Yocto

1) what is Yocto?

The Yocto Project is an open source project that helps you build custom Linux distribution for embedded systems. It's not a Linux distribution itself - It's a tool to build one tailored for your hardware.

Core components

1) BitBake : The build engine that executes tasks and recipes

2) Open embedded core : A set of common recipes and metadata for building packages

3) Metadata: Includes recipes (.bb files), configuration file and class files that tell Yocto what and how to build

4). layers: Modular collections of metadata [like lego blocks of featu

5) Poky : The reference Yocto distribution

6) machine configuration: : Defines hardware-specific settings like CPU architecture, Kernel and etc

7) Recipes: Instructions to fetch, configure, compile and install software.

2) Difference between open embedded and yocto

Feature	open embedded	yocto
what is it	Build systems & metadata	umbrella project including OE, BitBake, tools.
focus	core build tech	complete ecosystem.
main components	OE - core, layers.	BitBake, Poky, OE - core, SDK tools
Relationship	Yocto use open embedded core	yocto is powered by OE

3) Role of BitBake

*) BitBake is the task executor

*). It reads recipes (.bb files) and figures out how to fetch, compile, install and packages each software component

*) It resolves dependencies, schedules tasks and build images

Bitbake core-image-minimal.

This command tells BitBake to build the core-image-minimal image using all recipes

4) What are layers in yocto and why are they important?

A layer is a directory with related metadata, recipes, config files and etc. They allow you to separate functinally cleanly.

ex. layers

meta : OE core recipes

meta-poky : Poky-specific changes.

meta-myproject : your own custom layer.

---

5) what is a recipe (.bb file) in yocto? and its key fields

Ans

A recipe tells BitBake how to build a Software Pack

key fields

```
DESCRIPTION = "My APP"
LICENSE = "MIT"
SRC_URI = "https://github.com/jaswchan/tar.gz"
S = "${WORKDIR}/app"
DEPENDS = "glic"
inherit autotool
do_compile () {
        ./configure
        make
}
```

description → Describes the software.

License → declare the license type

SRC_URI → where to fetch the source.

.S → source directory path

DEPENDS → Build time dependencies

inherit → Reuses logic (autotools, cmake).

do_compile () → Custome compile steps

---

6). How does yocto handle dependecies between recipes

Yocto uses two main keywords

*) DEPENDS → Build time dependencies.

↳ ensures a package is built before.

*) RDEPENDS → Runtime dependencies.

↳ ensures a package is installed in the final image

---

7) diff btw PACKAGECONFIG, DEPENDS, and RDEPENDS

Keyword	Purpose	Type.
PACKAGECONFIG	optional features toggles	Build - time config.
DEPENDS	what must be built before.	Build - time
RDEPENDS	what must be installed with.	Run - time.

Ex

PACKAGECONFIG ??= "ssl"

PACKAGECONFIG [ssl] = "--enable -ssl, --disable -ssl, openssl"

DEPENDS = "zlib"

RDEPENDS_${PN} = "bash"

---

8) What are DISTRO-FEATURES and how do they affect the build?

DISTRO-FEATURES are flags that enable or disable features globally in your linux yocto

ex:

x11 → Add x11 GUI support

wifi → Include wi-Fi tools

loc

conf/distro/mydistro.conf.

---

9) How can you add a custom layer in yocto?

1) Create layer:

yocto-layer create meta-mycustom.

2) Add it to bblayers.conf

BBLayers += "${TOPDIR}/../meta-mycustom"

3) Add your recipes

meta-mycustom/
└ recipes-example/
  └ hello/
    ├ hello1.0.bb
    └ files/

4) Build the BitBake

bitbake hello

Q) What are machine configurations in yocto?

machine configs tell yocto how to build linux for a specific board or SOC.

loc

meta-myboard/conf/machine/myboard.conf

key variables

MACHINE = "myboard"

SOC_FAMILY = "imx"

KERNEL_IMAGETYPE = "zImage"

UBOOT_MACHINE = "myboard_defconfig"

Intermediate Questions (customization & Troubleshooting)

1) How do you add a new package to an existing yocto image?

Steps

1) Find or write a recipe (.bb) for the package.

