

Yocto on the Edge

Unusual challenges when
building not so embedded
systems

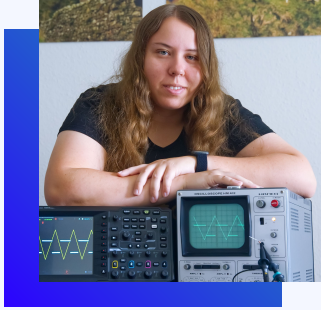
Anna-Lena Marx

Yocto Project Dev Day

September 19th, 2024 · Vienna



Anna-Lena Marx



Embedded Systems Developer

- since 2015 with inovex
- has a Master's degree in Embedded Systems
- studies Electrical Engineering as a hobby

Main Topics

- Embedded Systems
- Yocto Linux
- Linux Kernel
- AOSP/AAOS
- IoT



[Anna-Lena Marx](#)



anna-lena.marx@inovex.de



marx.engineer



work based on kirkstone



bugs caused by my distraction ninja



About the project and requirements

SaferCities is a New Zealand based company that provides independent subject matter expertise consulting around CCTV and related systems, as well as the **vGRID SaferCity Platform**. vGRID enables CCTV & ANPR asset owners to share with one another and law enforcement, regardless of CCTV system (or VMS), network configuration or brand. vGRID is used across the whole of New Zealand by New Zealand Police, and is used by other Police forces in Australia, and is undergoing trials in the USA.

For more information, please see <https://vgrid.io> or <https://safercities.com>



balena

switching to ...

**... to get
full platform control**

- containers and system
- updates

fulfill high security requirements

- user management
- system integrity
- encrypted data



Working with x86

Workflows, Debugging, Provisioning



- interfaces like UART, JTAG, ... directly accessible
- often SD cards for development and eMMC for production
- e.g. jumpers and/or manufacturer tools needed to flash eMMC

-> diverse, but known tooling

- mostly no accessible debug interfaces -> use screen and keyboard, screen capture tools, ...
- boots from USB for development, SSD/NVMe for production
- create a “self-installing image” from a USB drive

-> less diverse but unknown and “limited”

Provisioning - Self-installing Yocto from a USB drive

Updated on February 16, 2023 / Bennie Affleck

Self-installing Yocto Image from a USB drive

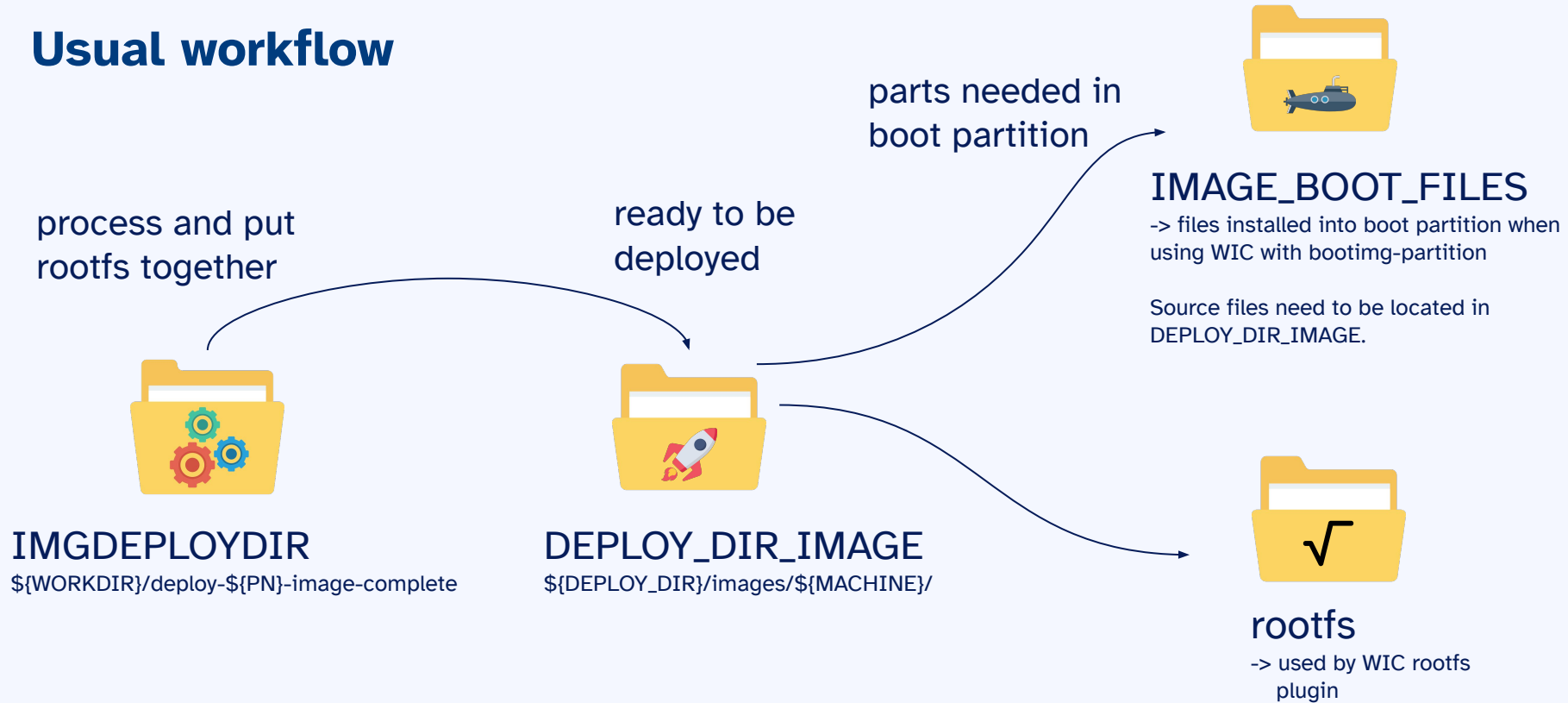
Tech Blog | Yocto

We are all familiar with the process of installing a desktop operating system onto a PC; simply insert the installation media, typically a USB stick, and boot the device. A minimal version of the OS then runs to perform the process of partitioning discs and installing the full OS.

