
Full Stack Python
by Matthew Makai

Table of Contents

    i. Data
    ii. Relational Databases
    iii. PostgreSQL
    iv. MySQL
    v. SQLite
    vi. Object-relational Mappers (ORMs)
    vii. NoSQL
7. Web APIs
    i. Web APIs
    ii. API Integration
    iii. API Creation
8. Web App Deployment
    i. Deployment
    ii. Servers
    iii. Platform-as-a-Service
    iv. Operating Systems
    v. Web Servers
    vi. Apache HTTP Server
    vii. Nginx
    viii. Caddy
    ix. WSGI Servers
    x. Source Control
    xi. App Dependencies
    xii. Static Content
    xiii. Task Queues
    xiv. Configuration Management
    xv. Continuous Integration
    xvi. Logging
    xvii. Monitoring
    xviii. Web Analytics
    xix. Docker
    xx. Caching
    xxi. Microservices
    xxii. DevOps
9. Testing
    i. Testing
    ii. Unit Testing
    iii. Integration Testing

Full Stack Python

Full Stack Python by Matthew Makai.

Introduction

You're knee deep in learning the Python programming language. The syntax is starting to make sense. The first few "*ahh-ha*!" moments are hitting you as you're learning conditional statements, `for` loops and classes while playing around with the open source libraries that make Python such an amazing language.

Now you want to take your initial Python knowledge and make something real, like an application that's available on the web that you can show off or sell as a service to other people. That's where Full Stack Python comes in. You've come to the right place to learn everything you need to create, deploy and run a production Python web application.

Purpose

This book provides a high-level overview of many Python concepts, especially in web development and deployment. The content here is the same as found on the Full Stack Python website, but this version comes nicely formatted in a PDF or EPUB format for easier consumption.

The material presented here supplements The Full Stack Python Guide to Deployments, which you should have also received if you're reading this book.

Revision History

2016-03-21: **March 2016 edition**. Contains all fullstackpython.com website content through March 21, 2016.

2016-02-20: **February 2016 edition**. This version contains all of the content on the Full Stack Python website as of February 20, 2016.

2015-07-28: **First edition**. Initial release.

Updates

Full Stack Python is an in-progress book, this is not its completed state. Certain chapters, especially ones such as Generators and Comprehensions, need a lot more content before they'll be finished.

This book will remain free online at www.fullstackpython.com. I will update this book on a regular basis until I consider Full Stack Python complete. Future updates will be available for free to all purchasers.

About the Book Cover

The cover photo is a picture I took while flying from Washington, D.C. to San Francisco, CA in 2014. The picture of looking down from above the clouds matches my intention of making Full Stack Python a high-level overview of Python development that anyone can use to get their bearings. Pixelmator was used to alter the image to give it a bit of a dreamy look. The cover picture stands in contrast to the ground-level photo on The Full Stack Python Guide to Deployments book and the step-by-step tutorial content found in that book.

Thank You

To the Python community, for all that you've taught me and continue to teach me every day.

Getting Started

Learning Programming

Learning to program is about understanding how to translate thoughts into source code that can be executed on computers to achieve one or more goals.

There are many steps in learning how to program, including

1. setting up a development environment
2. selecting a programming language, such as Python
3. understanding the syntax and commands for the language
4. writing code in the language, often using pre-existing code libraries and frameworks
5. executing the program
6. debugging errors and unexpected results
7. deploying an application so it can run for intended users

How should I learn programming?

There are several schools of thought on how a person should start learning to program. One school of thought is that a lower-level programming language such as Assembly or C are the most appropriate languages to start with because they force new developers to write their own data structures, learn about pointers and generally work their way through the hard problems in computer science.

There's certainly wisdom in this "low-level first" philosophy because it forces a beginner to gain a strong foundation before moving on to higher level topics such as web and mobile application development. This philosophy is the one most commonly used in university computer science programs.

The atomic units of progress in the "low-level first" method of learning are

1. aspects of programming language understood (type systems, syntax)
2. number of data structures coded and able to be used (stacks, queues)
3. algorithms in a developer's toolbelt (quicksort, binary search)

Another school of thought is that new developers should bootstrap themselves through working on projects in whatever programming language interests them enough to keep

working through the frustrations that will undoubtably occur.

In this "project-based" line of thinking, the number of projects completed that expand a programmer's abilities are the units of progress. Extra value is placed on making the projects open source and working with experienced mentors to learn what he or she can improve on in their programs.

Should I learn Python first?

Python is good choice in the project-based approach because of the extensive availability of free and low cost introductory resources, many of which provide example projects to build upon.

Note that this question of whether or not Python is a good first language for an aspiring programmer is highly subjective and these approaches are not mutually exclusive. Python is also widely taught in universities to explain the fundamental concepts in computer science, which is in line with the "low-level first" philosophy than the projects-first method.

In a nutshell, whether Python is the right first programming language to learn is up to your own learning style and what feels right. If Ruby or Java seem like they are easier to learn than Python, go for those languages. Programming languages, and the ecosystems around them, are human-made constructs. Find one that appears to match your personal style and give it a try, knowing that whatever you choose you'll need to put in many long days and nights to really get comfortable as a software developer.

Why Use Python?

Python's expansive library of open source data analysis tools, web frameworks, and testing instruments make its ecosystem one of the largest out of any programming community.

Python is an accessible language for new programmers because the community provides many introductory resources. The language is also widely taught in universities and used for working with beginner-friendly devices such as the Raspberry Pi.

Python's programming language popularity

Several programming language popularity rankings exist. While it's possible to criticize that these guides are not exact, every ranking shows Python as a top programming language within the top ten, if not the top five of all languages.

Most recently, the RedMonk January 2015 ranking had Python at #4.

The TIOBE Index, a long-running language ranking, has Python steady at #8.

The PopularitY of Programming Language (PYPL), based on leading indicators from Google Trends search keyword analysis, shows Python at #3.

GitHut, a visualization of GitHub language popularity, pegs Python at #3 overall as well.

These rankings provide a rough measure for language popularity. They are not intended as a precise measurement tool to determine exactly how many developers are using a language. However, the aggregate view shows that Python remains a stable programming language with a growing ecosystem.

Why does the choice of programming language matter?

Programming languages have unique ecosystems, cultures and philosophies built around them. You will find friction with a community and difficulty in learning if your

approach to programming varies from the philosophy of the programming language you've selected.

Python's culture values open source software, community involvement with local, national and international events and teaching to new programmers. If those values are also important to you and/or your organization then Python may be a good fit.

The philosophy for Python is so strongly held that it's even embedded in the language as shown when the interpreter executes "import this" and displays The Zen of Python.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

More perspectives on why to use Python

- The Python documentation has a HOWTO section specifically for Python advocacy.
- How to argue for Python's use explains that choosing a programming language can be complicated but that Python is a very good option for many use cases.
- Why I Push for Python explains one professor's rationale for promoting Python to teach programming to undergraduates.
- If you're wondering about the differences in Python's dynamically typed system

versus statically typed languages, be sure to read this thorough explanation of the topic.

Python 2 or 3?

The Python programming language is currently in the midst of a long-term transition from version 2 to version 3. New programmers typically have many questions about which version they should learn. It's confusing to hear that Python 3, which was originally released in 2008, is still not the default installation on many operating systems.

Here's the good news: you can't go wrong starting with either version. While there are differences in unicode and syntax, for the most part if you start with Python 2 and then learn Python 3 you won't be starting from scratch. Likewise, you'll be able to read and write Python 2 code if you started with Python 3.

My personal recommendation for new programmers as of right now is to start with Python 3. There are enough great resources out there that teach version 3 from the ground up.

However, if you are interested in DevOps-type work with configuration management tools such as Ansible or Fabric, then you'll have to stick to Python 2 because they have yet to upgrade to support Python 3. If you know there are libraries you must use in a project, check the Python Walls of Superpowers. If you're using Django, there is also a wall specifically for Python 3 compatibility of popular Django packages.

Python 2 to 3 resources

- Want to know all of the advantages and what's changed in Python 3 compared to Python 2? There's an official guide to Python 3 changes you'll want to read.

- The official porting code to Python 3 page links to resources on porting Python code as well as underlying C implementations. There is also a quick reference for writting code with Python 2 and 3 compatibility.

- Python 3 in 2016 explains that many newer Python developers have only used Python 3 and as that cohort continues to grow it will have an outsized impact on further adoption.

- Python 3 is winning presents data and graphs from PyPI to show that at the current rate, by mid-2016 overall Python 3 library support will overtake Python 2 support.

- The stages of the Python 3 transition provides perspective from a core Python developer on how the transition from Python 2 to 3 is going as of the end of 2015.
- Porting to Python 3 is like eating your vegetables explains that there are treats in Python 3 that are worth porting for and has some tips on making the transition easier.
- Moving from Python 2 to Python 3 is a PDF cheatsheet for porting your Python code.
- Django and Python 3 How to Setup pyenv for Multiple Pythons is a screencast showing how to run both Python 2 and 3 for different projects using pyenv.
- Scrapy on the road to Python 3 support explains from the perspective of a widely used Python project what their plan is for supporting Python 3 and why it has taken so long to make it happen.
- Python 3 Readiness is a visualization of which most popular 360 libraries (by downloads) are ready to be used with Python 3.

Enterprise Python

One of the misconceptions around Python and other dynamically-typed languages is that they cannot be reliably used to build enterprise-grade software. However, almost all commercial and government enterprises already use Python in some capacity, either as glue code between disparate applications or to build the applications themselves.

What is enterprise software?

Enterprise software is built for the requirements of an organization rather than the needs of an individual. Software written for enterprises often needs to integrate with legacy systems, such as existing databases and non-web applications. There are often requirements to integrate with authentication systems such as the Lightweight Directory Access Protocol (LDAP) and Active Directory (AD).

Organizations develop enterprise software with numerous custom requirements to fit the specific needs of their operating model. Therefore the software development process often becomes far more complicated due to disparate factions within an organization vying for the software to handle their needs at the expense of other factions.

The complexity due to the many stakeholders involved in the building of enterprise software leads to large budgets and extreme scrutiny by non-technical members of an organization. Typically those non-technical people place irrational emphasis on the choice of programming language and frameworks when otherwise they should not make technical design decisions.

Why are there misconceptions about Python in enterprise environments?

Traditionally large organizations building enterprise software have used statically typed languages such as C++, .NET and Java. Throughout the 1980s and 1990s large companies such as Microsoft, Sun Microsystems and Oracle marketed these languages as "enterprise grade". The inherent snub to other languages was that they were not appropriate for CIOs' difficult technical environments. Languages other than Java, C++

and .NET were seen as risky and therefore not worthy of investment.

In addition, "scripting languages" such as Python, Perl and Ruby were not yet robust enough in the 1990s because their core standard libraries were still being developed. Frameworks such as Django, Flask and Rails (for Ruby) did not yet exist. The Web was just beginning and most enterprise applications were desktop apps built for Windows. Python simply wasn't made for such environments.

Why is Python now appropriate for building enterprise software?

From the early 2000s through today the languages and ecosystems for many dynamically typed languages have greatly improved and often surpassed some aspects of other ecosystems. Python, Ruby and other previously derided languages now have vast, well-maintained open source ecosystems backed by both independent developers and large companies including Microsoft, IBM, Google, Facebook, Dropbox, Twilio and many, many others.

Python's open source libraries, especially for web development and data analysis, are some of the best maintained and fully featured pieces of code for any language.

Meanwhile, some of the traditional enterprise software development languages such as Java have languished due to underinvestment by their major corporate backers. When Oracle purchased Sun Microsystems in 2009 there was a long lag time before Java was enhanced with new language features in Java 7. Oracle also bundles unwanted adware with the Java installation, whereas the Python community would never put up with such a situation because the language is open source and does not have a single corporate controller.

Other ecosystems, such as the .NET platform by Microsoft have fared much better. Microsoft continued to invest in moving the .NET platform along throughout the early part of the new millennium.

However, Microsoft's enterprise products often have expensive licensing fees for their application servers and associated software. In addition, Microsoft is also a major backer of open source, especially Python, and their Python tools for Visual Studio provide a top-notch development environment.

The end result is that enterprise software development has changed dramatically over the past couple of decades. CIOs and technical executives can no longer ignore the progress of Python and the great open source community in the enterprise software development landscape if they want to continue delivering business value to their business side customers.

Open source enterprise Python projects

- Collab by the U.S. government's Consumer Financial Protection Bureau (CFPB) agency is a corporate intranet and collaboration platform for large organizations. The project is currently running and in-use by thousands of CFPB employees.

- Pants is a build system for software projects with many distinct parts and built with many different programming languages as is often the case in large organizations.

Enterprise Python software development resources

- There are a couple of solid demystifying articles in CIO magazine including this broad overview of Python in enterprises and this article on why dynamic languages are gaining share for enterprise development.

- JavaWorld wrote an interesting article about Python's inroads into enterprise software development.

- I gave a talk at DjangoCon 2014 on How to Solve the Top 5 Headaches with Django in the Enterprise which covered both Python and Django in large organizations.

- This StackExchange answer contains a solid explanation what differentiates enterprise software from traditional software.

- There was a Python subreddit thread about Python in the enterprise that's worth a look for broader opinions on Python compared to Java and .NET in enterprise environments.

Best Python Resources

The Python community is amazing at sharing detailed resources and helping beginners learn to program with the language. There are so many resources out there though that it can be difficult to know how to find them.

This page aggregates the best general Python resources with descriptions of what they provide to readers.

New to programming

If you're learning your first programming language these books were written with you in mind. Developers learning Python as a second or later language should skip down to the next section for "experienced developers".

- To get an introduction to Python, Django and Flask at the same time, consider purchasing the Real Python course by Fletcher, Michael and Jeremy.

- This short 5 minute video explains why it's better to think of projects you'd like to build and problems you want to solve with programming. Start working on those projects and problems rather than jumping into a specific language that's recommended to you by a friend.

- CS for All is an open book by professors at Harvey Mudd College which teaches the fundamentals of computer science using Python. It's an accessible read and perfect for programming beginners.

- If you've never programmed before check out the Getting Started page on Learn To Code with Me by Laurence Bradford. She's done an incredible job of breaking down the steps beginners should take when they're uncertain about where to begin.

- Learn Python the Hard Way is a free book by Zed Shaw.

- The Python projects tag on the Twilio blog is constantly updated with fun tutorials you can build to learn Python, such as the International Space Station Tracker with Flask and Redis-Queue, Choose Your Own Adventures Presentations using Flask

and WebSockets and Martianify Photos with OpenCV.

- Dive into Python 3 is an open source book provided under the Creative Commons license and available in HTML or PDF form.
- A Byte of Python is a beginner's tutorial for the Python language.
- Code Academy has a Python track for people completely new to programming.
- Introduction to Programming with Python goes over the basic syntax and control structures in Python. The free book has numerous code examples to go along with each topic.
- Google put together a great compilation of materials and subjects you should read and learn from if you want to be a professional programmer. Those resources are useful not only for Python beginners but any developer who wants to have a strong professional career in software.
- The O'Reilly book Think Python: How to Think Like a Computer Scientist is available in HTML form for free on the web.
- Python Practice Book is a book of Python exercises to help you learn the basic language syntax.
- Looking for ideas about what projects to use to learn to code? Check out this list of 5 programming project for Python beginners.
- There's a Udacity course by one of the creators of Reddit that shows how to use Python to build a blog. It's a great introduction to web development concepts through coding.
- I wrote a quick blog post on learning Python that non-technical folks trying to learn to program may find useful.
- Python for you and me is an approachable book with sections for Python syntax and the major language constructs. The book also contains a short guide at the end to get programmers to write their first Flask web application.

Python for specific occupations

Python is powerful for many professions. If you're seeking to use Python in a specific field, one of these guides may be the most appropriate for you.

- Python for Social Scientists contains a textbook, course outline and slides for a college course that taught social scientists to use Python for their profession.
- Practical Business Python is a blog that covers topics such as how to automate generating large Excel spreadsheets or perform analysis when your data is locked in Microsoft Office files.
- Python for the Humanities is a textbook and course on the basics of Python and text processing. Note if you've never worked with Python before the material ramps up quickly after the first chapter so you will likely want to combine it with some other introduction to Python resources.
- Practical Python for Astronomers provides open source workshop materials for teaching students studying astronomy to use Python for data analysis.

Experienced developers new to Python

If you can already program in another language, these resources are better for getting up to speed because they are more concise when explaining introductory topics.

- Learn Python in y minutes provides a whirlwind tour of the Python language. The guide is especially useful if you're coming in with previous software development experience and want to quickly grasp how the language is structured.
- Developers familiar with other languages often have difficulty adapting to accepted Python code style. Make sure to read the PEP8 code style guidelines as well as The Elements of Python Style to know the Python community standards.
- How to Develop Quality Python Code is a good read to begin learning about development environments, application dependencies and project structure.
- The Python module of the week chapters are a good way to get up to speed with the standard library. Doug Hellmann is also now updating the list for changes brought about from the upgrade to Python 3 from 2.x.
- Kenneth Reitz's The Hitchhiker's Guide to Python contains a wealth of information

both on the Python programming language and the community.

- Good to Great Python Reads is a collection of intermediate and advanced Python articles around the web focused on nuances and details of the Python language itself.

Videos, screencasts and presentations

Videos from conferences and meetups along with screencasts are listed on the best Python videos page.

Curated Python packages lists

- awesome-python is an incredible list of Python frameworks, libraries and software. I wish I had this page when I was just getting started.

- easy-python is like awesome-python although instead of just a Git repository this site is in the Read the Docs format.

Podcasts

- Talk Python to Me focuses on the people and organizations coding on Python. Each episode features a different guest interviewee to talk about his or her work.

- Podcast.__init__ is another podcast on "about Python and the people who make it great".

Newsletters

- Python Weekly is a free weekly roundup of the latest Python articles, videos, projects and upcoming events.

- PyCoder's Weekly is another great free weekly email newsletter similar to Python Weekly. The best resources are generally covered in both newsletters but they often cover different articles and projects from around the web.

- Import Python is a newer newsletter than Python Weekly and PyCoder's Weekly. So far I've found this newsletter often pulls from different sources than the other two. It's well worth subscribing to all three so you don't miss anything.

Best Python Videos

If you prefer to learn Python programming by watching videos then this is the resource for you. I've watched hundreds of live technical talks and combed through videos to pick out the ones with great speakers who'll teach you the most about the language and ecosystem.

This page links to the best free videos as well as other video lists so you can do your own searching through the huge backlog of conference and meetup talks from the past several years.

Live Coding

Learn by watching developers code. Follow my account on Twitch Creative then enable email alerts when I go live. On Wednesdays I code Python and Swift applications, answering any chat questions along the way. I also hold office hours on Fridays where you can drop in and ask questions about your coding issues.

Web development with Django, Flask and other frameworks

- Kate Heddleston gave a talk at PyCon 2014 called "Full-stack Python Web Applications" with clear visuals for how numerous layers of the Python web stack fit together. There are also slides available from the talk with all the diagrams.

- My EuroPython 2014 "Full Stack Python" talk goes over each topic from this guide and provides context for how the pieces fit together. The talk slides are also available.

- Kate Heddleston and I gave a talk at DjangoCon 2014 called Choose Your Own Django Deployment Adventure which walked through many of the scenarios you'd face when deploying your first Django website.

- The Discover Flask series is a detailed Flask tutorial on video with corresponding code examples on GitHub.

- Designing Django's Migrations covers Django 1.7's new migrations from the main programmer of South and now Django's built-in migrations, Andrew Godwin.
- GoDjango screencasts and tutorials are free short videos for learning how to build Django applications.
- Getting Started with Django is a series of video tutorials for the framework.
- The videos and slides from Django: Under the Hood 2014 are from Django core committers and provide insight into the ORM, internationalization, templates and other topics.
- DjangoCon US videos from 2014, 2013, 2012, 2011, as well as earlier US and DjangoCon EU conferences are all available free of charge.
- DjangoCon EU videos are also available from 2015.

Core Python language videos

- Jessica McKellar's Building and breaking a Python sandbox is a fascinating walk through the lower layers of the Python interpreter.
- Brandon Rhodes' All Your Ducks In A Row: Data Structures in the Std Lib and Beyond goes through how data structures are implemented, how to select a data structure appropriate to your application and how the list and dictionary can be used in many situations.
- The talk Python Descriptors by Simeon Franklin explains the what and why of this core Python language feature.
- David Beazley gives an amazing live coded performance to show Python concurrency using threads, event loops and coroutines. David makes the live coding look easy but a whole lot of work must've gone into that talk.

Screencasts and class recordings

- Google's Python Class contains lecture videos and exercises for learning Python.

- While there aren't always live streams online, it's worth checking out the Python category on Livecoding.tv to see if anyone is streaming or has a recording of working on a Python project. Even experienced developers can learn a whole lot from watching other developer's work on their projects. I also stream a couple times per week so if you follow me on my username you'll get an alert when I go online.

Video compilations

- PyVideo organizes and indexes thousands of Python videos from both major conferences and meetups.

- Incredible Technical Speakers is a repository I put together that features software developer speakers talking about programming language agnostic topics. The list is intended to emphasize professional software developers who also have the ability to engage an audience of peers with an exciting talk.

Development Environments

Development Environments

A development environment is a combination of a text editor and the Python interpreter. The text editor allows you to write the code. The interpreter provides a way to execute the code you've written. A text editor can be as simple as Notepad on Windows or more complicated as a complete integrated development environment (IDE) such as PyCharm which runs on any major operating system.

Why is a development environment necessary?

Python code needs to be written, executed and tested to build applications. The text editor provides a way to write the code. The interpreter allows it to be executed. Testing to see if the code does what you want can either be done manually or by unit and functional tests.

A development environment example

Here's what I (the author of Full Stack Python, Matt Makai) use to develop most of my Python applications. I have a Macbook Pro with Mac OS X as its base operating system. Ubuntu 14.04 LTS is virtualized on top with Parallels. My code is written in vim and executed with the Python 2.7.x interpreter via the command line. I use virtualenv to create separate Python interpreters with their own isolated application dependencies and virtualenvwrapper to quickly switch between the interpreters created by virtualenv.

That's a common set up but you can certainly write great code with a much less expensive set up or a cloud-based development environment.

Open source text editors

- vim is my editor of choice and installed by default on most *nix systems.
- emacs is another editor often used on *nix.

- Atom is an open source editor built by the GitHub team.

Proprietary (closed source) editors

- Sublime Text versions 2 and 3 (currently in beta) are popular text editors that can be extended with code completion, linting, syntax highlighting and other features using plugins.

- Komodo is a cross-platform text editor and IDE for major languages including Python, Ruby, JavaScript, Go and more.

Python-specific IDEs

- PyCharm is a Python-specific IDE built on JetBrains' platform. There are free editions for students and open source projects.

- Wing IDE is a paid development environment with integrated debugging and code completion.

- PyDev is a Python IDE plug in for Eclipse.

Hosted development environment services

In the past couple of years several cloud-based development environments have popped up. These can work great for when you're learning or stuck on a machine with only a browser but no way to install your own software. Most of these have free tiers for getting started and paid tiers as you scale up your application.

- Nitrous.io

- Cloud9

- Terminal

- Koding

Development environment resources

- If you're considering the cloud-based development environment route, check out this great article comparing Cloud9, Koding and Nitrous.io by Lauren Orsini. She also explains more about what a cloud IDE is and is not.

- Real Python has an awesome, detailed post on setting up your Sublime Text 3 environment as a full-fledged IDE.

- The Hitchhiker's Guide to Python has a page dedicated to development environments.

- Choosing the best Python IDE is a review of six IDEs. PyCharm, Wing IDE and PyDev stand out above the other three in this review.

- PyCharm: The Good Parts shows you how to be more efficient and productive with that IDE if it's your choice for writing Python code.

- JetBrains' PyCharm Blog is required reading if you're using the IDE or considering trying it. One of the core developers also an interview on the Talk Python to Me podcast that's worth listening to.

- PyCharm vs Sublime Text has a comparison of several features between the two editors.

Vim

Vim, short for Vi IMproved, is a configurable text editor often used as a Python development environment. Vim proponents commonly cite the numerous plugins, Vimscript and logical command language as major Vim strengths.

Why is Vim a good Python development environment?

Vim's philosophy is that developers are more productive when they avoid taking their hands off the keyboard. Code should flow naturally from the developer's thoughts through the keyboard and onto the screen. Using a mouse or other peripheral is a detriment to the rate at which a developer's thoughts become code.

Vim has a logical, structured command language. When a beginner is learning the editor she may feel like it is impossible to understand all the key commands. However, the commands stack together in a logical way so that over time the editor becomes predictable.

Configuring Vim with a Vimrc file

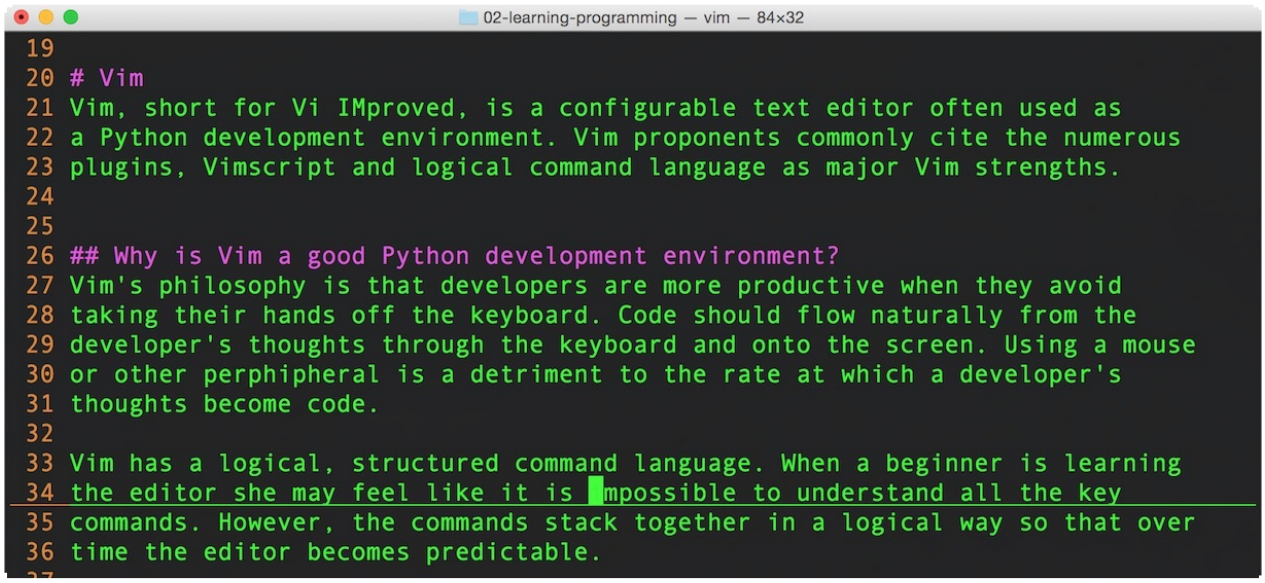
The Vimrc file is used to configure the Vim editor. A Vimrc file can range from nothing in it to very complicated with hundreds or thousands of lines of configuration commands.

