1. CI
2. Structure

* Application: source code of app
* config: contain all configuration files of application
* autoload.php: load libraries, helpers, models… automatically when a controller is initialized
* config.php: set base\_url,…
* routes.php: set default\_controller, re-map url,
* database.php: setting need to connect database
* cache: store the precessed data, support quick load, increase the speed of pages access
* **controllers: file control**
* **models: class entities access with database**
* **views: store files UI to render**
* **core: contain file to custom core CI**
* helpers: function support in application made by you
* libraries: class written by you
* language: support multi-languages
* third\_party: outside plugins
* logs: track application errors/exception
* hooks:
* System: core CI
* core:
* database:
* fonts:
* helpers:
* libraries:
* language:
* Custom core class by naming prefix: MY\_
* User\_guide: CI documentation

1. Usage:

* Route: [controller]/[action]/[params]
* Always extends CI\_Model/CI\_Controller for custom model/controller
* Use model in controller:

$this->load->model([name\_of\_model]);

$this->[name\_of\_model]->[action];

* Use library in controller:

$this->load-> library([name\_of\_library]);

$this->[name\_of\_library]->[action];

* Use helper in controller:

$this->load-> helper([name\_of\_helper]);

Function\_in\_helper();

* Load view in controller:

$this->load->view([name\_of\_ view], [data\_array]);

1. Integrate 3rd party

* 3 steps:
* Copy folder 3rd party library and parse into folder application/third\_party
* Create a library (in application/libraries) that is used to connect this 3rd party with CI

Include APPPATH.[path\_to\_3rd\_party\_library];

Class [library] extends [3rd\_party\_library] {...}

* Using 3rd party library like libraries in core CI

$this->load->library([name\_library]);

* Using composer require package
* Install through composer

composer require [name\_of\_package]

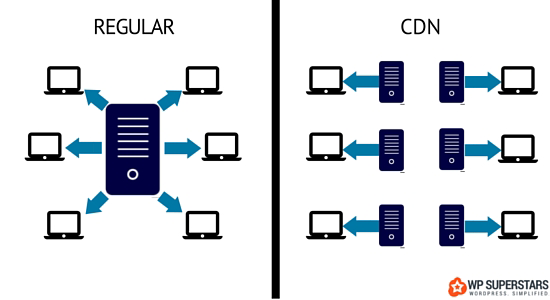
* Require vendor/autoload.php into file that need to use classes/entities in package
* Can make your own class to connect