*) place it an layer : meta-my-layer/recipes-example/myapp/myapp_1.0.bb

2) Include the package in your image :

IMAGE_INSTALL += "myapp"

3) OR globally in local.conf :

IMAGE_INSTALL:append = "myapp"

4) Build it

bitbake core-image-minimal.

---

2) Diff btw IMAGE_INSTALL and CORE_IMAGE_EXTRA_INSTALL

Field	used in	Purpose
IMAGE_INSTALL	Image recipe	Primary way to define packages to include.
CORE_IMAGE_EXTRA_INSTALL.	local.conf (or) distro.conf.	Append packages without modifying image recipe

Ex IMAGE_INSTALL += "nano"

CORE_IMAGE_EXTRA_INSTALL += "htop"

3) diff btw do_configure, do_compile and do_install.

These are standard BitBake tasks executed in sequence

Task	Purpose.
do_configure	Prepares the build system
do_compile	Compiles the source code.
do_install	Installs compiled binaries into sysroot staging directory

Important Path variable

${S} - Source directory

${B} - Build directory

${D} - Install destination directory

example override.

```
do_compile () {
    oe_runmake
}
```

4) How can you override a task in a yocto recipe?

You can redefine or extend the task using.

*) do_task() → override.

*) do_task_append() → Add steps after default

*) do_task_prepend() → Add steps before default

Ex

```
do_compile_append () {
    echo "post-build step"
}
```

```
do_install() {
    install -d ${D}${bindir}
    install -m 0755 mybinary ${D}${bindir}
}
```

5) How do you debug a recipe that fails to batch a source tarball?

** Debug steps

1) Check logs

-> bitbake -c batch <recipename>

look in:

tmp/work/.../temp/log.do_batch *

2) Check SRC_URI

* Is the URL is correct

* Is protocol supported (git, http).

3) Use -DDD or verbose mode:

bitbake -DDD -c batch myapp

4) Disable checksum temporarily

BB_NO_NETWORK = "0"

6) How does yocto handle source mirrors and local file storage?

yocto stores downloads in

DL_DIR ?= "${TOPDIR}/downloads"

in yocto sometimes:

1. your offline
2) The website is down

So yocto uses a system of mirror and caching to

*) Avoid re-downloading files
*) use your local files first
*) work offline.

What are source mirrors

A mirror is an alternative place to look for source files before going to the internet

So yocto supports

1. PREMIRRORS – check before fetching from original SRC_URI
2. MIRRORS – checked after the original URL fails.

---

7) How can you make bitbake fetch source from a specific git branch or commit.

SRC_URI = "git://github.com/org.... ;branch=mybranch"
SRCREV = "abcdef1234567" # specific commit hash.

8) What is devtool and how can it help during development.

+) devtool is a powerfull CLI tool to

- L> modify recipes
- L> Build and test packages
- L> work on external sources trees.

Common devtool commands

```
devtool modify <recipe>
devtool build <recipe>
devtool finish <layer> <recipe>
devtool oreset <recipe>
```

---

9) How do you apply a patch to a recipe in yocto.

Step 1 place your patch in the files/directory.

meta-my-layer/recipes-example/helloworld/files/mybpx.patch

Step 2 update recipe

```
SRC_URI += "file://mybpx.patch"
```

Step 3 Add do_patch() if needed, else Bitbake auto applies it between fetch and compile

Step 4 bitbake -c cleanall helloworld
bitbake helloworld.

