MENDER.io

Using “kas” to make Yocto manageable

Alan Martinovic - External consultant for Mender.io
YOCTO PROJECT SUMMIT 2022.05
May 17 – 19, 2022

Talk overview – this is about

1. What is kas?
2. kas for simplifying Customer Engineering tools at Mender
3. Conclusion
4. Q&A

Talk overview – this isn’t about

- Why kas is better/worse than git submodules
- Why kas is better/worse than the repo tool

What is kas?

What is kas?

Official definition[1]:

Setup tool for bitbake based projects

Unofficial definitions:

Build an image with yocto from a blank slate in less steps.

Python wrapper for reducing the nr. of steps in yocto workflow.

[1] kas github

What is kas? - Project health [github]

- Few commits a week
- Open to closed PR ratio: 1/23
- Open to closed issue ratio: 1/4
- docs live within the repository with the code
  - rendered on readthedocs

What is kas? – Installation

```
git clone -b 3.0.2 https://github.com/siemens/kas.git
cd kas
pip install .
```

- Containerized option also exists (kas-container)
- I only used it natively

What is kas? - Core features

Unofficial definition:

Build an image with yocto from a blank slate in less steps.

git clone ...

config.yml + kas checkout = layer repos cloned

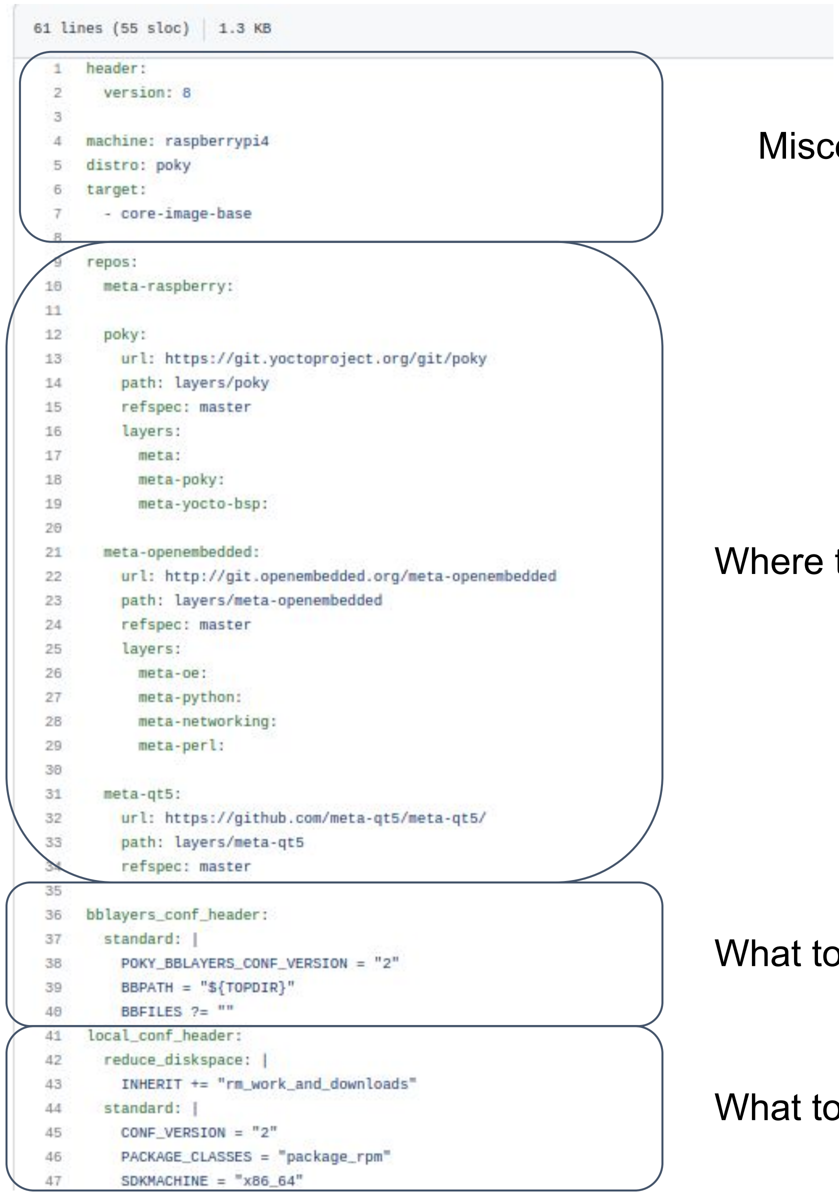
source build directory

config.yml + kas shell = bitbake environment

bitbake image

config.yml + kas build = yocto artifacts

What is kas? - Config file



61 lines (55 sloc) | 1.3 KB

```
header:
  version: 8

machine: raspberrypi4
distro: poky
target:
  - core-image-base

repos:
  meta-raspberry:

  poky:
    url: https://git.yoctoproject.org/git/poky
    path: layers/poky
    refspec: master
    layers:
      meta:
      meta-poky:
      meta-yocto-bsp:

  meta-openembedded:
    url: http://git.openembedded.org/meta-openembedded
    path: layers/meta-openembedded
    refspec: master
    layers:
      meta-oe:
      meta-python:
      meta-networking:
      meta-perl:

  meta-qt5:
    url: https://github.com/meta-qt5/meta-qt5/
    path: layers/meta-qt5
    refspec: master

bblayers_conf_header:
  standard: |
    POKY_BBLAYERS_CONF_VERSION = "2"
    BBPATH = "${TOPDIR}"
    BBFILES ?= ""
local_conf_header:
  reduce_diskspace: |
    INHERIT += "rm_work_and_downloads"
  standard: |
    CONF_VERSION = "2"
    PACKAGE_CLASSES = "package_rpm"
    SDKMACHINE = "x86_64"
```