Here's a short, commented example .vimrc file I use for Python development to get a feel for some of the configuration statements:

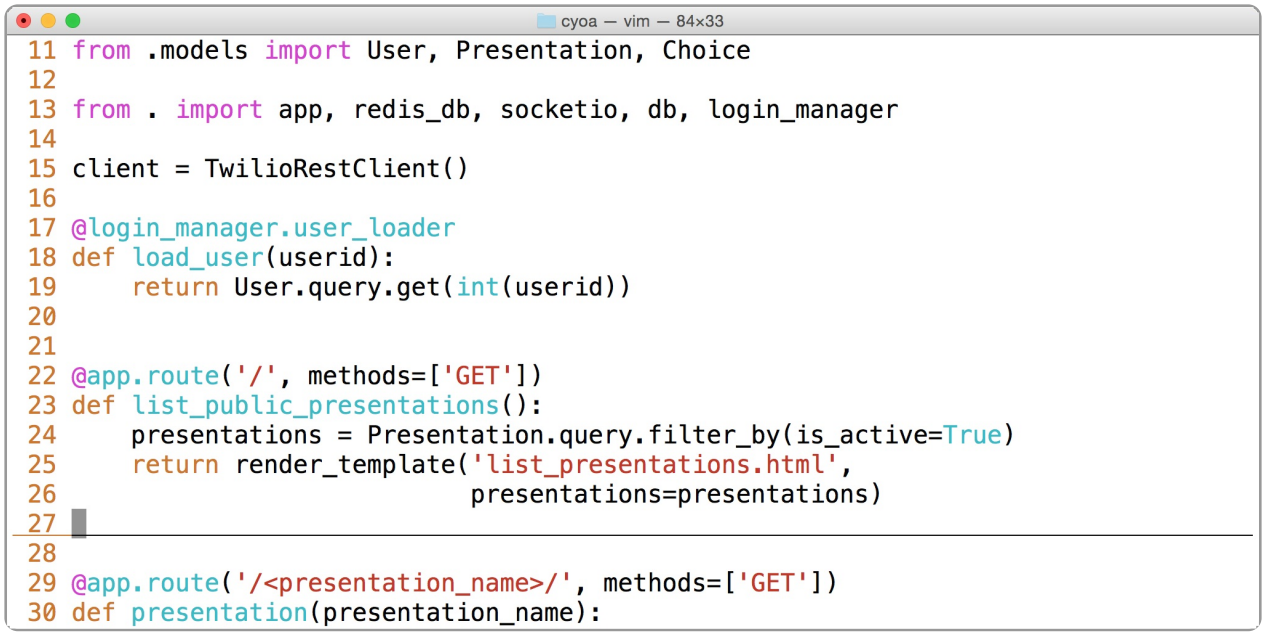
```
" enable syntax highlighting
syntax enable

" show line numbers
set number

" set tabs to have 4 spaces
set ts=4

" indent when moving to the next line while writing code
```

```
set autoindent

" expand tabs into spaces
set expandtab

" when using the >> or << commands, shift lines by 4 spaces
set shiftwidth=4

" show a visual line under the cursor's current line
set cursorline

" show the matching part of the pair for [] {} and ()
set showmatch

" enable all Python syntax highlighting features
let python_highlight_all = 1
```

Here is how these configuration options look with a dark background on Mac OS X while editing the markdown for this webpage (how meta!).



Take a look at another example using these configuration options, this time with a light background and editing Python code from my Choose Your Own Adventures Presentations project.



The Vimrc file lives under the home directory of the user account running Vim. For example, when my user account is 'matt', on Mac OS X my Vimrc file is found at `/Users/matt/.vimrc`. On Ubuntu Linux my .vimrc file can be found within the `/home/matt/` directory.

If a Vimrc file does not already exist, just create it within the user's home directory and it will be picked up by Vim the next time you open the editor.

Vim tutorials

Vim has a reputation for a difficult learning curve, but it's much easier to get started with these tutorials.

- Vim Adventures is a cute, fun browser-based game that helps you learn Vim commands by playing through the adventure.

- Learn Vim Progressively is a wonderful tutorial that follows the path I took when learning Vim: learn just enough to survive with it as your day-to-day editor then begin adding more advanced commands on top.

- A vim Tutorial and Primer is an incredibly deep study in how to go from beginner to knowledgeable in Vim.

- Vim as a Language explains the language syntax and how you can build up over time to master the editor.

- How to install and use Vim on a cloud server along with How to use Vim for advanced editing of code on a VPS are two detailed Digital Ocean guides for getting up and running with Vim, regardless of whether you're using it locally or on a cloud server.

- In Vim: revisited the author explains his on-again off-again relationship with using Vim. He then shows how he configures and uses the editor so it sticks as his primary code editing tool.

Vimrc resources

These are a few resources for learning how to structure a `.vimrc` file. I recommend adding configuration options one at a time to test them individually instead of going whole hog with a Vimrc you are unfamiliar with.

- A Good Vimrc is a fantastic, detailed overview and opinionated guide to configuring Vim. Highly recommended for new and experienced Vim users.

- Vim and Python shows and explains many Python-specific .vimrc options.

- This repository's folder with Vimrc files has example configurations that are well commented and easy to learn from.

- For people who are having trouble getting started with Vim, check out this blog post on the two simple steps that helped this author learn Vim.

Vim installation guides

These installation guides will help you get Vim up and running on Mac OS X, Linux and Windows.

- Upgrading Vim on OS X explains why to upgrade from Vim 7.2 to 7.3+ and how to do it using Homebrew.

- The easiest way to install Vim on Windows 7+ is to download and run the gvim74.exe file.

- On Linux make sure to install the vim package with `sudo apt-get install vim`.

- If you're using PyCharm as your IDE you won't need to install Vim as a separate text editor - instead use the IdeaVim PyCharm plugin to get Vim keybindings, visual/insert mode, configuration with ~/.ideavimrc and other Vim emulation features.

Using Vim as a Python IDE

Once you get comfortable with Vim as an editor, there are several configuration options and plugins you can use to enhance your Python productivity. These are the resources and tutorials to read when you're ready to take that step.

- VIM and Python - a Match Made in Heaven details how to set up a powerful VIM environment geared towards wrangling Python day in and day out.
- The python-mode project is a Vim plugin with syntax highlighting, breakpoints, PEP8 linting, code completion and many other features you'd expect from an integrated development environment.
- Vim as Your IDE discusses how to set up Vim for greater productivity once you learn the initial Vim language for using the editor.
- Vim as a Python IDE goes through the steps necessary to make Vim into a more comfortable environment for Python development.
- Setting up Vim for Python has a well written answer on Stack Overflow for getting started with Vim.
- If you're writing your documentation in Markdown using Vim, be sure to read this insightful post on a Vim setup for Markdown.

Vim Plugin Managers

- Vundle comes highly recommended as a plugin manager for Vim.
- Pathogen is a widely used used plugin manager.
- Vim-plug bills itself as a minimalistic Vim plugin manager.

Vim Plugin resources

- 5 Essential VIM Plugins That Greatly Increase my Productivity covers the author's experience with the Vundle, NERDTree, ctrlp, Syntastic and EasyMotion Vim plugins.
- Getting more from Vim with plugins provides a list of plugins with a description for each one on its usefulness. The comments at the bottom are also interesting as people have suggested alternatives to some of the plugins mentioned in the post.
- Powerline is a popular statusline plugin for Vim that works with both Python 2 and 3.

Emacs

Emacs is an extensible text editor that can be customized by writing Lisp code.

Why is Emacs a good choice for Python coding?

Emacs is designed to be customized via the built-in Lisp interpreter and package manager. The package manager, named package.el, has menus for handling installation. The largest Lisp Package Archive is Melpa, which provides automatic updates from upstream sources.

Macros are useful for performing repetitive actions in Emacs. A macro is just a recording of a previous set of keystrokes that can be replayed to perform future actions.

Hooks, which are Lisp variables that hold lists of functions to call, provide an extension mechanism for Emacs. For example, `kill-emacs-hook` runs before exiting Emacs so functions can be loaded into that hook to perform necessary actions before the exiting completes.

General Emacs resources

- GNU Emacs Manual provides an official in-depth review for how to use Emacs.
- Emacs - the Best Python Editor? continues the excellent Real Python series showing how to get started with editors. In addition to this Emacs post, there are also posts on Vim and Sublime Text 3 specifically for Python development.
- Emacs Redux is a blog with tips and tricks for how to use Emacs effectively.
- Emacs Rocks is a video tutorial series for Emacs.
- What the .emacs.d?! provides a bunch of tiny optimizations for Emacs' workflow.

Notable Elisp Packages

- Magit allows the user to inspect and modify Git repositories from within Emacs.
- company-mode creates a modular in-buffer completion framework.
- Flycheck provides syntax checking.
- anaconda-mode is specific to Python development and allows code navigation, documentation lookup and code completion. The jedi library is used under the hood.
- Tern is a stand-alone code-analysis engine for JavaScript. It can be integrated within a Django project via the tern-django package.

Popular user configurations

- Prelude is an enhanced Emacs version 24 distribution.
- A reasonable Emacs config shows a batteries-includes Emacs configuration bundle.
- Emacs settings is a repository of configurations used in the Emacs Rocks screencasts.
- Spacemacs mashes together Emacs' extensibility and Vim's ergonomic text editing features.

Language Concepts

Python Programming Language

The Python programming language is an open source, widely-used tool for creating software applications.

What is Python used for?

Python is often used to build and deploy web applications and web APIs. Python can also analyze and visualize data and test software, even if the software being tested was not written in Python.

General Python language resources

- The online Python tutor visually walks through code and shows how it executes on the Python interpreter.
- Python Module of the Week is a tour through the Python standard library.
- What the heck does "pythonic" mean? explains the difference in accepted Python coding style compared to other ways to write Python code that, while syntactically correct, are less maintainable and therefore should not be used.
- A Python interpreter written in Python is incredibly meta but really useful for wrapping your head around some of the lower level stuff going on in the language.
- A few things to remember while coding in Python is a nice collection of good practices to use while building programs with the language.
- Python internals: adding a new statement to Python
- Python tricks that you can't live without is a slideshow by Audrey Roy that goes over code readability, linting, dependency isolation, and other good Python practices.
- Python innards introduction explains how some of Python's internal execution happens.

- What is a metaclass in Python is one of the best Stack Overflow answers about Python.
- Armin Roacher presented things you didn't know about Python at PyCon South Africa in 2012.
- Writing idiomatic Python is a guide for writing Pythonic code.
- The thing that runs your Python is a summary of what one developer learned about PyPy while researching it.

Python ecosystem resources

There's an entire page on best Python resources with links but the following resources are a better fit for when you're past the very beginner topics.

- The Python Ecosystem: An Introduction provides context for virtual machines, Python packaging, pip, virtualenv and many other topics after learning the basic Python syntax.
- The Python Subreddit rolls up great Python links and has an active community ready to answer questions from beginners and advanced Python developers alike.
- The blog Free Python Tips provides posts on Python topics as well as news for the Python ecosystem.
- Python Books is a collection of freely available books on Python, Django, and data analysis.
- Python IAQ: Infrequently Asked Questions is a list of quirky queries on rare Python features and why certain syntax was or was not built into the language.

Generators

Generators are a Python core language construct that allow a function's return value to to behave as an iterator. A generator can allow more efficient memory usage by allocating and deallocating memory during the context of a large number of iterations. Generators are defined in PEP255 and included in the language as of Python 2.2 in 2001.

Python generator resources

- An introduction to Python generators by Intermediate Pythonista is a well done post with code examples.
- This blog post entitled Python Generators specifically focuses on generating dictionaries. It provides a good introduction for those new to Python.
- Python 201: An Intro to Generators is another short but informative read with example generators code.
- Iterators & Generators provides code examples for these two constructs and some simple explanations for each one.
- Python: Generators - How to use them and the benefits you receive is a screencast with code that walks through generators in Python.
- The question to Understanding Generators in Python? on Stack Overflow has an impressive answer that clearly lays out the code and concepts involved with Python generators.
- Generator Tricks for Systems Programmers provides code examples for using generators. The material was originally presented in a PyCon workshop for systems programmers but is relevant to all Python developers working to understand appropriate ways to use generators.

Comprehensions

Comprehensions are a Python language construct for concisely creating data in lists, dictionaries and sets. List comprehensions are included in Python 2 while dictionary and set comprehensions were introduced to the language in Python 3.

Why are comprehensions important?

Comprehensions are a more clear syntax for populating conditional data in the core Python data structures. Creating data without comprehensions often involves nested loops with conditionals that can be difficult for code readers to properly evaluate.

Example code

List comprehension:

```
>>> double_digit_evens = [e*2 for e in range(5, 50)]
>>> double_digit_evens
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50,
```

Set comprehension:

```
>>> double_digit_odds = {e*2+1 for e in range(5, 50)}
{11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51,
```

Dictionary comprehension:

```
>>> {e: e*10 for e in range(1, 11)}
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 7: 70, 8: 80, 9: 90, 10: 100}
```

Comprehension resources

- Intermediate Python's Python Comprehensions post gives a well written overview of comprehensions for the three core Python data structures.
- Python List Comprehensions: Explained Visually explains how the common idiom for iteration became syntactic sugar in the language itself and how you can use it in your own programs.
- The Python 3 Patterns and Idioms site has an overview of comprehensions including code examples and diagrams to explain how they work.
- Idiomatic Python: Comprehensions explains how Python's comprehensions were inspired by Haskell's list comprehensions. It also provides clear examples that show how comprehensions are shorthand for common iteration code, such as copying one list into another while performing some operation on the contained elements.
- Learning Python by example: list comprehensions gives an example of an incorrect list comprehension then shows how to correct its issues.
- List comprehensions in Python covers what the code for list comprehensions looks like and gives some example code to show how they work.
- An Introduction to Python Lists is a solid overview of Python lists in general and tangentially covers list comprehensions.

Web Development

Web Development

Web development is the umbrella term for conceptualizing, creating, deploying and operating web applications and application programming interfaces for the Web.

Why is web development important?

The Web has grown a mindboggling amount in the number of sites, users and implementation capabilities since the first website went live in 1989. Web development is the concept that encompasses all the activities involved with websites and web applications.

How does Python fit into web development?

Python can be used to build server-side web applications. While a web framework is not required to build web apps, it's rare that developers would not use existing open source libraries to speed up their progress in getting their application working.

Python is not used in a web browser. The language executed in browsers such as Chrome, Firefox and Internet Explorer is JavaScript. Projects such as pyjs can compile from Python to JavaScript. However, most Python developers are write their web applications in a combination of Python, which is executed on the server side and JavaScript, which is downloaded to the web browser client and executed there.

Web development resources

- Web application development is different and better provides some context for how web development has evolved from writing static HTML files into the complex JavaScript client-side applications produced today.

- While not Python-specific, Mozilla put together a Learning the Web tutorial for beginners and intermediate web users who want to build websites. It's worth a look for general web development learning.

- The Evolution of the Web visualizes how web browsers and related technologies

have changed over time as well as the overall growth of the Internet in the amount of data transferred. Note that the visualization unfortunately stops around the beginning of 2013 but it's a good way to explore what happened in the first 24 years.

- Web development involves HTTP communication between the server, hosting a website or web application, and the client, a web browser. Knowing how web browsers works is important as a developer, so take a look at this article on what's in a web browser.

- Three takeaways for web developers after two weeks of painfully slow Internet is a must-read for every web developer. Not everyone has fast Internet service, whether because they are in a remote part of the world or they're just in a subway tunnel. Optimizing sites so they work in those situations is important for keeping your users happy.

Web frameworks

A web framework is a code library that makes a developer's life easier when building reliable, scalable and maintainable web applications.

Why are web frameworks necessary?

Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application.

Common web framework functionality

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

1. URL routing
2. HTML, XML, JSON, and other output format templating
3. Database manipulation
4. Security against Cross-site request forgery (CSRF) and other attacks
5. Session storage and retrieval

Not all web frameworks include code for all of the above functionality. Frameworks fall on the spectrum from executing a single use case to providing every known web framework feature to every developer. Some frameworks take the "batteries-included" approach where everything possible comes bundled with the framework while others have a minimal core package that is amenable to extensions provided by other packages.

For example, the Django web application framework includes an Object-Relational Mapping (ORM) layer that abstracts relational database read, write, query, and delete operations. However, Django's ORM cannot work without significant modification on non-relational databases such as MongoDB.

Some other web frameworks such as Flask and Pyramid are easier to use with non-relational databases by incorporating external Python libraries. There is a spectrum between minimal functionality with easy extensibility on one end and including everything in the framework with tight integration on the other end.

Comparing web frameworks

Are you curious about how the code in a Django project is structured compared with Flask? Check out this Django web application tutorial and then view the same application built with Flask.

There is also a repository called compare-python-web-frameworks where the same web application is being coded with varying Python web frameworks, templating engines and object-relational mappers.

Do I have to use a web framework?

Whether or not you use a web framework in your project depends on your experience with web development and what you're trying to accomplish. If you are a beginner programmer and just want to work on a web application as a learning project then a framework can help you understand the concepts listed above, such as URL routing, data manipulation and authentication that are common to the majority of web applications.

On the other hand if you're an experienced programmer with significant web development experience you may feel like the existing frameworks do not match your project's requirements. In that case, you can mix and match open source libraries such as Werkzeug for WSGI plumbing with your own code to create your own framework. There's still plenty of room in the Python ecosystem for new frameworks to satisfy the needs of web developers that are unmet by Django, Flask, Pyramid, Bottle and many others.

In short, whether or not you need to use a web framework to build a web application depends on your experience and what you're trying to accomplish. Using a web framework to build a web application certainly isn't required, but it'll make most developers' lives easier in many cases.

Web framework resources

- "What is a web framework?" is an in-depth explanation of what web frameworks are and their relation to web servers.
- Check out the answer to the "What is a web framework and how does it compare to LAMP?" question on Stack Overflow.
- Frameworks is a really well done short video that explains how to choose between web frameworks. The author has some particular opinions about what should be in a framework. For the most part I agree although I've found sessions and database ORMs to be a helpful part of a framework when done well.
- Django vs Flask vs Pyramid: Choosing a Python Web Framework contains background information and code comparisons for similar web applications built in these three big Python frameworks.
- This Python web framework roundup covers Django, Flask and Bottle as well as several other lesser known Python frameworks.
- This fascinating blog post takes a look at the code complexity of several Python web frameworks by providing visualizations based on their code bases.
- Python's web frameworks benchmarks is a test of the responsiveness of a framework with encoding an object to JSON and returning it as a response as well as retrieving data from the database and rendering it in a template. There were no conclusive results but the output is fun to read about nonetheless.
- What web frameworks do you use and why are they awesome? is a language agnostic Reddit discussion on web frameworks. It's interesting to see what programmers in other languages like and dislike about their suite of web frameworks compared to the main Python frameworks.
- This user-voted question & answer site asked "What are the best general purpose Python web frameworks usable in production?". The votes aren't as important as the list of the many frameworks that are available to Python developers.

Web frameworks learning checklist

1. Choose a major Python web framework (Django or Flask are recommended) and stick with it. When you're just starting it's best to learn one framework first instead of bouncing around trying to understand every framework.

2. Work through a detailed tutorial found within the resources links on the framework's page.

3. Study open source examples built with your framework of choice so you can take parts of those projects and reuse the code in your application.

4. Build the first simple iteration of your web application then go to the deployment section to make it accessible on the web.

Django

Django is a widely-used Python web application framework with a "batteries-included" philosophy. The principle behind batteries-included is that the common functionality for building web applications should come with the framework instead of as separate libraries.

For example, authentication, URL routing, a templating system, an object-relational mapper (ORM), and database schema migrations (as of version 1.7) are all included with the Django framework. Compare that included functionality to the Flask framework which requires a separate library such as Flask-Login to perform user authentication.

The batteries-included and extensibility philosophies are simply two different ways to tackle framework building. Neither philosophy is inherently better than the other one.

Why is Django a good web framework choice?

The Django project's stability, performance and community have grown tremendously over the past decade since the framework's creation. Detailed tutorials and good practices are readily available on the web and in books. The framework continues to add significant new functionality such as database migrations with each release.

I highly recommend the Django framework as a starting place for new Python web developers because the official documentation and tutorials are some of the best anywhere in software development. Many cities also have Django-specific groups such

as Django District, Django Boston and San Francisco Django so new developers can get help when they are stuck.

There's some debate on whether learning Python by using Django is a bad idea. However, that criticism is invalid if you take the time to learn the Python syntax and language semantics first before diving into web development.

Django books and tutorials

There are a slew of free or low cost resources out there for Django. Since Django was released over 10 years ago and has had a huge number of updates since then, when you're looking for an up-to-date Django book check out the list below or read this post showing current Django books as of Django 1.7.

- Test-Driven Development with Python focuses on web development using Django and JavaScript. This book uses the development of a website using the Django web framework as a real world example of how to perform test-driven development (TDD). There is also coverage of NoSQL, websockets and asynchronous responses. The book can be read online for free or purchased in hard copy via O'Reilly.

- Tango with Django is an extensive set of free introductions to using the most popular Python web framework. Several current developers said this book really helped them get over the initial framework learning curve. It's recently been updated for Django 1.7!

- The Django Girls Tutorial is a great tutorial that doesn't assume any prior knowledge of Python or Django while helping you build your first web application.

- 2 Scoops of Django by Daniel Greenfeld and Audrey Roy is well worth the price of admission if you're serious about learning how to correctly develop Django websites.

- Effective Django is another free introduction to the web framework.

- The Django subreddit often has links to the latest resources for learning Django and is also a good spot to ask questions about it.

- Steve Losh wrote an incredibly detailed Django Advice guide.
- Lightweight Django has several nice examples for breaking Django into smaller simpler components.
- The Definitive Guide to Django Deployment explains the architecture of the resulting set up and includes Chef scripts to automate the deployment.
- Deploying a Django app on Amazon EC2 instance is a detailed walkthrough for deploying an example Django app to Amazon Web Services.
- This step-by-step guide for Django shows how to transmit data via AJAX with JQuery.
- django-awesome is a curated list of Django libraries and resources.
- Starting a Django Project answers the question, “How do I set up a Django (1.5, 1.6, or 1.7) project from scratch?”
- This Django tutorial shows how to build a project from scratch using Twitter Bootstrap, Bower, Requests and the Github API.
- The recommended Django project layout is helpful for developers new to Django to understand how to structure the directories and files within apps for projects.
- This Python Social Auth for Django tutorial will show you how to integrate social media sign in buttons into your Django application.
- Luke Plant writes about his approach to class based views (CBVs), which often provoke heated debate in the Django community for whether they are a time saver or "too much magic" for the framework.
- How to serve Django apps with uWSGI and Nginx on 14.04 and how to set up Django with PostgreSQL, Nginx and Gunicorn are detailed tutorials that walk through each step in the deployment process.
- Working with time zones is necessary for every web application. This blog post on pytz and Django is a great start for figuring out what you need to know.

Django videos

Are you looking for Django videos in addition to articles? There is a special section for Django and web development on the best Python videos page.

Django migrations

- Paul Hallett wrote a detailed Django 1.7 app upgrade guide on the Twilio blog from his experience working with the django-twilio package.
- Real Python's migrations primer explores the difference between South's migrations and the built-in Django 1.7 migrations as well as how you use them.
- Andrew Pinkham's "Upgrading to Django 1.7" series is great learning material for understanding what's changed in this major release and how to adapt your Django project. Part 1, part 2 and part 3 and part 4 are now all available to read.
- Channels are a new mechanism in Django 1.9 (as a standalone app, later for incorporation into the core framework) for real-time full-duplex communication between the browser and the server based on WebSockets. This tutorial shows how to get started with Django Channels in your project.
- Django migrations without downtimes shows one potential way of performing on-line schema migrations with Django.

Django testing

- Integrating Front End Tools with Django is a good post to read for figuring out how to use Gulp for handling front end tools in development and production Django sites.
- Getting Started with Django Testing will help you stop procrastinating on testing your Django projects if you're uncertain where to begin.
- Testing in Django provides numerous examples and explanations for how to test your Django project's code.
- Django views automated testing with Selenium gives some example code to get up

and running with Selenium browser-based tests.

Django with Angular (Djangular) resources

- Getting Started with Django Rest Framework and AngularJS is a very detailed introduction to Djangular with example code.
- Building Web Applications with Django and AngularJS is a very detailed guide for using Django as an API layer and AngularJS as the MVC front end in the browser.
- This end to end web app with Django-Rest-Framework & AngularJS part 1 tutorial along with part 2, part 3 and part 4 creates an example blog application with Djangular. There is also a corresponding GitHub repo for the project code.
- Django-angular is a code library that aims to make it easier to pair Django with AngularJS on the front end.

Django ORM resources

Django comes with its own custom object-relational mapper (ORM) typically referred to as "the Django ORM". Learn more about the Django ORM on the Python object-relational mappers page that includes a section specifically for the Django ORM as well as additional resources and tutorials.

Static and media files

Deploying and handling static and media files can be confusing for new Django developers. These resources along with the static content page are useful for figuring out how to handle these files properly.

- Using Amazon S3 to Store your Django Site's Static and Media Files is a well written guide to a question commonly asked about static and media file serving.
- Loading Django FileField and ImageFields from the file system shows how to load a model field with a file from the file system.
- Restricting access to user-uploaded files in Django provides a protection

mechanism for media files.

Open source Django example projects

- Browser calls with Django and Twilio shows how to build a web app with Django and Twilio Client to turn a user's web browser into a full-fledged phone. Pretty awesome!

- Txt 2 React is a full Django web app that allows audiences to text in during a presentation with feedback or questions.

- Openduty is a website status checking and alert system similar to PagerDuty.

- Courtside is a pick up sports web application written and maintained by the author of PyCoder's Weekly.

- These two Django Interactive Voice Response (IVR) system web application repositories part 1 and part 2 show you how to build a really cool Django application. There's also an accompanying blog post with detailed explanations of each step.

- Taiga is a project management tool built with Django as the backend and AngularJS as the front end.

Django project templates

- Caktus Group's Django project template is Django 1.6+ ready.

- Cookiecutter Django is a project template from Daniel Greenfeld, for use with Audrey Roy's Cookiecutter. The template results are Heroku deployment-ready.

- Two Scoops Django project template is also from the PyDanny and Audrey Roy. This one provides a quick scaffold described in the Two Scoops of Django book.

- Sugardough is a Django project template from Mozilla that is compatible with cookiecutter.

Django learning checklist

1. Install Django on your local development machine.
2. Work through the initial "polls" tutorial.
3. Build a few more simple applications using the tutorial resources found in the "Django resources" section.
4. Start coding your own Django project with help from the official documentation and resource links below. You'll make plenty of mistakes which is critical on your path to learning the right way to build applications.
5. Read 2 Scoops of Django to understand Django good practices and learn better ways of building Django web applications.
6. Move on to the deployment section to get your Django project on the web.

Flask

Flask is a Python web framework built with a small core and easy-to-extend philosophy.

Why is Flask a good web framework choice?

Flask is considered more Pythonic than Django because Flask web application code is in most cases more explicit. Flask is easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

For example, here's a valid "hello world" web application with Flask (the equivalent in Django would be significantly more code):

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

Flask was also written several years after Django and therefore learned from the Python community's reactions as the framework evolved. Jökull Sólberg wrote a great piece articulating to this effect in his experience switching between Flask and Django.

Flask resources

- The Flask mega tutorial by Miguel Grinberg is a perfect starting resource for using this web framework. Each post focuses on a single topic and builds on previous posts. The series includes 18 parts: #1 Hello World, #2 Templates, #3 Web Forms, #4 Database, #5 User Logins, #6 Profile Page and Avatars, #7 Unit Testing, #8 Followers, Contacts, and Friends, #9 Pagination, #10 Full Text Search, #11 Email Support, #12 Facelift, #13 Dates and Times, #14 I18n and L10n, #15 Ajax, #16 Debugging, Testing and Profiling, #17 Deployment on Linux and #18 Deployment on the Heroku Cloud. Miguel also wrote the O'Reilly Flask Web Development book which is also an excellent learning resource.

