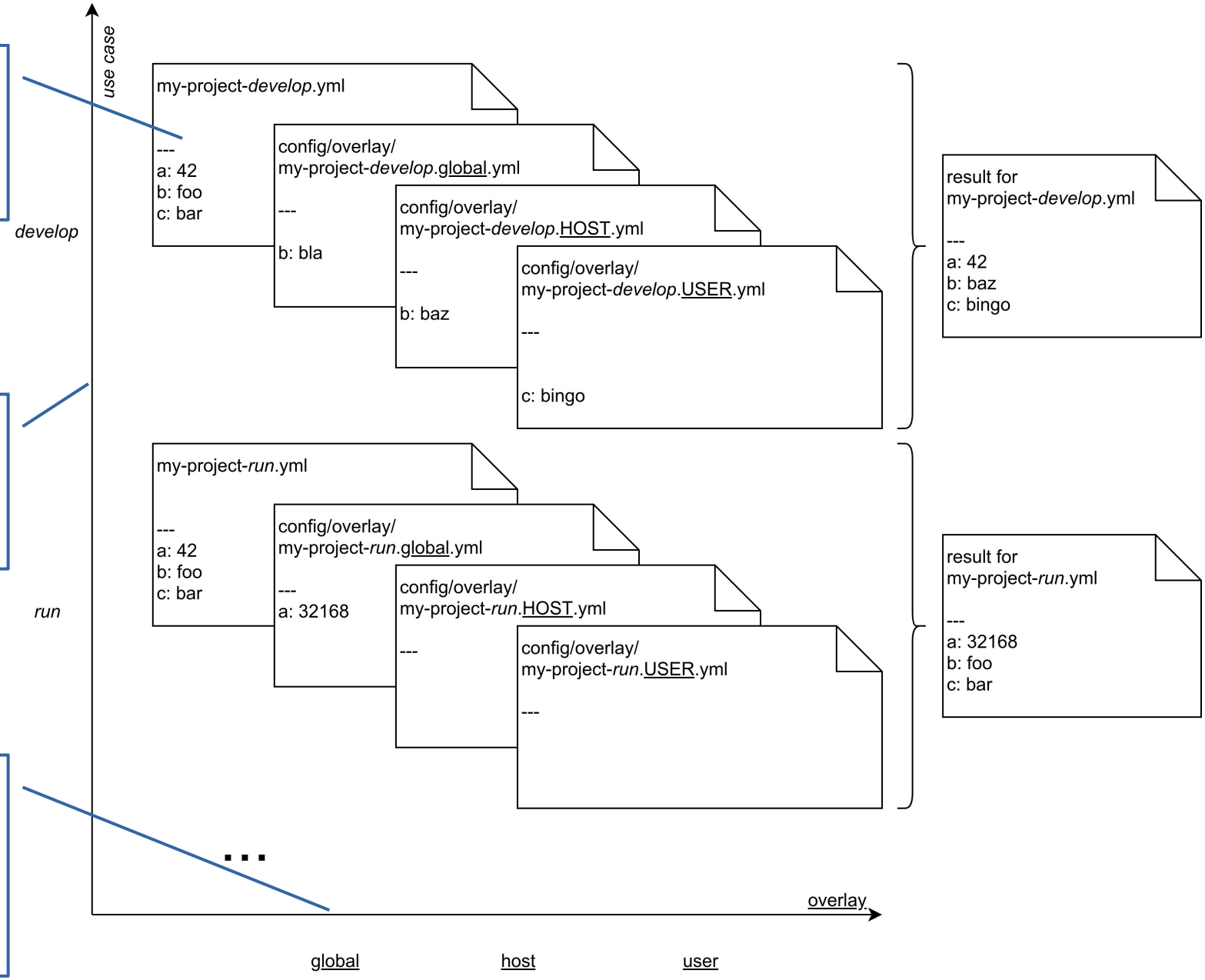


How can I configure the use cases?