Miscellaneous

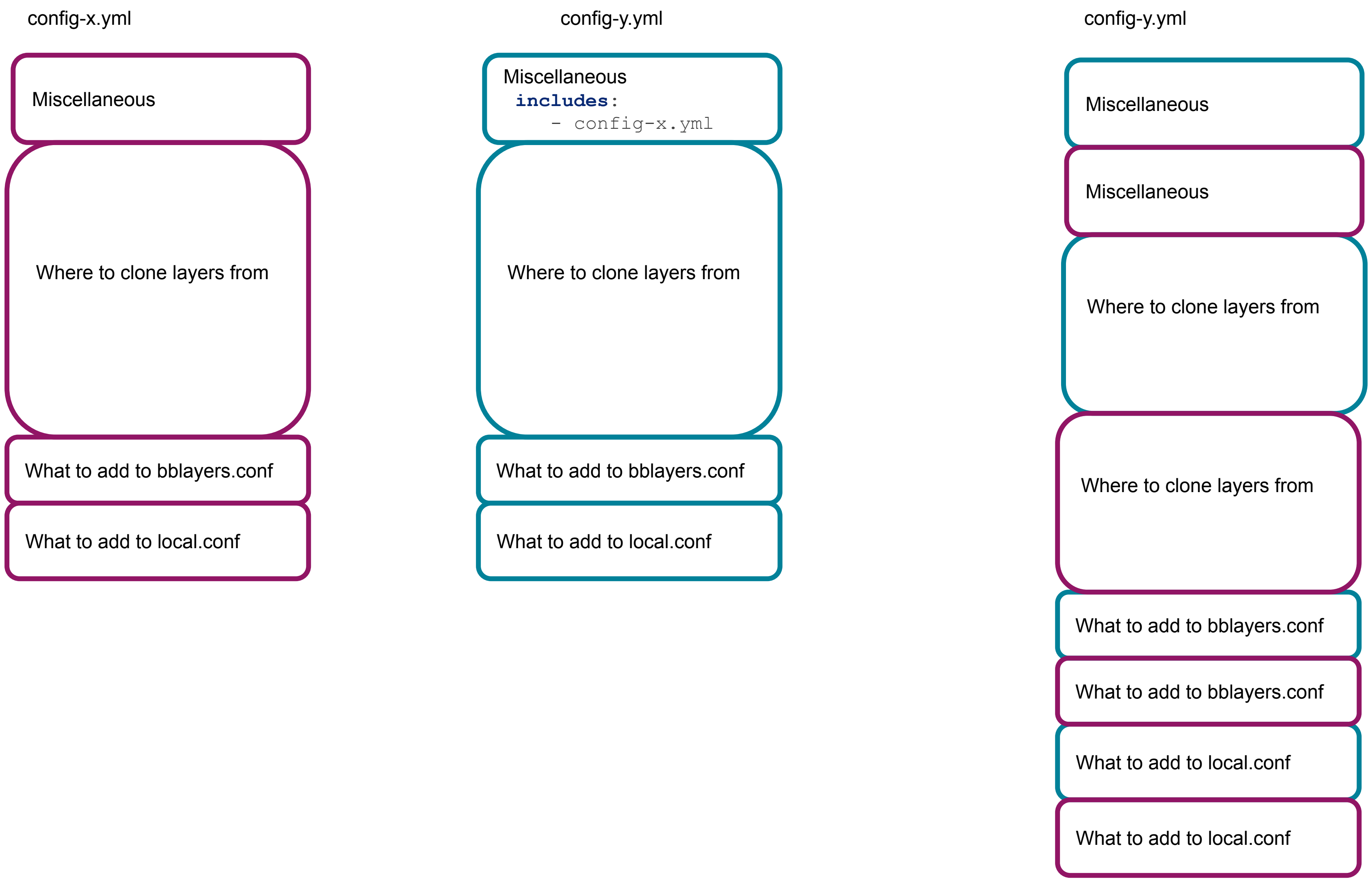
Where to clone layers from

What to add to bblayers.conf

What to add to local.conf

Example from
https://github.com/agherzan/meta-raspberrypi/blob/master/kas-poky-rpi.yml

What is kas? - Config file


config-x.yml
Miscellaneous
Where to clone layers from
What to add to bblayers.conf
What to add to local.conf
config-y.yml
Miscellaneous
Where to clone layers from
What to add to bblayers.conf
What to add to local.conf
config-y.yml
Miscellaneous
Miscellaneous
Where to clone layers from
Where to clone layers from
What to add to bblayers.conf
What to add to bblayers.conf
What to add to local.conf
What to add to local.conf

```
includes:
    - config-x.yml
```

kas in mender

Customer Engineering

kas at Mender - requirements

- Build multiple images for different HW (rpi3/4, bbb)
  - Diagnose customer issues
  - Test prospect requirements
- Diagnose Yocto issues

kas at Mender - requirements [rephrased]

- Minimal config and command overhead to build images
- local.conf doesn’t change much once set
    - I just want to make domain specific changes
- Simple to reach artifacts once they are built

- One step away from the regular bitbake env (if things break)

kas at Mender - example [once everything is configured]

```
source set_machine rpi3
# Edit auto.conf to set the dynamic build config
# i.e. MENDER_ARTIFACT_NAME = "version-1"

build_and_gather core-image-minimal
cd artifacts/rpi3/
```