- If you're looking for a fun tutorial with Flask and WebSockets, check out my blog post on creating Choose Your Own Adventure Presentations with Reveal.js, Python and WebSockets. Follow up that tutorial by building an admin interface in part 1, part 2 and part 3 that'll show you how to use forms and SQLAlchemy. There is also a companion open source GitHub repository for the app with tags for each step in the blog posts.

- This simple Flask app uses Twilio Voice to do voice calling with three participants. It's a fun introduction to Python and Flask I wrote for the Twilio blog.

- The Flask Extensions Registry is a curated list of the best packages that extend Flask. It's the first location to look through when you're wondering how to do something that's not in the core framework.

- Explore Flask is a public domain book that was previously backed on Kickstarter and cost money for about a year before being open sourced. The book explains best practices and patterns for building Flask apps.

- How I Structure My Flask Application walks through how this developer organizes the components and architecture for his Flask applications.

- Nice post by Jeff Knupp on Productionizing a Flask App.

- The Plank & Whittle blog has two posts, one on Packaging a Flask web app and another on Packaging a Flask app in a Debian package once you've built an app and want to deploy it.

- The Tuts+ Flask tutorial is another great walkthrough for getting started with the framework.

- Create Your Own Obnoxiously Simple Messaging App Just Like Yo is a silly walkthrough of very basic Flask web application that uses Nitrous.io to get started and Twilio for SMS.

- The blog post series "Things which aren't magic" covers how Flask's ubiquitous @app.route decorator works under the covers. There are two parts in the series, part 1 and part 2.

- Flask by Example: Part 1 shows the basic first steps for setting up a Flask project. Part 2 explains how to use PostgreSQL, SQLAlchemy and Alembic. Part 3 describes text processing with BeautifulSoup and NLTK. Part 4 shows how to build a task queue with Flask and Redis.

- Branded MMS Coupon Generation with Python and Twilio is a Flask tutorial I wrote for building a web application that can send branded barcode coupons via MMS. The post goes through every step from a blank directory until you have a working app that you can deploy to Heroku.

- How to Structure Large Flask Applications covers a subject that comes up quickly once you begin adding significant functionality to your Flask application.

- Flask Blueprint templates shows a way of structuring your `__init__.py` file with blueprints for large projects.

- Video streaming with Flask is another fantastic tutorial by Miguel Grinberg that covers video streaming.

- One line of code cut our Flask page load times by 60% is an important note about optimizing Flask template cache size to dramatically increase performance in some cases.

- Unit Testing Your Twilio App Using Python's Flask and Nose covers integrating the Twilio API into a Flask application and how to test that functionality with nose.

- The Flask documentation has some quick examples for how to deploy Flask with standalone WSGI containers.

- Handling Email Confirmation in Flask is a great walkthrough for a common use case of ensuring an email address matches with the user's login information.

- If you're not sure why `DEBUG` should be set to `False` in a production deployment, be sure to read this article on how Patreon got hacked.

Open source Flask example projects

- Choose Your Own Adventure Presentations combines Flask with Reveal.js and text messages to create presentations where the audience can vote on how the story should proceed. The code is all open source under an MIT license and also uses the Flask-SocketIO and Flask-WTF projects to support voting and form input.

- Skylines is an open source flight tracking web application built with Flask. You can check out a running version of the application.

- Microblog is the companion open source project that goes along with Miguel Grinberg's O'Reilly Flask book.

- Flaskr TDD takes the official Flask tutorial and adds test driven development and JQuery to the project.

- Here is a note-taking app along with the source code in Gists.

- Bean Counter is an open source Flask app for tracking coffee.

- FlaskBB is a Flask app for a discussion forum.

- psdash is an app built with Flask and psutils to display information about the computer it is running on.

Flask project templates

- Use the Flask App Engine Template for getting set up on Google App Engine with Flask.

- Flask Foundation is a starting point for new Flask projects. There's also a companion website for the project that explains what extensions the base project

includes.

- Cookiecutter Flask is a project template for use with Cookiecutter.
- Flask-Boilerplate provides another starting project with sign up, log in and password reset.

Flask framework learning checklist

1. Install Flask on your local development machine.
2. Work through the 18-part Flask tutorial listed first under "Flask resources" above.
3. Read through Flask Extensions Registry to find out what extensions you'll need to build your project.
4. Start coding your Flask app based on what you learned from the 18 part Flask tutorial plus open source example applications found below.
5. Move on to the deployment section to get your initial Flask project on the web.

Bottle

Bottle is a WSGI-compliant single source file web framework with no external dependencies except for the standard library included with Python.

Bottle resources

- The official Bottle tutorial provides a thorough view of basic concepts and features for the framework.
- Digital Ocean provides an extensive introductory post on Bottle.
- Developing With Bottle details how to create a basic application with Bottle.
- This tutorial provides a walkthrough for getting started with Bottle.
- Here's a short code snippet for creating a RESTful API with Bottle and MongoDB.
- This tutorial is another Bottle walkthrough for creating a RESTful web API.
- BAM! A Web Framework "Short Stack" is a walkthrough of using Bottle, Apache and MongoDB to create a web application.
- Bottle, full stack without Django does a nice job of connecting SQLAlchemy with Bottle and building an example application using the framework.
- Using bottle.py in Production has some good tips on deploying a Bottle app to a production environment.
- Jinja2 Templates and Bottle shows how to use Jinja instead of the built-in templating engine for Bottle page rendering.
- How to build a web app using Bottle with Jinja2 in Google App Engine provides a tutorial for using Bottle on the Google App Engine platform-as-a-service.

Open source Bottle example projects

- Pattle is a pastebin clone built with Bottle.
- Decanter is a library for structuring Bottle projects.

Bottle framework learning checklist

1. Download Bottle or install via pip with `pip install bottle` on your local development machine.
2. Work through the official Bottle tutorial.
3. Start coding your Bottle app based on what you learned in the official tutorial plus reading open source example applications found above.
4. Move on to the deployment section to get your initial Bottle application on the web.

Pyramid

Pyramid is an open source WSGI web framework based on the Model-View-Controller (MVC) architectural pattern.

Open source Pyramid example apps

- pyramid_blogr is an example project that shows how to build a blog with Pyramid modeled on the Flaskr tutorial.
- pyramid_appengine provides a project skeleton for running Pyramid on Google App Engine.

Pyramid resources

- Try Pyramid is a landing page that explains the advantages of the Pyramid framework. It also provides some sample "hello world!" code.
- The first Pyramid app is a good place to start getting your hands dirty with an example project.
- Six Feet Up explains why Pyramid is their choice for rapid development projects in that blog post.

- Build a chat app with Pyramid, SQLDB, and Bluemix is a Pyramid application walkthrough specific to IBM's Bluemix platform.

- Developing Web Apps Using the Python Pyramid Framework is a video from San Francisco Python with an overview of how to install, get started and build a web app with the Pyramid framework.

- This podcast interview with the primary author of the Pyramid framework explains how Pyramid sprang from Pylons and how Pyramid compares to other modern frameworks.

Morepath

Morepath is a micro web framework with a model-driven approach to creating web applications and web APIs.

Morepath's framework philosophy is that the data models should drive the creation via the web framework. By default the framework routes URLs directly to model code, unlike for example Django which requires explicit URL routing by the developer.

Why is Morepath an interesting web framework?

Simple CRUD web applications and APIs can be tedious to build when they are driven straight from data models without much logic between the model and the view. Learn more about how Morepath compares with other web frameworks from the creator.

With the rise of front end JavaScript frameworks, many Python web frameworks are first being used to build RESTful APIs that return JSON instead rendering HTML via a templating system. Morepath appears to have been created with the RESTful API model approach in mind and cuts out the assumption that templates will drive the user interface.

Morepath resources

- On the Morepath is a blog post by Startifact on how they use Morepath and some of the features of the framework.

- A Summer with Morepath describes the author's experience using Morepath, such as how he built a framework around Morepath's core functionality. He eventually became a core contributor to Morepath based on the application he created.

- Build a better batching UI with Morepath and Jinja2 is an introductory post on building a simple web application with the framework. The code for the application is also open source and available on GitHub.

- Morepath's creator gave a great talk on the motivation and structure for the new

framework at EuroPython 2014.

Other Web Frameworks

Python has a significant number of web frameworks outside the usual Django, Flask, Pyramid and Bottle suspects.

TurboGears2

TurboGears2 born as a full stack layer on top of Pylons is now a standalone web framework that can act both as a full stack solution (like Django) or as a micro framework.

Falcon

Falcon is a minimalist web framework designed with web application speed as a top priority.

web.py

web.py is a Python web framework designed for simplicity in building web applications.

- See this Reddit discussion on reasons why to not use web.py for some insight into the state of the project.

web2py

Web2py is a batteries-included philosophy framework with project structure based on model-view-controller patterns.

CherryPy

CherryPy is billed as a minimalist web framework, from the perspective of the amount of code needed to write a web application using the framework. The project has a long

history and made a major transition between the second and third release.

Muffin

Muffin is a web framework built on top of the asyncio module in the Python 3.4+ standard library. Muffin takes inspiration from Flask with URL routes defined as decorators upon view functions. The Peewee ORM is used instead of the more common SQLAlchemy ORM.

Other web framework resources

- This roundup of 14 minimal Python frameworks contains both familiar and less known Python libraries.
- The web micro-framework battle presentation goes over Bottle, Flask, and many other lesser known Python web frameworks.
- A Python newcomer asked the Python Subreddit to explain the differences between numerous Python web frameworks and received some interesting responses from other users.

Other frameworks learning checklist

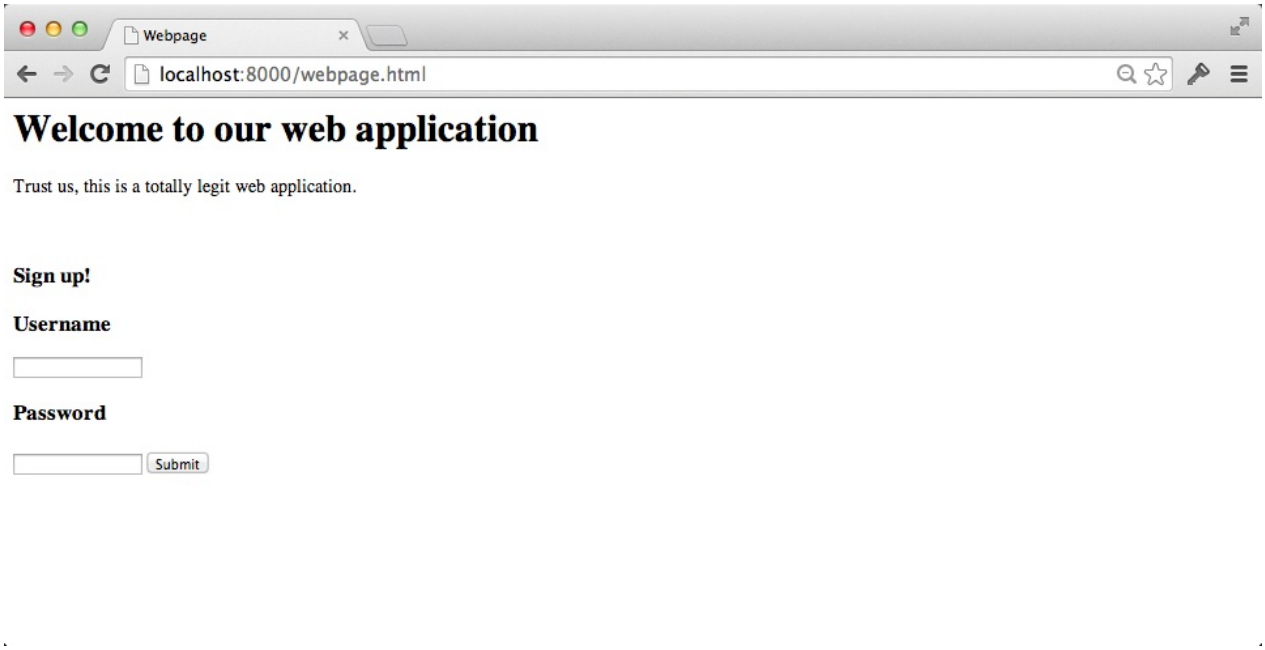
1. Read through the web frameworks listed above and check out their project websites.
2. It's useful to know what other web frameworks exist besides Django and Flask. However, when you're just starting to learn to program there are significantly more tutorials and resources for Django and Flask on the web. My recommendation is to start with one of those two frameworks then expand your knowledge from there.

Web Design

Web design is the creation of a web application's style and user interaction using CSS and JavaScript.

Why is web design important?

You wouldn’t use a web application that looked like the following screenshot, would you?



Creating web pages with their own style and interactivity so users can easily accomplish their tasks is a major part of building modern web applications.

Responsive design

Separating the content from the rules for how to display the content allows devices to render the output differently based on factors such as screen size and device type. Displaying content differently based on varying screen attributes is often called *responsive design*. The responsiveness is accomplished by implementing media queries in the CSS.

For example, a mobile device does not have as much space to display a navigation bar

on the side of a page so it is often pushed down below the main content. The Bootstrap Blog example shows that navigation bar relocation scenario when you resize the browser width.

Design resources

- Web Design Repo is a one stop shop for links to blogs, podcasts, inspiration, tutorials and tools related to web design.
- Frontend Guidelines is an amazing write up of good practices for HTML, CSS and JS.
- How I Work with Color is a fantastic article from a professional designer on how he thinks about color and uses it for certain effects in his designs.
- The Bootstrapping Design book is one of the clearest and concise resources for learning design that I've ever read. Highly recommended especially if you feel you have no design skills but need to learn them.
- Learn Design Principles is a well thought out clear explanation for how to think about design according to specific rules such as axis, symmetry, hierarchy and rhythm.
- Kuler is a complementary color picker by Adobe that helps choose colors for your designs.
- If you want to learn more about how browsers work behind the scenes, here's a blog post series on building a browser engine that will show you how to build a simple rendering engine.

Cascading Style Sheets (CSS)

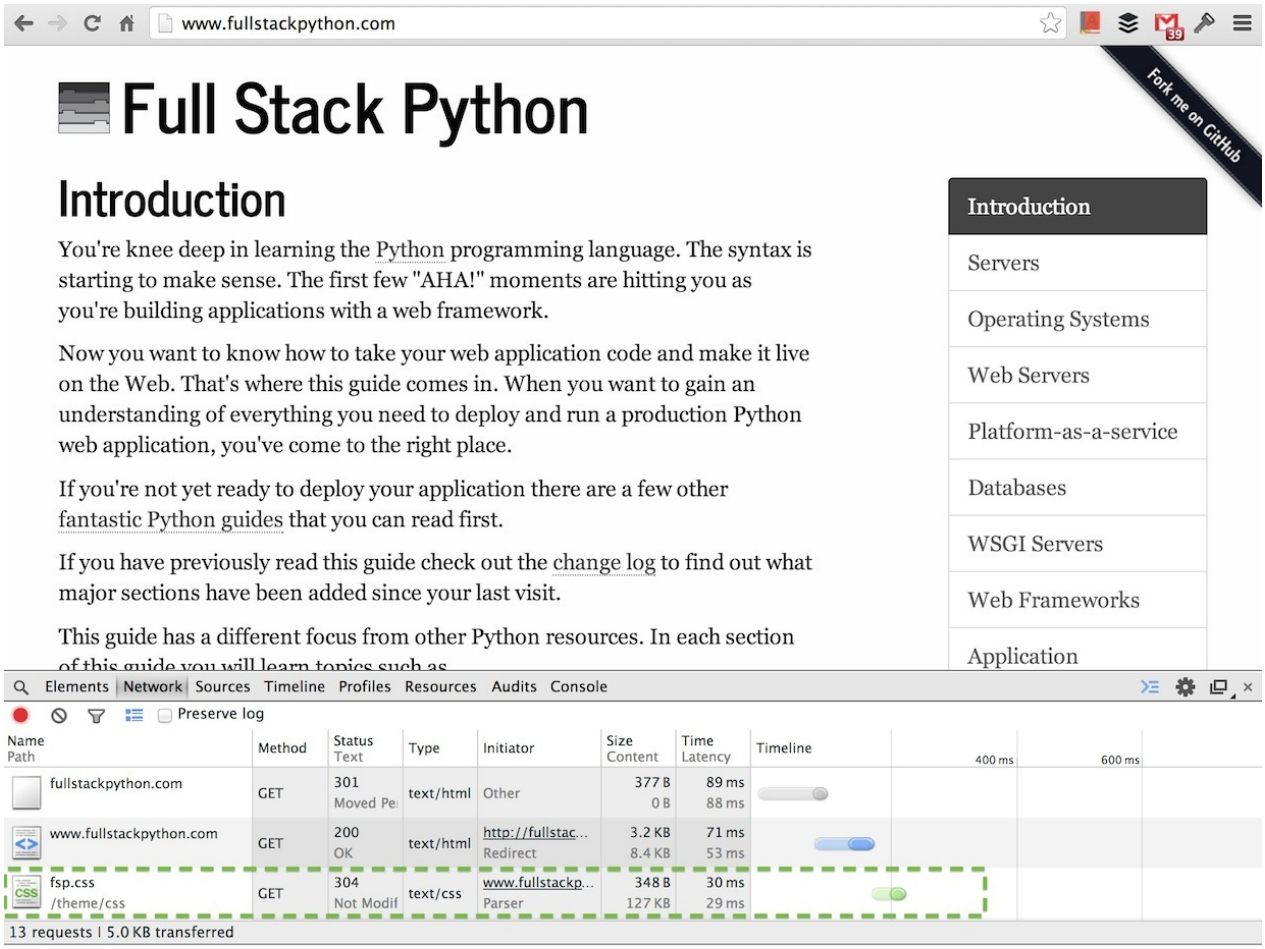
Cascading Style Sheet (CSS) files contain rules for how to display and lay out the HTML content when it is rendered by a web browser.

Why is CSS necessary?

CSS separates the content contained in HTML files from how the content should be displayed. It is important to separate the content from the rules for how it should be rendered primarily because it is easier to reuse those rules across many pages. CSS files are also much easier to maintain on large projects than styles embedded within the HTML files.

How is CSS retrieved from a web server?

The HTML file sent by the web server contains a reference to the CSS file(s) needed to render the content. The web browser requests the CSS file after the HTML file as shown below in a screenshot captured of the Chrome Web Developer Tools network traffic.



That request for the fsp.css file is made because the HTML file for Full Stack Python contains a reference to `theme/css/fsp.css` which is shown in the view source screenshot below.

CSS preprocessors

A CSS preprocessor compiles a processed language into plain CSS code. CSS preprocessing languages add syntax such as variables, mixins and functions to reduce code duplication. The additional syntax also makes it possible for designers to use these

basic programming constructs to write maintainable front end code.

- Sass is currently the favored preprocessor in the design community. Sass is considered the most powerful CSS preprocessor in terms of advanced language features.
- LESS is right up there with Sass and has an ace up its sleeve in that the Bootstrap Framework is written in LESS which brings up its popularity.
- Stylus is often cited as the third most popular CSS preprocessing language.

CSS preprocessor resources

- The Advanced Guide to HTML and CSS book has a well-written chapter on preprocessors.
- Sass vs LESS provides a short answer on which framework to use then a longer more detailed response for those interested in understanding the details.
- How to choose the right CSS preprocessor has a comparison of Sass, LESS and Stylus.
- Musings on CSS preprocessors contains helpful advice ranging from how to work with preprocessors in a team environment to what apps you can use to aid your workflow.

CSS frameworks

CSS frameworks provide structure and a boilerplate base for building a web application's design.

- Bootstrap
- Foundation
- Gumby
- Compass

- Profound Grid
- Skeleton
- HTML5 Boilerplate

CSS resources

- Frontend Development Bookmarks is one of the largest collections of valuable resources for frontend learning both in CSS as well as JavaScript.
- CSS refresher notes is incredibly helpful if you've learned CSS in bits and pieces along the way and you now want to fill in the gaps in your knowledge.
- Mozilla Developer Network's CSS page contains an extensive set of resources, tutorials and demos for learning CSS.
- CSS Positioning 101 is a detailed guide for learning how to do element positioning correctly with CSS.
- CSS3 cheat sheet
- Learn CSS layout is a simple guide that breaks CSS layout topics into chapters so you can learn each part one at a time.
- Google's Web Fundamentals class shows how to create responsive designs and performant websites.
- Tailoring CSS for performance is an interesting read since many developers do not consider the implications of CSS complexity in browser rendering time.
- Can I Use... is a compatibility table that shows which versions of browsers implement specific CSS features.

CSS learning checklist

1. Create a simple HTML file with basic elements in it. Use the `python -m SimpleHTTPServer` command to serve it up. Create a `<style></style>` element within

the `<head>` section in the HTML markup. Play with CSS within that style element to change the look and feel of the page.

2. Check out front end frameworks such as Bootstrap and Foundation and integrate one of those into the HTML page.

3. Work through the framework's grid system, styling options and customization so you get comfortable with how to use the framework.

4. Apply the framework to your web application and tweak the design until you have something that looks much better than generic HTML.

JavaScript

JavaScript is a small scripting programming language embedded in web browsers to enable dynamic content and interaction.

Why is JavaScript necessary?

JavaScript executes in the client and enables dynamic content and interaction that is not possible with HTML and CSS alone. Every modern Python web application uses JavaScript on the front end.

Front end frameworks

Front end JavaScript frameworks move the rendering for most of a web application to the client side. Often these applications are informally referred to as "one page apps" because the webpage is not reloaded upon every click to a new URL. Instead, partial HTML pages are loaded into the document object model or data is retrieved through an API call then displayed on the existing page.

Examples of these front end frameworks include:

- Angular.js
- Backbone.js
- Ember.js

Front end frameworks are rapidly evolving. Over the next several years consensus about good practices for using the frameworks will emerge.

How did JavaScript originate?

JavaScript is an implementation of the ECMAScript specification which is defined by the Ecma International Standards Body.

JavaScript resources

- Frontend tooling in 2015 shows the results of a survey for what frontend developers are using for CSS pre- and post-processing and other build steps.
- How Browsers Work is a great overview of both JavaScript and CSS as well as how pages are rendered in a browser.
- A re-introduction to JavaScript by Mozilla walks through the basic syntax and operators.
- Coding tools and JavaScript libraries is a huge list by Smashing Magazine with explanations for each tool and library for working with JavaScript.
- Superhero.js is an incredibly well designed list of resources for how to test, organize, understand and generally work with JavaScript.
- Unheap is an amazing collection of reusable JQuery plugins for everything from navigation to displaying media.
- The State of JavaScript in 2015 is an opinion piece about favoring small, single-purpose JavaScript libraries over larger frameworks due to churn in the ecosystem.
- The Modern JavaScript Developer's Toolbox provides a high-level overview of tools frequently used on the client and server side for developers using JavaScript in their web applications.

JavaScript learning checklist

1. Create a simple HTML file with basic elements in it. Use the `python -m SimpleHTTPServer` command to serve it up. Create a `<script type="text/javascript"></script>` element at the end of the `<body>` section in the HTML page. Play with JavaScript within that element to learn the basic syntax.
2. Download JQuery and add it to the page above your JavaScript element. Start working with JQuery and learning how it makes basic JavaScript easier.
3. Work with JavaScript on the page. Incorporate examples from open source projects

listed below as well as JQuery plugins. Check out Unheap to find a large collection of categorized JQuery plugins.

4. Check out the JavaScript resources below to learn more about advanced concepts and open source libraries.

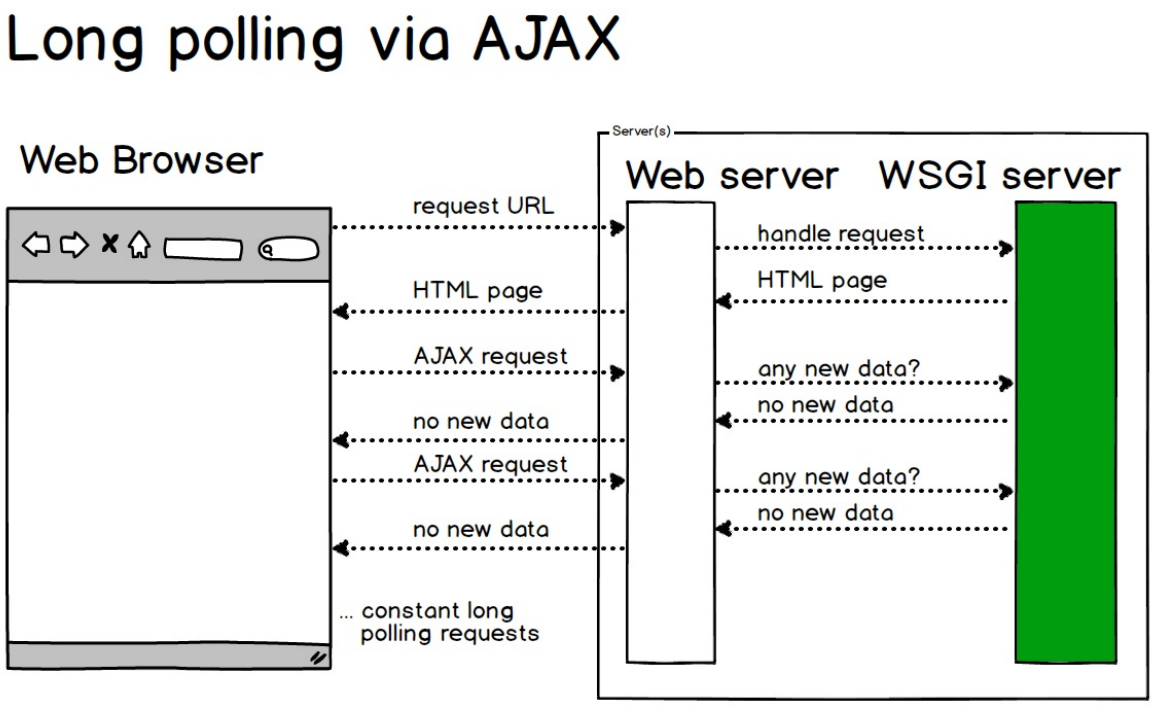
5. Integrate JavaScript into your web application and check the static content section for how to host the JavaScript files.

