Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

Today

- Integrating with Web Services
- Apache & nginx
- Django

Integrating with Web Services

- GNU/Linux is the most popular platform for web services
- This is because web service support and integration is prevalent on Linux
- Similarly to the application services in the last lecture, integrating with web services on Linux is well supported
- Web service integration on Linux leverages many of same tools for application services, as well as many new ones

Integrating with Web Services

- Web services are supported in a plethora of languages on Linux
- As web services and applications use standard languages (PHP, Python, Java, etc), all integration tools/libraries that exist for those languages can be leveraged even if they are not specific to the web
- There are many extremely well supported frameworks and server engines on Linux;

Integrating with Web Services

- Highly Popular Frameworks for Linux:

Java
Struts
JSF
Spring
Google Web Toolkit

Ruby
Ruby on Rails

Flash
Adobe Flex

Python
Django
Turbogears
CherryPy
Pylons

PHP
CakePHP
CodeIgniter
Symfony
Zephyr
Zend
Drupal (really a CMS)

Integrating with Web Services

- Due to all of the open source languages and integration tools for Linux, it has become a very popular platform

- There are large number of highly supported open source web servers available for Linux:

  - **Apache**
  - **nginx**
  - **Lighttpd**
  - **Hiawatha**
  - **Cherokee**

Integrating with Web Services

- You very often will hear about LAMP:

  LAMP - Linux, Apache, MySQL and PHP/Python/Perl

- There are many similar setups available and commonly used Replacements:

  Apache - nginx, Lighttpd, Hiawatha, Cherokee
  MySQL - Postgresql, SQLite, CouchDB
  PHP - Python, Perl, Ruby, Java

Apache & nginx

- Of the many web servers available for Linux, Apache and nginx have the greatest momentum
- Apache and nginx are currently the two most popular web servers
- Apache itself is the most popularly used and mentioned web server
- nginx is quickly rising as another open source web server, but it is still very new (2004) compared to Apache (1995)

Apache & nginx

- Both Apache and nginx are strongly supported on Linux, with most packages available for both server

- Apache likely has more support overall as it has been around longer, but nginx is mature and mostly has equivalent support

- nginx is an event driven server, while Apache is thread/process driven:

  Thread/Process Driven - Individual threads for each connection
  Event Driven - Event fires an action instead of continually checking to see if action occurred

  *Note: Events and Threads are not mutually exclusive

Apache

- In lab, you will be working with Apache
- Apache is configured using several configuration files
- Here are some of the main configuration files in Apache

apache2.conf - **Top level Apache file, contains global settings**

httpd.conf - **Main Apache file, contains user specific configuration options**

ports.conf - **Configure which ports Apache is listening on**

Apache

- Apache stores website configurations in a folder called sites-available:

  /etc/apache2/sites-available

- This folder houses configuration files for each website and their settings

- The configuration files point to a folder where each website resides

- These folders usually contain an index.html file and all of the site pages, css, javascript, images, etc

Apache

- By default, the sites-available folder has a file called default, which is a page you can use to test Apache
- This gives you a starting point for making your own configuration file as it can be referenced for commands and syntax
- The default file points to a pre-made Apache directory that contains the testing pages

Apache

- Once you have a site configured, you can enable it with the command **a2ensite**

- This command stands for “Apache2 Enable Site”

- Once a site is enable, a duplicate of the configuration file from the sites-available folder is made in the folder **sites-enabled**

  /etc/apache2/sites-enabled

Apache

- Sites are disabled with the command **a2dissite** which stands for "Apache2 Disable Site"
- When this command is run, the configuration file is removed from the sites-enabled folder, and will only reside in sites-available
- If a site is enabled, you may connect to it with the address given in the configuration file
- When connecting from the local machine, the address is given as http://localhost

Apache

- At this point, Apache can run, but it will not allow HTTPS connections yet

- To enable a secure connection, you first need a security certificate

- Generating a certificate is done in three steps:

  1) Generate RSA Key
  2) Generate Certificate Signing Request (CSR)
  3) Generate Privacy Enhanced Mail Certificate (PEM)

Apache

- Generating an RSA key is done with openssl using the genrsa flag

  openssl genrsa -des3 -out mySite.key 2048

- Once you have the RSA key, you can generate the certificate signing request using openssl and the req flag

  openssl req -new -key mySite.key -out mySite.csr

- Finally, you can make the certificate with openssl and the x509 flag

  openssl x509 -req -days 365 -in mySite.csr -signkey mySite.key -out mySite.pem

Apache

- After you have generated the certificate, it can be moved into

  /etc/ssl/certs/

- This is a folder designed to hold all of your certificates

- You will also need to move the RSA key to

  /etc/ssl/private

- This folder is intended to hold your private keys

Apache

- Apache will automatically reference certificates and keys in /etc/ssl/certs and /etc/ssl/private
- To enable Apache to do this, you need to use the command **a2enmod ssl**
- A2enmod stands for “Apache2 Enable Module”
- This will enable Apache to use SSL certificates and keys

Apache

- Once the SSL module has been enabled, you will find a default HTTPS configuration file in sites-available called **default-ssl**

- If you open the file, you will find that it references Apache's own test key and certificate called:

  ssl-cert-snakeoil.pem
  ssl-cert-snakeoil.key

- This serves as a testing and reference configuration file for SSL enabled sites

Apache

- When you make your own SSL site configuration files, you need to use your own key and certificate in configuration file
- If you were working with a certificate authority to make the SSL site a legitimate site, you would give your certificate them to validate
- After you finish your SSL site configuration file, you can enable the site as normal using **a2ensite**
- This time, however, you need to restart Apache to load the SSL mod

