1. **Preparation**
2. **Overview & Architecture**
3. **File System**
4. **Development Operation**
5. **DI & Object Manager**
6. **Event**
7. **Module Configuration**
8. **Preparation**

Magento 2 requires the familiar LAMP stack (Linux, Apache, MySQL, and PHP) along with PHP Composer

* **Installation**

Cloning repository

Using Composer

* **Magento 2 Mode**

Default

Developer

Production

1. **Overview & Architecture**

**Magento Overview**

* Magento 2 continues to provide the same flexibility and extensibility of Magento 1,Built on a new and modern technology stack, Magento 2 integrates better with third-party solutions, and is more accessible and open to frontend developers.
* With an improved implementation process, partners and merchants will realize faster deployments, simplified upgrades, and faster return on investment.

There were six key goals in designing and producing the Magento 2 platform:

* Update the technology stack
* Improve performance and scalability
* Reduce upgrade efforts and costs
* Simplify integrations
* Provide high-quality tested code, testing resources, and documentation (and increase engagement with the Magento community)

**Magento 2 Platform & Architecture**

1. **Magento 2 Areas**

* Adminhtml
* Frontend
* Crontab
* Rest web api
* Soap web api
* Install

1. **Magento 2 Essentials**

* Modular code structure
* Themes
* Layout files
* Merged config file
* Object Installation "magic"
* Naming conventions for controllers and layout
* Event & Plugins

1. **Magento 2 Components**

* Configuration(app/etc/) ex. env.php, modules.php
* Framework (lib/internal/Magento/Framework)
* Modules (app/code/Magento)
* Command-line Tool (bin/magento)
* Themes (app/design)
* Dev Tools(dev)

1. **Magento 2 Paths**

* **Modules**

Using composer(vendor/magento/module-\*)

Cloning repo(app/code/Magento)

* **Framework**

using composer (vendor/magento/framework)

Clone repo(lib/internal/Magento/Framework)

* **Themes**

Using composer ( vendor / magento / theme-frontent-\* , vandor / magento / theme-adminhtml-backend ) clone repo ( app / design / adminhtml | frontend / Magento)

1. **Magento 2 File Types**

* Configuration File(XML files & a few PHP file)
* PHP Classes
* Layout instaruction (\*.xml)
* Templates (\*.phtml)
* JavaScript modules(\*.js)
* Javascript Templates(\*.html)
* Static assets(css & images)
* **Configuration File (XML files & a few PHP file)**

1. **Global** - Magento instance configuration like magento > store > configuration
2. **Module** - Defains a module's behavior (ex. routes & di,events)
3. **Theme** - Defains name of theme and it's parent theme details and configuration

* **PHP Classes**

1. Model, Resource and Collection (use for interact with database add,update,delete)
2. Api interfaces (use to create api oprations & CRUD oprations)
3. Controllers (handle web request)
4. Block (it's represet part of the page, it's connected to template file)
5. Observer (Event class container)
6. Plugins (use of plugin around any public method of any class. A developer can modify a method’s behavior by using plugins , before - after - around)
7. Helpers (encapsulate some useful functions)
8. Setup [install / upgrade] (upgrade a database schema, or add data to a database)
9. UI Components (Magento 2 allows a developer to create a component -- an independent element on adminhtml)
10. **Enable Custom Code**

* Create & register a module
* Run bin/magento setup:upgrade
* modify core classes by plugins & preferse
* create observer
* controllers
* system configuration

1. **Magento 2 Module**

* Module is basically a package of code
* Module encaptulate business feature & set of features
* Module containing .php and .xml files (ex. block, controller & model, ....helpres...etc)
* Location

1. app/code/<vandor>/<modulename>/
2. vendor/<vandor-composer-namespace>/<module\_name>/

* Naming a Module

1. Namespace(Vendorname)
2. Module (Your modulename)
3. **Magento 2 Registering a Module | Empty Module Structure**

* Two file require to register module to magento 2 module.xml and registration.php files.
* **module.xml**