WebSockets

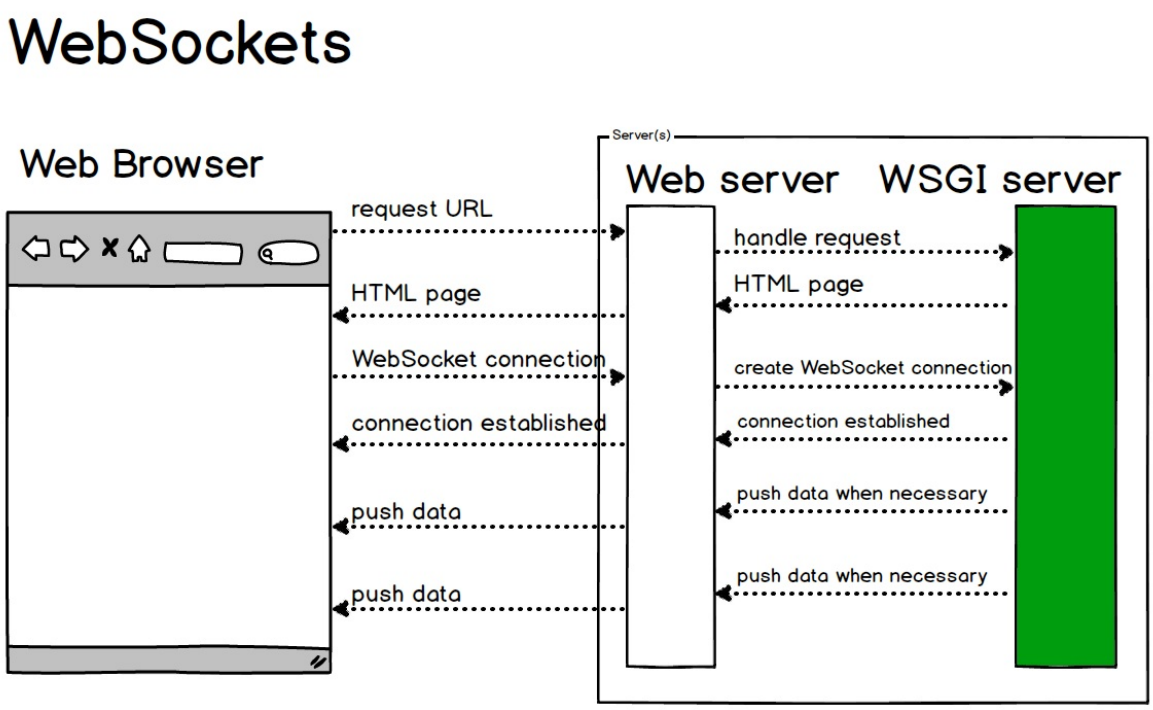
A WebSocket is a standard protocol for two-way data transfer between a client and server. The WebSockets protocol does not run over HTTP, instead it is a separate implementation on top of TCP.

Why use WebSockets?

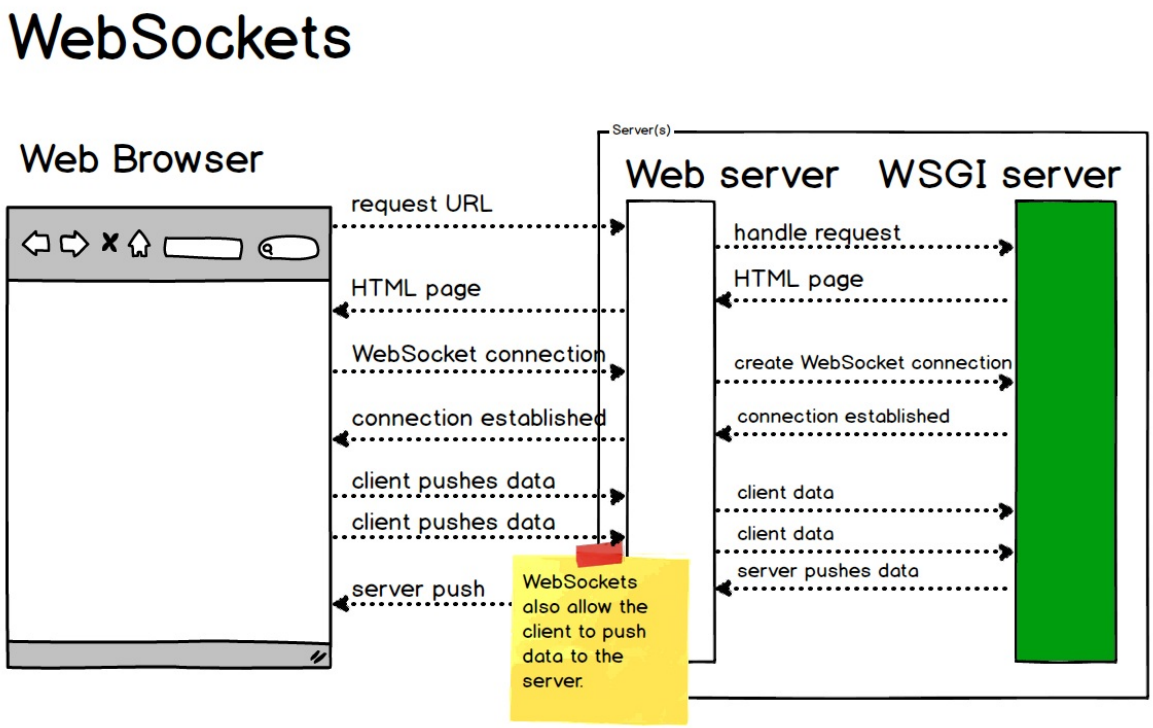
A WebSocket connection allows full-duplex communication between a client and server so that either side can push data to the other through an established connection. The reason why WebSockets, along with the related technologies of Server-sent Events (SSE) and WebRTC data channels, are important is that HTTP is not meant for keeping open a connection for the server to frequently push data to a web browser. Previously, most web applications would implement long polling via frequent Asynchronous JavaScript and XML (AJAX) requests as shown in the below diagram.



Server push is more efficient and scalable than long polling because the web browser does not have to constantly ask for updates through a stream of AJAX requests.



While the above diagram shows a server pushing data to the client, WebSockets is a full-duplex connection so the client can also push data to the server as shown in the diagram below.



The WebSockets approach for server- and client-pushed updates works well for certain categories of web applications such as chat room, which is why that's often an example

application for a WebSocket library.

Implementing WebSockets

Both the web browser and the server must implement the WebSockets protocol to establish and maintain the connection. There are important implications for servers since WebSockets connections are long lived, unlike typical HTTP connections.

A multi-threaded or multi-process based server cannot scale appropriately for WebSockets because it is designed to open a connection, handle a request as quickly as possible and then close the connection. An asynchronous server such as Tornado or Green Unicorn monkey patched with gevent is necessary for any practical WebSockets server-side implementation.

On the client side, it is not necessary to use a JavaScript library for WebSockets. Web browsers that implement WebSockets will expose all necessary client-side functionality through the WebSockets object.

However, a JavaScript wrapper library can make a developer's life easier by implementing graceful degradation (often falling back to long-polling when WebSockets are not supported) and by providing a wrapper around browser-specific WebSocket quirks. Examples of JavaScript client libraries and Python implementations are found below.

JavaScript client libraries

- Socket.io's client side JavaScript library can be used to connect to a server side WebSockets implementation.
- web-socket-js is a Flash-based client-side WebSockets implementation.

Python implementations

- Autobahn uses Twisted or asyncio to implement the WebSockets protocol.
- Crossbar.io builds upon Autobahn and includes a separate server for handling the

WebSockets connections if desired by the web app developer.

Nginx WebSocket proxying

Nginx officially supports WebSocket proxying as of version 1.3. However, you have to configure the Upgrade and Connection headers to ensure requests are passed through Nginx to your WSGI server. It can be tricky to set this up the first time.

Here are the configuration settings I use in my Nginx file as part of my WebSockets proxy.

```
# this is where my WSGI server sits answering only on localhost
# usually this is Gunicorn monkey patched with gevent
upstream app_server_wsgiapp {
  server localhost:5000 fail_timeout=0;
}

server {

  # typical web server configuration goes here

  # this section is specific to the WebSockets proxying
  location /socket.io {
    proxy_pass http://app_server_wsgiapp/socket.io;
    proxy_redirect off;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_read_timeout 600;
  }
}
```

Note if you run into any issues with the above example configuration you'll want to scope out the official HTTP proxy module documentation.

The following resources are also helpful for setting up the configuration properly.

- Nginx has an official page for WebSocket proxying.

- WebSockets in Nginx walks through the Nginx WebSockets configuration directives.
- Proxying WebSockets with Nginx shows how to proxy with Socket.io.

Open source Python examples with WebSockets

- The python-websockets-example contains code to create a simple web application that provides WebSockets using Flask, Flask-SocketIO and gevent.
- The Flask-SocketIO project has a chat web application that demos sending server generated events as well as input from users via a text box input on a form.

General WebSockets resources

- The official W3C candidate draft for WebSockets API and the working draft for WebSockets are good reference material but can be tough for those new to the WebSockets concepts. I recommend reading the working draft after looking through some of the more beginner-friendly resources list below.
- WebSockets 101 by Armin Ronacher provides a detailed assessment of the subpar state of HTTP proxying in regards to WebSockets. He also discusses the complexities of the WebSockets protocol including the packet implementation.
- The "Can I Use?" website has a handy WebSockets reference chart for which web browsers and specific versions support WebSockets.
- Mozilla's Developer Resources for WebSockets is a good place to find documentation and tools for developing with WebSockets.
- WebSockets from Scratch gives a nice overview of the protocol then shows how the lower-level pieces work with WebSockets, which are often a black box to developers who only use libraries like Socket.IO.
- websocketd is a WebSockets server aiming to be the "CGI of WebSockets". Worth a look.

Python-specific WebSockets resources

- The "Async Python Web Apps with WebSockets & gevent" talk I gave at San Francisco Python in January 2015 is a live-coded example Flask web app implementation that allows the audience to interact with WebSockets as I built out the application.

- Real-time in Python provides Python-specific context for how the server push updates were implemented in the past and how Python's tools have evolved to perform server side updates.

- websockets is a WebSockets implementation for Python 3.3+ written with the asyncio module (or with Tulip if you're working with Python 3.3).

- The Choose Your Own Adventure Presentations tutorial uses WebSockets via gevent on the server and socketio.js for pushing vote count updates from the server to the client.

- Adding Real Time to Django Applications shows how to use Django and Crossbar.io to implement a publish/subscribe feature in the application.

- Async with Bottle shows how to use greenlets to support WebSockets with the Bottle web framework.

- If you're deploying to Heroku, there is a specific WebSockets guide for getting your Python application up and running.

- The Reddit thread for this page has some interesting comments on what's missing from the above content that I'm working to address.

- Creating Websockets Chat with Python shows code for a Twisted server that handles WebSockets connections on the server side along with the JavaScript code for the client side.

- Synchronize clients of a Flask application with WebSockets is a quick tutorial showing how to use Flask, the Flask-SocketIO extension and Socket.IO to update values between web browser clients when changes occur.

Template Engines

Template engines process template files, which provide an intermediate format between your Python code and a desired output format, such as HTML or PDF.

Why are template engines important?

Template engines allow developers to generate a desired content type, such as HTML, while using some of the data and programming constructs such as conditionals and for loops to manipulate the output. Template files that are created by developers and then processed by the template engine consist of prewritten markup and template tag blocks where data is inserted.

For example, look at the first ten source lines of HTML of this webpage:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="author" content="Matt Makai">
    <meta name="description" content="Template engines provide programmatic output of
    <link rel="shortcut icon" href="//static.fullstackpython.com/fsp-fav.png">
```

Every one of the HTML lines above is standard for each page on Full Stack Python, with the exception of the `<meta name="description"...` line which provides a unique short description of what the individual page contains.

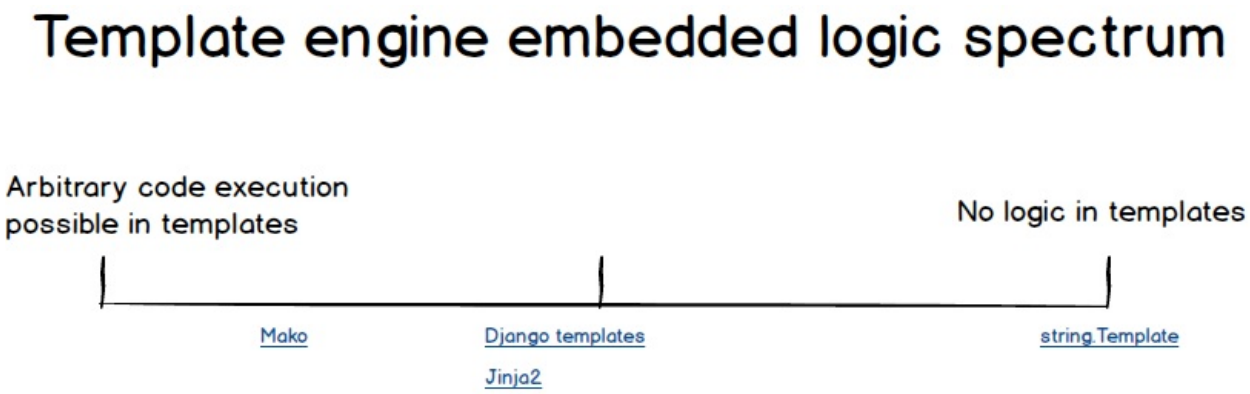
The base.html Jinja template used to generate Full Stack Python allows every page on the site to have consistent HTML but dynamically generate the pieces that need to change between pages when the static site generator executes. The below code from the `base.html` template shows that the meta description is up to child templates to create.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="author" content="Matt Makai">
    {% block meta_header %}{% endblock %}
    <link rel="shortcut icon" href="//static.fullstackpython.com/fsp-fav.png">
```

In a typical WSGI application, the template engine would generate the HTML output response when an HTTP request comes in for a particular URL.

Python template engines

There are several popular Python template engines. A template engine implementation will fall somewhere on the spectrum between allowing arbitrary code execution and granting only a limited set of capabilities via template tags. A rough visual of the code in template spectrum can be seen below for four of the major Python template engines.



Jinja

Jinja, also known as "Jinja2", is a popular Python template engine written as an independent open source project, unlike some template engines that are provided as part of a larger web framework.

Major Python open source applications such as the configuration management tools Ansible and SaltStack as well as the static site generator Pelican use the Jinja template engine by default for generating output files.

There's a whole lot more to learn about Jinja on the Jinja2 page.

Django templating

Django comes with its own template engine in addition to supporting (as of Django 1.9) drop-in replacement with other template engines such as Jinja.

Mako

Mako is the default templating engine for the Pyramid web framework and has wide support as a replacement template engine for many other web frameworks.

Template engine resources

- A template engine in 500 lines or less is an article by Ned Batchelder provides a template engine in 252 lines of Python that can be used to understand how template engines work under the cover.
- A Primer on Jinja Templating shows how to use the major parts of this fantastic template engine.
- Template fragment gotchas is a collection of situations that can trip up a developer or designer when working with templates.
- This template engines site contains a range of information from what templates engines are to listing more esoteric Python template engines.

Web Application Security

Website security must be thought about while building every level of the web stack. However, this section includes topics that deserve particular treatment, such as cross-site scripting (XSS), SQL injection, cross-site request forgery and usage of public-private keypairs.

Security open source projects

- Bro is a network security and traffic monitor.
- quick NIX secure script for securing Linux distributions.

HTTPS resources

- How does HTTPS actually work? is a well-written overview of the protocol including certificates, signatures, signing and related topics.
- These introduction to HTTPS videos explain what HTTPS is and how to implement it.
- This question asking what is the difference between TLS and SSL? explains that TLS is a newer version of SSL and should be used because SSL through version 3.0 is insecure.
- If you have wondered what all the SSL/TLS acronyms and settings mean, read the Security/Server Side TLS guide which Mozilla uses to operationalize its servers.
- If you're having users submit sensitive information to your site you need to use SSL/TLS. Anything before TLS is now insecure. Check out this handy guide that goes over some of the nuances of the subject.
- The Sorry State of SSL details the history and evolution of SSL/TLS. There are important differences between the versions and Hynek explains why TLS should always be used. The talk prompted work to improve Python's SSL in 2.7.9 based on

the upgrades in Python 3 outlined in The not-so-sorry state of SSL in Python.

- How HTTPS Secures Connections is a guide for what HTTPS does and does not secure against.
- When and How to Deploy HTTPS
- The first few milliseconds of an HTTPS connection provides a detailed look at the SSL handshake process that is implemented by browsers based on the RFC 2818 specification.
- Qualy SSL Server Test can be used to determine what's in place and what is missing for your server's HTTPS connection. Once you run the test read this article on Getting an A+ on Qualy's SSL Labs Tester to improve your situation.

General security resources

- The Open Web Application Security Project (OWASP) has cheat sheets for security topics.
- This page contains a fantastic curated list of security reading material from beginning to advanced topics.
- The /r/netsec subreddit is one place to go to learn more about network and application security.
- Hacking Tools Repository is a great list of password cracking, scanning, sniffing and other security penetration testing tools.
- Securing an Ubuntu Server
- Securing Ubuntu
- Security Tips from Apache
- Securing a Linux Server
- The EFF has a well written overview on what makes a good security audit. It's broad but contains some of their behind the scenes thinking on important considerations

with security audits.

- Ars Technica wrote posts on securing your website along with how to set up a safe and secure web server: part 1 and part 2 to explain HTTPS and SSL without much required pre-existing knowledge.

- Crypto 101 is an introductory course on cryptography for programmers.

- An in-depth analysis of SSH attacks on Amazon EC2 shows how important it is to secure your web servers, especially when they are hosted in IP address ranges that are commonly scanned by malicious actors.

- Cloud Security Auditing: Challenges and Emerging Approaches is a high-level overview of some of security auditing problems that come with cloud deployments.

- Wondering how the common buffer overflow attack works? Check out this article on buffer overflows that explains the attack in layman's terms.

- 7 Security Measures to Protect Your Servers provides a good overview of the fundamentals for how servers should be configured for baseline security.

- As you're developing on Linux, you'll want to read and follow this Linux workstation security document to make sure your code and environment are not compromised. If you're on Mac OS X, check out this securing Yosemite guide which covers that environment.

- TLS and Nginx Web Server Hardening explains a secure server configuration for the Nginx web server.

- Timing attacks are one form of vulnerability that can be used to defeat HTTPS in certain configurations. Understanding how those attacks work is important in keeping your users' connections secure.

Web security learning checklist

1. Read and understand the major web application security flaws that are commonly exploited by malicious actors. These include cross-site request forgery (CSRF), cross-site scripting (XSS), SQL injection and session hijacking. The OWASP top 10 web application vulnerabilities list is a great place to get an overview of these topics.

2. Determine how the framework you've chosen mitigates these vulnerabilities.

3. Ensure your code implements the mitigation techniques for your framework.

4. Think like an attacker and actively work to break into your own system. If you do not have enough experience to confidently break the security consider hiring a known white hat attacker. Have her break the application's security, report the easiest vulnerabilities to exploit in your app and help implement protections against those weaknesses.

5. Recognize that no system is ever totally secure. However, the more popular an application becomes the more attractive a target it is to attackers. Reevaluate your web application security on a frequent basis.

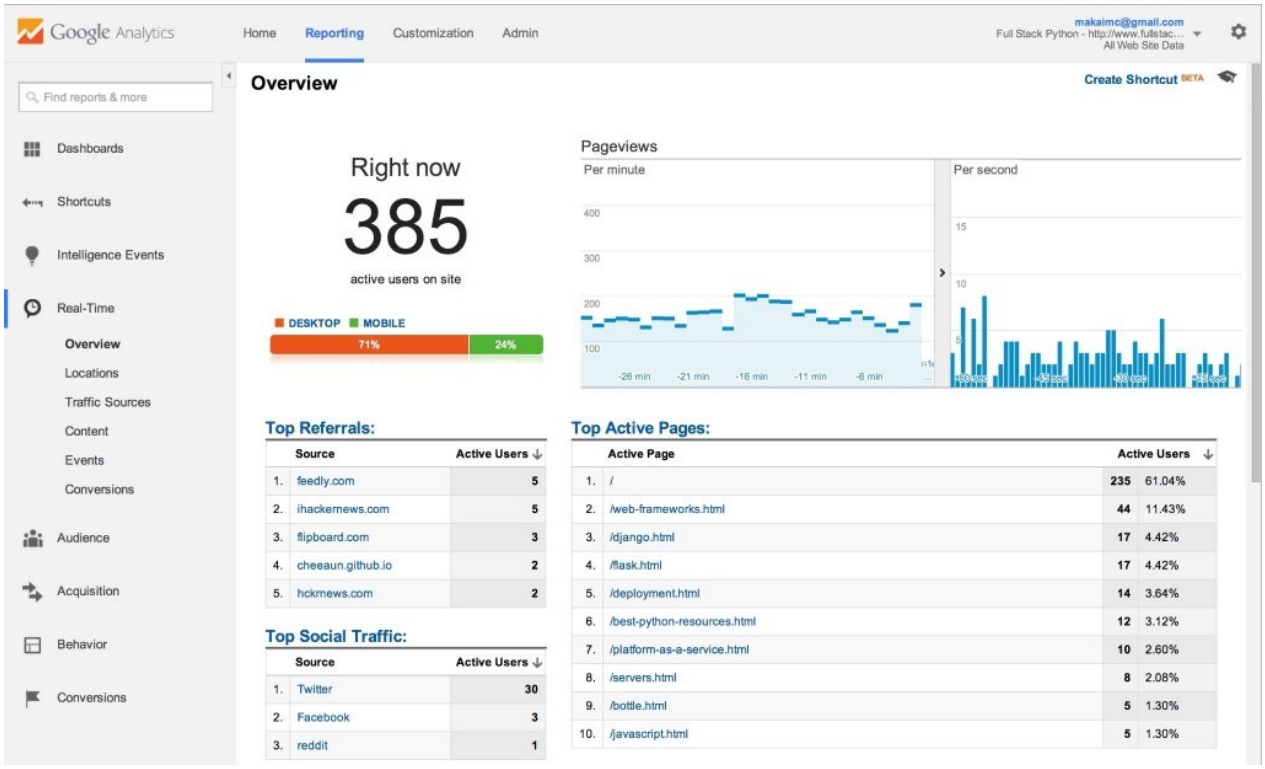
Static Site Generator

A static website generator combines a markup language, such as Markdown or reStructuredText, with a templating engine such as Jinja, to produce HTML files. The HTML files can be hosted and served by a web server or content delivery network (CDN) *without* any additional dependencies such as a WSGI server.

Why are static site generators useful?

Static content files such as HTML, CSS and JavaScript can be served from a content delivery network (CDN) for high scale and low cost. If a statically generated website is hit by high concurrent traffic it will be easily served by the CDN without dropped connections.

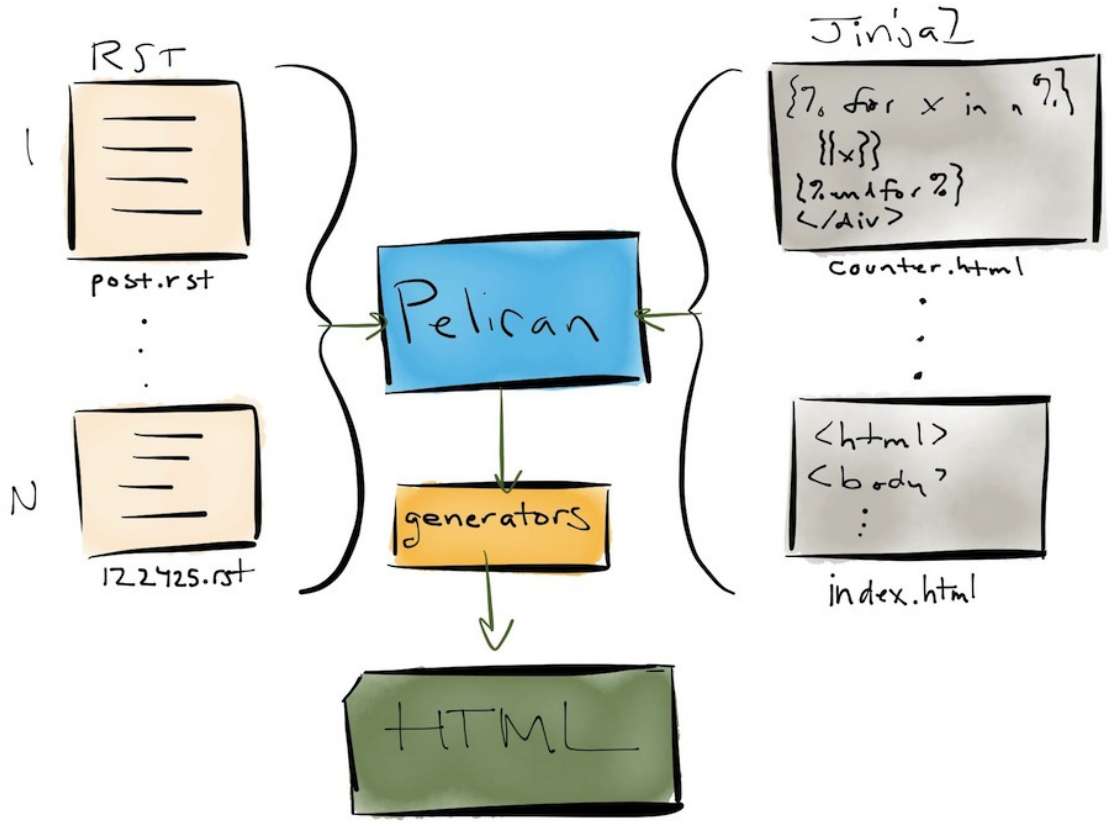
For example, when Full Stack Python was on the top of Hacker News for a weekend, GitHub Pages was used as a CDN to serve the site and didn't have any issues even with close to 400 concurrent connections at a time, as shown in the following Google Analytics screenshot captured during that traffic burst.



How do static website generators work?

Static site generators allow a user to create HTML files by writing in a markup language and coding template files. The static site generator then combines the markup language and templates to produce HTML. The output HTML does not need to be maintained by hand because it is regenerated every time the markup or templates are modified.

For example, as shown in the diagram below, the Pelican static site generator can take in reStructuredText files and Jinja2 template files as input then combine them to output a set of static HTML files.



What's the downside to using static site generators?

The major downside is that code cannot be executed after a site is created. You are stuck with the output files so if you're used to building web applications with a traditional web framework you'll have to change your expectations.

Content that is typically powered by a database, such as comments, sessions and user

data can only be handled through third party services. For example, if you want to have comments on a static website you'd need to embed Disqus's form and be completely reliant upon their service.

Many web applications simply cannot be built with only a static site generator. However, a static website generator can create part of a site that will be served up by a web server while other pages are handled by the WSGI server. If done right, those web applications have the potential to scale better than if every page is rendered by the WSGI server. The complexity may or may not be worth it for your specific application.

Python implementations

Numerous static website generators exist in many different languages. The ones listed here are primarily coded in Python.

- Pelican (source code) is a commonly used Python static website generator which is used to create Full Stack Python. The primary templating engine is Jinja and Markdown, reStructuredText and AsciiDoc are supported with the default configuration.

- MkDocs (source code) uses a YAML configuration file to take Markdown files and an optional theme to output a documentation site. The templating engine is Jinja, but a user doesn't have to create her own templates unless a custom site is desired at which point it might make more sense to use a different static site generator instead.

- Nikola (source code) takes in reStructuredText, Markdown or Jupyter (IPython) Notebooks and combines the files with Mako or Jinja2 templates to output static sites. It is compatible with both Python 2.7 and 3.3+. Python 2.7 will be dropped in early 2016 while Python 3.3+ will continue to be supported.

- Acrylamid (source code) uses incremental builds to generate static sites faster than recreating every page after each change is made to the input files.

- Hyde (source code) started out as a Python rewrite of the popular Ruby-based Jekyll static site generator. Today the project has moved past those "clone Jekyll" origins. Hyde supports Jinja as well as other templating languages and places more emphasis on metadata within the markup files to instruct the generator how to

produce the output files. Check out the Hyde-powered websites page to see live examples created with Hyde.

- Grow SDK (source code) uses projects, known as pods, which contain a specific file and directory structure so the site can be generated. The project remains in the "experimental" phase.

- Lektor (source code) is a Python static content management system that can deploy to any webserver. It uses Jinja2 as a template engine.

