31.07.25.

Yocto.

1. What is the yocto project and what is the core component?

The yocto project is an open source collaboration project that provides a tools and templates for creating custom linux distribution for embedded system. It is not a linux distribution but a build system that enable developers to create a tailored linux as for the specific hardware and application needs.

↳ Create a minimal, full-featured linux images for embedded devices

↳ Supports the cross compilation for different target architecture.

Core Components :-

- Bitbake :-
  - → In the Bitbake basically performs the same functionality as of make
  - → It is a task scheduler that parse Python and shell script mixed code.
  - → In the bitbake is a generic task execution engine that parse recipes and dependencies to build software package.

openEmbedded - core :- A core set of metadata (recipes, classes, configuration) shared across yocto based project.
refers to the

Metadata -> -> Meta data / a build instructions
→ commands and data used to indicate what version of software are used.
→ meta data is used to fix the bugs or customize the software for use in particular situation.
→ Files written .bb (Bitbake) format that define how to build software packages.

Layers:- Modular collections of metadata (recipe, configuration).

Poky :- The reference distribution of yocto. It include Bitbake, OE - core and some configurations.

SDK Generator :-
Allows generation of a cross development toolchain (SDK) for application development.

Devtool ⇒ A command line tool that helps developer easily patch, modify and test software in the yocto workflow.

2. What is the difference between yocto project and open Embedded ?

Feature	Yocto Project	Open Embedded.
What is	A collaborative project that provide tools and metadata to build the custom Linux distribution	A build framework for embedded linux system. Predates yocto.
Includes	Bitbake, poky (reference distro), OE-core, tools like Toaster / devtool.	Bitbake (originally developed by OE core), meta layers like OE-core.
Relation to Yocto.	Uses of openEmbedded core as the base meta data.	Provides OE-core, which is used by yocto.
Main tool.	Bitbake (borrowed from OE)	Bitbake (original creation).

2) Explain the role of bitbake in the yocto build system.

Bitbake reads and interprets various form of metadata, including recipes (.bb files), classes (.bbclass files) and configuration files. the metadata defines how software components fetched, configured, compiled, installed and packaged.

It similarly work make,

→ Determines which packages depend on others and in what order they must be built.

→ Run build step like do_fetch(), do_unpack(), do_compile(), do_install(), do_package().

Bitbake is the build engine used by yocto Project. It main role is to automate the process of building custom embedded linux images. It reads the metadata files, called recipes, which describe how to fetch, configure, compile and package software components.

Bitbake resolves dependencies between packages, executes build tasks in right order and finally assembles a complete root file system and creates images.

Bitbake take care of running these task efficiently, with supports for parallel builds and caching.

If I run "bitbake core-image-minimal" the bitbake analyze all required packages, build each one in order - starting from source fetching to final image creation..

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

A layer is a directory containing a collection of metadata (recipes, configuration files and classes) that tell bitbake how to build software.

Core layer → Basic recipes and configuration

BSP layer → Board support package - adds supports for hardware.

Distro layer → Customize linux distribution
Ex:- meta-poky)

layers help in structuring the build system in a reusable and maintainable way.

5. What is recipe (.bb file) in yocto? Explain its key fields?

The recipe provides the build instructions how to build the system..

key fields include DESCRIPTION, LICENSE, SRC_URI, Inherit.

```
DISCRIPTION = "nano text editor" // Short description of the software
HOMEPAGE = "https://www.nano-edit.org" // URL of the software home page
LICENSE = "GPLV3"
SRC_URI = "source location" // Where the source code will be download from
SRC_URI[sha 256sum] = "abcdef1234567"
S = "${WORKDIR}/nano-5.9" // The source directory inside the build
Inherit tools autotools // inherit predefined build logic from classes like
do_install() {
    make install DESTDIR = ${D}.
} // function to install the compiled output to the correct place
```

6. How does yocto handle dependencies between recipes?

yocto manage the dependencies between recipes using DEPENDS for built time and RDEPENDS for runtime.

Bitbake parse these fields to build dependency graph.

7. What is the difference between PACKAGECONFIG, DEPENDS, RDEPENDS?

variable	Purpose	affects	used for.
PACKAGECONFIG	Enable or disable optinal features in a package	Build time & runtime (Indirectly)	Feature flags (Eg. enable SSL, disable x11).
DEPENDS	Declares build time	. Build process	
RDEPENDS	Declares run time	Final Image.	