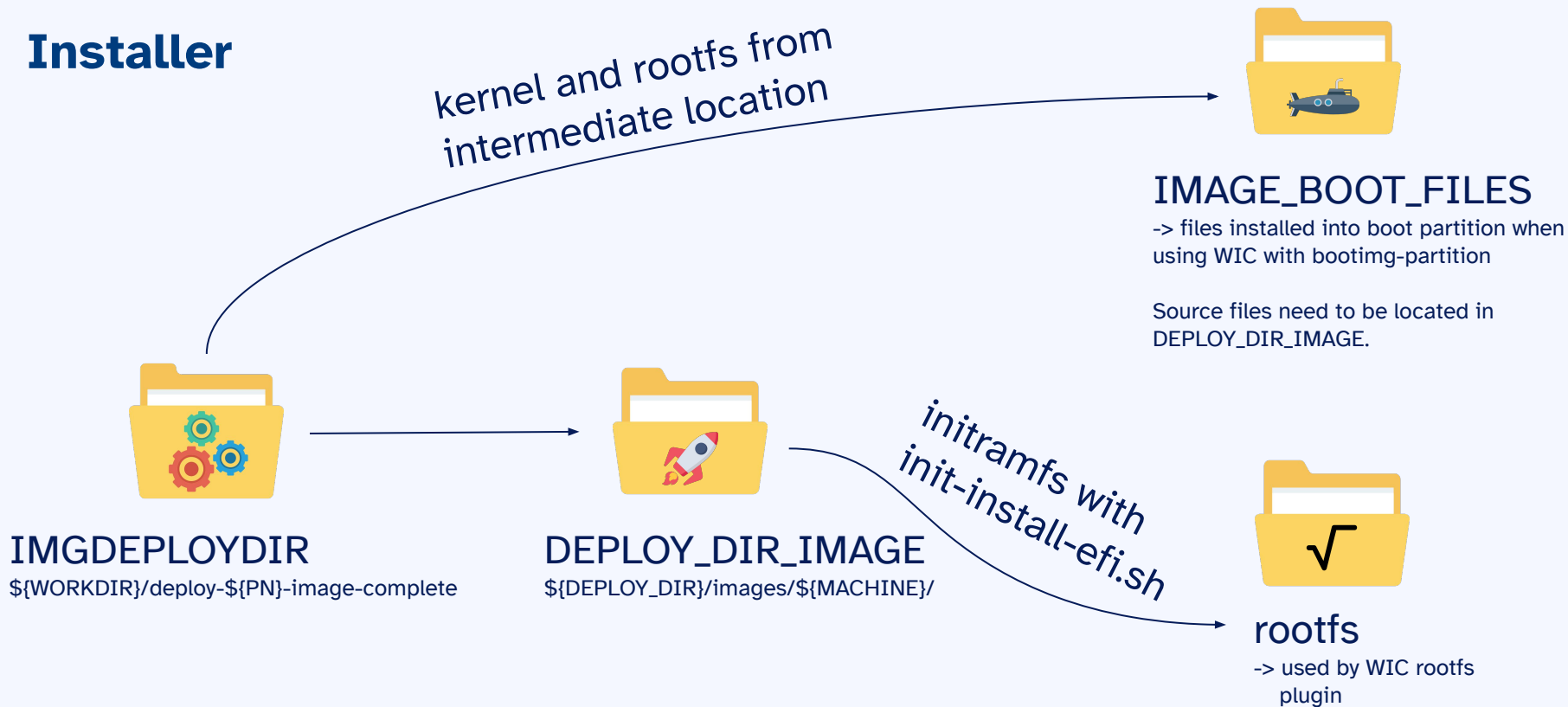
Yocto also has the ability to generate a self-installing image, just like a desktop operating system. For an embedded system that can boot from a USB drive this gives some significant advantages:

<https://www.thegoodpenguin.co.uk/blog/self-installing-yocto-image-from-a-usb-drive/>