- Complexity (source code) is a site generator for users who like to work in HTML. It uses HTML for templating but has some functionality from Jinja for inheritance. Works with Python 2.6+, 3.3+ and PyPy.

- Cactus (source code) uses the Django templating engine that was originally built with front-end designers in mind. It works with both Python 2.x and 3.x.

Open source Python static site generator examples

- This site is all open source in its own GitHub repository under the MIT license. Fork away!

- Django REST Framework uses MkDocs to create its documentation site. Be sure to take a look at the mkdocs.yml file to see how large, well-written docs are structured for that project.

- Practicing web development uses Acrylamid to create its site. The code is open source on GitHub.

- Linux Open Admin Days (Loadsys) has their site open source and available for viewing.

- The Pythonic Perambulations blog has a fairly standard theme but is also open source on GitHub.

Static site generator resources

- The Long Road to Building a Static Blog with Pelican is a fantastic read that really gets into the details throughout the walkthrough.

- Staticgen lists static website generators of all programming languages sorted by various attributes such as the number of GitHub stars, forks and issues.

- Static site hosting with S3 and Cloudflare shows how to set up an S3 bucket with Cloudflare in front as a CDN that serves the content with HTTPS. You should be able to accomplish roughly the same situation with Amazon Cloudfront, but as a Cloudflare user I like their service for these static site configurations.

- Getting Started with Pelican and GitHub Pages is a tutorial I wrote for getting up and running with Full Stack Python's source code, which uses Pelican to generate the site.

- The title is a big grandiose, but there's some solid detail in this article on why static website generators are the next big thing. I'd argue static website generators have been big for a long time now.

- Static site generators can be used for a range of websites from side projects up to big sites. This blog post by WeWork on why they use a static site generator explains it from the perspective of a large business.

- Getting started with Pelican and GitHub pages is a tutorial I wrote to use the Full Stack Python source code to create and deploy your first static site.

- Ditching Wordpress and becoming one of the cool kids is one developer's experience moving away from Wordpress and onto Pelican with reStructuredText for his personal blog.

Jinja2

Jinja2 is a Python template engine used to create HTML, XML or other markup formats that are returned to the user via an HTTP response.

Why is Jinja2 useful?

Jinja2 is useful because it has consistent template tag syntax and the project is cleanly extracted as an independent open source project so it can be used a dependency by other code libraries.

Jinja2 strikes a thoughtful balance on the template engine spectrum where on one end you can embed arbitrary code in the templates and the other end a developer can code whatever she wants.

Jinja2 origin and development

The first recorded public released of Jinja2 was in 2008 with 2.0rc1. Since then the engine has seen numerous updates and remains in active development.

Jinja2 engine certainly wasn't the first template engine. In fact, Jinja2's syntax is inspired by Django's built-in template engine, which was released several years earlier. There were many template systems, such as JavaServer Pages (JSPs), that originated almost a decade before Jinja2. Jinja2 built upon the concepts of other template engines and today is widely used by the Python community.

What projects depend on Jinja2?

Jinja2 is a commonly-used templating engine for web frameworks such as Flask, Bottle Morepath and, as of its 1.8 update, optionally Django as well. Jinja2 is also used as a template language by configuration management tool Ansible and the static site generator Pelican, among many other similar tools.

The idea is that if a developer already knows Jinja2 from working with one project then

the exact same syntax and style can be used in another project that requires templating. The re-use reduces the learning curve and saves the open source project author from having to reinvent a new templating style.

Jinja2 resources

- Real Python has a nice Jinja2 primer with many code examples to show how to use the template engine.
- The second part of the Flask mega tutorial is all about Jinja2 templates. It walks through control flow, template inheritance and other standard features of the engine.
- Upgrading to Jinja2 Templates in Django 1.8 With Admin shows how to fix an issue that can occur with Django 1.8 and using Jinja2 as the template engine.
- The official Jinja2 template designer documentation is exceptionally useful both as a reference as well as a full read-through to understand how to properly work with template tags.
- When you want to use Jinja2 outside of a web framework or other existing tool, here's a handy quick load function snippet so the template engine can be easily used from a script or the REPL.
- When working with Jinja2 in combination with LaTeX, some of Jinja2's blocks can conflict with LaTeX commands. Check out this post on LaTeX templates with Python and Jinja2 to generate PDFs to resolve those issues.
- When you use Jinja2 for long enough, eventually you'll want to escape large blocks of Jinja2-like text in your templates. To do so, you'll need the "raw" template tag.

Data

Data

Data is an incredibly broad topic but it can be broken down into many subsections, including (in no particular order):

- data processing / wrangling
- machine learning
- data analysis
- visualization
- geospatial mapping
- persistence via relational databases and NoSQL data stores
- object-relational mappers
- natural language processing (NLP)
- indexing, search and retrieval

The Python community has built and continues to create open source libraries and tutorials for all of the above topics.

Why is Python a great language choice for data tasks?

Python has a wide array of open source code libraries available and a diverse community of people with different backgrounds who contribute to make those libraries better each day.

In addition, Python data manipulation code can be combined with web frameworks and web APIs to build software that would be difficult to create with a single other language. For example, Ruby is a fantastic language for building web applications but its data analysis and visualization libraries are very limited compared to what is currently available in the Python ecosystem.

How did Python become so widely used for working with data?

Python is a general purpose programming language and can be applied to many problem areas. Over the past couple of decades, Python has become increasingly popular in the scientific and financial communities. Projects such as pandas grew out of a hedge-fund while NumPy and SciPy were created in academic environments then improved by the broader open source community.

The question is: why Python was used to created these projects? The answer is a mix of luck, the growth of the open source community as Python was maturing and wide adoption by people not formally trained as computer scientists. The pragmatic syntax and explicit style helped very intelligent people without programming backgrounds to pick up the language and get their work done with less fuss than other programming languages. Over time the code used in the financial world and scientific community was shared at the same time global open source communities were developing, further spreading their usage among a broader base of software developers.

There's no doubt some of the momentum behind Python's wide adoption for all types of data manipulation was that it happened to be the right language in the right place at the right time. Nevertheless, it was ultimately the hard work of a massive number of engineers and scientists around the world who created the incredible mix of data code libraries available today.

General Python data resources

- PyData is a community for developer and users of Python data tools. They put on fantastic conferences around the and fund the continued development of open source data-related libraries.

- Continuum Analytics is one of the leading Python companies that pours a tremendous amount of time and funding into the data community.

- A crash course in Python for scientists provides an overview of the Python language with iPython Notebook for those in scientific fields.

- The videos of Travis Oliphant on Python's Role in Big Data Analytics: Past, Present, and Future and Building the PyData Community give historical perspective on how the Python data tools have evolved over the past 20ish years based on his first-hand experience as a leader and member in that community.

Databases

A database is an abstraction on top of an operating system's file system to ease creating, reading, updating, and deleting persistent data.

Why are databases necessary?

At a high level web applications store data and present it to users in a useful way. For example, Google stores data about roads and provides directions to get from one location to another by driving through the Maps application. Driving directions are possible because the data is stored in a structured format.

Databases make structured storage reliable and fast. They also give you a mental framework for how the data should be saved and retrieved instead of having to figure out what to do with the data every time you build a new application.

Relational databases

The database storage abstraction most commonly used in Python web development is sets of relational tables. Alternative storage abstractions are explained on the NoSQL page.

Relational databases store data in a series of tables. Interconnections between the tables are specified as *foreign keys*. A foreign key is a unique reference from one row in a relational table to another row in a table, which can be the same table but is most commonly a different table.

Databases storage implementations vary in complexity. SQLite, a database included with Python, creates a single file for all data per database. Other databases such as PostgreSQL, MySQL, Oracle and Microsoft SQL Server have more complicated persistence schemes while offering additional advanced features that are useful for web application data storage. These advanced features include but are not limited to:

1. data replication between a master database and one or more read-only slave instances

2. advanced column types that can efficiently store semi-structured data such as JavaScript Object Notation (JSON)
3. sharding, which allows horizontal scaling of multiple databases that each serve as read-write instances at the cost of latency in data consistency
4. monitoring, statistics and other useful runtime information for database schemas and tables

Typically web applications start with a single database instance such as PostgreSQL with a straightforward schema. Over time the database schema evolves to a more complex structure using schema migrations and advanced features such as replication, sharding and monitoring become more useful as database utilization increases based on the application users' needs.

Most common databases for Python web apps

PostgreSQL and MySQL are two of the most common open source databases for storing Python web applications' data.

SQLite is a database that is stored in a single file on disk. SQLite is built into Python but is only built for access by a single connection at a time. Therefore is highly recommended to not run a production web application with SQLite.

PostgreSQL

PostgreSQL is the recommended relational database for working with Python web applications. PostgreSQL's feature set, active development and stability contribute to its usage as the backend for millions of applications live on the Web today.

Learn more about using PostgreSQL with Python on the PostgreSQL page.

MySQL

MySQL is another viable open source database implementation for Python applications. MySQL has a slightly easier initial learning curve than PostgreSQL but is not as feature rich.

Find out about Python applications with a MySQL backed on the dedicated MySQL page.

Connecting to a database with Python

To work with a relational database using Python, you need to use a code library. The most common libraries for relational databases are:

- psycopg2 for PostgreSQL
- MySQLdb for MySQL
- cx_Oracle for Oracle

SQLite support is built into Python 2.7+ and therefore a separate library is not necessary. Simply "import sqlite3" to begin interfacing with the single file-based database.

Object-relational Mapping

Object-relational mappers (ORMs) allow developers to access data from a backend by writing Python code instead of SQL queries. Each web application framework handles integrating ORMs differently. There's an entire page on object-relational mapping (ORMs) that you should read to get a handle on this subject.

Database third-party services

Numerous companies run scalable database servers as a hosted service. Hosted databases can often provide automated backups and recovery, tightened security configurations and easy vertical scaling, depending on the provider.

- Amazon Relational Database Service (RDS) provides pre-configured MySQL and PostgreSQL instances. The instances can be scaled to larger or smaller configurations based on storage and performance needs.
- Google Cloud SQL is a service with managed, backed up, replicated, and auto-patched MySQL instances. Cloud SQL integrates with Google App Engine but can

be used independently as well.

- BitCan provides both MySQL and MongoDB hosted databases with extensive backup services.

- ElephantSQL is a software-as-a-service company that hosts PostgreSQL databases and handles the server configuration, backups and data connections on top of Amazon Web Services instances.

General database resources

- How does a relational database work? is a detailed longform post on the sorting, searching, merging and other operations we often take for granted when using an established relational database such as PostgreSQL.

- Why I Love Databases is a great read on the CAP Theorem, distributed systems and other topics that are at the core of database theory and implementation. Well worth the time to read.

- Writing better SQL is a short code styling guide to make your queries easier to read.

- DB-Engines ranks the most popular database management systems.

- DB Weekly is a weekly roundup of general database articles and resources.

- Databases integration testing strategies covers a difficult topic that comes up on every real world project.

- Asynchronous Python and Databases is an in-depth article covering why many Python database drivers cannot be used without modification due to the differences in blocking versus asychronous event models. Definitely worth a read if you are using WebSockets via Tornado or gevent.

- PostgreSQL vs. MS SQL Server is one perspective on the differences between the two database servers from a data analyst.

Databases learning checklist

1. Install PostgreSQL on your server. Assuming you went with Ubuntu run `sudo apt-get install postgresql`.

2. Make sure the psycopg2 library is in your application's dependencies.

3. Configure your web application to connect to the PostgreSQL instance.

4. Create models in your ORM, either with Django's built-in ORM or SQLAlchemy with Flask.

5. Build your database tables or sync the ORM models with the PostgreSQL instance, if you're using an ORM.

6. Start creating, reading, updating and deleting data in the database from your web application.

PostgreSQL

PostgreSQL, often written as "Postgres" and pronounced "Poss-gres", is an open source relational database implementation frequently used by Python applications as a backed for data storage and retrieval.

How does PostgreSQL fit within the Python stack?

PostgreSQL is the default database choice for many Python developers, including the Django team when testing the Django ORM. PostgreSQL is often viewed as more feature robust and stable when compared to MySQL, SQLServer and Oracle. All of those databases are reasonable choices. However, because PostgreSQL tends to be used by Python developers the drivers and example code for using the database tend to be better documented and contain fewer bugs for typical usage scenarios. If you try to use an Oracle database with Django, you'll see there is far less example code for that setup compared to PostgreSQL backend setups.

Why is PostgreSQL a good database choice?

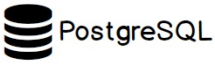
PostgreSQL's open source license allows developers to operate one or more databases without licensing cost in their applications. The open source license operating model is much less expensive compared to Oracle or other proprietary databases, especially as replication and sharding become necessary at large scale. In addition, because so many people ranging from independent developers to multinational organizations use PostgreSQL, it's often easier to find developers with PostgreSQL experience than other relational databases.

The PostgreSQL core team also releases frequent updates that greatly enhance the database's capabilities. For example, in the PostgreSQL 9.4 release the jsonb type was added to enhance JavaScript Object Notation (JSON) storage capabilities so that in many cases a separate NoSQL database is not required in an application's architecture.

Connecting to PostgreSQL with Python

To work with relational databases in Python you need to use a database driver, which is also referred to as a database connector. The most common driver library for working with PostgreSQL is psycopg2. There is a list of all drivers on the PostgreSQL wiki, including several libraries that are no longer maintained. If you're working with the asyncio Python stdlib module you should also take a look at the aiopg library which wraps psycopg2's asychronouos features together.

To abstract the connection between tables and objects, many Python developers use an object-relational mapper (ORM) with to turn relational data from PostgreSQL into objects that can be used in their Python application. For example, while PostgreSQL provides a relational database and psycopg is the common database connector, there are many ORMs that can be used with varying web frameworks, as shown in the table below.

web framework	Bottle	Flask	Flask	Django
ORM	Peewee	Pony ORM	SQLAlchemy	Django ORM
database connector	psycopg	psycopg	psycopg	psycopg
relational database				

Learn more about Python ORMs on that dedicated topic page.

PostgreSQL data safety

If you're on Linux it's easy to get PostgreSQL installed using a package manager. However, once the database is installed and running your responsibility is just beginning. Before you go live with a production application, make sure to:

1. Lock down access with a whitelist in the `pg_hba.conf` file
2. Enable replication to another database that's preferrably on different infrastructure in a separate location
3. Perform regular backups and test the restoration process
4. Ensure your application prevents SQL injection attacks

When possible have someone qualified do a PostgreSQL security audit to identify the biggest risks to your database. Small applications and bootstrapped companies often cannot afford a full audit in the beginning but as an application grows over time it becomes a bigger target.

The data stored in your database is the lifeblood of your application. If you have ever accidentally dropped a production database or been the victim of malicious activity such as SQL injection attacks, you'll know it's far easier to recover when a bit of work has been performed beforehand on backups, replication and security measures.

Python-specific PostgreSQL resources

Many quickstarts and tutorials exist specifically for Django, Flask and other web application frameworks. The ones below are some of the best walkthroughs I've read.

- This post on using PostgreSQL with Django or Flask is a great quickstart guide for either framework.
- This article explains how and why PostgreSQL can handle full text searching for many use cases. If you're going down this route, read this blog post that explains how one developer implemented PostgreSQL full text search with SQLAlchemy.
- django-postgres-copy is a tool for bulk loading data into a PostgreSQL database based on Django models. Say hello to our new open-source software for loading bulk data into PostgreSQL is an introduction to using the tool in your own projects.
- How to speed up tests in Django and PostgreSQL explains some hacks for making your schema migration-backed run quicker.
- Full Text Search in Django using Database Back-Ends provides code for both PostgreSQL and MySQL for adding simple full text search into your application.
- Records is a wrapper around the psycopg2 driver that allows easy access to direct SQL access. It's worth a look if you prefer writing SQL over using an ORM like SQLAlchemy.

General PostgreSQL resources

PostgreSQL tutorials not specific to Python are also really helpful for properly handling your data.

- PostgreSQL: The Nice Bits is a good overview slideshow of why PostgreSQL is a great relational database.
- PostgreSQL Weekly is a weekly newsletter of PostgreSQL content from around the web.
- Braintree wrote about their experiences scaling PostgreSQL. The post is an inside

look at the evolution of Braintree's usage of the database.

- This post estimates the costs of a PostgreSQL connection.
- There is no such thing as total security but this IBM article covers hardening a PostgreSQL database.
- Craig Kerstiens wrote a detailed post about understanding PostgreSQL performance.
- Handling growth with Postgres provides 5 specific tips from Instagram's engineering team on how to scale the design of your PostgreSQL database.
- Inserting And Using A New Record In Postgres shows some SQL equivalents to what many developers just do in their ORM of choice.
- Following a Select Statement Through Postgres Internals provides a fascinating look into the internal workings of PostgreSQL during a query.
- If you're just getting started with PostgreSQL here are 10 beginner tasks you should know how to execute.
- The title's a bit presumptuous but here's a useful list of 7 PostgreSQL data migration hacks you should be using, but aren't.
- awesome-postgres is a list of code libraries, tutorials and newsletters focused specifically on PostgreSQL.
- This guide to PostgreSQL monitoring is handy for knowing what to measure and how to do it.
- While you can use a graphical interface for working with PostgreSQL, it's best to spend some time getting comfortable with the command-line interface.
- Backing up databases is important because data loss can and does happen. This article explains how to back up a PostgreSQL database hosted on an Amazon Web Services EC2 instance if managing your own database on a cloud server is your preferred setup.
- How to fix undead PostgreSQL queries shows a bit of a hack for when what to do

when you can't kill certain PostgreSQL queries.

- Is bi-directional replication (BDR) in PostgreSQL transactional? explores a relatively obscure topic with the final result that BDR is similar to data stores with eventual consistency rather than consistency as a requirement.

- PostgreSQL-metrics is a tool built by Spotify's engineers that extracts and outputs metrics from an existing PostgreSQL database. There's also a way to extend the tools to pull custom metrics as well.

- This article on performance tuning PostgreSQL covers how to find slow queries, tune indexes and modify your queries to run faster.

- Creating a Document-Store Hybrid in Postgres 9.5 explains how to store and query JSON data, similar to how NoSQL data stores operate.

- PostgreSQL Indexes: First Principles is a detailed look at what indexes are, what they are good for and how to use them in PostgreSQL.

MySQL

MySQL is an open source relational database implementation for storing and retrieving data.

MySQL or PostgreSQL?

MySQL is a viable open source database implementation for Python web applications. MySQL has a slightly easier initial learning curve than PostgreSQL. However, PostgreSQL's design is often preferred by Python web developers, especially when data migrations are run as an application evolves.

What organizations use MySQL?

The database is deployed in production at some of the highest trafficked sites such as Twitter, Facebook and many others major organizations. However, since MySQL AB, the company that developed MySQL, was purchased by Sun Microsystems (which was in turn purchased by Oracle), there have been major defections away from the database by Wikipedia and Google. MySQL remains a viable database option but I always recommend new Python developers learn PostgreSQL if they do not already know MySQL.

MySQL resources

- 28 Beginner's Tutorials for Learning about MySQL Databases is a curated collection on various introductory MySQL topics.
- This tutorial shows how to install MySQL on Ubuntu.
- Graph Data From MySQL Database in Python is an interesting study with code of how to pull data out of MySQL and graph the data with Plotly.
- Pinterest open sourced many of their MySQL tools to manage instances of the database.
- Bye Bye MySQL & MongoDB, Guten Tag PostgreSQL goes into details for why the company Userlike migrated from their MySQL database setup to PostgreSQL.
- Terrible Choices: MySQL is a blog post about specific deficiencies in MySQL's implementation that hinder its usage with Django's ORM.

SQLite

SQLite is an open source relational database included with the Python standard library as of Python 2.5. The pysqlite database driver is also included with the standard library so that no further external dependencies are required to access a SQLite database from within Python applications.

SQLite resources

- sqlite3 - embedded relational database is an extensive tutorial showing many of the common create, read, update and delete operations a developer would want to do with SQLite.
- A simple step-by-step SQLite tutorial walks through creating databases as well as inserting, updating, querying and deleting data.
- My list of SQLite resources is a nice roundup of useful tools to use with SQLite and tutorials for learning more about the database.
- Introduction to SQLite3 with Python's Flask shows how to write SQL within a Flask application to work with a SQLite backend.
- Extending SQLite with Python uses the Peewee object-relational mapper (ORM) to implement virtual tables and aggregates on top of SQLite.

- Using SQLite with Flask explains how Flask code can directly query a SQLite database without an ORM.
- SQLite Browser is an open source graphical user interface for working with SQLite.
- The official sqlite3 module in the Python stdlib docs contains a bunch of scenarios with code for how to use the database from a Python application.
- Using the SQLite JSON1 and FTS5 Extensions with Python shows how to compile SQLite 3.9.0+ with json1 and fts5 (full-text search) support to use these new features.
- SQLite with a fine-toothed comb digs into the internals of SQLite and shows some bugs found (and since fixed) while the author was researching the SQLite source code.

Object-relational mappers (ORMs)

An object-relational mapper (ORM) is a code library that automates the transfer of data stored in relational databases tables into objects that are more commonly used in application code.

Why are ORMs useful?

ORMs provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to create, read, update and delete data and schemas in their database. Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

For example, without an ORM a developer would write the following SQL statement to retrieve every row in the USERS table where the `zip_code` column is 94107:

```
SELECT * FROM USERS WHERE zip_code=94107;
```

The equivalent Django ORM query would instead look like the following Python code:

```
# obtain everyone in the 94107 zip code and assign to users variable
users = Users.objects.filter(zip_code=94107)
```

The ability to write Python code instead of SQL can speed up web application development, especially at the beginning of a project. The potential development speed boost comes from not having to switch from Python code into writing declarative paradigm SQL statements. While some software developers may not mind switching back and forth between languages, it's typically easier to knock out a prototype or start a web application using a single programming language.

ORMs also make it theoretically possible to switch an application between various relational databases. For example, a developer could use SQLite for local development and MySQL in production. A production application could be switched from MySQL to

PostgreSQL with minimal code modifications.

In practice however, it's best to use the same database for local development as is used in production. Otherwise unexpected errors could hit in production that were not seen in a local development environment. Also, it's rare that a project would switch from one database in production to another one unless there was a pressing reason.

Do I have to use an ORM for my web application?

Python ORM libraries are not required for accessing relational databases. In fact, the low-level access is typically provided by another library called a *database connector*, such as psycopg (for PostgreSQL) or MySQL-python (for MySQL). Take a look at the table below which shows how ORMs can work with different web frameworks and connectors and relational databases.

web framework	None	Flask	Flask	Django
ORM	SQLAlchemy	SQLAlchemy	SQLAlchemy	Django ORM
database connector	(built into Python stdlib)	MySQL-python	psycopg	psycopg
relational database	SQLite	MySQL	PostgreSQL	PostgreSQL

The above table shows for example that SQLAlchemy can work with varying web frameworks and database connectors. Developers can also use ORMs without a web framework, such as when creating a data analysis tool or a batch script without a user interface.

What are the downsides of using an ORM?

There are numerous downsides of ORMs, including

1. Impedance mismatch
2. Potential for reduced performance
3. Shifting complexity from the database into the application code

Impedance mismatch

The phrase "impedance mismatch" is commonly used in conjunction with ORMs. Impedance mismatch is a catch-all term for the difficulties that occur when moving data between relational tables and application objects. The gist is that the way a developer uses objects is different from how data is stored and joined in relational tables.

This article on ORM impedance mismatch does a solid job of explaing what the concept is at a high level and provides diagrams to visualize why the problem occurs.

Potential for reduced performance

One of the concerns that's associated with any higher-level abstraction or framework is potential for reduced performance. With ORMs, the performance hit comes from the translation of application code into a corresponding SQL statement which may not be tuned properly.

ORMs are also often easy to try but difficult to master. For example, a beginner using Django might not know about the `select_related()` function and how it can improve some queries' foreign key relationship performance. There are dozens of performance tips and tricks for every ORM. It's possible that investing time in learning those quirks may be better spent just learning SQL and how to write stored procedures.

There's a lot of hand-waving "may or may not" and "potential for" in this section. In large projects ORMs are good enough for roughly 80-90% of use cases but in 10-20% of a project's database interactions there can be major performance improvements by having a knowledgeable database administrator write tuned SQL statements to replace the ORM's generated SQL code.

Shifting complexity from the database into the app code

The code for working with an application's data has to live somewhere. Before ORMs were common, database stored procedures were used to encapsulate the database logic. With an ORM, the data manipulation code instead lives within the application's

Python codebase. The addition of data handling logic in the codebase generally isn't an issue with a sound application design, but it does increase the total amount of Python code instead of splitting code between the application and the database stored procedures.

Python ORM Implementations

There are numerous ORM implementations written in Python, including

1. The Django ORM
2. SQLAlchemy
3. Peewee
4. PonyORM
5. SQLObject

There are other ORMs, such as Canonical's Storm, but most of them do not appear to currently be under active development. Learn more about the major active ORMs below.

Django's ORM

The Django web framework comes with its own built-in object-relational mapping module, generally referred to as "the Django ORM" or "Django's ORM".

Django's ORM works well for simple and medium-complexity database operations. However, there are often complaints that the ORM makes complex queries much more complicated than writing straight SQL or using SQLAlchemy.

It's technically possible to drop down to SQL but it ties the queries to a specific database implementation. The ORM is coupled closely with Django so replacing the default ORM with SQLAlchemy is currently a hack workaround. Note though that some of the Django core committers believe it is only a matter of time before the default ORM is replaced with SQLAlchemy. It will be a large effort to get that working though so it's likely to come in Django 1.9 or later.

Since the majority of Django projects are tied to the default ORM, it's best to read up on advanced use cases and tools for doing your best work within the existing framework.

SQLAlchemy

SQLAlchemy is a well-regarded Python ORM because it gets the abstraction level "just right" and seems to make complex database queries easier to write than the Django ORM in most cases. SQLAlchemy is typically used with Flask as the database ORM via the Flask-SQLAlchemy extension.

Peewee

Peewee is a Python ORM written to be "simpler, smaller and more hackable" than SQLAlchemy. The analogy used by the core Peewee author is that Peewee is to SQLAlchemy as SQLite is to PostgreSQL. An ORM does not have to work for every exhaustive use case in order to be useful.

Pony

Pony ORM is another Python ORM with a slight twist in its licensing model. The project is multi-licensed. Pony is free for use on open source projects but has a commercial license that is required for commercial projects. The license is a one-time payment and does not necessitate a recurring fee.

SQLObject

SQLObject is an ORM that has been under active open source development since before 2003.

Schema migrations

Schema migrations, for example when you need to add a new column to an existing table in your database, are not technically part of ORMs. However, since ORMs typically lead to a hands-off approach to the database (at the developers peril in many cases), libraries to perform schema migrations often go hand-in-hand with Python ORM usage on web application projects.

Database schema migrations are a complex topic and deserve their own page. For now, we'll lump schema migration resources under ORM links below.

General ORM resources

- There's also a detailed overview of what ORMs are on another page of the website.
- This example GitHub project implements the same Flask application with several different ORMs: SQLAlchemy, Peewee, MongoEngine, stdnet and PonyORM.
- Martin Fowler addresses the ORM hate in an essay about how ORMs are often misused but that they do provide benefits to developers.
- If you're confused about the difference between a connector, such as MySQL-python and an ORM like SQLAlchemy, read this StackOverflow answer on the topic.