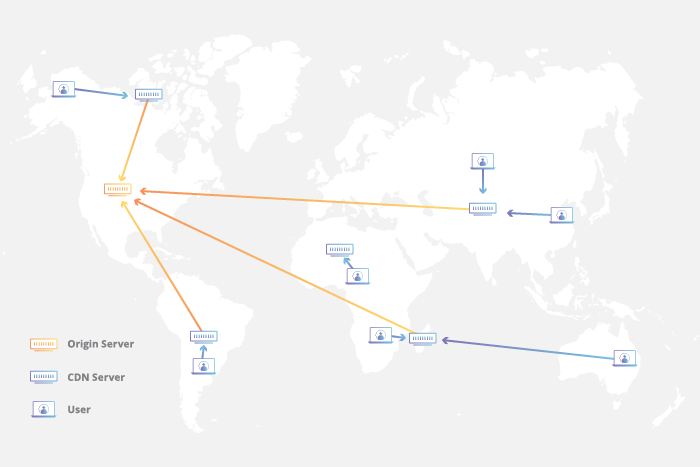
1. Smarty
2. Webapp
3. Web app



1. Web database
2. CDN

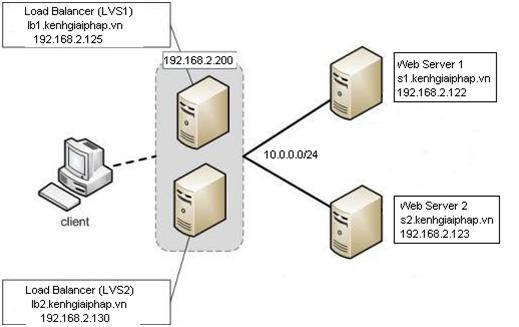
* Content Delivery Network (mạng phân phối nội dung)
* Giải pháp tiết kiệm băng thông máy chủ và tăng tốc độ website
* Lưu bản sao (cache) của các nội dung tĩnh bên trong website tại nhiều địa điểm khác nhau (thường được gọi là PoP – Point of Presence) và từ các PoP đó nó sẽ gửi tới cho người dùng khi họ truy cập vào website.
* TP chính:
* PoP: point of presence: nhiều caching server
* Caching server: lưu trữ và truyền tải các File đã được Cached
* Phân loại:
* Pull http/static: PoP CDN tự động truy cập tới web và lưu lại bản sao toàn bộ nội dụng tĩnh bên trong website
* Post/push: tải các nội dung cần phân phối qua CDN lên máy chủ của họ qua các giao thức FTP/HTTP
* Streaming: hỗ trợ phát trực tiếp video
* Lợi ích:
* Tiết kiệm bang thông cho máy chủ gốc
* Tang tốc độ truy cập
* Tiết kiệm dung lượng lưu trữ trên máy chủ
* Improve security
* Nên sử dụng:
* Máy chủ đăt xa người dung
* Lượt truy cập lớn/trên nhiều quốc gia





* Load balancing (can bang tai) vs Fail over (chiu loi)

Solution: layer 4 switching



1. Header status code

* 200
* 300
* 400
* 500

1. Integrate HMVC CI – Smarty
2. Structure



Find template file

1. Hoạt động

Find & Add module paths

Locate module

routing

Auto load module

loader

Index.php

caching

compile

render

Assign data to template

Create\_template

Output->append\_output

Smarty->fetch

Parser->parse

* Output class:
* To send the finalized web page to the requesting browser. It is also responsible

for [caching](https://www.codeigniter.com/user_guide/general/caching.html) your web pages, if you use that feature.

* This class is initialized automatically by the system so there is no need to do it manually.
* Parser class:
* can perform simple text substitution for pseudo-variables contained within your view files. It can parse simple variables or variable tag pairs
* Loader class:
* is used to load elements
* libraries (classes) View files, Drivers, Helpers, Models, or your own files

1. Demo
2. DB
3. InnoDB vs MyISAM
4. Fulltext search indexes

<http://forum.gocit.vn/threads/mysql-full-text-search.323/>

CREATE TABLE opening\_lines (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

opening\_line TEXT(500),

author VARCHAR(200),

title VARCHAR(200),

FULLTEXT idx (opening\_line)

) ENGINE=InnoDB;

CREATE FULLTEXT INDEX idx ON opening\_lines(opening\_line);

SELECT COUNT(\*) FROM opening\_lines WHERE MATCH(opening\_line) AGAINST('Ishmael');

<http://www.kode-blog.com/codeigniter-hmvc/>

<https://www.smarty.net/about_smarty>

<http://www.php-dev-zone.com/2015/08/introduction-to-smarty-template-engine.html>

<https://www.theprimacy.com/blog/application-development-with-php-codeigniter/>

1. Output cache
2. Syntax:

$this->output->cache(time\_to\_live);

1. Usage:

A page request is made. Codeigniter (before very much of the framework has even been loaded) does a hash of the current url and if it finds that filename in the cache directory, it serves that.

The only way you can get fresh data is to manually delete the files. When codeigniter doesn't find the file from the hash it generated, it dynamically creates the page.

Codeigniter's implementation is called "full page" caching.

* Normal URL vs query string
* Normal URL: controller/action/params

Refresh and create new cache file if the params change

* Using query string: controller/action?params=value

Render the old cache file and no data change if this params modified

To use cache like normal, set in config.php file:

$config['cache\_query\_string'] = TRUE;

* Type of cache:
* Full file
* Object (cache driver)
* Database