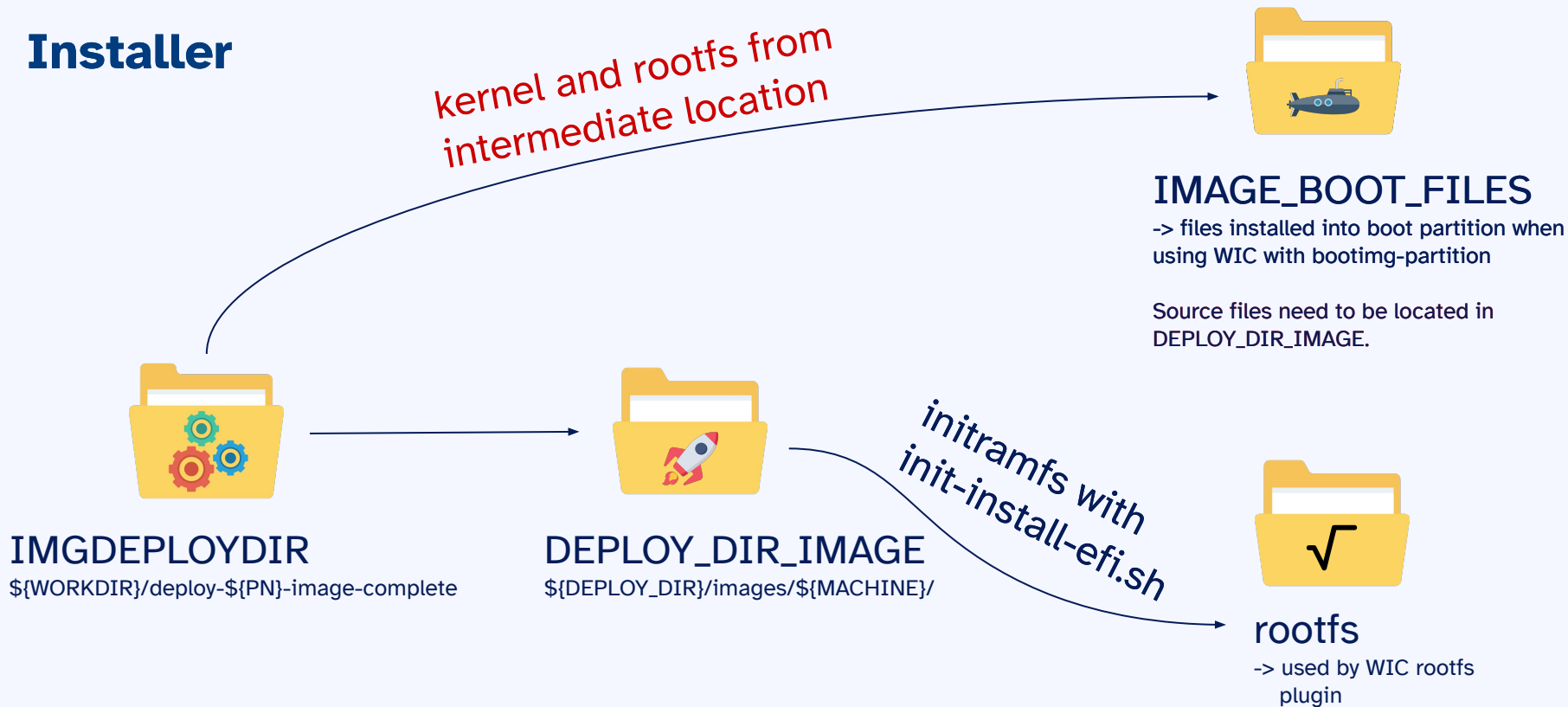
Usual workflow



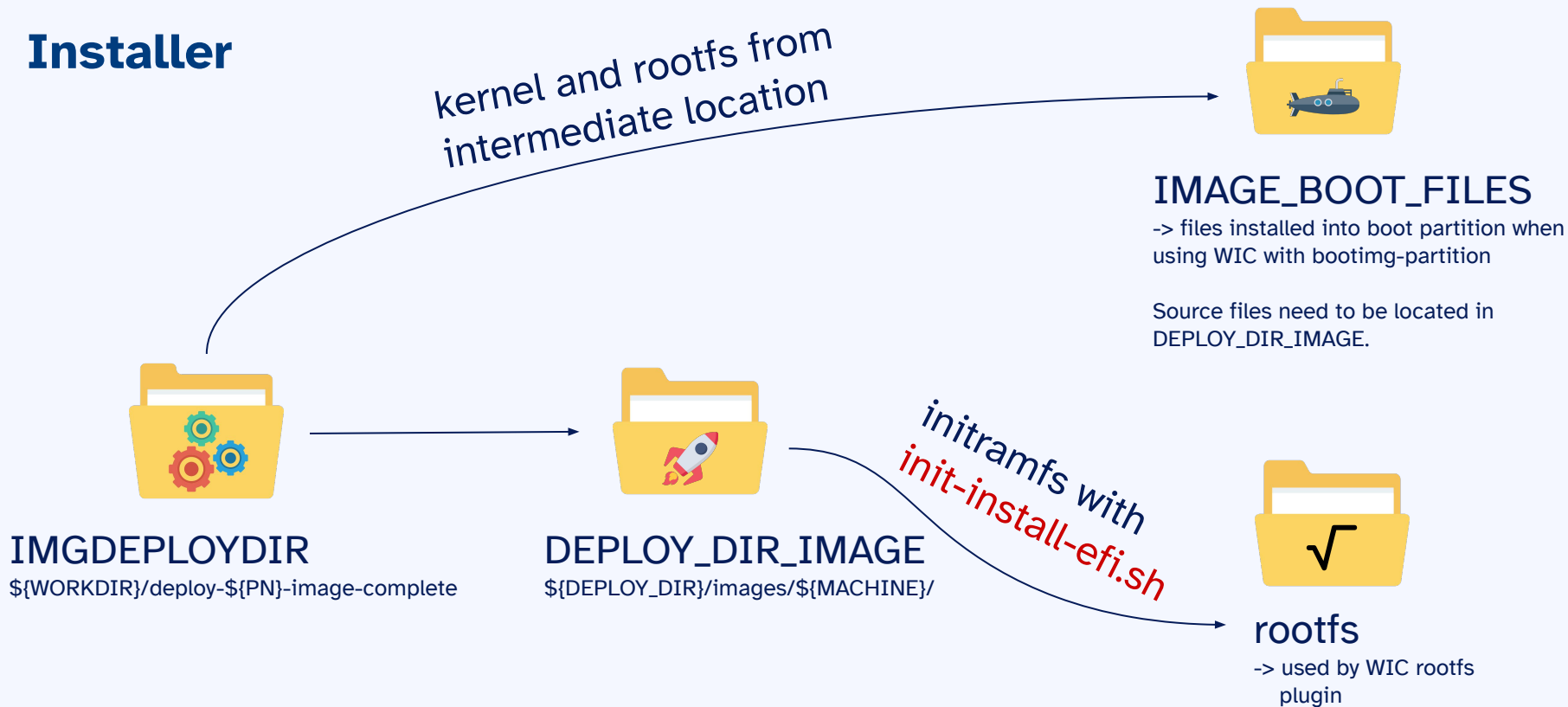
Installer



Installer



Installer



```
bootfs=${device}${part_prefix}1  
rootfs=${device}${part_prefix}2  
swap=${device}${part_prefix}3
```

```
echo "*****"  
echo "Boot partition size:"  
echo "Rootfs partition size:"  
echo "Swap partition size:"  
echo "*****"  
echo "Deleting partition table"  
dd if=/dev/zero of=${device}
```