---

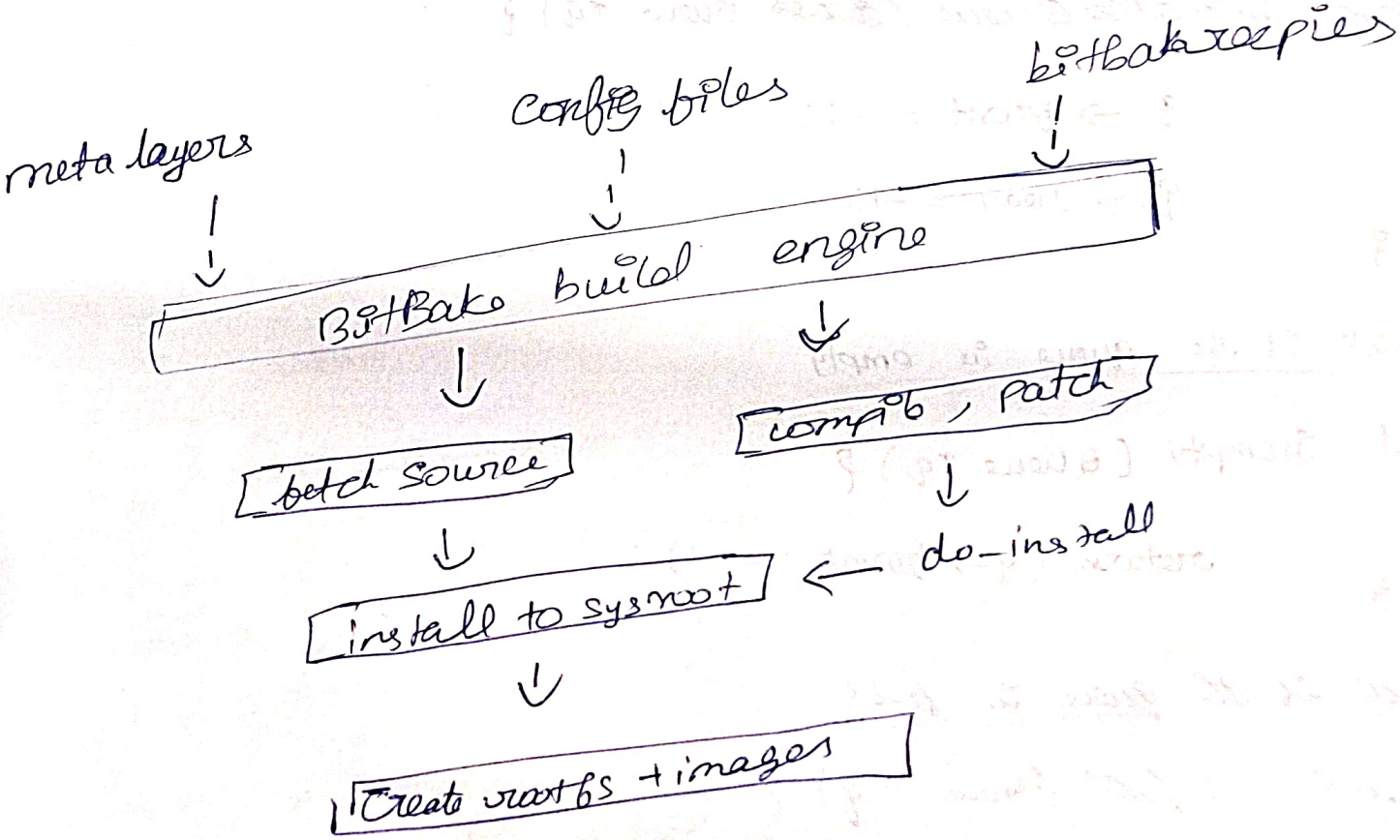
10) explain inherit in yocto and give example.

ans Inherit Bbtys in reusable classes that add build logic.

common classes

Class	Purpose
autotools	Adds ./configure && make && make install
cmake	Adds CMake support
systemd	enable systemd unit installation
update-rc.d	init script handling
kernel	linux kernal - specific rules
image	Required for image recipes
deploy	for deploying artifacts

Yocto Build flow



Yocto

Advanced Questions

Yocto

The yocto Project is an Open Source Project that helps you build custom linux distribution itself. It is a tool to build one tailored for your hardware.

1) How would you create a custom image with one specific packages:

A Yocto image is a complete root filesystem + Kernel + Bootloader. You can create a custom image recipe that includes only the package you want – no extras.

ex:

1) Create a new image recipe in your layer.

```
mkdir -p meta-mydynamic/recipes-core/image
cd meta-mydynamic/recipes-core/image
touch custom-minimal-image.bb
```

2) .bb

```
Summary = "custom minimal image"
LICENSE = "MIT"
inherit core-image.
Image_Install += "\ busybox \ dropbear \ mycalculator \ "

Image_FEATURES+= "ssh-server-dropbear"
```

3) edit conf

```
IMAGE_FSTYPES = "ext4"
```

And set it to build your custom Image

```
bitbake custom-minimal-image.
```

2) How do you build and Integrate a custom Kernel in Yocto?