Django ORM resources

- Django models, encapsulation and data integrity is a detailed article by Tom Christie on encapsulating Django models for data integrity.
- Django Debug Toolbar is a powerful Django ORM database query inspection tool. Highly recommended during development to ensure you're writing reasonable query code. Django Silk is another inspection tool and has capabilities to do more than just SQL inspection.
- Making a specific Django app faster is a Django performance blog post with some tips on measuring performance and optimizing based on the measured results.
- Why I Hate the Django ORM is Alex Gaynor's overview of the bad designs decisions, some of which he made, while building the Django ORM.
- Going Beyond Django ORM with Postgres is specific to using PostgreSQL with Django.
- Migrating a Django app from MySQL to PostgreSQL is a quick look at how to move from MySQL to PostgreSQL. However, my guess is that any Django app that's been running for awhile on one relational database will require a lot more work to port over to another backend even with the power of the ORM.
- Django Model Descriptors discusses and shows how to incorporate business logic into Django models to reduce complexity from the views and make the code easier to reuse across separate views.
- Supporting both Django 1.7 and South explains the difficulty of supporting Django

1.7 and maintaining South migrations for Django 1.6 then goes into how it can be done.

- Adding basic search to your Django site shows how to write generic queries that'll allow you to provide site search via the Django ORM without relying on another tool like ElasticSearch. This is great for small sites before you scale them up with a more robust search engine.

- How to use Django's Proxy Models is a solid post on a Django ORM concept that doesn't frequently get a lot of love or explanation.

- Tightening Django Admin Logins shows you how to log authentication failures, create an IP addresses white list and combine fail2ban with the authentication failures list.

- Django Migrations - a Primer takes you through the new migrations system integrated in the Django core as of Django 1.7, looking specifically at a solid workflow that you can use for creating and applying migrations.

- Django 1.7: Database Migrations Done Right explains why South was not directly integrated into Django, how migrations are built and shows how backwards migrations work.

- Squashing and optimizing migrations in Django shows a simple example with code for how to use the migrations integrated into Django 1.7.

- Sorting querysets with NULLs in Django shows what to do if you're struggling with the common issue of sorting columns that contain NULL values.

SQLAlchemy resources

- If you're interested in the differences between SQLAlchemy and the Django ORM I recommend reading SQLAlchemy and You by Armin Ronacher.

- There is an entire chapter in the Architecture of Open Source Applications book on SQLAlchemy . The content is detailed and well worth reading to understand what's happening under the covers.

- SQLAlchemy vs Other ORMs provides a detailed comparison of SQLAlchemy

against alternatives.

- Most Flask developers use SQLAlchemy as an ORM to relational databases. If you're unfamiliar with SQLAlchemy questions will often come up such as what's the difference between flush and commit? that are important to understand as you build out your app.

Peewee resources

- Managing database connections with Peewee explains the connection pool and ExecutionContext of the ORM.

- An encrypted command-line diary with Python is an awesome walkthrough explaining how to use SQLite, SQLCipher and Peewee to create an encrypted file with your contents, diary or otherwise.

- The official Peewee quickstart documentation along with the example Twitter clone app will walk you through the ins and outs of your first couple Peewee-powered projects.

- Shortcomings in the Django ORM and a look at Peewee from the author of the Peewee ORM explains how some of the design decisions made in Peewee were in reaction to parts of the Django ORM that didn't work so well in practice.

- How to make a Flask blog in one hour or less is a well written tutorial that uses the Peewee ORM instead of SQLAlchemy for the blog back end.

Pony ORM resources

- Why you should give Pony ORM a chance explains some of the benefits of Pony ORM that make it worth trying out.

- An intro to Pony ORM shows the basics of how to use the library, such as creating databases and manipulating data.

- The Pony ORM author explains on a Stack Overflow answer how Pony ORM works behind the scenes. Worth a read whether or not you're using the ORM just to find out how some of the magic coding works.

SQLObject resources

- This post on Object-Relational Mapping with SQLObject explains the concept behind ORMs and shows the Python code for how they can be used.

- Ian Bicking presented on SQLObject back in 2004 with a talk on SQLObject and Database Programming in Python.

NoSQL Data Stores

Relational databases store the vast majority of web application persistent data. However, there are several alternative classifications of storage representations.

1. Key-value pair
2. Document-oriented
3. Column-family table
4. Graph

These persistent data storage representations are commonly used to augment, rather than completely replace, relational databases. The underlying persistence type used by the NoSQL database often gives it different performance characteristics than a relational database, with better results on some types of read/writes and worse performance on others.

Key-value Pair

Key-value pair data stores are based on hash map data structures.

Key-value pair data stores

- Redis is an open source in-memory key-value pair data store. Redis is often called "the Swiss Army Knife of web application development." It can be used for caching, queuing, and storing session data for faster access than a traditional relational database, among many other use cases. Redis-py is a solid Python client to use with Redis.

- Memcached is another widely used in-memory key-value pair storage system.

Key-value pair resources

- What is a key-value store database? is a Stack Overflow Q&A that straight on answers this subject.

Redis resources

- "How To Install and Use Redis" is a guide for getting up with the extremely useful in-memory data store.
- Getting started with Redis and Python is a walkthrough for installing and playing around with the basics of Redis.
- This video on Scaling Redis at Twitter is a detailed look behind the scenes with a massive Redis deployment.
- Walrus is a higher-level Python wrapper for Redis with some caching, querying and data structure components build into the library.
- Writing Redis in Python with Asyncio shows a detailed example for how to use the new Asyncio standard library in Python 3.4+ for working with Redis.
- Pentesting Redis servers shows that security is important not only on your application but also the databases you're using as well.
- Redis, just as with any relational or NoSQL database, needs to be secured based on security guidelines. There is also a post where the main author of Redis cracks its security to show the tradeoffs purposely made between ease of use and security in the default settings.

Document-oriented

A document-oriented database provides a semi-structured representation for nested data.

Document-oriented data stores

- MongoDB is an open source document-oriented data store with a Binary Object Notation (BSON) storage format that is JSON-style and familiar to web developers. PyMongo is a commonly used client for interfacing with one or more MongoDB instances through Python code. MongoEngine is a Python ORM specifically written for MongoDB that is built on top of PyMongo.

- Riak is an open source distributed data store focused on availability, fault tolerance and large scale deployments.

- Apache CouchDB is also an open source project where the focus is on embracing RESTful-style HTTP access for working with stored JSON data.

Document-oriented data store resources

- MongoDB for startups is a guide about using non-relational databases in green field environments.

- The creator and maintainers of PyMongo review four decisions they regret from building the widely-used Python MongoDB driver.

    1. start_request
    2. use_greenlets
    3. copy_database
    4. MongoReplicaSetClient

- The Python and MongoDB Talk Python to Me podcast has a great interview with the maintainer of the Python driver for MongoDB.

Column-family table

A the column-family table class of NoSQL data stores builds on the key-value pair type. Each key-value pair is considered a row in the store while the column family is similar to a table in the relational database model.

Column-family table data stores

- Apache HBase

- Apache Cassandra

Graph

A graph database represents and stores data in three aspects: nodes, edges and

properties.

A *node* is an entity, such as a person or business.

An *edge* is the relationship between two entities. For example, an edge could represent that a node for a person entity is an employee of a business entity.

A *property* represents information about nodes. For example, an entity representing a person could have a property of "female" or "male".

Graph data stores

- Neo4j is one of the most widely used graph databases and runs on the Java Virtual Machine stack.
- Cayley is an open source graph data store written by Google primarily written in Go.
- Titan is a distributed graph database built for multi-node clusters.

Graph data store resources

- Introduction to Graph Databases covers trends in NoSQL data stores and compares graph databases to other data store types.

NoSQL third-party services

- MongoHQ provides MongoDB as a service. It's easy to set up with either a standard LAMP stack or on Heroku.

NoSQL data store resources

- NoSQL databases: an overview explains what NoSQL means, how data is stored differently than in relational systems and what the Consistency, Availability and Partition-Tolerance (CAP) Theorem means.
- CAP Theorem overview presents the basic constraints all databases must trade off in operation.

- This post on What is a NoSQL database? Learn By Writing One in Python is a detailed article that breaks the mystique behind what some forms of NoSQL databases are doing under the covers.

- The CAP Theorem series explains concepts related to NoSQL such as what is ACID compared to CAP, CP versus CA and high availability in large scale deployments.

- NoSQL Weekly is a free curated email newsletter that aggregates articles, tutorials, and videos about non-relational data stores.

- NoSQL comparison is a large list of popular, BigTable-based, special purpose, and other datastores with attributes and the best use cases for each one.

- Relational databases such as MySQL and PostgreSQL have added features in more recent versions that mimic some of the capabilities of NoSQL data stores. For example, check out this blog post on using MySQL as a key-value pair store and this post on storing JSON data in PostgreSQL.

NoSQL data stores learning checklist

1. Understand why NoSQL data stores are better for some use cases than relational databases. In general these benefits are only seen at large scale so they may not be applicable to your web application.

2. Integrate Redis into your project for a speed boost over slower persistent storage. Storing session data in memory is generally much faster than saving that data in a traditional relational database that uses persistent storage. Note that when memory is flushed the data goes away so anything that needs to be persistent must still be backed up to disk on a regular basis.

3. Evaluate other use cases such as storing transient logs in a document-oriented data store such as MongoDB.

Web APIs

Application Programming Interfaces

Application programming interfaces (APIs) provide machine-readable data transfer and signaling between applications.

Why are APIs important?

HTML, CSS and JavaScript create human-readable webpages. However, those webpages are not easily consumable by other machines.

Numerous scraping programs and libraries exist to rip data out of HTML but it's simpler to consume data through APIs. For example, if you want the content of a news article it's easier to get the content through an API than to scrap the text out of the HTML.

Key API concepts

There are several key concepts that get thrown around in the APIs world. It's best to understand these ideas first before diving into the API literature.

- Representation State Transfer (REST)
- Webhooks
- JavaScript Object Notation (JSON) and Extensible Markup Language (XML)
- Endpoints

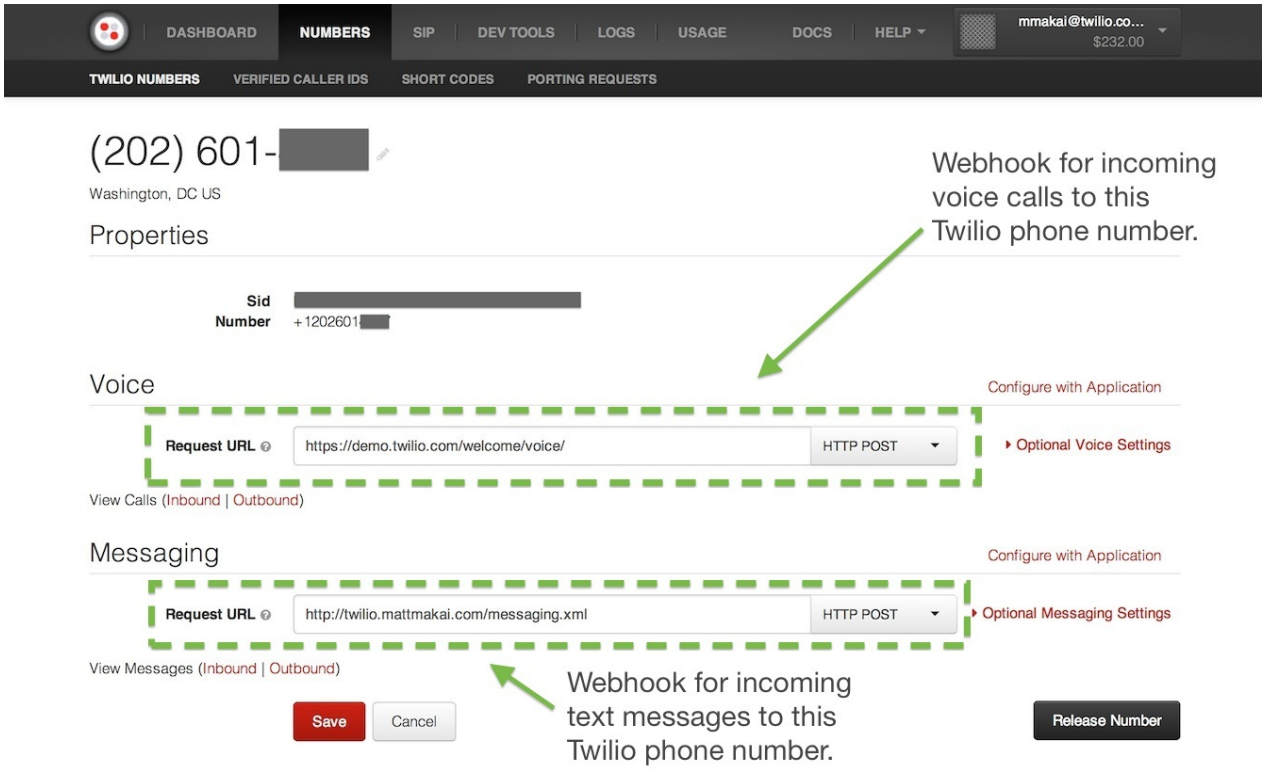
Webhooks

A webhook is a user-defined HTTP callback to a URL that executes when a system condition is met. The call alerts the second system via a POST or GET request and often passes data as well.

Webhooks are important because they enable two-way communication initiation for APIs. Webhook flexibility comes in from their definition by the API user instead of the

API itself.

For example, in the Twilio API when a text message is sent to a Twilio phone number Twilio sends an HTTP POST request webhook to the URL specified by the user. The URL is defined in a text box on the number's page on Twilio as shown below.



API open source projects

- Swagger is an open source project written in Scala that defines a standard interface for RESTful APIs.

API resources

- Zapier has an APIs 101 free guide for what APIs are, why they are valuable and how to use them properly.

- What RESTful actually means does a fantastic job of laying out the REST principles in plain language terms while giving some history on how they came to be.

- What is a webhook? by Nick Quinlan is a plain English explanation for what webhooks are and why they are necessary in the API world.

- Simplicity and Utility, or, Why SOAP Lost provides context for why JSON-based web services are more common today than SOAP which was popular in the early 2000s.

- API tools for every occasion provides a list of 10 tools that are really helpful when working with APIs that are new in 2015.

APIs learning checklist

1. Learn the API concepts of machine-to-machine communication with JSON and XML, endpoints and webhooks.

2. Integrate an API such as Twilio or Stripe into your web application. Read the API integration section for more information.

3. Use a framework to create an API for your own application. Learn about web API frameworks on the API creation page.

4. Expose your web application's API so other applications can consume data you want to share.

API Integration

The majority of production Python web applications rely on several externally hosted application programming interfaces (APIs). APIs are also commonly referred to as third party services or external platforms. Examples include Twilio for messaging and voice services, Stripe for payment processing and Disqus for embedded webpage comments.

There are many articles about proper API design but best practices for integrating APIs is less commonly written about. However, this subject continuously grows in importance because APIs provide critical functionality across many implementation areas.

Hosted API testing services

- Runscope is a service specifically designed for APIs that assists developers with automated testing and traffic inspection.
- Apiary provides a blueprint for creating APIs so they are easier to test and generate clean documentation.

API Integration Resources

- Some developers prefer to use Requests instead of an API's helper library. In that case check out this tutorial on using requests to access web APIs.
- Product Hunt lists many commonly used commercial and free web APIs to show "there's an API for everything".
- There's a list of all government web APIs at 18F's API-All-the-X list. The list is updated whenever a new API comes online.
- If you use Requests check out this handy guide on gracefully handling HTTP errors with Python.
- John Sheehan's "Zen and the Art of API Maintenance" slides are relevant for API integration.

- This post on "API Driven Development" by Randall Degges explains how using APIs in your application cuts down on the amount of code you have to write and maintain so you can launch your application faster.
- Safe Sex with Third Party APIs is a funny high level overview of what you should do to protect your application when relying on third party services.
- Retries in Requests is a nice tutorial for easily re-executing failed HTTP requests with the Requests library.
- My DjangoCon 2013 talk dove into "Making Django Play Nice With Third Party Services."
- If you're looking for a fun project that uses two web APIs within a Django application, try out this tutorial to Build your own Pokédex with Django, MMS and PokéAPI.
- vcr.py is a way to capture and replay HTTP requests with mocks. It's extremely useful for testing API integrations.
- Caching external API requests is a good post on how to potentially limit the number of HTTP calls required when accessing an external web API via the Requests library.

API integration learning checklist

1. Pick an API known for top notch documentation. Here's a list of ten APIs that are a good starting point for beginners.
2. Read the API documentation for your chosen API. Figure out a simple use case for how your application could be improved by using that API.
3. Before you start writing any code, play around with the API through the commandline with cURL or in the browser with Postman. This exercise will help you get a better understanding of API authentication and the data required for requests and responses.
4. Evaluate whether to use a helper library or work with Requests. Helper libraries are usually easier to get started with while Requests gives you more control over the

HTTP calls.

5. Move your API calls into a task queue so they do not block the HTTP request-response cycle for your web application.

API Creation

Creating and exposing APIs allows your web application to interact with other applications through machine-to-machine communication.

API creation frameworks

- Django REST framework and Tastypie are the two most widely used API frameworks to use with Django. The edge currently goes to Django REST framework based on rough community sentiment. Django REST framework continues to knock out great releases after the 3.0 release mark when Tom Christie ran a successful Kickstarter campaign.

- Flask-RESTful is widely used for creating web APIs with Flask. It was originally open sourced and explained in a blog post by Twilio then moved into its own GitHub organization so engineers from outside the company could be core contributors.

- Flask API is another common library for exposing APIs from Flask web applications.

- Sandman is a widely used tool to automatically generate a RESTful API service from a legacy database without writing a line of code (though it's easily extensible through code).

- Cornice is a REST framework for Pyramid.

- Restless is a lightweight API framework that aims to be framework agnostic. The general concept is that you can use the same API code for Django, Flask, Bottle, Pyramid or any other WSGI framework with minimal porting effort.

- Eve is a Python REST framework built with Flask, MongoDB and Redis. The framework's primary author Nicola Iarocci gave a great talk at EuroPython 2014 that introduced the main features of the framework.

- Falcon is a fast and lightweight framework well suited to create RESTful APIs.

- Hug built on-top of Falcon and Python3 with an aim to make developing Python

driven APIs as simple as possible, but no simpler. Hug leverages Python3 annotations to automatically validate and convert incoming and outgoing API parameters.

- Pycnic is a JSON-API-only framework designed with REST in mind.

API testing projects

Building, running and maintaining APIs requires as much effort as building, running and maintaining a web application. API testing frameworks are the equivalent of browser testing in the web application world.

- zato-apitest invokes HTTP APIs and provides hooks for running through other testing frameworks.

Hosted API testing services

- Runscope is an API testing SaaS application that can test both your own APIs and external APIs that your application relies upon.

- API Science is focused on deep API testing, including multi-step API calls and monitoring of external APIs.

- SmartBear has several API monitoring and testing tools for APIs.

API creation resources

- An API is only as good as its documentation is a strongly held mantra in the web API world because so many APIs have poor documentation that prevents ease-of-use. If an API is not well documented then developers who have options to use something else will just skip it.

- Adventures in running a free, public API is a quick story of a developer's geolocation API being abused and his lack of resources for preventing further abuse. Eventually he had to shut down the free plan and only provide a paid plan in addition to allowing others to host the open source code. Fraud and malware prevention are

difficult problems so keep an eye on server utilization and endpoint calls growth to separate legitimate from illegitimate traffic.

- API Doc JS allows a developer to embed markup in their documentation that will generate a site based on the endpoints available in the API.

- 10 Reasons Why Developers Hate Your API (And what to do about it) goes through the top difficulties and annoyances developers face when working with APIs and how you can avoid your API falling into the same traps.

- Versioning of RESTful APIs is a difficult and contentious topic in the web API community. This two-part series covers various ways to version your API and how to architect a version-less API.

- NARWHL is a practical API design site for developers confused about what is appropriate for RESTful APIs.

- 18F's API standards explains the details behind their design decisions on creating modern RESTful APIs.

- Design a beautiful REST API reviews common design decisions regarding endpoints, versioning, errors and pagination. There is also a source material YouTube video where this blog post derives its recommendations from.

- Move Fast, Don't Break Your API are slides and a detailed blog post from Amber Feng at Stripe about building an API, separating layers of responsibility, hiding backwards compatibility and a whole slew of other great advice for developers and API designers.

- Self-descriptive, isn't. Don't assume anything. is an appeal that metadata makes a difference in whether APIs are descriptive or not.

- Designing the Artsy API has their recommendations list for building an API based on their recent experiences.

- Some REST Best Practices is a high level summary of rules to follow while creating your API.

- Hacker News had a discussion on what's the best way to write an API spec? that provides a few different viewpoints on this topic.

- Apigee's Web API Design ebook is free and contains a wealth of practical advice for what design decisions to make for your web API.
- 1-to-1 Relationships and Subresources in REST APIs tells the story of design decisions that were made during an API's creation and why those choices were made.
- How many status codes does your API need? gives an answer from a Dropbox API developer as to their decision making process.
- This API Design Guide is based on Heroku's best practices for the platform's API.

Python-specific API creation resources

- Choosing an API framework for Django by PyDanny contains questions and insight into what makes a good API framework and which one you should currently choose for Django.
- Create a REST API in Minutes with Pyramid and Ramses is a thorough tutorial from start to finish that uses the Pyramid web framework along with Ramses, a library that uses YAML files to generate a RESTful API.
- RESTful web services with Python is an interesting overview of the Python API frameworks space.
- Implementing a RESTful Web API with Python & Flask is a good walkthrough for coding a Flask app that provides standard web API functionality such as proper HTTP responses, authentication and logging.
- REST Hooks is an open source Python project that makes it easier to implement subscription-based "REST hooks". These REST hooks are similar to webhooks, but provide a different mechanism for subscribing to updates via a REST interface. Both REST hooks and webhooks are far more efficient than polling for updates and notifications.
- Serialization is common for transforming objects into web API JSON results. One company found the serialization performance of Django REST framework was lacking so they created Serpy and wrote a blog post with the results of its

performance.

- Building better API docs shows how Square used Swagger with React to create more helpful docs.

Django REST Framework resources

- To create an API to export your data in comma-separated values, check out this blog post on exporting data as CSV format with Django REST framework.

- This multi-part series on getting started with Django REST framework and AngularJS (part 1) along with its second part do a good job of showing how a RESTful API can serve as the backend for a client front end built with a JavaScript MVC framework.

- If you're looking for a working example of a Django REST framework project, check out the PokeAPI, open sourced under the BSD license.

API creation learning checklist

1. Pick an API framework appropriate for your web framework. For Django I recommend Django REST framework and for Flask I recommend Flask-RESTful.

2. Begin by building out a simple use case for the API. Generally the use case will either involve data that users want in a machine-readable format or a backend for alternative clients such as an iOS or Android mobile app.

3. Add an authentication mechanism through OAuth or a token scheme.

4. Add rate limiting to the API if data usage volume could be a performance issue. Also add basic metrics so you can determine how often the API is being accessed and whether it is performing properly.

5. Provide ample documentation and a walkthrough for how the API can be accessed and used.

6. Figure out other use cases and expand based on what you learned with the initial API use case.

Web App Deployment

Deployment

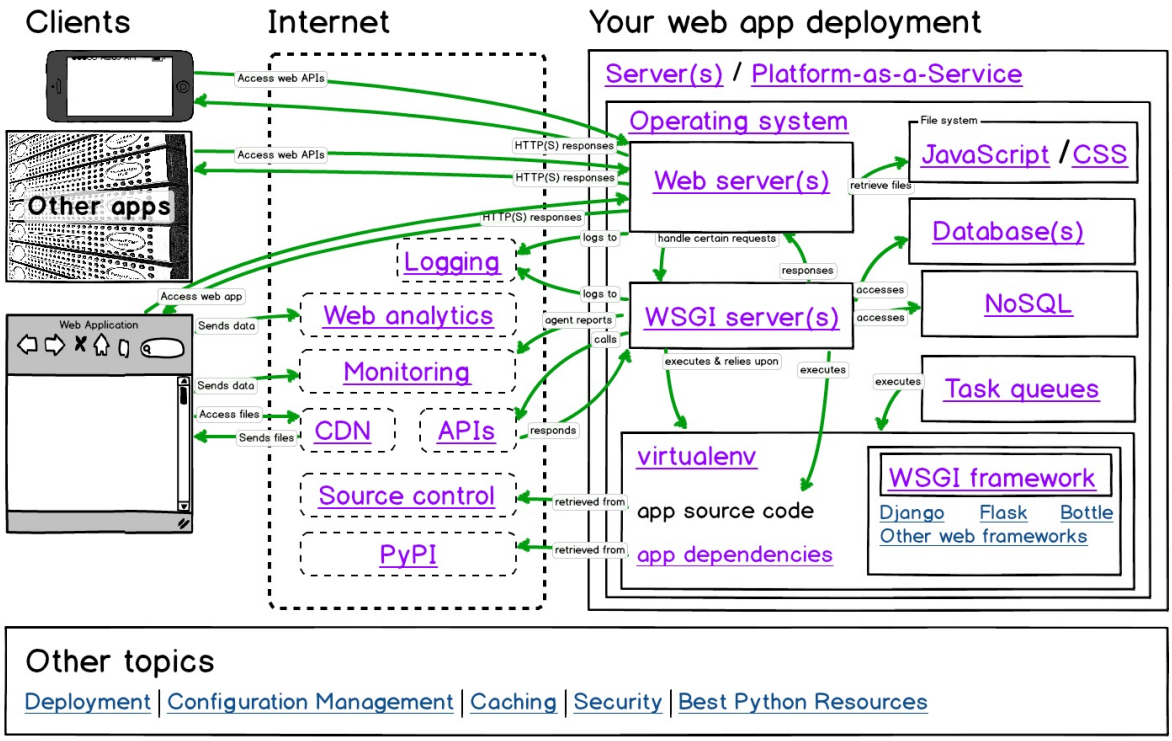
Deployment involves packaging up your web application and putting it in a production environment that can run the app.

Why is deployment necessary?

Your web application must live somewhere other than your own desktop or laptop. A production environment is the canonical version of your current application and its associated data.

Deployment topics map

Python web application deployments are comprised of many pieces that need to be individually configured. Here is a map that visually depicts how each deployment topic relates to each other. Click the image to pull up a PDF version.



Deployment hosting options

There are four options for deploying and hosting a web application:

1. "Bare metal" servers
2. Virtualized servers
3. Infrastructure-as-a-service
4. Platform-as-a-service

The first three options are similar. The deployer needs to provision one or more servers with a Linux distribution. System packages, a web server, WSGI server, database and the Python environment are then installed. Finally the application can be pulled from source and installed in the environment.

Note that there are other ways of installing a Python web application through system-specific package management systems. We won't cover those in this guide as they are considered advanced deployment techniques.

Deployment resources

- If you need a step-by-step guide to deploying a Python web application, I wrote a whole book on exactly this topic called The Full Stack Python Guide to Deployments that you'll find super helpful.
- Deploying Python web applications is an episode of the great Talk Python to Me podcast series where I discuss deploying web applications based on a fairly traditional virtual private server, Nginx and Green Unicorn stack.
- Thoughts on web application deployment walks through stages of deployment with source control, planning, continuous deployment and monitoring the results.
- Deploying Software is a long must-read for understanding how to deploy software properly.
- Practical continuous deployment defines delivery versus deployment and walks

through a continuous deployment workflow.

- In this free video by Neal Ford, he talks about engineering practices for continuous delivery. He explains the difference between continuous integration, continuous deployment and continuous delivery. Highly recommended for an overview of deployment concepts and as an introduction to the other videos on those subjects in that series.

