

# Web Development Platforms

## A comparative study

Ayaz Ahmed Khan

February 26, 2007

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Web Frameworks, Templating Systems, and Python, oh my!</b>	<b>2</b>
2.1	CherryPy and Cheetah . . . . .	2
2.1.1	CherryPy . . . . .	2
2.1.2	Cheetah . . . . .	4
2.2	Pylons . . . . .	5
<b>3</b>	<b>Perl and CGI</b>	<b>6</b>
<b>4</b>	<b>Ruby on Rails</b>	<b>7</b>
<b>5</b>	<b>So?</b>	<b>8</b>

# 1 Introduction

A web developer should not be concerned much with anything other than the logic that goes into building web applications. The details of how incoming requests and messages are handled, processes are initiated and managed, which part of application is invoked when, etc, only rest to clutter up the time of a web developer. That is why there are web frameworks. A web framework takes care of all the low-level details, provides layers of abstractions, and leaves the developer to concentrate on the actual application.

What follows is a brief result of a not so detailed comparative study which took into consideration one or more web frameworks from a couple of different platforms. Deciding which web framework for which platform is best is not only nearly impossible, but is an exercise in futility. In the end, almost always, the decision of which web framework and platform to use boils down to personal taste and, more importantly, familiarity with web frameworks and how well they size up in providing a means to achieve requirements and final goal. To further emphasize, this study draws no conclusions other than what has already been highlighted, and serves only as a small reference touching upon some of the web frameworks being used popularly.

## 2 Web Frameworks, Templating Systems, and Python, oh my!

They say that *“Python has more web frameworks and templating systems than reserved keywords.”* And it is true. If you stroll through the lanes of the World Wide Web, looking for a web application framework and templating system to use to build a web application in Python, suffice it to say, you will find yourself overwhelmed by the dearth of web frameworks and templating systems available.

There are simply too many web frameworks in Python to choose from. It may seem like a good thing to have a lot many different web frameworks to pick from, and it is a good thing, but it also acts as a double-edged sword in that it is extremely difficult at times to decide which web framework is best for your needs. If you are familiar with a single framework and biased towards it, good for you. But if you have to sit down, review most, if not all, of the large and advanced frameworks there are, you would find the exercise annoyingly exhaustive and futile. There is no best or worst. What is best is what works for you, your particular requirements, and your particular environment. And what may be best for you might hardly be satisfactory for another. [Python Web Frameworks]

The following subsections address two powerful, easy to use web frameworks in Python. One notable templating engine, which has stood out from others, is also taken into account.

### 2.1 Cherrypy and Cheetah

#### 2.1.1 Cherrypy

Anyone who plans to do serious web development in Python *must* give Cherrypy a good look. Cherrypy is a pythonic web framework that makes it simple to develop web applications in an object oriented manner. The best way to

appreciate what CherryPy offers to web programmers is to actually try it out. Here's a small example:

```
1 import cherrypy
2
3 class HomePage:
4     def index(self):
5         html = '''<center><h1>TITLE</h1></center>
6             <p style="text-align:center;" >
7                 <h2>SUBTITLE</h2>
8             </p>
9             <br />
10            <br />
11            <p>BODY</p>
12            '''
13         return html
14
15     index.exposed = True
16
17 cherrypy.tree.mount(HomePage())
```

This is all you need to create a simple HTML page in Python using CherryPy. The following are few notably good things CherryPy has to offer:

**Simplicity** A web framework cannot possibly get simpler than this without complicating a whole lot of other things. CherryPy handles almost all the low-level details for web programmers, leaving them to concentrate on the logic of web applications. All the while, it does not restrict web developers to anything: the layers of abstractions are there, but they are easily customisable and can be adapted to suit requirements.

**Flexibility** Granted, big shot web frameworks like Django, Pylons, Turbogears, Ruby on Rails etc go way ahead to provide web developers with everything they need to develop web applications fast, but in so doing, they tend to restrict developers in a number of ways. Two of such restrictions that get the most heat are template and database interface/access support. Big web frameworks impose restrictions on which templating engine and database UI they support, which means that if a certain web framework you like a lot does not let you use the templating engine you find more intuitive than the custom one supported by the framework, you can't do anything about. Similar is the case with object-relational mapping engine support. CherryPy, on the other hand, imposes no such restrictions. If there is a templating engine you like and it can be used in Python, cherryPy will merrily let you use it. Like the CherryPy website says it aptly, "CherryPy pretty much stays out of your way. You are free to use any kind of templating, data access etc. technology you want".