Yocto uses Kernel recipes (linux-yocto, linux-stable) To use a custom kernel you write a new kernel recipe (or modify an existing one.

Steps

step1

mkdir -p metadynamic/recipes-kernel/linux && cd meta-dynamic/recipes-kernel/linux

step 2

linux-mycustom.bb

```
DESCRIPTION = "my custom kernel"
LICENSE = "GPLV2"
LIC_FILES_CHKSUM = "file://COPYING; md5=xyz.."

SRC_URI = "git://github.com/jawahar/linux-custom.git; branch=mybranch"

SRCREV = "abcd123"
S = "${WORKDIR}/git"
inherit kernel.
```

step3

PREFERRED_PROVIDER_virtual/kernel = "linux-mycustom"

step 4

bitbake virtual/kernel.

3) explain how you would add support for a new hardware board in yocto

Adding support for a new hardware board in Yocto involves creating a BSP to define the machine-specific configurations. The MACHINE variable in Yocto determines which board conf is used. affecting the Kernel, Bootloader and other components.

steps

step 1: Create a BSP layer

↳ yocto-layer create myboard.

Add this in bblayers.conf

```
BBLAYERS += "${TOPDIR}/../meta-myboard"
```

step 2: Define the machine configuration.

```
// Architecture
    TARGET_ARCH = "arm"
    DEFAULT TUNE = "armv7a"
// machine features
    MACHINE_FEATURES = "ext2 keyboard screen wifi"
    .....
// Kernel
    PREFERRED_VERSION_linux-mycustom = "5.15%."
    .....
// serial console
    SERIAL_console = "115200;ttyS0"
```

step 3 set up the Kernel.

step 4: configure the Bootloader.

step 5 update local.conf

step 6 Build the Image.

4) How do you generate an SDK for application development using Yocto.

Concept

Yocto lets you generate an SDK (tool chain + environment) tailored to your image. This is super useful for compiling app outside the Yocto tree.

Steps

1) generate SDK:

bitbake custom-minimal-image -c populate_sdk

Popular sdk builder.

2) output

You will get .sh installer file

poky-glibc-x86-64-custom-image-armv7at2hf-neon-toolchain-3.1.sh.

3) install it

chmod +x poky-*.sh

./poky-*.sh

4) use sdk

source /opt/poky/3.1/environment-setup-armv7a-poky-linux-gnueabi arm-poky-linux-gnueabi-gcc. myapp.c -o myapp

5) what is EXTRA_OECONF and how is it used

when a recipe inherits autotools, it call ./configure. you use EXTRA_OECONF to pass extra ./configure options

example

```
inherit autotools

EXTRA_OECONF += "--enable --disable-debug"
```

6) How does Yocto manage root file system generation and compression

ROOT FS is generated based on image recipe + installed packages. Then it's compressed using tools like gzip, bzip2, xz.

components involved

*) classes:

• -> image.bbclass -> builds rootfs.

-> rootfs - ${IMAGE_FSTYPES}.bbclass -> handles compression.

*) Types

IMAGE_FSTYPES = "ext4.gz tar.bz2 squashfs"

*) compression tools

Yocto uses fakeroot, tar, mksquashfs, etc.

7) explain the purpose of tmp and Sstate-cache directories.

tmp:

Temporary working directory for builds

contains

1) tmp/work → task work dirs

2) tmp/deploy → output image, SDKs

3) tmp/log → task log

Sstate-cache:

1) Shared state cache: reuse previous builds

2) Boosts rebuild speed

3) makes CI/CD (or) multiple machine builds faster.

8) How do you reduce the image size in yocto

1) minimal image:

start with core-image-minimal and add only what you need.