8. What are DISTRO_FEATURES. and how do they affect the build?

DISTRO_FEATURES is a variable in yocto. that defines the global features.

meta-poky/conf/distro/poky.conf.

Recipes can enable or disable feature based on what's in DISTRO FEATURES.

Common features :-

x11	Enable x window systems.
Wayland	
WiFi	Enable wifi support
bluetooth	Enable bluetooth hfp support.

Ex:-

DISTRO_FEATURES = "Wifi systemd alsa"

Q. How can you add a custom layer in yocto?

Step 1 :- create a custom layer

↳ It provides the structure of the layer.

- "bitbake-layers create-layer" this command to generate the structure, now this layer registered in bblayers.conf.

Step 2 :- Add layer to bblayers.conf.

conf/bblayers.conf.

Step 3:- Add recipe to your layer.

To add one recipe file like .bb extension.

Step 4:- Build and test.

bitbake hello.

Q. What are the machine configuration in yocto? Give Examples?

Machine Configuration defines the hardware specific setting needed to build an Image for a particular board or device.

The target Hardware (CPU, SOC, board).

Meta - <layer> / conf / machine

common fields on the machine configuration.

Variable	Description
MACHINE	Name of the machine
DEFAULT_TUNE	Architecture and tuning
KERNEL_DEVICETREE	Device tree binary used for booting
SERIAL_CONSOLES	serial port for login/debug.

Intermediate Questions :-

10. How do you add a new package to an existing yocto image?

Step 1 :- First ensure the package recipe is available. If not to modify the recipe image.

Step 2 :- add the package to the image.

create a new recipe in your custom layer.

Create a recipe file like a.

recipes - mypackage / mypackage/mp_pack.bb

Step 3 :- Add your package in your layers.

→ conf / bblayers.conf.

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

Step 4 :- Modify local.conf (qualk test).

IMAGE_INSTALL : append = " your package name".

location :- /build/conf/local.conf.

steps :- Rebuild the image
→ bitbake core-image-minimal.

12). What is the difference between IMAGE_INSTALL and CORE_IMAGE_EXTRA_INSTALL?

Feature	IMAGE_INSTALL	CORE_IMAGE_EXTRA_INSTALL.
definition location	Used inside image recipes (.bb file).	used in configuration files (local.conf).
Purpose	primary list of packages for an image.	Append extra pack. without changing image recipe.
Overridable	Yes, but more tightly controlled.	Yes, and intended for user customization
Typical use case	Defining base contents of a custom image.	Adding debug tools or temporary packages testing
Scope	part of image logic.	External to the image recipe.

13. Explain the difference between do_compile; do_configure, do_install.

Do_configure:-

Prepare the source code for building, usually by running configurations scripts like ./configure or cmake. Checks for dependencies and set up makefiles or build scripts.

configure build options.

do_compile :-

compile the source code into the binaries using

make.

→ converts source code into executables or

libraries.

→ uses toolchain specified by yocto.

do_Install :-

Installed the compiled binaries and resources into a temporary staging area (${D}) which is later packed into .ipk, .deb, .rpm.

each of these tasks is customizable in a bitbake recipe to handle different build system or special needs.

14). How can you override a task (eg do_compile) in a yocto project?

Syntax:

```
do_compile() {
        ## custom lines
}
```

Ex:-

Build a simple C application without using autotools, cmake.

```
do_compile() {
        ${CC} ${CFLAGS} -o myapp myapp.c
}
```

append to do_compile :-

```
do_compile : append ( ) {
        echo "custom step after compile" >>
                                    ↳ source directory.
        ${S}/compile_log.txt.
```

append :- to add commands after the default task.

15) How do you debug a recipe that fails to fetch a source tarball?

If a recipe fails during do_fetch(), check the error to find out whether it's a broken URL, a network issue or a checksum mismatch. Verify the URL, use bitbake -c fetch -V for more info and inspect the SR_URI. and checksum.

16. How does yocto handle source mirrors and local file storage?

local file storage (DL_DIR) :-

⇒ All downloads files are stored in the DL_DIR directory.

⇒ This local file storage prevents re-downloading the same files for future builds or different projects.

DL_DIR ?= "${TOPDIR}/downloads".

Mirrors in yocto :-

yocto allows using alternative locations (mirrors) to fetch sources before trying the original URLs.

TYPES :-

→ PREMIRRORS - checked before trying the actual SRC_URI.

→ MIRRORS → checked after a download attempt from the original URL fails.
→ Act as a fallback mirror.