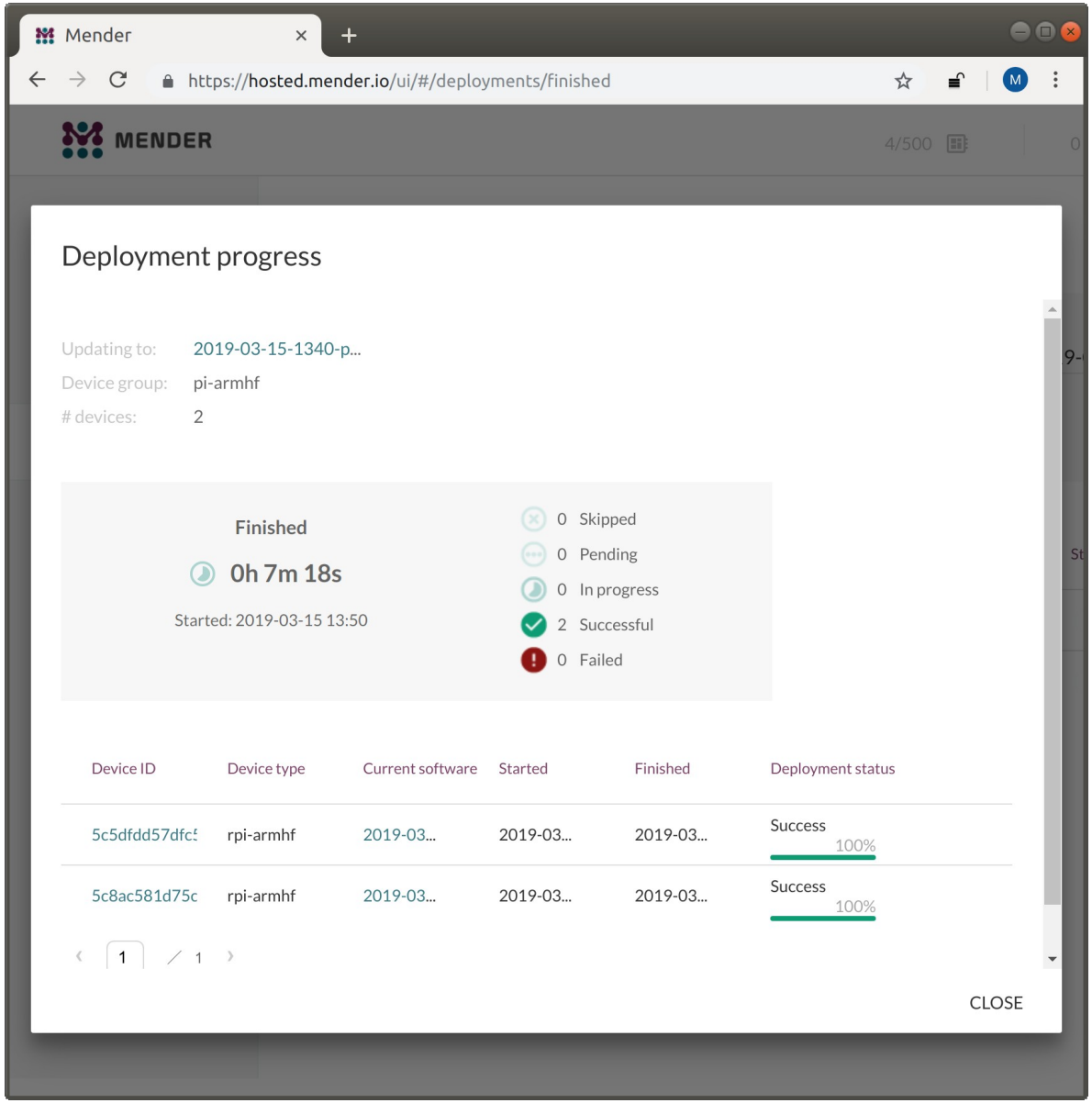
Configuration files are written in yaml and Jinja2.

edi supports as many use cases as required.

Use cases can be adjusted globally, per host and per user.