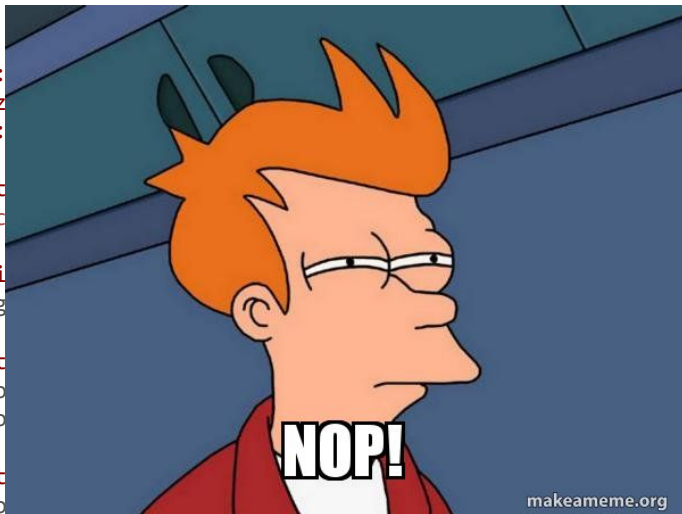
```
echo "Creating new partition table"  
parted ${device} mklabel gpt
```

```
echo "Creating boot partition"  
parted ${device} mkpart boot fat 1 100%  
parted ${device} set 1 boot
```

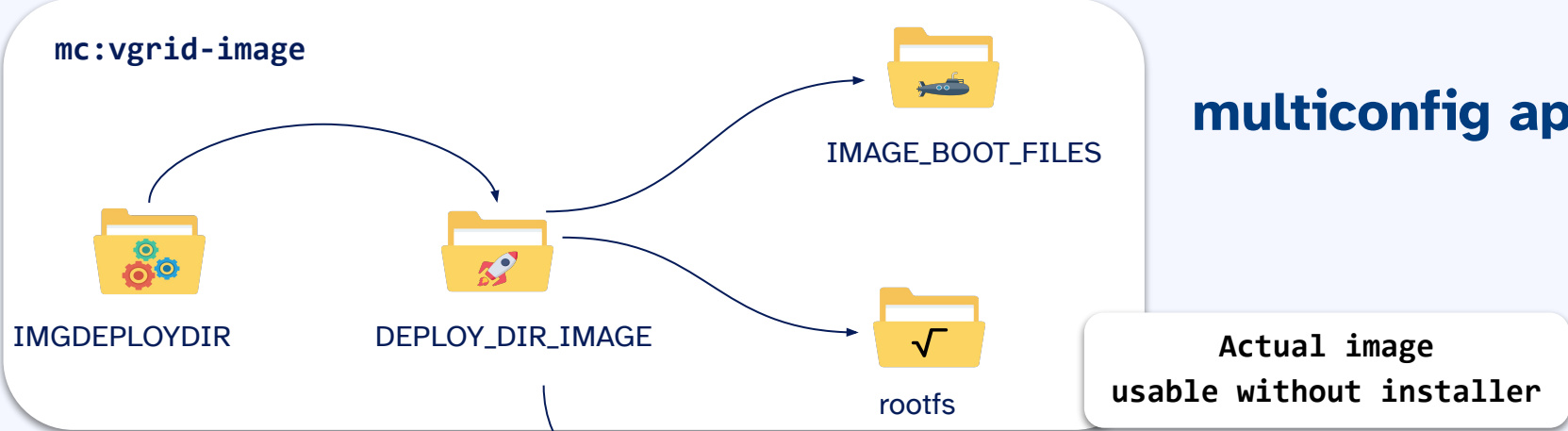
```
echo "Creating rootfs partition"  
parted ${device} mkpart root ext4 100% 100%
```

```
echo "Creating swap partition on $swap"  
parted ${device} mkpart swap linux-swaps $swap_start 100%
```