- Modulename

- Version

- Dependency other module or source & set sequence of module

* **registration.php**

It's use to add module to nonComposerComponentAutoloader.php file & register the this module to magento component (using Magento \ Framework \ Component \ ComponentRegistrar :: register('module','vendor\_modulename',\_\_DIR\_\_))

1. **Magento 2 Module Dependencies**

* The concept of dependencies, or using and being dependent upon another module’s features, is important in Magento.
* Dependency usually means that a module is loaded after another module.
* Modules can be dependent upon the following components:

1. Other modules
2. PHP extensions
3. Libraries (either Magento Framework Library or third-party libraries)

* Types of Module Dependencies

1. Hard Dependencies (module cannot work without depended module)
2. Soft Dependencies (module is able to work without depended module)
3. **File System**

* **app**: includes core code (in the case of repo cloning installation), custom modules, themes, and global configuration.
* **bin**: has the “magento” script, which is the Magento 2 command-line tool described earlier.
* **dev**: includes different scripts and tools useful for developers.
* **lib**: includes external libraries not installed through the composer. For the repo-cloning installation, it includes the Magento framework: lib/internal/Magento/Framework.
* **vendor**: the composer folder, which contains all the dependencies. It may also include Magento framework and core modules for composer-based installation.
* **pub**: folder that should be public on the host. This folder has a static subfolder where all static files need to be placed in production. Static files can be located in themes and modules, so moving them all to pub/static is not a trivial task. There is a special command available within the command-line tool which allows for that. However, as it takes a while to accomplish this, and is not convenient for development, there is also a static.php file that locates and returns static files from the modules and themes.
* **setup**: folder that contains installation-specific files.
* **var**: an important folder, where cache, generated code, logs and other files (like uploaded csv files) are stored.

1. **App Folder Contents**

app/etc: includes global configuration.

app/code: custom modules, and in the case of cloning-repo installation, core modules.

app/design: themes.

app/i18n: specific translation files.

app/Bootstrap.php, app/autoload.php, and app/functions.php are three special PHP files that must be included at the very

beginning of execution process.

1. **Framework & Core Modules**
2. Core modules (app/code/Magento/\* | vendor/magento/module-\* )
3. Framework module ( lib / internal / Magento / Framework/ | vendor / magento / framework)
4. **Core Module contains for**

* Backend
* Bundle
* Catalog
* Checkout
* Customer
* Cms
* Eav
* Index
* Install
* Payment
* Sales
* Tax
* Whishlist ....and more

1. **Framework module contains for**

* Acl
* Api
* App
* Archive
* Authorization
* Autoload
* Backup
* Cache
* Component
* Composer
* Config
* Console
* Controller
* Convert ...... and more

1. **Magento 2 Module Structure**

* Api
* Block
* Console
* Controller
* etc
* Helper
* i18n
* Model
* Observer
* Plugin
* Setup
* Test
* Ui
* view
* registration.php...... and more

1. **Magento 2 Module View File Types**

* Ui Components
* Tamplates
* Layout
* Static files
* Javascript modules
* Javascript templates

1. **Magento 2 Module tempate area**

* Frontent
* Adminhtml
* Base

1. **Development Operations**
2. **Modes**
3. **Command-Line Interface**
4. **Caching**
5. **Magento 2 Modes**

* developer
* production
* default

1. **Developer Mode in Magento 2**

* This mode for use module development time
* Visible error message
* Static file generated every time
* Exceptions are displayed in the browser
* System logging in var/report is highly detailed in this mode.