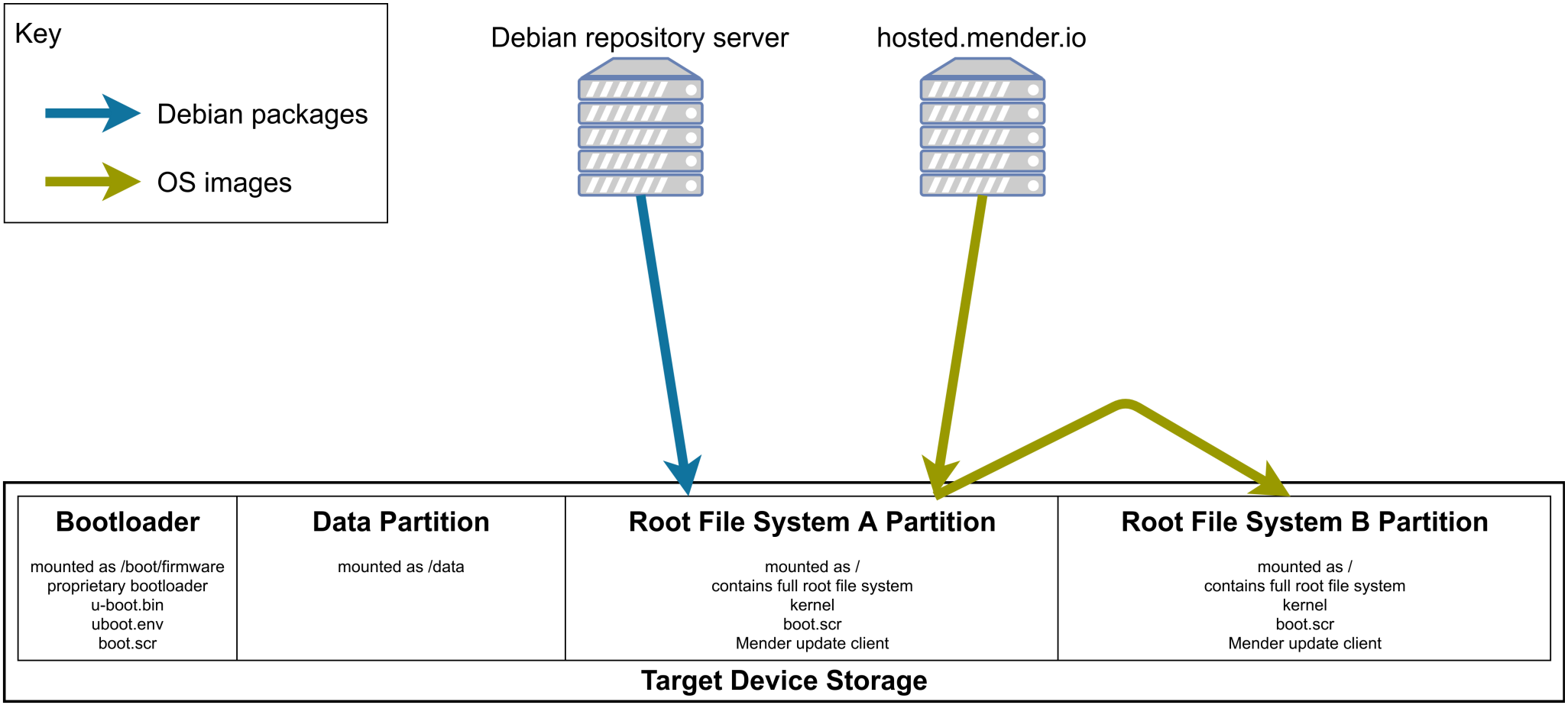


Can I update my system over the air?



More information: https://www.get-edi.io/Updating-a-Debian-Based-IoT-Fleet/

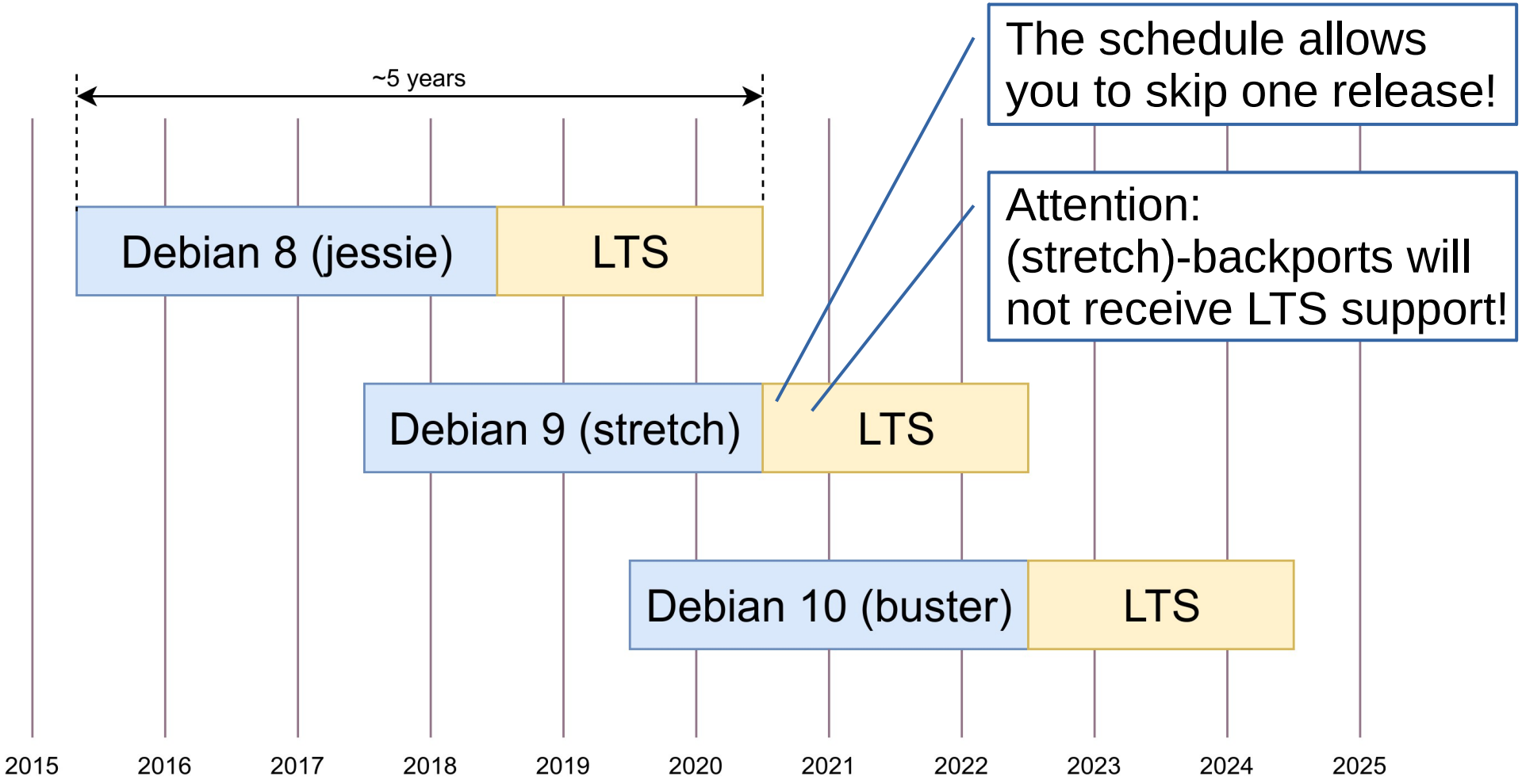
What partition layout shall I choose?