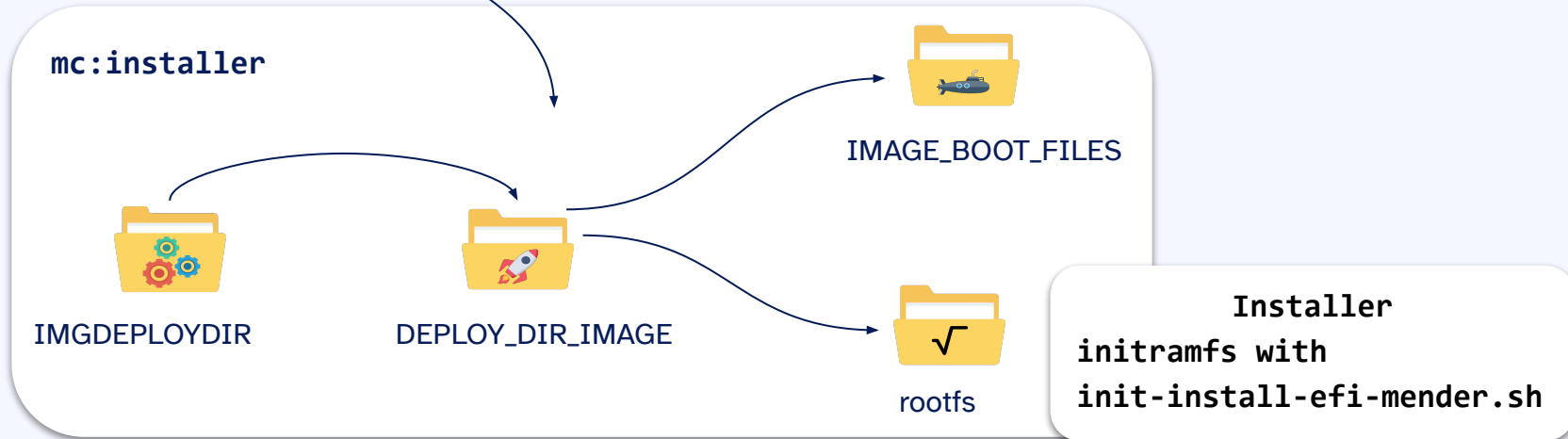
```
parted ${device} print
```



multiconfig approach



`mender-enabled-image.uefiimg.bz2`




```
SUMMARY = "Build the actual images as an inner payload"

PACKAGE_ARCH = "${MACHINE_ARCH}"
PACKAGES = "${PN}"

INHIBIT_DEFAULT_DEPS = "1"

# Variables to control where images are found: the multiconfig name, and the deploy dir.
CONTAINER_PACKAGE_MC ?= "vgrid-images"

CONTAINER_PACKAGE_DEPLOY_DIR = "${TOPDIR}/tmp-vgrid-images-glibc/deploy/images/intel-corei7-64"

do_install[mcdepends] += "mc:vgrid-installer:vgrid-images:vg-image-provisioning:do_image_complete"
do_install[mcdepends] += "mc:vgrid-installer:vgrid-images:vg-image-update:do_image_complete"

do_install() {
    install ${CONTAINER_PACKAGE_DEPLOY_DIR}/${IMAGE}.uefiimg.bz2 ${D}/${BASE}.uefiimg.bz2
}

FILES:${PN} = "${BASE}.uefiimg.bz2"

# we don't need those!
do_configure[noexec] = "1"
do_compile[noexec] = "1"
deltask do_populate_sysroot
```

vgrid-image-installer.bb

```
SUMMARY = "Minimal installer image"

require recipes-core/images/core-image-base.bb

PACKAGE_INSTALL:append = " vgrid-images-buildtask"
```

vgrid-distro-installer.bb

```
IMAGE_BOOT_FILES:append = " \
    ${IMAGE_ROOTFS}/${PAYLOAD_IMG_NAME}.uefiimg.bz2 \
"
```