14. How can you make BitBake fetch source from a specific Git branch or commit ?

To make BitBake fetch source code from a specific Git branch or commit in yocto, you configure the SRC_URI and SRCREV variables in your recipe (.bb file).

SRC_URI = " Repository & branch "

SRCREV = " " < commit hash > ".

18. What is devtool and how can it help during development ?

Devtool is a powerful command-line utility provided by yocto's extensible SDK and the core layer.

→ Work with recipes and source code more easily.

→ Add or modify packages outside the main yocto layers.

add a new recipe to use

→ devtool add <recipe name> <source url-dir>.

Modify existing recipe

→ devtool modify <recipe-name>.

Build modified recipe.

devtool build <recipe-name>.

Deploy to target

devtool deploy-target <recipe-name> <target-ip>

update the recipe with your changes.

devtool update-recipe <recipe-name>.

Finish and clean up.

devtool reset <recipe-name>

19). How do you apply a patch to a recipe in Yocto?

step1:- place your patch in the files/directory.

meta-my-layer/recipe-example/helloworld/files/myfix.patch

step2:- update recipe:-

SRC_URI += "file://myfix.patch"

Step 3:- Add do_patch() if needed, else bitbake auto applies it between fetch and compile.

step 4:-

bitbake -c cleanall hellocworld.

bitbake hellocworld.

20. Explain inherit in yocto and give examples (eg. autotools, cmake, systemd).

The inherit keyword is used in recipe, to reuse the common functionality provided by classes. The classes encapsulated build logic.

→ It includes predefined tasks and variables into your recipe.

Autotools :-

- inherit autotools.
  - ↳ Automatically defines do_configure, do_compile, etc.

cmake :-

- inherit cmake.
  - ↳ Defines how to call cmake and make.

Systemd :- inherit systemd.

↳ adds support for systemd unit files

↳ allows an automatic installation of service files.

↳ works with SYSTEMD_SERVICE_${PN} and SYSTEMD_AUTO_ENABLE.

Advanced Questions

21. How to create a custom image with only specific packages?

Step 1:- create a custom layer

yocto-layer create mylayer.

Step2:- Create a custom recipe

inside a layer to write a .bb file

Step 3:- Build your custom image.

bitbake my-custom-image.

22. How do you build and integrate a custom kernel in yocto?

It contains the two method.

↳ use a custom kernel source (eg. your Git repo)

↳ modify an existing kernel recipe (eg. linux-yocto)

Step 1:-

create a custom layer

Yocto-layer create mylayer

add this layer in the bblayers.conf.

bitbake-layers add-layer .../meta-layer

steps :- create a kernel recipe.

mkdir -p meta-mylayer/recipes-kernel/linux.

cd meta-mylayer/recipes-kernel/linux.

touch linux-myboard.bb.

steps:- to write a kernel recipe in the .bb file.

step4 :- add a defconfig

mkdir -p linux-myboard/myboard.

cp your-defconfig ./linux-myboard/myboard/defconfig.

steps :- modify the machine file.

PREFERRED_PROVIDER_VIRTUAL/kernel =
"linux-board".

step6:- Build the kernel.

↳ bitbake linux-board

↳ bitbake core-image-minimal.

23. 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 new machine configurations and possibly a custom BSP layer.

step1 :- create a custom layer

⇒ yocto-layer create my-bsp-layer.

⇒ bitbake-layers add-layer ../meta-my-bsp-layer.

Step 2:- Create a Machine configuration files.

mkdir -p meta-my-bsp-layer/conf/machine

touch meta-my-bsp-layer/conf/machine/myboard-conf

Step 3:- Add a kernel Recipe.

create a directory and write a kernel recipe in the .bb file.

Step 4: TO add the bootloader in the recipe file.

recipes-bsp/u-boot/u-boot_myboard.bb

and add

PREFERRED-PROVIDER_u-boot = "u-boot-myboard"

Steps :- Build the image.

Source oe-init-build-env.

to edit conf/local.conf :-

MACHINE = "myboard"

=> Bitbake core-image-minimal.

22). How do you genrate an SDK for application development using yocto?

BitBake to create a cross compilation toolchain tailored for your image and target hardware.

Step1:- Build image

Bitbake core-image-minimal.

Step2:- Cremate the SDK.

Bitbake core-image-minimal -c populate_SDK.

Step3:- Find the SDK installer

tmp/deploy/sdk/.

Step4:- Install the SDK on the host.