1. **Production Mode in Magento 2**

* Exception is not display to browser
* Not generate static file
* Docroot is only readonly permission
* Error is store in error log file

1. **Default Mode in Magento 2**

* Do not display excption to browser & write to log file
* Auto static file is create if you pass any frontend or backend request

1. **Maintenance Mode in Magento 2**

* Maintenance mode is used when you want to make the site unavailable to the public during updates or other changes.
* Create a flag file (var/.maintenance.flag) to enable the mode

1. **Command-Line Interface**

* This magento feature base on symfony, that perform both installation & configuration
* The new interface performs multiple tasks, including:
* Install Magento
* Clear cache
* Manage indexes
* Generate non-existent classes (factories, interceptors for plugins, DI configuration for the object manager)
* Deploy static view files; clear static files (to comply with static file fallback rules)
* List & Enable/disable available modules: ... & more

1. **Caching**

* Caching in Magento 2 is similar to Magento 1.
* Cache type are located in app/etc/env.php
* Ex. Layout XML, HTML Block Output caches, Page cache...etc
* Cleaning cache

1. From Admin (System > Cache Management)
2. Using command line (bin/magento cache:clean | cache:flush)
3. Remove cache file (rm -rf var/cache | var/page\_cache | var/view\_procecess)
4. **DI & Object Manager**
5. **Dependency Injection**
6. **Object Manager**

1. Dependency Injection

- Dependecy is inject object to create object instance, It's depend on other module class & it's assign to custructor or methods perameter

Class Instantiation in Magento 2

- constructor (original) class is key. It contains a list of declarations (dependencies) that could be other classes, interfaces, classes, factories, and others..

Different Classes Instantiation

1. Singleton-type classes ...insted by di

2. Entry classes ... created by factory or repositories.

3. Factories ... auto-genrated classes

2. Object Manager

- Creating Object

- Implementing singleton pattern

- Maneging dependencies

- Automatically instanting parameters

defains:

- Paramenter : varialable declared the custructor signature.

- Arguments : value passed to the custructore when class instance is created.

Object Manager method

1. get() - get/create singleton object

2. create() - create new object for every time

Object Manager Auto-generated Classes

1. Factory (auto generated class)

2. Controller (auto generated class)

- Generated code is located in var/generation & generation folder for (2.2.X)

- Auto generated class include "Interceptors" & "Proxies"

Interceptors

- Allows the plugin functionality to work.

Object Manager Configuration

- di.xml is define configuration which object(instance) to deliver into the constructor of a class

- Each module have multiple di.xml files (with deffrent area)

1. **Plugins**

- Magento provied the best feature is plugin.

- But plugin only create in public method or class

- final or private & protected method dose not create plugin

- plugin dose not conflict with each other because it's run before or after.

- Declare plugin

<config>

<type name="Objecttype">

<plugin name="Pluginname"

type="Pluginclassname"

sortOrder="1"

disabled="true" />

</config>

1. Object Type: A class, interface, or virtual type

2. Pluginname : name of plugin

3. sortOrder : Order to call this plugin

4. disabled : set true or false

1. Before Plugin (inject before call Object method)

2. After Plugin (inject after method called)

3. Around Plugin (replace the original method functinality)

1. **Events**

- Magento 2 provied event same as magento 1. Magento 2 include some more events in that list

1. **Module Configuration**

1. Configuration File

Predefined configuration file include : config.xml, module.xml, di.xml, events.xml, menu.xml, system.xml, routes.xml, crontab.xml....etc

app/etc/config.php : declaration of all modules

app/etc/env.php : database connection and magento cache & mode configuration

etc/config.xml : set default value of admin configuration.

etc/di.xml | etc/[area]/di.xml : configuration about dependency injection

etc/events.xml | etc/[area]/events.xml : event configuration

etc/[area]/routes.xml : routes configuration

2. Storage of configration

1. core\_config\_data [database]

2. config.xml, acl.xml, events.xml ...etc

1. **Configuration scope**

1. Globle

2. Frontend

3. Admin

4 Configuration Load order

- Primary configuration : load only config file need for app start & installation-spacific configuration

- Global configuration : include config file comman accross all app area from all modules

- Area specific : adminhtml or frontent configuration

5. Creating a Custom Config File

1. XML file

2. XSD Schema

3. Config php file

4. Config reader

5. Schema locator

6. Converter