**Template and ORM Engines** As has been highlighted, CherryPy lets you use whichever templating engine and ORM module you want to use, as long as it can be used in Python. CherryPy has its own template engine as well, but again, which you decide to use is your choice entirely, and CherryPy does not interfere in your decision process.

**Stability** Cherrypy is stable. Period. It is old (but not outdated), mature and time tested. It has proven its reliability and stability over time. Not only that, it has been in constant development for the past three years, and in heavy use by the community.

**Stand-alone Applications** Cherrypy web applications are stand-alone Python applications which run on top of the multi-threaded, small webserver Cherrypy provides. This is a good thing because you can bring up your applications anywhere you can get Python to run. Cherrypy web applications are really stand-alone. And unlike the stand-alone webservers other web frameworks provide, Cherrypy beats everything in speed and stability.

Again, the best way to judge Cherrypy is by using it.

### 2.1.2 Cheetah

Cheetah is a powerful yet simple to use templating and code generation tool. It provides everything a templating system ought to provide. Further, Cheetah provides access to most (or all) of Python constructs within the templates. You can have flow control in Cheetah templates all the while keeping yourself from falling into the mixing-logic-with-presentation trap.

The drawback with most templating systems is that, if the programmer wants even minimal amount of flow control, they are forced to put certain amount of logic into the template. Cheetah, on the other hand, with its easy syntax and access to python constructs, allows achieving flow control with the minimal amount of logic required.

Some of things that make Cheetah a great templating engine are:

**Simple Syntax** The templating syntax is exceptionally simple. For flow control, the constructs are Python-like. If you know Python, you can start using Cheetah in a jiffy (well, maybe more than a jiffy, but just a little bit more). Templating systems are infamous for forcing programmers to learn strange templating languages before programmers can even think about using templates with their applications. With Cheetah, the template language is so Python-like, you will often take it as Python code and not Cheetah code. An example of Cheetah template code that introduces flow control constructs like the for, if, and set directives:

```
1 | #for $link , $title , $comment in $main_links :
2 |     #if $comment
3 |         #set $comment = "- " + $comment
4 |     #end if
5 |     <a href="$link">$title</a> $comment
```

**Importing User-Defined Templates** Cheetah not only makes possible including templates defined in different files into other templates, but, going one step ahead, allows for subclassing of templates as well. So, for example, if there is a parent template which defines the header and footer each HTML page will have, any subsequent template file can easily subclass the parent template with a simple declaration like:

```

1 | #from parent import parent
2 |
3 | #extends parent

```

**Compile into Classes** This is by far the best thing about Cheetah: Template definitions can be compiled into Python classes. These classes then can be used from any Python code like any other class. For example, if you have a template, `index_page.tmpl`, you can compile it into a class, and from within your main Python application, you can import the `index_page` class like any normal class:

```

1 | $ cheetah compile parent.tmpl
2 |
3 | from index_page import index_page
4 |
5 | return index_page(searchList = [{
6 |     'title': self.title,
7 |     'css': self.css,
8 |     'irc': self.irc,
9 |     'fortune': fortune[0],
10 |    'author': fortune[1],
11 |    'projects': self.projects_list,
12 |    'main_links': self.main_links }]).respond()

```

**Generate Any Type of Output** Cheetah does not only support generation of HTML output. It can generate pretty much anything from HTML, XML, text, to, and this should not go without mention, Python code.

**L<sup>A</sup>T<sub>E</sub>X Documents** Cheetah can generate any type of output other than HTML as well. It can work with L<sup>A</sup>T<sub>E</sub>X documents as well. `plasTeX`, for example, a python framework for L<sup>A</sup>T<sub>E</sub>X processing, fully supports Cheetah templates. [plasTeX]

These are only few of the things Cheetah is exceptionally good at doing. All in all, Cheetah is a great templating engine with a very flat learning curve. A modest comparison of Cheetah with other templating platforms and web content generation tools is given at [Cheetah vs Other Template Engines].