Adjust your partition layout before you ship the first product!

More information:
https://www.get-edi.io/Updating-a-Debian-Based-IoT-Fleet/
https://www.get-edi.io/Booting-Debian-with-U-Boot/

What is the longevity of my OS image?



More information:
https://www.freexian.com/services/debian-lts.html

Pitfalls

What can go wrong?

- ssh host keys
- software installation without Debian packages
- choosing a local Debian mirror
- skip signature checks
- not reproducible customization
- login credentials

What else can go wrong?

- service startup during image creation
- machine ID
- systemd preset behavior
- file system permissions
- sacrificing legal compliance for image size

More details:
https://www.get-edi.io/11-Traps-to-Avoid-When-Building-Debian-Images/
https://www.get-edi.io/Secure-by-Default-ssh-Setup/

Summary

Best Practices

- Do not take Debian for very resource constrained devices (consider using Yocto, ptxdist, buildroot etc. for such use cases).
- Make sure that your hardware at least supports ARMv7 with VFPv3 (required for Debian armhf).
- Make sure that the majority of your application can be developed and tested on the development host (within a digital twin of the target hardware):

  - faster development cycle
  - easier to test (also in virtual environment)
  - portable to future hardware
- Use standard interfaces like USB and Ethernet to improve readiness for future hardware and emulated environments.
- Properly package your software as Debian packages.

Conclusion

- Nowadays, Debian is a great choice for many embedded use cases.
- Since Debian stretch the cross compiler packages are part of the main repository.
- Debian development requires some infrastructure (build server, package server, etc.).
- If your software is not a one-man business, it is advisable to automate the setup of your infrastructure.
- Take whatever IDE you like.
- edi (https://www.get-edi.io) will help you to efficiently handle your Debian environment.
- Try out edi and comment, contribute, ...

Key technologies:

Debian, Ubuntu, Ansible, LXD, Python, Yaml, Jinja2

Debian packages for:

Ubuntu (xenial, bionic, disco)
Debian (stretch)

Security:

https://www.debian.org/security/
5 year LTS life cycle!

License compliance:

integration through machine-readable debian/copyright files planned

Open source:

LGPL license
https://github.com/lueschem/edi

Hardware testbed:

sample configurations for the Raspberry Pi are available

Quality assurance:

edi gets automatically tested and has a around 90% code coverage

https://www.get-edi.io

Update strategies:

supports incremental (packages) and full OS updates (e.g. Mender)

Digital twin:

for development and cross compilation

Supported architectures:

supports the Debian architectures (host + target)

Support:

ask question using disqus, report issues using GitHub

Community:

worldwide usage, small number of contributors