kas-installer.yml

```
header:
  version: 16
  includes:
    - kas/common-intel.yml

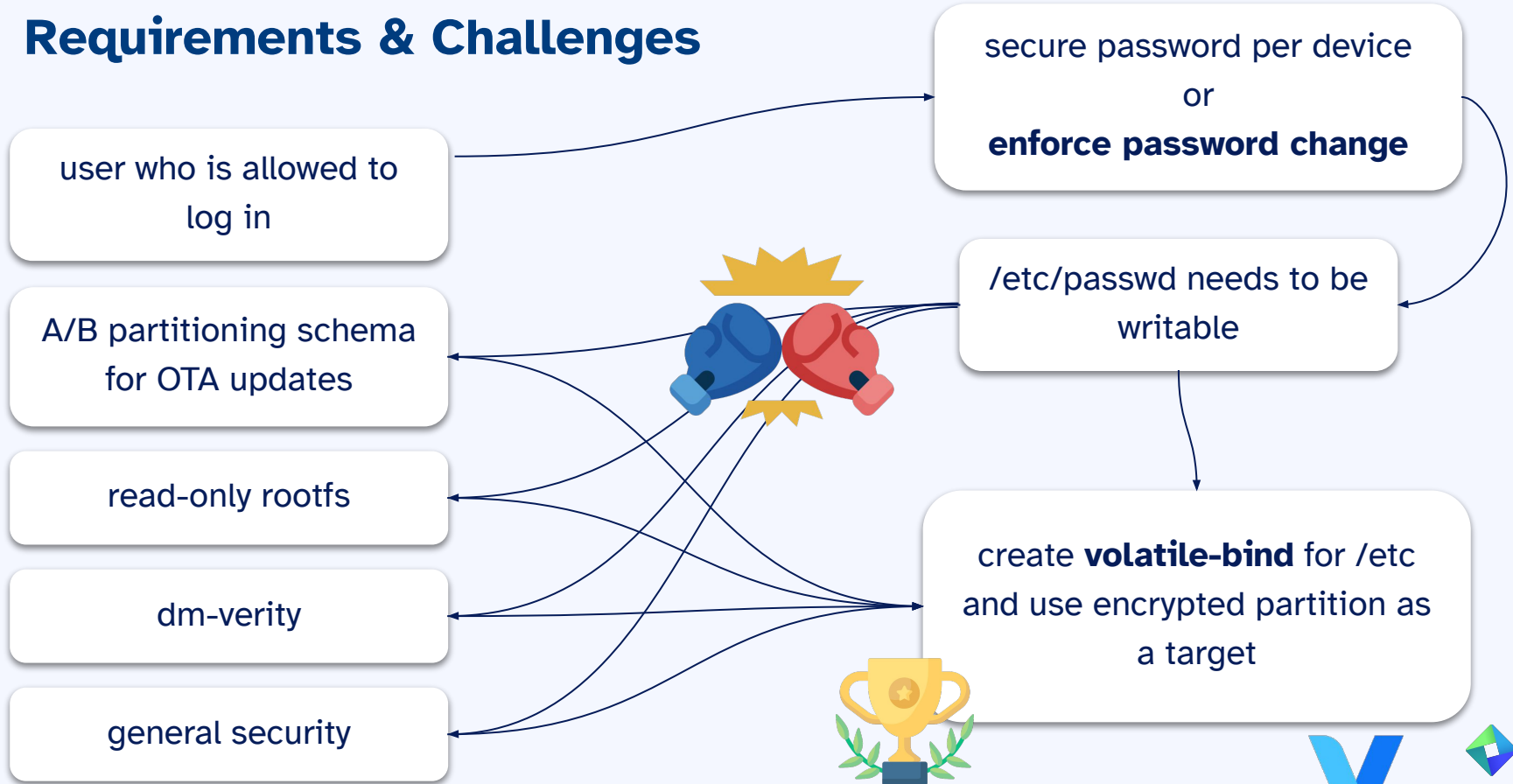
distro: vgrid-distro-installer
target: mc:vgrid-installer:vg-image-installer

local_conf_header:
  usb-provisioning: |
    BBMULTICONFIG = "vgrid-images vgrid-installer"
```

Login Users

**with read-only rootfs and A/B updates
- but sound and secure**

Requirements & Challenges



```
# We need to overlay /etc for writeable system configurations,  
# e.g. changing the user's password.  
VOLATILE_BINDS:append = " \  
    /data/config /etc\n \  
"  
  
# We need to make sure the target directories exist otherwise  
# the bind-mount will fail  
FILES:${PN} += " \  
    /data/config \  
"  
  
do_install:append() {  
    install -d ${D}/data/config  
}
```

Other possible implementations:

- overlayfs.bbclass
- overlayfs-etc.bbclass

[OverlayFS and its use in Yocto Project](#)

vgrid-image.bb

```
ROOTFS_POSTPROCESS_COMMAND +=  
'${@bb.utils.contains_any('VG_FEATURES', 'vg-expire-password', 'set_password_expiry; ', '', d)}'  
  
set_password_expiry() {  
    export PSEUDO="${FAKEROOTENV} ${STAGING_DIR_NATIVE}${bindir}/pseudo"  
    flock -x ${IMAGE_ROOTFS}${sysconfdir} -c "$PSEUDO chage -R ${IMAGE_ROOTFS} -d0 ${AUTHORIZED_USER_NAME}"  
}
```

Encrypted data partition

Utilizing LUKS2, TPM2 and systemd

Encrypt on runtime



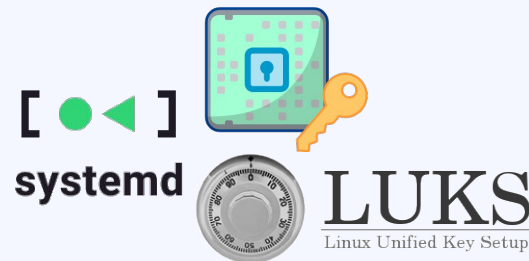
systemd service

- triggers encryption
- runs before **data.mount**



encryption script

- write LUKS header on existing partition
`cryptsetup reencrypt --encrypt`
- actual online encryption
`cryptsetup reencrypt`



encryption script

- link TPM to LUKS device
`systemd-cryptenroll`

Prepare in image

fstab

```
- UUID=<data-uuid> /data ext4 rw 0 2  
+ /dev/mapper/data /data ext4 rw 0 2
```

fstab is modified by Mender in a
ROOTFS_POSTPROCESS_COMMAND !

crypttab

```
data UUID=<data-uuid> none tpm2-device-auto,tpm2-pcrs=7
```

At early boot and when the system manager configuration is reloaded, `/etc/crypttab` is translated into `systemd-cryptsetup@.service` units by [systemd-cryptsetup-generator\(8\)](#).

Dealing with Limits

Using KAS to patch .bbclass, .inc or similar
“unpatchable” files in Yocto

```
meta-mender:
  layers:
    meta-mender-core:
      branch: kirkstone
      url: https://github.com/mendersoftware/meta-mender.git
      path: sources/meta-mender
      patches:
        encrypted-data:
          repo: meta-vgrid
          path: patches/meta-mender/mender-use-mapper-device-for-data.patch
```

kas/common.yml

meta-vgrid/patches/meta-mender/mender-use-mapper-device-for-data.patch

```
diff --git a/meta-mender-core/classes/mender-setup-image.inc b/meta-mender-core/classes/mender-...
index 85383d27..3fc4097a 100644
--- a/meta-mender-core/classes/mender-setup-image.inc
+++ b/meta-mender-core/classes/mender-setup-image.inc
@@ -58,7 +58,7 @@ mender_update_fstab_file() {
     fi

     mkdir -p ${IMAGE_ROOTFS}/data
-    printf "%-20s %-20s %-10s %-21s %-2s %s\n" ${tmpDataPart} /data ${MENDER_DATA_PART_FSTYPE} ...
+    printf "%-20s %-20s %-10s %-21s %-2s %s\n" ${MENDER_DATA_PART_CRYPT} /data ${MENDER_DATA_PA...
```

Why and when am I using this mechanism?