## 2.2 Pylons

Pylons, a lightweight web framework for agile web development, can be thought of as a port of Ruby on Rails for Python. Pylons, strictly speaking, isn't *completely* a port of Ruby on Rails. It implements ideas Ruby on Rails uses, incorporates direct ports of few useful Rails components, and, all in all, allows for very rapid development of web applications. While still not mature, Pylons offers compelling reasons to choose it as the web framework of choice for rapid web development. Some of the features worth mentioning about Pylons are briefly described below. [Pylons, Pylons Features]

- If you are a fan of Ruby on Rails' scaffolding magic, you'd be excited to learn that Pylons has something similar to offer.

- While built on top of Myghty (python port of HTML::Mason, a Perl templating system) and Paste (what has come to be known as the Python web glue), Pylons, like Cherrypy, pretty much stays out of your way by letting you use the templating engine and ORM (among other things) of your choice. This is a great thing, considering that most other heavy web frameworks either don't support particular sets of templating systems and ORMs, or offer only custom-made alternatives.
- Applications written using Pylons are stand-alone, much like Cherrypy, and can be run either on the custom webserver that comes with Pylons or any other web platform such as mod\_python, Apache, WSGI, CGI, FastCGI, etc. And quite like Cherrypy, all you really need to run a Pylons application is a system with a Python interpreter and standard library.
- The direct port of Ruby on Rails' AJAX Helpers provide for AJAX support in Pylons.
- Web frameworks simplify things, a lot of things, but at the cost of speed and underlying complexity. Websites running on top of many powerful, feature-rich web frameworks tend to suffer from performance degradation in terms of slow response speeds mostly. And it is quite natural for them to suffer so. Pylons, on the other hands, through using a component-based model, boasts of less speed and performance penalties. If you only need part of the functionality a web framework offers, it makes sense to use only that functionality and discard the rest. Pylons offers just that.

All in all, Pylons seems a promising, young web framework that is gaining popularity and support quickly.

### 3 Perl and CGI

The biggest benefit of working with Perl is the nearly unlimited supply of third-party modules and packages available through CPAN (and elsewhere). There are so many that there isn't a thing that some CPAN module does not already do. Not only that, documentation for almost everything is just out there.

Perl has web frameworks (quite few though), such as Catalyst, but at the heart of them all is the CGI module. The CGI module can do a lot of things but it isn't, strictly speaking, a web framework. You can write dynamic websites using CGI, but when CGI is talked about, the mind automatically drifts towards forms and forms-based content (as that is what it was designed for).

The following points highlight some advantages and disadvantages of working with Perl and CGI.

- Perl is a scripting language, and as such is unstructured. There are ten ways (so to speak) to do a single thing, and it is very easy to write unintelligible and strange-looking code.
- Perl CGI scripts are nothing more than scripts that run off web servers. They have to be loaded into memory and interpreted each time a web service that relies on such scripts is requested. This leaves behind a lot of load on the web server.

- Serving dynamic content using Perl + CGI has been the norm for quite a long time now. However, the trend is turning towards full-stack, feature-rich web frameworks that manage all the dirty work and provide intuitive layers of abstractions to work with, all the while giving developers the freedom to use ORM and templating technology of their choice.
- The Object Oriented interface is hardly intuitive, being nothing more than a hack. The code is highly prone to getting complex-looking (if not ugly), and very difficult to maintain if not carefully written and commented.
- ORMs in Perl have a terrible interface. You might (and people do tend to) get the hang of it eventually, but since ORM technology relies heavily on Object Oriented concepts and since Perl's Object Oriented features are hardly anything to boast about, the interface ORMs in Perl present is pretty scary. However, no doubt, they work pretty well, but the resulting code, again, isn't exactly what you would call pretty code. Plus, support for databases in Perl is huge, reliable, stable, and time tested.
- There are a number of templating modules available which can be used to abstract out presentation from logic while using CGI. There is `HTML::Mason`, `Template Toolkit`, and `HTML::Template`. All of these are powerful and useful, especially `Template Toolkit` which is not only powerful, but fast. [Perl Template Engines]
- Furthermore, `Template Toolkit` has a `LaTeX` plugin that makes possible generation of static `LaTeX` content from templates. [LaTeX Plugin]