Apache

- When Apache restarts, you will be able to connect to your site page using an HTTPS connection
- Your browser will give you an untrusted connection error if you have not given the certificate to a valid certificate authority
- The browser will not have any way of validating the certificate unless it can connect to a certificate authority
- You may add the certificate to the browser to see your page

Apache

- You will occasionally see sites that do not use a valid CA and thus their certificate gives an untrusted connection error
- Many times companies do this on their internal network, where the site is not outside facing
- In these instances, they are acting as their own certificate authority and validating the certificate on their own network
- The browser will still not show a legitimate connection as there are no certificate authorities validating the certificate

Django

- Apache is a web server and it controls web connections
- Apache will use the configuration files to use specific folders containing web site files, but it is only hosting static pages
- Apache itself does not generate dynamic pages/content that many sites use
- Instead, this requires software able to generate dynamic content

Django

- Dynamic content is generated one of two ways:
  - Using custom code/engine
  - Using a framework
- Using a framework has more overhead than custom code, but it is much easier and tends to have better security measures in place
- Many of the more popular frameworks are scalable and have existing code to integrate with many different open source technologies

Django

- In Lab, you will be looking at Django, a popular Python framework

- Django is a model-view-controller (MVC) framework that is designed for ease of use, security and scalability

- Django includes many development tools, such as :

-validation system
-serialization
-internationalization
-unit-testing
-caching
-middleware support
-templates

Django

- Django can work with many open source technologies
- Django has support for various web servers, common gateway interfaces, databases and modules
- One of the nice features Django has, is that development can be done on its own and tested against Django's internal development-only server

Django

- Django's internal development-only server can not be used for hosting live pages, as it is not secure and not designed for performance

- The internal development server allows you to thoroughly test pages as if they were live before deploying to a web server

- This allows you to do web development and testing before building a server environment or deploying to an existing environment

Django

- We are using Django in lab because:
  - It is one of the easiest frameworks to use,
  - It is very secure
  - It is one of the fastest
  - It illustrates how well open source technologies integrate together
- Django has tremendous integration support for many technologies out of the box
- In lab, you will look at integrating Django with MySQL

Django

- The first step in setting up Django, is to choose your database

- In the database, you need to create a user for Django to use and create a database named “Django”

  *Note: your database user must have access to the Django table

- Django will use this database to store all of its data and user data

  *Note: The database does not necessarily need to be a relational database; Many NoSQL databases are supported as well

Django

- Once the database has been configured, you need to create a folder to house your Django project

- Inside of that folder, you can initiate Django with

  django-admin startproject myProjectName

- This will create a directory called myProjectName filled with Django configuration files

Django

- In the new folder, there will be be a file called settings.py
- This is a global setting configuration file for your Django project
- In the file there is an 'ENGINE' variable set to django.db.backends.
- You can interact with many databases by tacking on the name of the database to the end of the variable

  Example: django.db.backends.mysql

Django

- Below the engine line are variables for the database name, database username and database user password

- Once these have been filled in and the file has been saved, Django can setup the database using the manage script in the same folder

  ./manage.py syncdb

- Once the database is setup, you can test Django using Django's internal development server

  ./manage.py runserver

Django

- Once the development server is running, you should be able to connect to it on your local machine on port 8000

- If you go to https://127.0.0.1:8000 in a web browser, a web page should load saying “It Worked”

- After Django is configured, you can start creating apps with

  ./manage.py startapp myAppName

Django

- After startapp has been run, there will be a folder called myAppName that is full of application configuration files

- After the app has been made, you can open the settings.py file again and find the INSTALLED_APPS variable;

  At the end of the variable you can add your new application name

  'myAppName'

- This allows Django to know which apps you want to be running

Django

- In Django, content is generated dynamically
- When a request is made, Django calls a function to generate the page you want them to have based on their specific data
- You will notice in the main Django folder, there is a file called “urls.py”
- This file directs all urls to the functions you want them to call based on regular expressions (regex)

Django

- You will also find that there is a urls.py file inside of the app folder

- This is because the first urls.py is Django's global level url management;

  This file determines which app to call based on the url

- After the app is called, the app takes the url into its own urls.py file to determine which page to generate

Django

- An example of the global level urls regex

  url(r'^social/', include('socialNetwork.urls'))

  This line would take any urls that begin with “social” after the address and call the application called “socialNetwork”

- An example of the application level urls regex

  (r'^login/$', 'createLoginPage')

  This line would take any urls that end in “login/” and call the function createLoginPage to generate the page

Django

- These functions that generate pages are in the file **views.py**
- The views.py file contains functions for each urls call in order to generate any data necessary
- Most of the time pages are generated using html pages that have been created at templates
- The views function can call these templates and fill in any customized data the page needs

Django

- Template html pages can be written like normal html pages, but can contain variables and evaluation statements that Django can alter

- Variables in Django html pages are made with

  {{ variableName }}

- Django can then grab that variable name in a function and alter it to the value it desires in proper html format

Django

- Similarly, Django can use evaluation statements in html pages

- Evaluation statements are done with

  {% for v in variableName %}

  {% endfor %}

- This allows html and variables in between the statements to be generated as needed by the loop;

  Django allows many different types of evaluation to be done

Django

- With the powerful features Django offers and its strong integration with open source, dynamic web pages can be made quickly
- Django easily integrates with many open source technologies with little configuration or extras to install
- As Django itself is open source, new developments and integration libraries are being made often