- patching files where `.bbappend` is not possible, e.g. `.bbclass`, `.inc`, `ROOTFS_POSTPROCESS_COMMAND` functions, ...
 - when copying the file to our own layer does not make sense, e.g.
 - just a one-line change
 - want to get updates and see if the original file changes
 - when copying the file to our own layer does not work, e.g.
 - an e.g. `.inc` file that's already used in the original layer
 - priority issues
- > not Yocto-by-book, but a really clean and maintainable way to work around limits
- > no replacement for `.bbappend`!

Encrypted data partition

Utilizing LUKS2, TPM2 and systemd

vgrid-distro.conf

```
PACKAGECONFIG:append:pn-cryptsetup = " cryptsetup veritysetup udev luks2"  
PACKAGECONFIG:append:pn-systemd = " cryptsetup tpm2"
```

systemd_%.bbappend

```
do_install:append() {  
    install -d ${D}${libdir}/cryptsetup  
    install -m 0755 $f ${D}${libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \  
    install -m 0755 $f ${D}${libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \  
}
```

Fixed in scarthgap!

```
FILES:${PN}:append = "  
    ${base_libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \  
    ${libdir}/cryptsetup/libcryptsetup-token-systemd-tpm2.so \  
"
```

encrypt-data.sh

```
echo "Generating a new key..."
/usr/bin/openssl rand -base64 44 > ${tmp_key_file}

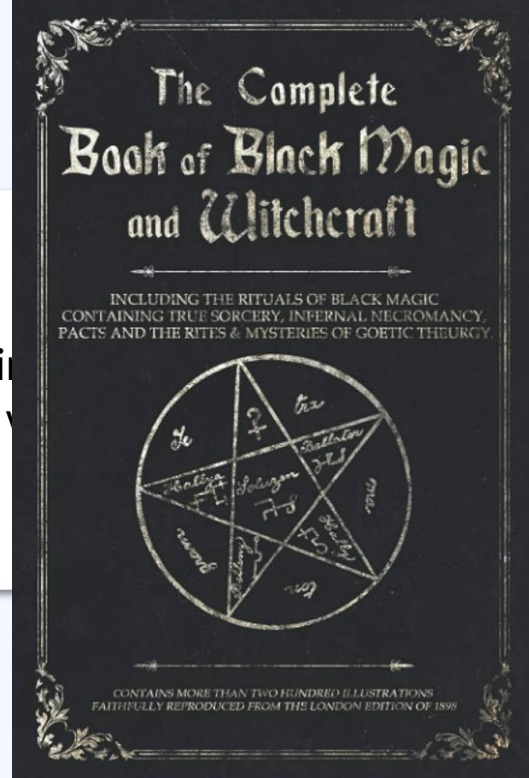
echo "Writing encryption headers..."
/bin/cat "${tmp_key_file}" | /usr/sbin/cryptsetup reencrypt --encrypt --type luks2 --key-slot=1 \
--batch-mode --init-only --reduce-device-size 32M --offset="${OFFSET}" "${data_dev}" data

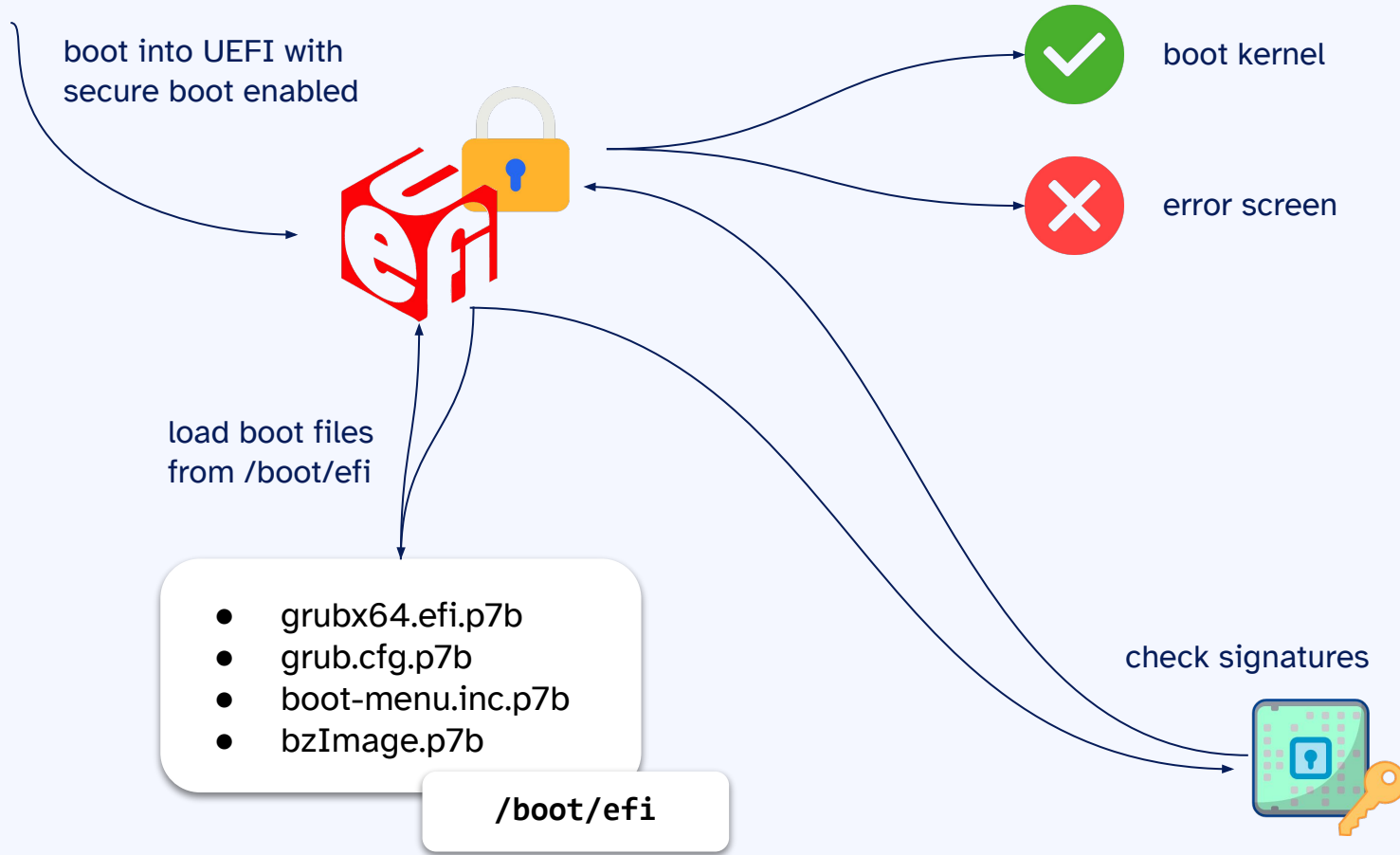
# Enrolling the TPM2 integration only works after the online encryption step is finished.
echo "Encrypting the data partition..."
/bin/cat ${tmp_key_file} | /usr/sbin/cryptsetup reencrypt --offset="${OFFSET}" "${data_dev}"

echo "Deploying TPM2 keys..."
export PASSWORD="$(cat ${tmp_key_file})"
/usr/bin/systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 "${data_dev}"
unset PASSWORD
```