Mender relevant artifacts available here

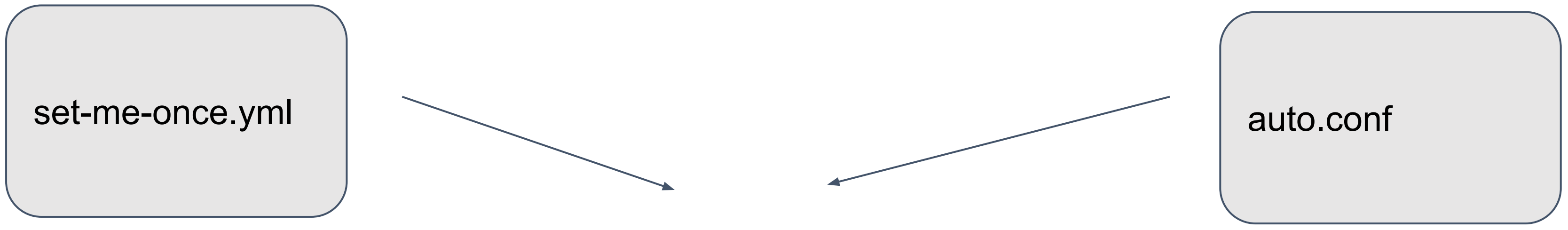
```
_kas_shell()
```

Go to initialized build dir with all config preset

kas at Mender - configuration

Set once and don’t touch

Quick change in local.conf style
Known config format
Simpler README



```
source set_machine rpi3
build_and_gather core-image-minimal
```

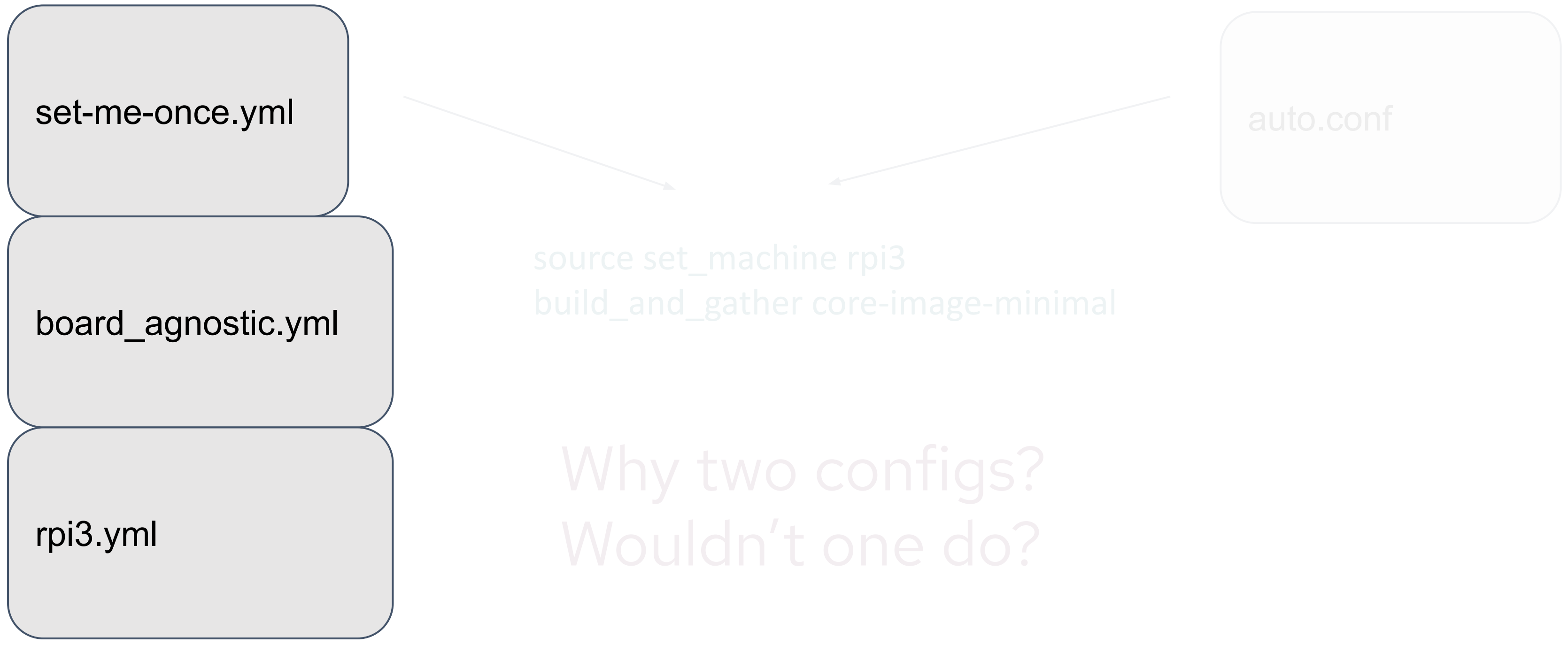
Why two configs?
Wouldn’t one do?