2) Remove debug symbols

INHIBIT_PACKAGE_DEBUG_SPLIT = ""

3) Strip binaries:

INHIBIT_PACKAGE_STRIP = "0"

4) Avoid locale data

IMAGE_LINGUAS = " "

5) Use compressed FS

IMAGE_FSTYPES = "squashfs xz"

9) steps to include a new init system.

Yocto defaults to sysVinit unless you explicitly choose systemd.

1) Add systemd to DISTRO_FEATURES:

In conf/local.conf:

```
DISTRO_FEATURES:append = "systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"
```

2) ensure Packages support systemd:

Some packages provide systemd unit files via pkgname.bb (or) pkgname.bbappend.

3) disable conflicting init

```
VIRTUAL-RUNTIME_initscripts = ""
```

10) ways to implement (OTA) updates in Yocto.

1) swupdate.	2) mender	3) RAUC
*) Designed for robust OTA	*) Full-blown OTA platform	*) lightweight, secure. OTA with update bundles
*) Add recipe meta-swupdate	*) uses A/B partitions and rollback.	*) Add meta-rauc.
*) Handles dual updates, U-Boot Integration	*) layer: meta-mender-core, meta-mender-<board>	

Advanced Debugging Scenarios

Q) How do you use BitBake -e and what insights can it give?

Ans

bitbake -e <recipe> dumps the entire environment BitBake sees when building a recipe.

Why it's useful

*) Shows final value of each variable

*) Helps debug why your SRC_URI, DEPENDS or do_install isn't behaving as expected.

ex

bitbake -e myapp | grep ^SRC_URI=

It shows where the source is being fetched from.

Q) How do you diagnose issues in a slow Yocto build?

Ans

Common slowdown causes:

*) unnecessary rebuilds due to recipe changes (or) missing cache

*) I/O bottlenecks

*) parallelism not fully utilized.

Tuning tips

*) Use BB_LOCK_CONFIG = "true" to analyze parallel jobs.

*) use bitbake -p to parse and benchmark layer parsing time.

*) check disk I/O, RAM, CPU

*) Check .sstate reuse

Tools

*) time bitbake core-image-minimal

*) iotop, htop, vmstat during builds

---

3) what causes a BitBake task to re-run when you didn't change anything?

Reasons:

*) Variable change [eg SRCREV, EXTRA_OECONF]

*) environment variables (eg. MACHINE, DISTRO)

*) Functions changed (do_compile, do_install)

*) do_batch (triggered by timestamp mismatch in downloads.

Solution

*) use bitbake -e <recipe> to inspect hashes

*) check task signature using:

bitbake-diffsigs <recipe> <old.sigdata> <new.sigdata>

---

4) How do you cache and reuse build artifacts across teams/machines.

sstate cache (shared state)

Yocto's magic for binary reuse

How it works

*) stores output of tasks like do_compile, do_install etc

*) can be reused across builds/machines if

↳ same machine + distro + env vars

↳ same BB_SIGNATURE_HANDLER. task

Team Hosting setup

SSTATE_MIRRORS = "file://.* http://myteam/sstate/PATH"

---

5) explain how BB_NO_Network and BB_HASHBASE_WHITELIST help debugging.

Ans

BB_NO_Network.

+) Forces BitBake to not use the network.

+) If You forgot to mirror / Download source it will throw a failure

+) Useful in:

+) CI builds

+) Offline validation

+) Reproducibility testing.

BB_HASHBASE_WHITELIST

+) Prevents certain variables from affecting task hashes.

+) Some variables change every build - causing unnecessary rebuild

+) Whitelisting them tells BitBake "Ignore them when hashing"

Ex:

BB_HASHBASE_WHITELIST ?= "DATE TIME BUILD_USER"

Real - world scenarios

6) If a Package is not appearing in your final rootfs image, how would you troubleshoot it?

StepBy-step

1) Check your image recipe:

Did you add the Package to IMAGE_INSTALL

IMAGE_INSTALL += "myapp"

2) Check if the recipe was actually built.

bitbake -s | grep myapp

3) Inspect the installed packages in the rootfs

bitbake -c rootfs <image>

Then look inside

tmp/work/<machine>/<os-image-*>/image/

4) Use oe-pkgdata-util to check runtime dependencies

oe-pkgdata-util list-pkg-files myapp

5) Maybe it was built, but removed by

IMAGE_FEATURES (like minimal)

1) Your Image Boots but network does not come up — how do you debug?