Step5:- Set up Environment for cross-compilation.

this stage to set the cc, cxx, LD and

8. other environment variables for cross compiling.

Step6:- Start Application development.

$cc hello.c -o hello.

Q5) what is the EXTRA_OECONF and how is it used?

EXTRA_OECONF is a bitbake variable used in

Yocto recipes that inherit the autotools class.

when you build software that uses the

GNU autotools build system.

Yocto autotools class automatically runs ./configure

-e.

Ex:-

Enable or Disable specific feature, specify

dependency path or toggle optional modules

26). How does yocto manage root file system generation and compression?

yocto uses Bitbake tasks like do_rootfs. to genrate the root file system and do_image to create compressed image formats. It installs all selected packages from IMAGE-INSTALL into a staging directory, resolves dependencies and build the final rootfs. the output format is is controlled by the IMAGE_FSTYPES. Variable.

27) Explain the purpose of tmp and sstate-cache directories.

The tmp- directory is the main build output directory in yocto. It contains all the intermediate and final build

→ compiled packages and binaries (tmp/work).
→ Final images and SDK's (temp/deploy).
→ The sysroot for host and target (tmp/sysroots)

The state-cache stores precompiled tasks outputs and metadata. This allow the build system to reuse results.

→ To speed up rebuilds by avoiding redundant works.

→ State-cache can be shared across machine or builds to save time.

28) How do you reduce the Image Size in Yocto?

-> Removed unused packages

↳ Review and minimize what goes into your Image.

↳ Edit your IMAGE_INSTALL variable in the Image recipe or local.conf.

IMAGE_INSTALL:remove = "_____"

↳ This step do follow the necessary

Examination

bitbake core-image-minimal.

↳ This is used to what you need

that file only to add the Yocto Image file

Enable Image compression

-> Use final image to compress like gzip, xz or lz4.

IMAGE_FSTYPES = "ext4.gz"

29. Describe the steps to include a new init system (eg systemd) into your image?

To include a new init system like systemd in a Yocto Image, you need to configure both your machine setting and Image recipe to use and support systemd.

30). what are the ways to implement over-the-Air (OTA) updates in Yocto-based system?

Implementing the over the-Air updates in a Yocto-Based system involves integrating update frameworks or building your own update mechanism to deliver new firmware, kernel or rootfile system to embedded devices.

OTA update methods in Yocto.

↳ Swupdate (Software update)

↳ RAUC (Robust Auto update controller).

↳ Mender

↳ OSTree + Systemd.

↳ custom scripts mechanisms.

---

31). How do you use bitbake -e and what-° oneigets can it give?

bitbake -e gives to expanded the values of all variables

SRC_URI ⇒ source location

FILEEXTRAPATHS → path used to find patches or files.

DEPENDS → recipe dependencies.

BMAGE_INSTALL →

32) How do you diagnose Issues in a slow Yocto build ?

→ Monitor System Resource

TO checking CPU, RAM, I/O and swap.

usage using like tools htop, iotop, vmstat.

→ check Parallel Build settings.

BB_NUMBER_THREDS and PARALLEL_MAKE.

are configured to utilize available CPU cores fully.

→ Profile the Build.

This systematic approach has helped me significantly reduce build times and pinpoint inefficiencies in complex Yocto-based projects.

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

A BitBake task can re-run even when I didn't change the recipe code directly, due to few key reasons.

→ changed Input Variable

BitBake tracks the values of variables like SRC_URI, EXTRA_OECONF, DEPENDS. etc.

→ To debug such rebuilds, typically run bitbake -diffsigs or inspect the task logs in temp/log.do_<task> to identify the root causes and optimize the build.

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

To cache and reuse build artifacts across teams or machines in Yocto, we use a combination of shared sstate-cache, downloads cache, and optionally a build artifacts server like NFS, HTTP or SSTATE_MIRRORS.

35). Explain how BB_NO_NETWORK and BB_HASHBASE_WHITELIS help debugging.

Both BB_NO_NETWORK and BB_HASHBASE_WHITELIST are helpful tools for debugging and ensuring build reproducibility in Yocto.

BB_NO_NETWORK :-

↳ Prevents BitBake from accessing the network during a build.

BB_HASHBASE_WHITELIST :-

↳ Prevents certain variable changes from triggering task re-execution by whitelisting them.

→ Helps me avoid unnecessary task rebuilds by ignoring benign variable changes.

26). If a package is not appearing in your final rootfs image, how would you troubleshoot it?