Don't let this fool you into thinking that no-one uses Perl CGI for serious web development. To the contrary, it has been used extensively. Slashdot, for example, runs on Perl, and has a complete project over that manages development of Slashdot. [The Slash Code]

## 4 Ruby on Rails

The three letter phrase we have all come, out of nowhere, to hear a lot about, Ruby on Rails is a web development framework for rapid web development using Ruby. I am not going to go into the details of the good things Ruby on Rails has to offer, because it offers almost most of the things a modern, rich web framework for rapid development ought to offer. However, I would like to make a passing mention of some of the drawbacks that become clear once you start to get to know Ruby on Rails better.

**ORM and Database Support** Ruby on Rails offers a custom, raw database access interface which at times requires you to manually access and make changes to database through whatever low-level interface the database of your choice provides. Furthermore, support for different databases is still limited.

**Templating Engine** Ruby on Rails uses a custom templating language (ERB or `.rhtml`) which is nothing more than Ruby code embedded within blocks in HTML documents (pretty much like ASP). The fine line between logic

and presentation would appear blurred if not absent if you start working with the template engine.

**Old yet New** Ruby, though as old as Python, has only recently started getting attention, and as such, is a new kid around the block. Say whatever you want, Ruby is still not ubiquitous.

**Limited Third-party Module Base** Ruby is hardly blessed with half the number of third-party modules and libraries Perl and Python have available in their repertoires. This is a major drawback for Ruby on Rails.

**It's All About Rails** Most of the development in Ruby centers around Rails. There is hardly anything going in Ruby that isn't about Rails that gets people's attention.

**L<sup>A</sup>T<sub>E</sub>X Support** The R<sub>T</sub> plugin add supports for templates containing L<sup>A</sup>T<sub>E</sub>X markups and embedded Ruby code and for PDF generation from such templates. However, most anything available regarding R<sub>T</sub> on the web leads to a dead end, which makes it difficult to say anything about it. [R<sub>T</sub>]

**Dynamic** There is little, if anything, static where Ruby on Rails goes. Everything is dynamic, so much so, that Rails developers find themselves increasingly inclined towards doing things dynamically all the time.

This does not mean that Ruby on Rails hasn't got what it takes to be a big shot web framework. It actually has most of the things. Plus, it makes web application development real fast. It is just that it has a number of unfortunate restrictions and drawbacks that make it not such a likely platform of choice for web development.

## 5 So?

There is no best. Use what works for you, for your requirements. If you are a Perl developer, you would certainly want to take the Perl route. If you are more of a developer who puts value in structured code (among other things), then you can pick between Python and Ruby on Rails. If you want a lot of third-party module support, or want to choose from different web frameworks and a set of template engines and ORMs, then Python is clearly the winner. Whichever you are, in the end, it boils down to, and I'd like to emphasize again, what works for you and your immediate requirements. Everything else is, well, a boatload of crap.

## References

- [Cheetah vs Other Template Engines] [http://www.cheetahtemplate.org/docs/users\\_guide\\_html/users\\_guide.html#SECTION00022000000000000000](http://www.cheetahtemplate.org/docs/users_guide_html/users_guide.html#SECTION00022000000000000000)
- [RTex] <http://wiki.rubyonrails.org/rails/pages/RTex+Plugin>  
<http://rubyforge.org/projects/rtex/>
- [plasTeX] <http://plastex.sourceforge.net/>
- [Perl Template Engines] <http://www.masonhq.com/>  
<http://www.template-toolkit.org/>  
<http://search.cpan.org/dist/HTML-Template/Template.pm>
- [LaTeX Plugin] <http://search.cpan.org/~andrewf/Template-Latex-2.17/lib/Template/Plugin/Latex.pm>
- [Python Web Frameworks] <http://wiki.python.org/moin/WebFrameworks>
- [Pylons] <http://pylonshq.com/>
- [Pylons Features] <http://pylonshq.com/information.html>
- [The Slash Code] <http://www.slashcode.com/>