Questionable benefits :)

kas at Mender - kas underneath

Set once and don't touch

Quick change in local.conf style
Known config format
Simpler README



source set_machine rpi3
build_and_gather core-image-minimal

Why two configs?
Wouldn't one do?

kas shell

kas checkout

Questionable benefits :)

kas at Mender - a wrapper around a wrapper

_kas_shell()

Wrapper around “kas shell”

Why another wrapper?!?

To propagate the auto.conf to the build environment

Results depend on a hidden state (ENV vars) instead of explicit parameters ANTI PATTERN in development! Reduces cognitive load as a cli tool

Add an implicit mechanism for extending env with custom config

kas at Mender - including features as kas config

```
includes:
  - kas-base.yml
  - extend.yml
```

=

```
kas shell kas-base.yml:extend.yml
```

Some mender features need multiple lines in local.conf

This allows for simpler feature inclusion

Add an implicit mechanism for extending env with custom config

kas at Mender - simplify internal tool usage/readme

```
# Clone the repositories
git clone http://git.openembedded.org/meta-openembedded
git clone https://github.com/mendersoftware/meta-mender
git clone https://github.com/agherzan/meta-raspberrypi
git clone https://git.yoctoproject.org/git/poky
source poky/oe-init-build-env

# Please add the following to your local.conf
DL_DIR= "/home/workspace/build_hard/yocto_cache/downloads"
SSTATE_DIR= "/home//workspace/build_hard/yocto_cache/sstate-together"
MENDER_SERVER_URL = "https://hosted.mender.io"
# MENDER_SERVER_URL = "https://staging.hosted.mender.io"
DISTRO_FEATURES_append = " systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"
VIRTUAL-RUNTIME_initscripts = ""
# INHERIT += "rm_work"
IMAGE_LINK_NAME_append = "-${MENDER_ARTIFACT_NAME}"
IMAGE_FEATURES += "ssh-server-openssh allow-empty-password debug-tweaks"
IMAGE_INSTALL_append = " python3"
# Stuff from meta-mender-ce
IMAGE_INSTALL_append = " mender-monitor-crasher-app"
INHERIT += "mender-full"
RPI_USE_U_BOOT = "1"
IMAGE_FSTYPES_remove += " rpi-sdimg"
MENDER_FEATURES_ENABLE_append = " mender-uboot mender-image-sd"
MENDER_FEATURES_DISABLE_append = " mender-grub mender-image-uefi"
MENDER_BOOT_PART_SIZE_MB = "40"
LICENSE_FLAGS_WHITELIST_append=" commercial_mender-binary-delta"
IMAGE_FEATURES_append= " read-only-rootfs"
IMAGE_INSTALL_append= " mender-binary-delta"
…

# Edit local.conf to set the dynamic build config
MENDER_ARTIFACT_NAME = "version-1"

bitbake core-image-minimal

# Find the correct images
cd tmp/deploy/images/raspberrypi3/
```

```
source set_machine rpi3
# Edit auto.conf to set the dynamic build config
# i.e. MENDER_ARTIFACT_NAME = "version-1"

build_and_gather core-image-minimal
cd artifacts/rpi3/

_kas_shell()
```

Mender relevant artifacts available here

Go to initialized build dir with all config preset

Conclusion

Conclusion

- kas replaces extra commands with config files
- kas config files can be included into each other
    - as config syntax and as cli parameters

- in mender CE kas is used to do the heavy lifting with regards yocto setup
- It significantly reduces the complexity of helper scripts making them maintainable

Q&A

Thank you