1) check ifconfig (or) ip a output

↳ is interface like eth0, wlan0 present

2) check drivers

is the right kernel module included

↳ dmesg | grep eth

↳ lsmod

→ check systemd or init logs

journalctl -xe.

4) was connman, networkmanager (or) busybox ifupdown in

↳ if not → no Network config to configure interfaces

5) check /etc/Network/interfaces (or) /etc/systemd/network.

↳ Maybe misconfigured or missing

1) is DHCP client included?

*) udhcpc

*) dhclient

2) use static IP if needed for quick fix

*) ip addr add 192.168.1.100/24 dev eth0

*) ip link set eth0 up

8) Diff [illegible] Prebuilt SDK vs building from Yocto tree

Feature	Prebuilt SDK (populate_sdk)	Building Yocto Tree
Toolchain	Prebuilt (cross compiler, sysroot)	compiled as part of native build.
Setup time	Fast for apps, easy to hand off to teams	Slower, full build context needed.
Customization	Limited - rebuilt SDK to reflect changes	Full control over all layers, dependencies
Use case	App developers, CI/CD pipelines	BSP/kernel/distro working on build system
Environment	exported via environment-setup-* script	Bit-Bake native

9) How do you handle license compliance in a Yocto build system?

Yocto has powerful tools to track and report licenses automatically.

1) LIC_FILES_CHKSUM:

a) every recipe must declare license file + checksum

LICENSE = "MIT"

LIC_FILES_CHKSUM = "file://LICENSE;md5=abcd..."

2) Set INHERIT += "license" in conf/local.conf.

3) generate license report.

bitbake my-image -c archive-lic

output will be in:

tmp/deploy/licenses/<image>/

4) controlled allowed licenses:

INCOMPATIBLE_LICENSE = "GPLv3"

---

10) How do you generate and deploy signed images & prevent unauthorized modification of images using cryptographic signatures.

1) signing Kernel / Bootloader

↳ use sign_image class (or) manual post-processing

↳ sign with RSA (or) ECDSA

2) secure update Frameworks.

↳ create signed bundles (.raucb, .swu)

↳ sign with GPG keys (or) CMS signatures

3) Root filesystem signing:

↳ use wic image signing

IMAGE_CLASSES += "image-signature"

SIGN-IMAGE_CMD = "gpg --detach-sign

$1 (DEPLOY_DIR_IMAGE)/$ [IMAGE_NA

OTA integration

Integrate with mender, RAUC, SWUpdate to enforce signed updates.

File System, Init, Bootloader and Deployment:

1) How do you create a dual-partition setup for A/B OTA with Yocto.

Goal:
A/B Setup = 2 identical rootfs partitions → one is active the other is updated → if update fails, rollback

Key components
*) .wic file defines partitions [wic image creator for Bootable media]
*) Bootloader support dual-rootfs boot logic
*) update mechanism switches partitions.

i) Create .wic layout with 2 rootfs partitions

bitbake WKS_FILE = "dual-rootfs.wks"

wic
Part --Source rootfs --ondisk mmcblk0 --label rootA --size 256M
Part --Source rootfs --ondisk mmcblk0 --label rootB --size 256M
Part --Source bootloader --ondisk mmcblk0 --label boot --active --alisn 4 --size 16M

ii) Enable A/B update.
↳ For RAUC
*) Create bundle with rootfs.0 and rootfs.1
*) Define which slot is primary/backup in system.conf.

iii) U-Boot logic to detect boot success / failure

use Boot environment variables

```
bootcmd = A;
```

- Bootcount, upgrade_available, etc.

iv) Persistent data in separate partition

- Add data partition to avoid overwrite of user Data.

---

2) How do you include and configure U-Boot for a custom Board.

steps

1. Add U-Boot to your Image:

PREFERRED_PROVIDER_u-boot = "u-boot-myboard

2) create custom U-Boot recipe

meta-myboard/recipes-bsp/u-boot/u-boot-myboard.bb

3) Set machine specific U-Boot configs.