- If you're using Flask this detailed post on deploying it to Ubuntu is a great way to familiarize yourself with the deployment process.

Deployment learning checklist

1. If you're tight on time look at the platform-as-a-service (PaaS) options. You can deploy a low traffic project web app for free or low cost. You won't have to worry about setting up the operating system and web server compared to going the traditional server route. In theory you should be able to get your application live on the web sooner with PaaS hosting.

2. Traditional server options are your best bet for learning how the entire Python web stack works. You'll often save money with a virtual private server instead of a platform-as-a-service as you scale up.

3. Read about servers, operating systems, web servers and WSGI servers to get a broad picture of what components need to be set up to run a Python web application.

Servers

Servers are the physical infrastructure to run all the layers of software so your web application can respond to requests from clients such as web browsers.

Why are servers necessary?

Your web application must live somewhere other than your own desktop or laptop. Servers should ideally be accessible 24 hours a day, 7 days a week, with no unplanned downtime. The servers that host your web application for actual users (as opposed to test users) are known as *production* servers. Production servers hold real data (again as opposed to test data) and must be secure against unauthorized access.

Bare metal servers

The term *bare metal* refers to purchasing the actual hardware and hooking it up to the Internet either through a business-class internet service provider (ISP) or co-locating the server with other servers. A "business-class" ISP is necessary because most residential Internet service agreements explicitly prohibit running web servers on their networks. You may be able to get away with low traffic volume but if your site serves a lot of traffic it will alert an ISP's filters.

The bare metal option offers the most control over the server configuration, usually has the highest performance for the price, but also is the most expensive upfront option and the highest ongoing maintenance. With bare metal servers the ongoing operating cost is the electricity the server(s) use as well as handling repairs when server components malfunction. You're taking on manual labor working with hardware as well as the rest of the software stack.

Buy actual hardware from a vendor either pre-built or as a collection of components that you assemble yourself. You can also buy pre-configured servers from Dell or HP. Those servers tend to be in smaller case form factors (called "blades") but are correspondingly more expensive than putting off-the-shelf components together yourself in a standard computer case.

Virtualized servers

Virtual private servers (VPSs) are slices of hardware on top of a larger bare metal server. Virtualization software such as Xen and VMWare allow providers such as Linode and prgmr (as well as a many others) to provide fractions of a full server that appear as their own instances. For example, a server with an 8-core Xeon processor and 16 gigabytes of memory can be sliced into 8 pieces with the equivalent of 1-core and 2 gigabytes of memory.

The primary disadvantage of virtualized servers is that there is resource overhead in the virtualization process. In addition, physical constraints such as heavy I/O operations by a single virtualized instance on persistent storage can cause performance bottlenecks for other virtualized instances on the shared server. Choosing virtualized server hosting should be based on your needs for urgency of service ticket requests and the frequency you require for ongoing maintenance such as persistent storage backups.

Virtualized servers resources

- Choosing a low cost VPS reviews the factors that you should weigh when deciding on hosting providers.
- How to set up your Linode for maximum awesomeness shows how to work with a VPS once you've got the server up and running.
- CPU Load Averages explains how to measure CPU load and what to do about it.

Infrastructure-as-a-service

Infrastructure-as-a-service (IaaS) overlaps with virtualized servers because the resources are often presented in the same way. The difference between virtualized servers and IaaS is the granularity of the billing cycle. IaaS generally encourages a finer granularity based on minutes or hours of server usage instead of on monthly billing cycles.

IaaS can be used in combination with virtualized servers to provide dynamic upscaling for heavy traffic. When traffic is low then virtualized servers can solely be used. This combination of resources reduces cost at the expense of greater complexity in the

dynamically scaled infrastructure.

The most common IaaS platforms are Amazon Web Services and Rackspace Cloud.

The disadvantage to IaaS platforms is the lock-in if you have to write custom code to deploy, dynamically scale, and generally understand your infrastructure. Every platform has its quirks. For example, Amazon's standard Elastic Block Store storage infrastructure has at least an order of magnitude worse I/O throughput than working with your local disk. Your application's database queries may work great locally but then when you deploy the performance is inadequate. Amazon has higher throughput EBS instances but you will pay correspondingly more for them. EBS throughput is just one of many quirks you need to understand before committing to an IaaS platform.

Infrastructure-as-a-service resources

- The cloud versus dedicated servers
- 5 common server setups for your web application is a great introduction to how hosting can be arranged.
- Apache Libcloud is a Python library that provides a unified API for many cloud service providers.
- Amazon Web Services has official documentation for running Python web applications.
- boto is an extensive and well-tested Python library for working with Amazon Web Services.
- Poseidon is a Python commandline interface for managing Digital Ocean droplets (servers).

Servers learning checklist

1. Sign up for a hosting provider. I recommend getting a Linode VPS to set up your initial infrastructure and deploy your web application there. Digital Ocean and prgrmr are other VPS options. You can change hosting providers later after the deployment process is automated.

2. Provision your first server. It will be ready but in a shutdown state while awaiting your instructions.

3. Move to the operating systems section to learn how to load Ubuntu 14.04 LTS as a base OS for Python web applications.

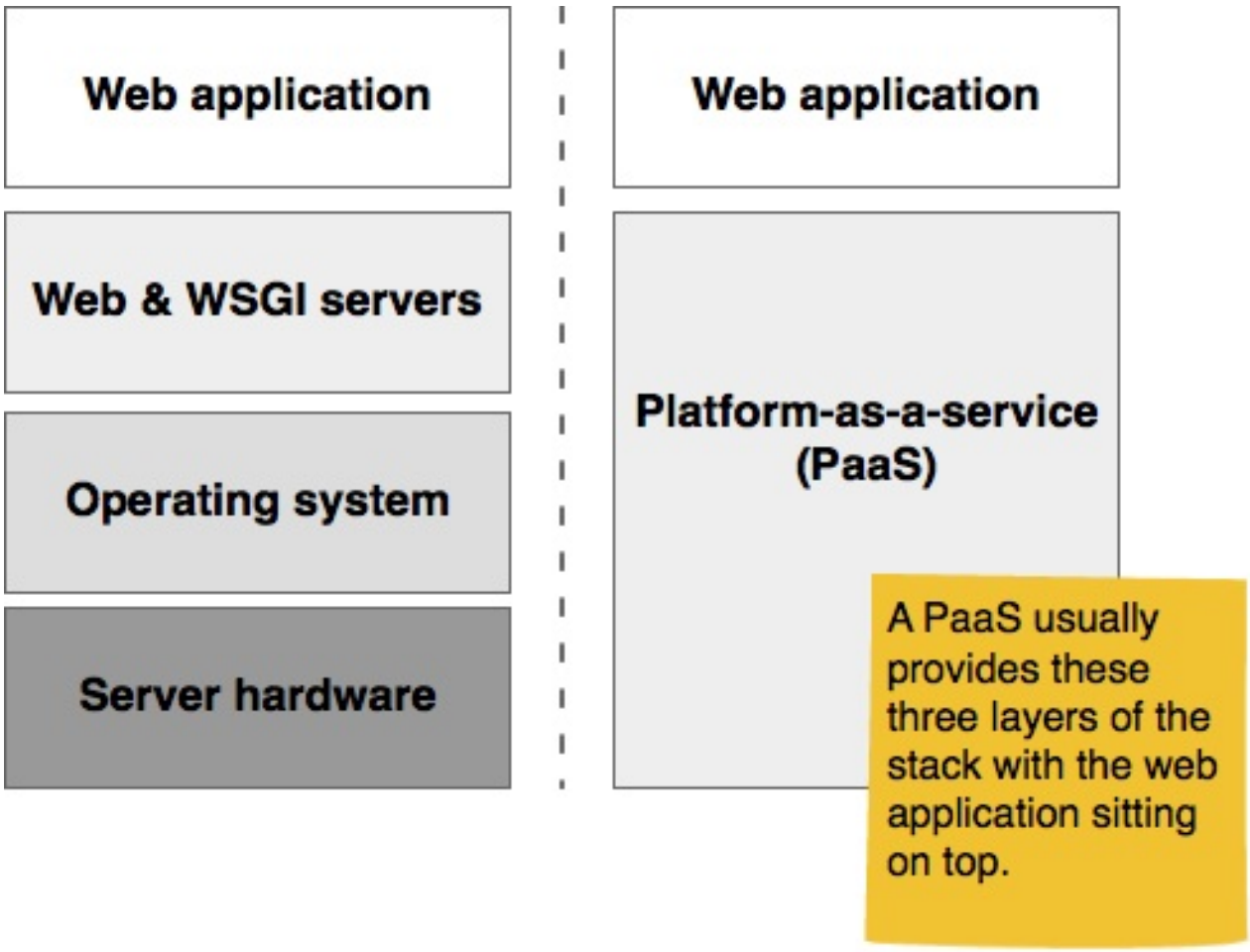
Platform-as-a-service

A platform-as-a-service (PaaS) provides infrastructure and a software layer on which a web application is deployed. Running your web application from a PaaS removes the need to know as much about the underlying servers, operating system, web server, and often the WSGI server.

Note: If you are not interested in deploying to a PaaS you can move ahead to the WSGI servers section.

The PaaS layer defines how the application accesses resources such as computing time, files, and external services. The PaaS provides a higher-level abstraction for working with computing resources than deploying an application to a server or IaaS.

A PaaS makes deployment and operations easier because it forces the developer to conform applications to the PaaS architecture. For example, Heroku looks for Python's `requirements.txt` file in the base directory of the repository during deployment because that is the file's de facto community standard location.



If you go the PaaS route, you can skip configuring an operating system and web server prebaked into PaaS offerings. PaaS offerings generally start at the WSGI server layer.

Platform-as-a-service responsibilities

Although PaaS offerings simplify setting up and maintaining the servers, operating system, and web server, developers still have responsibilities for other layers of their web stack.

While it's useful to know the operating system that underpins your PaaS, for example Heroku uses Ubuntu 10.04, you will not have to know as much about securing the operating system and server level. However, web applications deployed to a PaaS are just as vulnerable to security breaches at the application level as a standard LAMP stack. It's still your responsibility to ensure the web application framework and your app itself is up to date and secured. See the security section for further information.

Platforms-as-a-service that support Python

- Heroku
- Google App Engine
- Gondor
- PythonAnywhere
- OpenShift
- AWS Elastic Beanstalk

Platform-as-a-service resources

- PaaS bakeoff: Comparing Stackato, OpenShift, Dotcloud and Heroku for Django hosting and deployment by Nate Aune.
- Deploying Django by Randall Degges is another great free resource about Heroku.
- AWS in Plain English shows what current Amazon Web Services individual services are currently called and what they could've been called to be more clear to users.
- 5 AWS mistakes you should avoid explains how common beginner practices such as manually managing infrastructure, not using scaling groups and underutilizing instances can create problems you'd be better off avoiding altogether.
- Heroku's Python deployment documentation provides clear examples for how to work with virtualenv, pip and `requirements.txt` to get a applications deployed to their platform.
- Miguel Grinberg's Flask tutorial contains an entire post on deploying Flask applications to Heroku.
- This series on DevOps Django by Randall Degges is great reading for using the Heroku service:
    - Part One: Goals
    - Part Two: The Pain of Deployment
    - Part Three: The Heroku Way

    - Part Four: Choosing Heroku
- Deploying a Django App to AWS Elastic Beanstalk is a fantastic post that shows how to deploy to Amazon Web Service's own PaaS.

- Deploy your hack in 3 steps: Intro to AWS and Elastic Beanstalk shows how to deploy a simple Ruby Sinatra app, but the steps are generally applicable to Python web apps as well.

- Are you wondering what it will cost to deploy a reasonable sized production app on a platform-as-a-service like Heroku? Check out Cushion's transparent costs list where they include their expenses from using a PaaS as well as other services.

- The beginner's guide to scaling to 11 million users on AWS is a useful list of services you'll need to look at as you grow an application from 10 to 100 to 1000 to 500,000 and beyond to millions of users.

- How to Separate Your AWS Production and Development Accounts is a basic post on keeping developer sandbox accounts separate from production AWS environments.

- How much is Spotify Paying Google Cloud? provides some insight into how Spotify runs all of their infrastructure on Google Cloud and posits what they may be paying to run their service.

Platform-as-a-service learning checklist

1. Review the potential Python platform-as-a-service options listed above.

2. Sign up for a PaaS account at the provider that appears to best fit your application needs. Heroku is the PaaS option recommended for starters due to their detailed documentation and walkthroughs available on the web. However, the other options are also viable since their purpose is to make deploying applications as easy as possible.

3. Check if there are any PaaS-specific configuration files needed for your app to run properly on the PaaS after it is deployed.

4. Deploy your app to the PaaS.

5. Sync your application's configuration with the database.

6. Set up a content delivery network for your application's static content unless your PaaS provider already handles this deployment step for you.

7. Check if the application's functionality is working and tweak as necessary.

Operating Systems

An operating system runs on the server or virtual server and controls access to computing resources. The operating system also includes a way to install programs necessary for running your Python web application.

Why are operating systems necessary?

An operating system makes many of the computing tasks we take for granted easy. For example, the operating system enables writing to files, communicating over a network and running multiple programs at once. Otherwise you'd need to control the CPU, memory, network, graphics card, and many other components with your own low-level implementation.

Without using an existing operating system like Linux, Mac OS X or Windows, you'd be forced to write a new operating system as part of your web application. It would be impossible to write features for your Python web application because you'd be too busy hunting down a memory leak in your assembly code, if you even were able to get that far.

Fortunately, the open source community provides Linux to the Python world as a rock solid free operating system for running our applications.

Recommended operating systems

The only recommended operating system for production Python web stack deployments is Linux. There are several Linux distributions commonly used for running production servers. Ubuntu Long Term Support (LTS) releases, Red Hat Enterprise Linux, and CentOS are all viable options.

Mac OS X is fine for development activities. Windows and Mac OS X are not appropriate for production deployments unless there is a major reason why you must use them in lieu of Linux.

Ubuntu

Ubuntu is a Linux distribution packaged by the Canonical Ltd company. Ubuntu uses the Debian distribution as a base for packages, including the aptitude package manager. For desktop versions of Ubuntu, GNOME (until the 11.04 release) or Unity (11.10 through current) is bundled with the distribution to provide a user interface.

Ubuntu Long Term Support (LTS) releases are the recommended versions to use for deployments. LTS versions receive five years of post-release updates from Canonical. Every two years, Canonical creates a new LTS release, which allows for an easy upgrade path as well as flexibility in skipping every other LTS release if necessary. As of November 2014, 14.04 Trusty Tahr is the latest Ubuntu LTS release.

Ubuntu Python Packages

There are several Aptitude packages found on Linux servers running a Python stack. These packages are:

- python-dev for header files and static library for Python
- python-virtualenv for creating and managing Python virtualenvs to isolate library dependencies

Red Hat and CentOS

Red Hat Enterprise Linux (RHEL) and Community ENTerprise Operating System (CentOS) are the same distribution. The primary difference between the two is that CentOS is an open source, liberally licensed free derivative of RHEL.

RHEL and CentOS use a different package manager and command-line interface from Debian-based Linux distributions: RPM Package Manager (RPM) and the Yellowdog Updater, Modified (YUM). RPM has a specific .rpm file format to handle the packaging and installation of libraries and applications. YUM provides a command-line interface for interacting with the RPM system.

Operating system resources

- What is a Linux distribution and how do I choose the right one?
- Lifehacker's guide to choosing a Linux distro.

- The Ops School curriculum is a comprehensive resource for learning about Linux fundamentals and how to perform the work that system administrators typically handle.
- Since Linux is your go-to production operating system, it's important to get comfortable with the Unix/Linux commands and philosophy. Study up on this introduction to Unix tutorial to become more familiar with the operating system.
- Choosing a Linux Distribution
- First 5 Minutes on a Server
- Digital Ocean has a detailed walkthrough for setting up Python web applications on Ubuntu.
- linux-internals is a series of posts about how Linux works under the covers, starting from the low level booting process.

Operating system learning checklist

1. Choose either a Debian-based Linux distribution such as Ubuntu or a Fedora-based distribution like CentOS.
2. Harden the security through a few basic steps. Install basic security packages such as fail2ban and unattended-upgrades. Create a new user account with sudo privileges and disable root logins. Disable password-only logins and use a public-private keypair instead. Read more about hardening systems in the resources listed below.
3. Install Python-specific packages to prepare the environment for running a Python application. Which packages you'll need to install depends on the distribution you've selected.
4. Read up on web servers as installing one will be the next step in the deployment process.

Web servers

Web servers respond to Hypertext Transfer Protocol (HTTP) requests from clients and send back a response containing a status code and often content such as HTML, XML or JSON as well.

Why are web servers necessary?

Web servers are the ying to the web client's yang. The server and client speak the standardized language of the World Wide Web. This standard language is why an old Mozilla Netscape browser can still talk to a modern Apache or Nginx web server, even if it cannot properly render the page design like a modern web browser can.

The basic language of the Web with the request and response cycle from client to server then server back to client remains the same as it was when the Web was invented by Tim Berners-Lee at CERN in 1989. Modern browsers and web servers have simply extended the language of the Web to incorporate new standards.

Web server implementations

The conceptual web server idea can be implemented in various ways. The following web server implementations each have varying features, extensions and configurations.

- The Apache HTTP Server has been the most commonly deployed web server on the Internet for 20+ years.
- Nginx is the second most commonly used server for the top 100,000 websites and often serves as a reverse proxy for Python WSGI servers.
- Caddy is a newcomer to the web server scene and is focused on serving the HTTP/2 protocol with HTTPS.

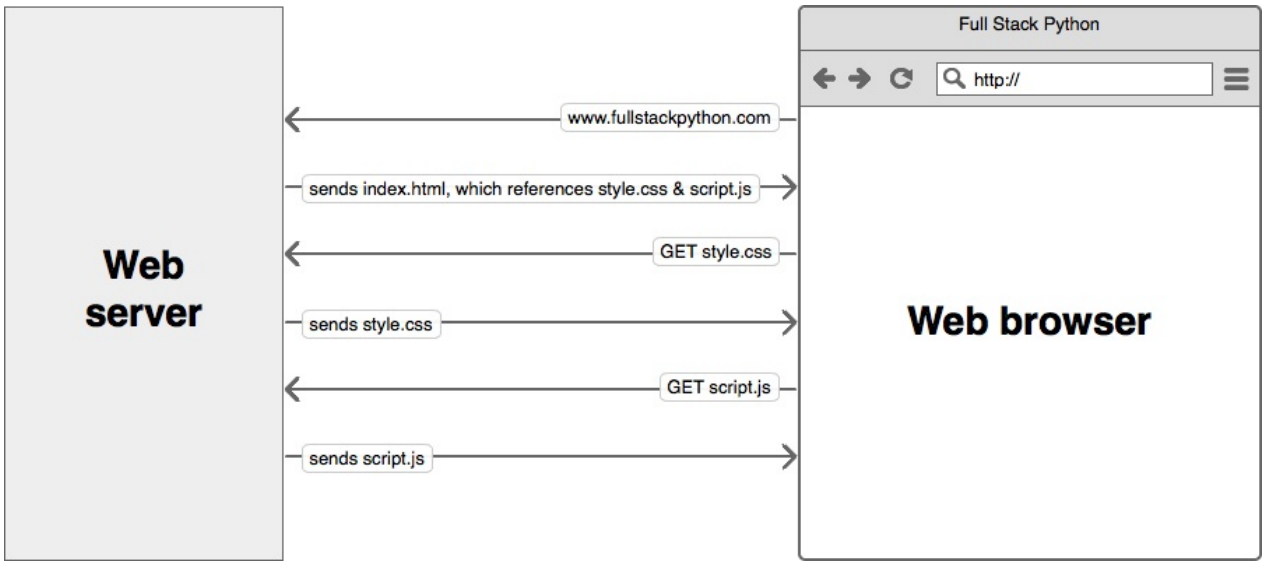
Client requests

A client that sends a request to a web server is usually a browser such as Internet Explorer, Firefox, or Chrome, but it can also be a

- headless browser, commonly use for testing, such as phantomjs
- commandline utility, for example wget and cURL
- text-based web browser such as Lynx
- web crawler.

Web servers process requests from the above clients. The result of the web server's processing is a response code and commonly a content response. Some status codes, such as 204 (No content) and 403 (Forbidden), do not have content responses.

In a simple case, the client will request a static asset such as a picture or JavaScript file. The file sits on the file system in a location the web server is authorized to access and the web server sends the file to the client with a 200 status code. If the client already requested the file and the file has not changed, the web server will pass back a 304 "Not modified" response indicating the client already has the latest version of that file.



A web server sends files to a web browser based on the web browser's request. In the first request, the browser accessed the "www.fullstackpython.com" address and the server responded with the index.html HTML-formatted file. That HTML file contained references to other files, such as style.css and script.js that the browser then requested from the server.

Sending static assets (such as CSS and JavaScript files) can eat up a large amount of bandwidth which is why using a Content Delivery Network (CDN) to serve static assets is important when possible.

Web server resources

- HTTP/1.1 Specification
- How to set up a safe and secure Web server
- A reference with the full list of HTTP status codes is provided by W3C.
- 4 HTTP Security Headers You Should Always Be Using
- If you're looking to learn about web servers by building one, here's part one, part two and part three of a great tutorial that shows how to code a web server in Python.
- rwasa is a newly released web server written in Assembly with no external dependencies that tuned to be faster than Nginx. The benchmarks are worth taking a look at to see if this server could fit your needs if you need the fastest performance trading off for as of yet untested web server.

Web servers learning checklist

1. Choose a web server. Nginx is often recommended although Apache is also a great choice.
2. Create an SSL certificate. For testing use a self-signed certificate and for a production app buy one from Digicert. Configure the web server to serve traffic over SSL. You'll need SSL for serving only HTTPS traffic and preventing security issues that occur with unencrypted user input.
3. Configure the web server to serve up static files such as CSS, JavaScript and images.
4. Once you set up the WSGI server you'll need to configure the web server as a pass through for dynamic content.

Apache HTTP Server

The Apache HTTP Server is a widely deployed web server that can be used in combination with a WSGI module, such as mod_wsgi or a stand-alone WSGI server to run Python web applications.

Why is the Apache HTTP Server important?

Apache remains the most commonly deployed web server with a reign of 20+ years. Its wide usage contributes to the large number of tutorials and open source modules created by developers for anyone to use.

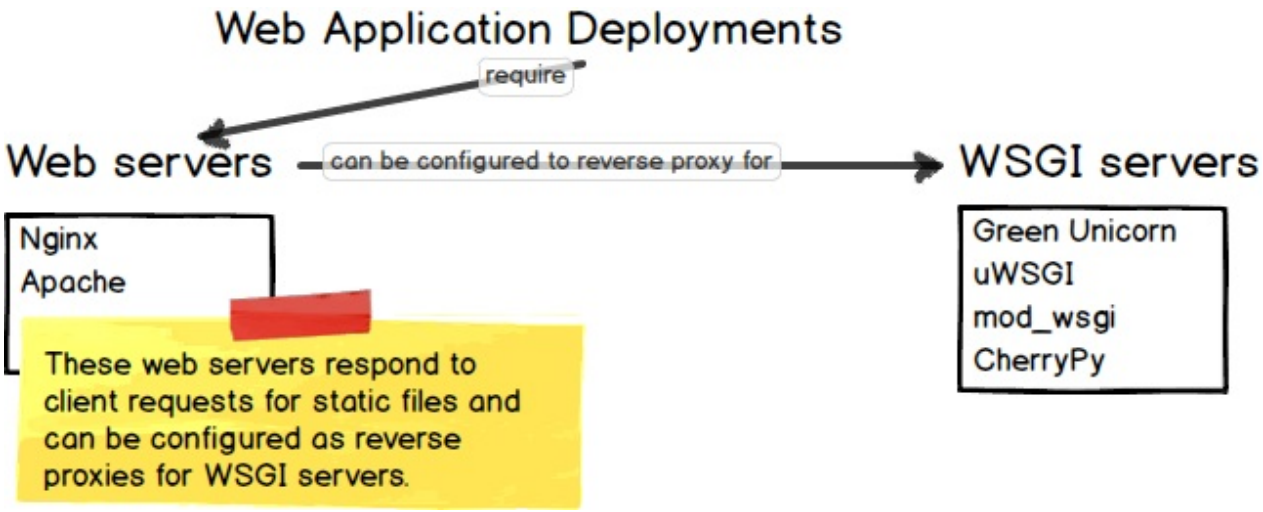
Apache's development began in mid-1994 as a fork of the NCSA HTTP Server project. By early 1996, Apache overtook the previously dominant but suddenly stagnant NCSA server as NCSA's progress stalled due to signficantly reduced development attention.

Apache HTTP Server resources

- The official project documentation page contains a section with How-Tos and Tutorials to handle authentication, security and dynamic content.
- Reverse proxies shows how to set up Apache as a reverse proxy using `mod_proxy`.
- Apache Web Server on Ubuntu 14.04 LTS explains how to install Apache on Ubuntu 14.04, which is still a supported release. Note however, do *not* install mod_python because it is now insecure and made obsolete by mod_wsgi and WSGI servers.
- Deploy Django on Apache with Virtualenv and mod_wsgi provides instructions for what packages to install to get Apache up and running with mod_wsgi on Ubuntu.
- Apache and mod_wsgi on Ubuntu 10.04 is an older post that shows how to set up Apache on the now out-of-support Ubuntu 10.04 LTS release. This setup isn't recommended in 2016 and beyond but if you are already using 10.04 as your base operating system you might need to reference this material.

Nginx

Nginx is the second most common web server among the top 100,000 websites. Nginx also functions well as a reverse proxy to handle requests and pass back responses for Python WSGI servers or even other web servers such as Apache.



Should I use Nginx or the Apache HTTP Server?

Let's be clear about these two "competing" servers: they are both fantastic open source projects and either will serve your web application deployment well. In fact, many of the top global web applications use both servers in their deployments to function in many steps throughout the HTTP request-response cycle.

I personally use Nginx more frequently than Apache because Nginx's configuration feel easier to write, with less boilerplate than alternatives.

There's also a bit of laziness in the usage: Nginx works well, it never causes me problems. So I stick with my battle-tested Ansible configuration management files that set up Nginx with HTTPS and SSL/TLS certificates

Nginx resources

- The Nginx chapter in the Architecture of Open Source Applications book has a great

chapter devoted to why Nginx is built to scale a certain way and lessons learned along the development journey.

- Inside Nginx: How we designed for performance and scale is a blog post from the developers behind Nginx on why they believe their architecture model is more performant and scalable than other approaches used to build web servers.

- Nginx web server tutorials are oldies but goodies on setting up previous versions of Nginx.

- Nginx for Developers: An Introduction

- An example of an Nginx security configuration.

- A faster Web server: ripping out Apache for Nginx explains how Nginx can be used instead of Apache in some cases for better performance.

- Nginx vs Apache: Our view is a first-party perspective written by the developers behind Nginx as to the differences between the web servers.

- Rate Limiting with Nginx covers how to mitigate against brute force password guessing attempts using Nginx rate limits.

- Nginx with dynamic upstreams is an important note for setting up your upstream WSGI server(s) if you're using Nginx as a reverse proxy with hostnames that change.

- Nginx Caching shows how to set up Nginx for caching HTTP requests, which is often done by Varnish but can also be handled by Nginx with the `proxy_cache` and related directives.