Combining BitBake tools, log analysis and an understanding of package dependencies and split packages, I systematically trace why a package didn't appear in the final image.

27. your image boots but network does not come up - how do you debug?

→ the image boots but the network does not come up, I approach the problem methodically as follows.

→ ip link.

→ ifconfig -a.

28. Explain the difference between using a prebuilt SDK vs. building from yocto tree?

The main difference between using a prebuilt SDK and building from the yocto tree lies in development scope, flexibility, and complexity.

Prebuilt SDK :-

used for application development without needing the full yocto build system.

→ Include cross-toolchain, target sysroot, and environment setup scripts

→ Useful for developers just building apps or libraries for a target image.

→ can be distributed to teams independently of the yocto tree.

Building from the Yocto Tree

↳ used for full system development - image, recipes, kernel, drivers, rootfs etc.

↳ full control over the entire embedded linux stack.

↳ you can add/remove packages, tweak layers, modify the kernel or init system.

39. How do you handle license compliance in a Yocto-build system?

→ Yocto provides built-in mechanisms to help track and manage license compliance.

→ To using Yocto's license infrastructure, I can automate license tracking, restrict unwanted licenses, and generate reports and artifacts required for legal compliance and product release.

40. How do you generate and deploy signed Images in yocto?

To generate and deploy signed Images in yocto, I follow a process that typically includes signing the bootloader, kernel or root filesystem - depending on the platform's secure boot implementation. The overall steps are.

→ Enable Image signing in yocto
→ sign the bootloader
→ sig the kernel and Device Tree.
→ create and sign Root Filesystem Images.

41. How do you create a dual-partition setup for A/B OTA with yocto?

To implement an A/B OTA partioning scheme in yocto, I create a custom .WIC image layout with two root filesystem partitions and integrate an OTA client (like RAUC, Mender, or SWupdate) to manage the switching and updates.

→ Design the partition layout.

To create a custom .wic file that defines the A/B scheme.

→ create a custom WIC Image type.

local.conf.

⇒ use a bootloader that support A/B boot switching.

⇒ Integrate an OTA update Framework.

RAUC → A flexible update framework that works well with Yocto.

Mender → Provides full A/B OTA with rollback and hosted management.

⇒ Manage Active / Inactive partition logic.

⇒ Installs update to the inactive rootfs

⇒ Marks it as bootable.

⇒ Set rollback mechanism if new rootfs fails boot

10. How do you include and configure U-Boot for a custom board?

⇒ To include and configure U-Boot for a custom board in Yocto, I follow these steps.

⇒ Add a U-Boot recipe:-

Yocto typically provides U-Boot through the meta or vendor BSP layer.

⇒ Configure the U-Boot Build for the board.

⇒ Include U-Boot in the Image build

⇒ Flash layout and deployment.

43). What's the role of .wic files and how do you customize them?

.wic files in Yocto are used to define how the final disk image is structured. They describe the partition layout, filesystem types, bootloader placement and where content is written such as rootfs, boot or data partitions.

44. How would you integrate TPM or Secure Boot into a Yocto-based system?

=> To integrate TPM or secure Boot in yocto I follow seperate but complementary strategies as they target different parts of the secure boot chain.

TPM (Trusted platform Module) Integration :-

TPM enables measured boot, key storage, attestation and more.

Secure Boot Integration :-

Secure Boot ensures that only trusted bootloaders and kernels run.

45. How do you set default system service using systemd in Yocto?

To set default systemd services in a Yocto-build system, I ensure the services are installed and then enable or mask them using the SYSTEMD_SERVICE and SYSTEMD_AUTO-ENABLE variables in the recipe or image configuration.

46. How do you integrate third-party binary blobs into a yocto image?

To integrate third-party binary BLOB blobs like GPU firmware into a yocto image, I typically create a custom recipe that fetches, installs and licenses the binary correctly.

47. How do you test a yocto-built image using QEMU?

Yocto provides a built in support for testing images QEMU. I follow these steps to test a yocto built image in QEMU.

→ Build a QEMU-compatible Machine and Images
⇒ Run the image using runqemu.
⇒ Interact with the system.

48. How do you cross compile a C++ application outside the yocto tree but link to the rootfs?

This compilation process first need Yocto SDK

Step1:- Build the SDK from yocto.

bitbake <your-image-name> -c populate_SDK.

ex:- bitbake core-image-minimal -c populate SDK.

Step2:- Install the SDK.

chmod +x poky-*.sh.

./poky.sh.