UBOOT_CONFIG = "myconfig"

UBOOT_CONFIG[myconfig] = "defconfig,bin,flash

4) Patch U-Boot for board specific configs.

modify include/configs/myboard.h

Set memory, serial, bootcmd.

5) Add to Image.

IMAGE_FSTYPES += "wic"

3) what's the role of .wic files and how do you customize them

.WIC = Image Creator for Bootable media.

used for:

*) Define partitions (rootfs, boot, data)

*) Adding bootloaders (U-Boot, GRUB)

*) Flash-ready SD/eMMC images

1) Create Custom .wks file

Part --source bootloader --ondisk mmcblk0 --active --align 4 --size 16M

Part --source rootfs --ondisk mmcblk0 --label rootfs --size 256M

Part --source rawcopy --sourceparams="file=some_extra.img" --label mypart --size 50M

2) Add to your Image build:

WKS_FILE = "myimage.wks"

IMAGE_FSTYPES += "wic"

check with:

wic create myimage -e myimage.

4) How would you integrate TPM for Secure Boot into a Yocto based system.

Secure Boot

+) Sign u-Boot / Kernel / FIT images using GPG / RSA

+) Verify Signature in Bootloader.

+) Use Yocto Signing-keys, image-signing class.

TPM integration

1) Include TPM driver.

MACHINE_FEATURES += "tpm"

2) Add Software stack.

3) Secure Key storage, attestation app use TPM

4) Configure Kernel for IMA/EVM to validate integrity

5] How do you set default System services using SystemD in Yocto.?

Ans Use Systemd bbclass

DISTRO_FEATURES += "systemd"

VIRTUAL-RUNTIME_init_manager = "systemd"

To enable your service.

SYSTEMD_SERVICE_${PN} = "myapp.service"

SYSTEMD_AUTO_ENABLE_${PN} = "enable"

ex:

```
do_install() {
    install -D -m 0644 ${WORKDIR}/myapp.service
    ${D}${systemd_system_unitdir}/myapp.service
}
```

Advanced open Questions

6) How do you integrate third-party binary blobs into a yocto image.

Option 1 if it's a .bin (or) .so

1) Add to SRC_URI:

```
SRC_URI += "file://vendor_blob.bin"
```

2) Install in do_install():

```
install -m 644 vendor_blob.bin ${D}${libdir}/
                                    firmware.
```

option 2 if its a full SDK:

*) wrap SDK with custom sdk.bb recipe

*) Extract and deploy headers/libs during build.

*) Configure EXTRA_OECONF, CFLAGS, LDFLAGS for build.

7) How do you test a Yocto-build image using QE

Steps

1) Build QEMU image:

bitbake core-image-minimal

2) Run with QEMU:

runqemu qemux86-64 core-image-minimal

3) Test SSH, login, networking, etc.

[write tests using oeqa.runtime for automated checks]

8] How do you cross compile a C++ application outside the Yocto tree but link it to the rootfs?

Steps

1) Build SDK.

Bitbake core-image-minimal -c populate_sdk

2) Install SDK:

./tmp/deploy/sdk/*.....sh

3) Source environment

Source /opt/poky/*/environment-setup-*.

4) Build your app:

$CXX myapp.cpp -o myapp `pkg-config --cflags --libs

q) what would be good approach to ... based yocto system?

2 options

1) Host Docker / Podman inside Yocto

*) Add docker -ce (or) Podman to your image

*) use docker.service to run container apps.

2) Build container image using meta-virtualization

*) Create OCI containers via Yocto

*) use container-image.bbclass to output

.tar (or) .oci

---

q) How Do you maintain multiple products and hardware variants using same Yocto tree

use BSP + layer isolation.

1) use MACHINE variable to switch boards.

Source oe-init-build-env build-imx6

Source oe-init-build-env build-rpi4.

2) create custom layer.

meta-product1/

meta-product2/

3) use .bbappend, MACHINE_FEATURES, and DISTRO to control what's included

4) store components go in meta-common.

Benefits ① One Yocto tree ② Different outputs ③ Scalable CI/CD