secure-boot

and Mender





meta-secure-core/meta-efi-secure-boot

- grub-efi, efitools, mokutil, shim, ...
- patches for GRUB
- signing tasks for kernel, bootloader, ...
- TPM integration for keys and certificates
- automated certificate provisioning boot step

... but all examples and documentation utilize an initramfs!

Best read for kirkstone: [Discussion on Mender and secure boot](#)

vgrid-distro.conf

```
MACHINE_FEATURES:append = " efi tpm2"
DISTRO_FEATURES:append = " \
    security \
    tpm2 \
    efi-secure-boot \
"

MOK_SB = ""
SIGNING_MODEL = "user"
```

vgrid-image.bb

```
IMAGE_INSTALL:append = " \
    tpm2-tools \
    libtss2-tcti-device \
    efityls \
    seloader \
    update-signed-kernel-state-script \
"

# Deploy kernel and signature to boot partition.
# They cannot be verified on rootfs partition.
#
# Mender processes IMAGE_BOOT_FILES and has some issues with multiline parsing
IMAGE_BOOT_FILES:append = " ${KERNEL_IMAGETYPE} ${KERNEL_IMAGETYPE}${SB_FILE_EXT}"
```

...

```
mender_kernel_path=""
if [ "${drop_to_grub_prompt}" = "no" ];
then
    search --no-floppy --label --set=root boot
    if linux "${mender_kernel_path}/${kernel_imagetype}" root="${mender_kernel_root}" ${bootargs};
    then
        if test -n "${initrd_imagetype}" -a test -e "${mender_kernel_path}/${initrd_imagetype}";
        then
            initrd "${mender_kernel_path}/${initrd_imagetype}"
        fi
        maybe_pause "Pausing before booting."
        boot
    fi
    maybe_pause "Pausing after failed boot."
fi
```

It works now, but ...

the kernel image within the boot partition is **NOT** managed by Mender!

- use Mender state scripts to copy the kernel to /boot/efi
 - executed as **Enter** or **Leave** action to the Mender states
 - **ArtifactInstall_Leave_05_kernel_update**
 - **ArtifactRollback_Leave_05_kernel_rollback**

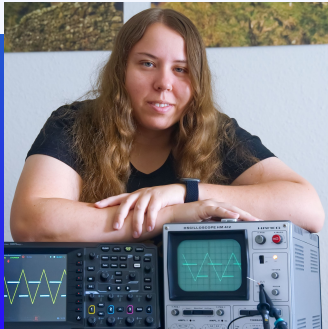
And in the end?



We have built a ~~hopefully~~ rock-solid and secure system,
with clean, understandable and maintainable code.
And a lot time spent into bringing pieces together!

Thank you!

Time for questions.



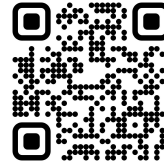
Anna-Lena Marx
Embedded Systems Developer

anna-lena.marx@inovex.de



<https://vgrid.io>

<https://safercities.com>



inovex is an IT project center driven by innovation and quality, focusing its services on 'Digital Transformation'.

- founded in 1999
- 500+ employees
- 8 offices across Germany



www.inovex.de