Caddy

Caddy is a relatively new HTTP server created in 2015 and written in Go. The server's philosophy and design emphasize HTTPS-everywhere along with the HTTP/2 protocol.

How can Caddy be used with Python deployments?

Caddy can be used both for testing during local development or as part of a production deployment as an HTTP server and a reverse proxy with the proxy directive.

General Caddy resources

- A look inside Caddy shows and explains some of the Go code written to build the server.
- The official Caddy server docs are the spot to look for what directives can be placed into a Caddy configuration file
- Caddy a modern web server supporting HTTP/2 is a quick synopsis on installing Caddy along with a short example configuration file.
- HTTP 2.0 on localhost with Caddy shows how to use a self-signed certificate with Caddy to do local development with an HTTP/2 web server.
- Is Caddy free? explains the donation and sponsorships model that Caddy uses to continue development on the server. The gist is that the server is free to clone, download and use. Sponsors and optional donations are currently used to fund ongoing development.

WSGI Servers

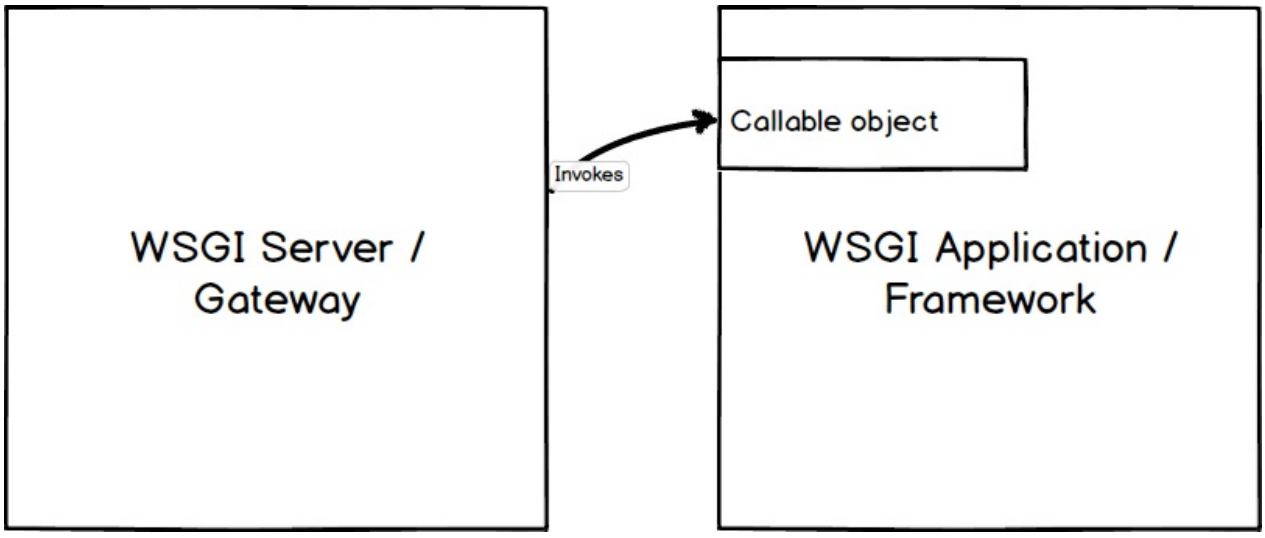
A Web Server Gateway Interface (WSGI) server implements the web server side of the WSGI interface for running Python web applications.

Why is WSGI necessary?

A traditional web server does not understand or have any way to run Python applications. In the late 1990s, a developer named Grisha Trubetskoy came up with an Apache module called mod_python to execute arbitrary Python code. For several years in the late 1990s and early 2000s, Apache configured with mod_python ran most Python web applications.

However, mod_python wasn't a standard specification. It was just an implementation that allowed Python code to run on a server. As mod_python's development stalled and security vulnerabilities were discovered there was recognition by the community that a consistent way to execute Python code for web applications was needed.

Therefore the Python community came up with WSGI as a standard interface that modules and containers could implement. WSGI is now the accepted approach for running Python web applications.



As shown in the above diagram, a WSGI server simply invokes a callable object on the WSGI application as defined by the PEP 3333 standard.

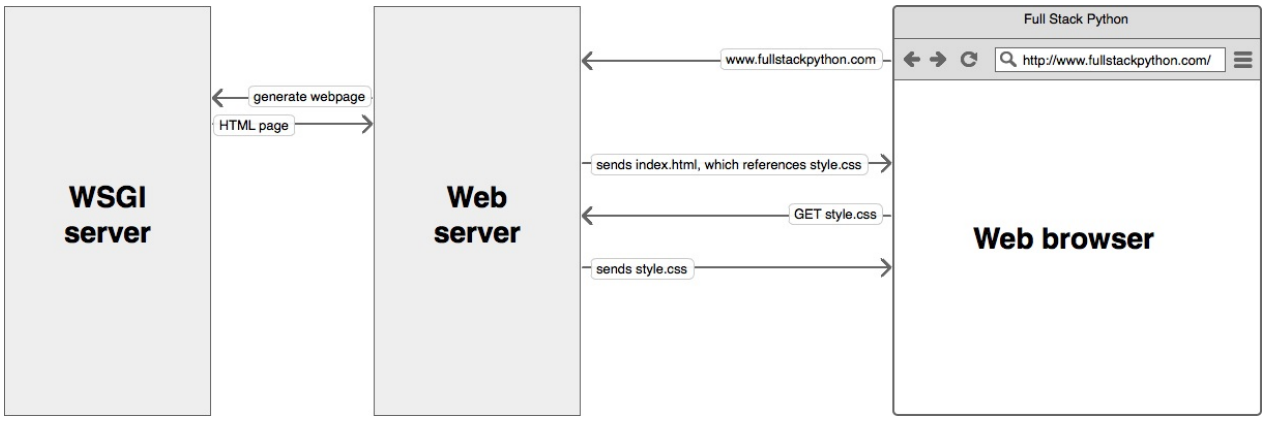
WSGI's Purpose

Why use WSGI and not just point a web server directly at an application?

- **WSGI gives you flexibility**. Application developers can swap out web stack components for others. For example, a developer can switch from Green Unicorn to uWSGI without modifying the application or framework that implements WSGI. From PEP 3333:

  The availability and widespread use of such an API in web servers for Python [...] would separate choice of framework from choice of web server, freeing users to choose a pairing that suits them, while freeing framework and server developers to focus on their preferred area of specialization.

- **WSGI servers promote scaling**. Serving thousands of requests for dynamic content at once is the domain of WSGI servers, not frameworks. WSGI servers handle processing requests from the web server and deciding how to communicate those requests to an application framework's process. The segregation of responsibilities is important for efficiently scaling web traffic.



WSGI is by design a simple standard interface for running Python code. As a web developer you won't need to know much more than

- what WSGI stands for (Web Server Gateway Inteface)

- that a WSGI container is a separate running process that runs on a different port than your web server

- your web server is configured to pass requests to the WSGI container which runs

your web application, then pass the response (in the form of HTML) back to the requester

If you're using a standard web framework such as Django, Flask, or Bottle, or almost any other current Python framework, you don't need to worry about how frameworks implement the application side of the WSGI standard. Likewise, if you're using a standard WSGI container such as Green Unicorn, uWSGI, mod_wsgi, or gevent, you can get them running without worrying about how they implement the WSGI standard.

However, knowing the WSGI standard and how these frameworks and containers implement WSGI should be on your learning checklist though as you become a more experienced Python web developer.

Official WSGI specifications

The WSGI standard v1.0 is specified in PEP 0333. As of September 2010, WSGI v1.0 is superseded by PEP 3333, which defines the v1.0.1 WSGI standard. If you're working with Python 2.x and you're compliant with PEP 0333, then you're also compliant with 3333. The newer version is simply an update for Python 3 and has instructions for how unicode should be handled.

wsgiref in Python 2.x and wsgiref in Python 3.x are the reference implementations of the WSGI specification built into Python's standard library so it can be used to build WSGI servers and applications.

Example web server configuration

A web server's configuration specifies what requests should be passed to the WSGI server to process. Once a request is processed and generated by the WSGI server, the response is passed back through the web server and onto the browser.

For example, this Nginx web server's configuration specifies that Nginx should handle static assets (such as images, JavaScript, and CSS files) under the /static directory and pass all other requests to the WSGI server running on port 8000:

```
# this specifies that there is a WSGI server running on port 8000
upstream app_server_djangoapp {
```

```
    server localhost:8000 fail_timeout=0;
}

# Nginx is set up to run on the standard HTTP port and listen for requests
server {
  listen 80;

  # nginx should serve up static files and never send to the WSGI server
  location /static {
    autoindex on;
    alias /srv/www/assets;
  }

  # requests that do not fall under /static are passed on to the WSGI
  # server that was specified above running on port 8000
  location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    if (!-f $request_filename) {
      proxy_pass http://app_server_djangoapp;
      break;
    }
  }
}
```

Note that the above code is a simplified version of a production-ready Nginx configuration. For real SSL and non-SSL templates, take a look at the Underwear web server templates on GitHub.

WSGI servers

There is a comprehensive list of WSGI servers on the WSGI Read the Docs page. The following are WSGI servers based on community recommendations.

- Green Unicorn is a pre-fork worker model based server ported from the Ruby Unicorn project.
- uWSGI is gaining steam as a highly-performant WSGI server implementation.
- mod_wsgi is an Apache module implementing the WSGI specification.
- CherryPy is a pure Python web server that also functions as a WSGI server.

WSGI resources

- PEP 0333 WSGI v1.0 and PEP 3333 WSGI v1.0.1 specifications.
- This basics of WSGI post contains a simple example of how a WSGI-compatible application works.
- A comparison of web servers for Python web apps is a good read to understand basic information about various WSGI server implementations.
- A thorough and informative post for LAMP-stack hosting choices is presented in the "complete single server Django stack tutorial."
- The Python community made a long effort to transition from mod_python to the WSGI standard. That transition period is now complete and an implementation of WSGI should always be used instead mod_python.
- Nicholas Piël wrote an interesting benchmark blog post of Python WSGI servers. Note that the post is a few years old. Benchmarks should be considered for their specific tested scenarios and not quickly extrapolated as general "this server is faster than this other server" results.
- How to Deploy Python WSGI Applications with CherryPy answers why CherryPy is a simple combination web and WSGI server along with how to use it.
- Another Digital Ocean walkthrough goes into How to Deploy Python WSGI Apps Using Gunicorn HTTP Server Behind Nginx.
- The uWSGI Swiss Army Knife shows how uWSGI can potentially be used for more than just running the Python web application - it can also serve static files and handle caching in a deployment.

WSGI servers learning checklist

1. Understand that WSGI is a standard Python specification for applications and servers to implement.
2. Pick a WSGI server based on available documentation and tutorials. Green Unicorn

is a good one to start with since it's been around for awhile.

3. Add the WSGI server to your server deployment.

4. Configure the web server to pass requests to the WSGI server for appropriate URL patterns.

5. Test that the WSGI server responds to local requests but not direct requests outside your infrastructure. The web server should be the pass through for requests to and responses from the WSGI server.

Source control

Source control, also known as *version control*, stores software code files with a detailed history of every modification made to those files.

Why is source control necessary?

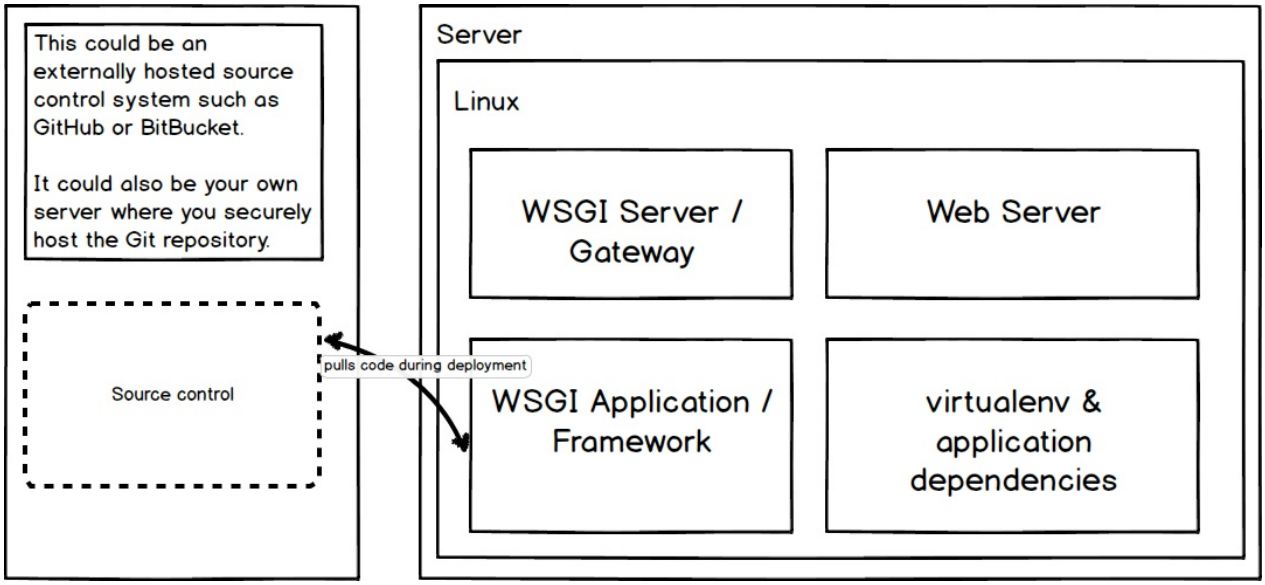
Version control systems allow developers to modify code without worrying about permanently screwing something up. Unwanted changes can be easily rolled back to previous working versions of the code.

Source control also makes team software development easier. One developer can combine her code modifications with other developers' code through diff views that show line-by-line changes then merge the appropriate code into the main code branch.

Version control is a necessity on all software projects regardless of development time, codebase size or the programming language used. Every project should immediately begin by using a version control system such as Git or Mercurial.

Source control during deployment

Pulling code during a deployment is a potential way source control systems fit into the deployment process.



Note that some developers recommend deployment pipelines package the source code to deploy it and never have a production environment touch a source control system directly. However, for small scale deployments it's often easiest to pull from source code when you're getting started instead of figuring out how to wrap the Python code in a system installation package.

Source control projects

Numerous source control systems have been created over the past several decades. In the past, proprietary source control software offered features tailored to large development teams and specific project workflows. However, open source systems are now used for version control on the largest and most complicated software projects in existence. There's no reason why your project should use anything other than an open source version control system in today's Python development world. The two primary choices are:

- Git is a free and open source distributed version control system.
- Mercurial is similar to Git, also a free and open source distributed version control system.

Hosted source control services

Git and Mercurial can be downloaded and run on your own server. However, it's easy

and cheap to get started with a hosted version control service. You can transition away from the service at a later time by moving your repositories if your needs change. A couple of recommended hosted version control services are:

- GitLab has both a self-hosted version of its open source software as well as their hosted version with pricing for businesses that need additional hosting support.
- GitHub provides free open source repositories and paid private repositories for Git.
- BitBucket also has free Git and Mercurial repositories for open projects, but adds private repositories for up to five users. Users pay for hosting private repositories with more than five users.

General source control resources

- Staging Servers, Source Control & Deploy Workflows, And Other Stuff Nobody Teaches You is a comprehensive overview by Patrick McKenzie of why you need source control.
- Version control best practices is a good write up of how to work with version control systems. The post is part of an ongoing deployment guide written by the folks at Rainforest.
- This lighthearted guide to the ten astonishments in version control history is a fun way to learn how systems developed over the past several decades.
- A visual guide to version control is a detailed article with real-life examples for why version control is necessary in software development.
- An introduction to version control shows the basic concepts behind version control systems.
- What Is Version Control? Why Is It Important For Due Diligence? explains the benefits and necessity of version control systems.
- About version control reviews the basics of distributed version control systems.

Git resources

- Pro Git is a free open source book that walks through all aspects of using the version control system.
- Git in Six Hundred Words is a clear and concise essay explaining the fundamental concepts of Git.
- A Hacker's Guide to Git covers the basics as well as more advanced Git commands while explaining each step along the way.
- A practical git introduction is exactly what the title says it is. This is a well written guide with plenty of code snippets to get you up to speed with Git.
- Git from the inside out demonstrates how Git's graph-based data structure produces certain behavior through example Git commands. This is a highly recommended read after you've grasped the basics and are looking to go deeper with Git.
- git ready has a nice collection of blog posts based on beginner, intermediate and advanced Git use cases.
- git-flow details a Git branching model for small teams.
- GitHub Flow builds on git-flow, goes over some of the issues that arise with it and presents a few solutions to those problems.
- Git Workflows That Work is a helpful post with diagrams to show how teams can create a Git workflow that will help their development process.
- "Our Git Workflow" by Braintree goes over how this payments company uses Git for development and merging source code.
- Code Sleuthing with Git shows how to review past changes when a deployment goes wrong to figure out what the heck happened.

Source control learning checklist

1. Pick a version control system. Git is recommended because on the web there are a significant number of tutorials to help both new and advanced users.
2. Learn basic use cases for version control such as committing changes, rolling back

to earlier file versions and searching for when lines of code were modified during development history.

3. Ensure your source code is backed up in a central repository. A central repository is critical not only if your local development version is corrupted but also for the deployment process.

4. Integrate source control into your deployment process in three ways. First, pull the project source code from version control during deployments. Second, kick off deployments when code is modified by using webhooks or polling on the repository. Third, ensure you can roll back to a previous version if a code deployment goes wrong.

Application Dependencies

Application dependencies are the libraries other than your project code that are required to create and run your application.

Why are application dependencies important?

Python web applications are built upon the work done by thousands of open source programmers. Application dependencies include not only web frameworks but also libraries for scraping, parsing, processing, analyzing, visualizing, and many other tasks. Python's ecosystem facilitates discovery, retrieval and installation so applications are easier for developers to create.

Finding libraries

Python libraries are stored in a central location known as the Python Package Index (PyPi). PyPi contains search functionality with results weighted by usage and relevance based on keyword terms.

Besides PyPi there are numerous resources that list common or "must-have" libraries. Ultimately the decision for which application dependencies are necessary for your project is up to you and the functionality you're looking to build. However, it's useful to browse through these lists in case you come across a library to solve a problem by reusing the code instead of writing it all yourself. A few of the best collections of Python libraries are

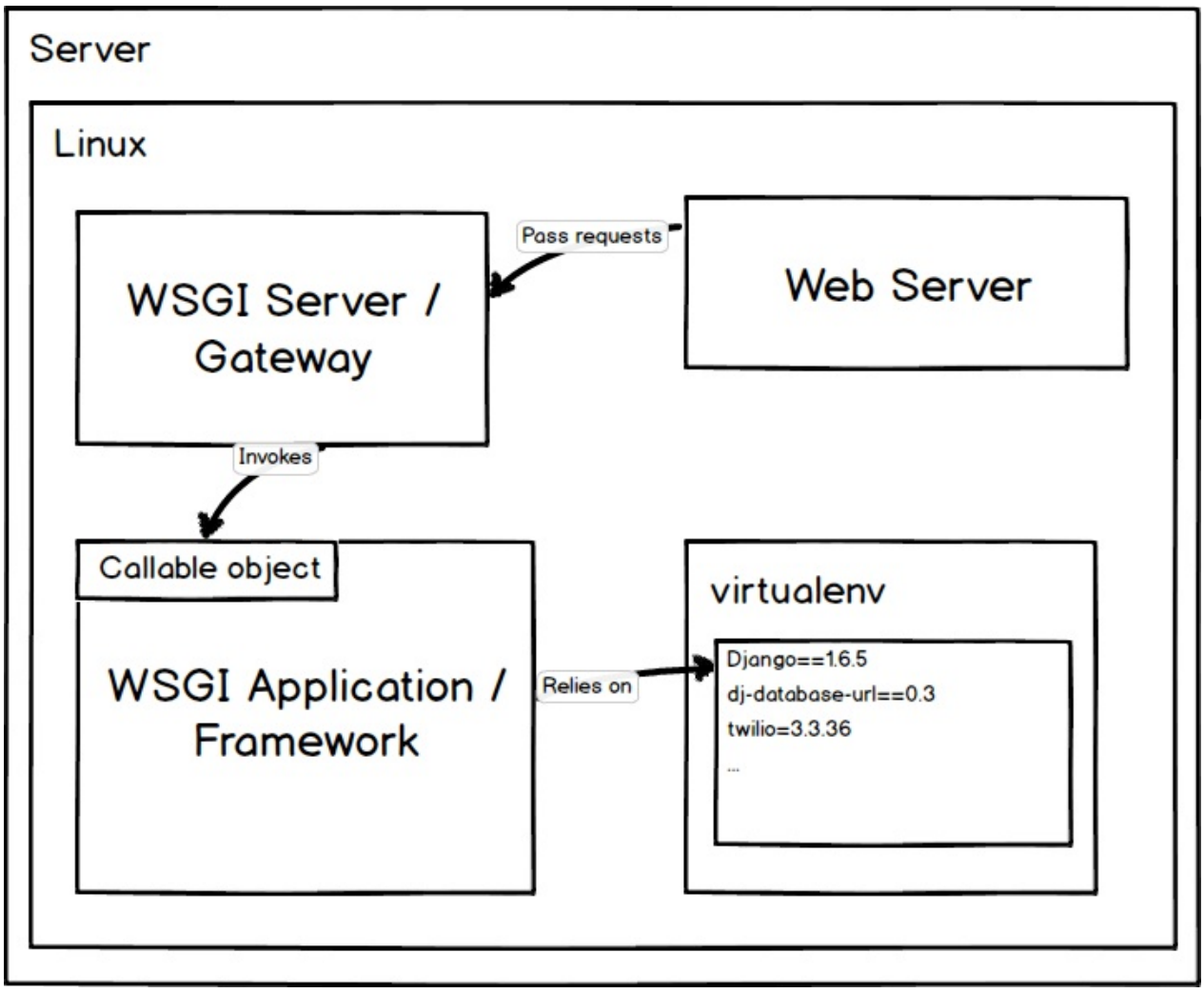
- Python.org's useful modules which groups modules into categories.
- GitHub Explore Trending repositories shows the open source Python projects trending today, this week, and this month.
- This list of 20 Python libraries you can't live without is a wide-ranging collection from data analysis to testing tools.

- Wikipedia actually has an extensive page dedicated to Python libraries grouped by categories.

Isolating dependencies

Dependencies are installed separately from system-level packages to prevent library version conflicts. The most common isolation method is virtualenv. Each virtualenv is its own copy of the Python interpreter and dependencies in the site-packages directory. To use a virtualenv it must first be created with the virtualenv command and then activated.

The virtualenv stores dependencies in an isolated environment. The web application then relies only on that virtualenv instance which has a separate copy of the Python interpreter and site-packages directory. A high level of how a server configured with virtualenv can look is shown in the picture below.



Installing Python dependencies

The recommended way to install Python library dependencies is with the pip command when a virtualenv is activated.

Pip and virtualenv work together and have complementary responsibilities. Pip downloads and installs application dependencies from the central PyPi repository.

requirements.txt

The pip convention for specifying application dependencies is with a `requirements.txt` file. When you build a Python web application you should include `requirements.txt` in the base directory of your project.

Python projects' dependencies for a web application should be specified with pegged dependencies like the following:

```
django==1.6
bpython==0.12
django-braces==0.2.1
django-model-utils==1.1.0
logutils==0.3.3
South==0.7.6
requests==1.2.0
stripe==1.9.1
dj-database-url==0.2.1
django-oauth2-provider==0.2.4
djangorestframework==2.3.1
```

Pegged dependencies with precise version numbers or Git tags are important because otherwise the latest version of a dependency will be used. While it may sound good to stay up to date, there's no telling if your application actually works with the latest versions of all dependencies. Developers should deliberately upgrade and test to make sure there were no backwards-incompatible modifications in newer dependency library versions.

setup.py

There is another type of dependency specification for Python libraries known as setup.py. Setup.py is a standard for distributing and installing Python libraries. If you're

building a Python library, such as requests or underwear you must include setup.py so a dependency manager can correctly install both the library as well as additional dependencies for the library. There's still quite a bit of confusion in the Python community over the difference between requirements.txt and setup.py, so read this well written post for further clarification.

Application dependency resources

- Jon Chu wrote a great introduction on virtualenv and pip basics.
- A non-magical introduction to virtualenv and pip breaks down what problems these tools solve and how to use them.
- Tools of the modern Python hacker contains detailed explanations of virtualenv, Fabric, and pip.
- Occasionally arguments about using Python's dependency manager versus one of Linux's dependency managers comes up. This provides one perspective on that debate.
- This Stack Overflow question details how to set up a virtual environment for Python development.
- Another Stack Overflow page answers how to set environment variables when using virtualenv.
- Tips for using pip + virtualenv + virtualenvwrapper shows how to use shell aliases and postactivate virtualenvwrapper hooks to make life easier when using these tools.
- Major speed improvements were made in pip 7 over previous versions. Read this article about the differences and be sure to upgrade.
- How to submit a package to PyPI is a short and sweet introduction that'll help you quickly get your first package on PyPI.

Open source app dependency projects

- Autoenv is a tool for activating environment variables stored in a `.env` file in your projects' home directories. Environment variables aren't managed by virtualenv and although virtualenvwrapper has some hooks for handling them, it's often easiest to use a shell script or `.env` file to set them in a development environment.

- Pipreqs searches through a project for dependencies based on imports. It then generates a `requirements.txt` file based on the libraries necessary to run those dependencies. Note though that while this could come in handy with a legacy project, the version numbers for those libraries will not be generated with the output.

Application dependencies learning checklist

1. Ensure the libraries your web application depends on are all captured in a requirement.txt file with pegged versions.

2. An easy way to capture currently installed dependencies is with the `pip freeze` command.

3. Create a fresh virtualenv and install the dependencies from your `requirements.txt` file by using the `pip install -r requirements.txt` command.

4. Check that your application runs properly with the fresh virtualenv and only the installed dependencies from the `requirements.txt` file.

Static content

Some content on a website does not change and therefore should be served up either directly through the web server or a content delivery network (CDN). Examples include JavaScript, image, and CSS files.

Types of static content

Static content can be either assets created as part of your development process such as images on your landing page or user-generated content. The Django framework calls these two categories *assets* and *media*.

Content delivery networks

A content delivery network (CDN) is a third party that stores and serves static files. Amazon CloudFront, Akamai, and Rackspace Cloud Files are examples of CDNs. The purpose of a CDN is to remove the load of static file requests from web servers that are handling dynamic web content. For example, if you have an nginx server that handles both static files and acts as a front for a Green Unicorn WSGI server on a 512 megabyte virtual private server, the nginx server will run into resource constraints under heavy traffic. A CDN can remove the need to serve static assets from that nginx server so it can purely act as a pass through for requests to the Green Unicorn WSGI server.

CDNs send content responses from data centers with the closest proximity to the requester.

Static Content Resources

- The super stupid idiot's guide to getting started with Django, Pipeline, and S3 shows how to host static content on S3 and use those files with Django.
- Crushing, caching and CDN deployment in Django shows how to use django-compressor and a CDN to scale static and media file serving.

- Uploading with Django and Amazon S3 walks through each step in getting buckets set up so you can upload files to them via Django.
- Using Amazon S3 to host your Django static files
- CDNs fail, but your scripts don't have to
- django-storages is a Django library for managing static and media files on services such as Amazon S3 and other content delivery networks.
- RevSys has a nice article on a range of important static file optimizations such as setting cache headers, optimizing JavaScript and reducing the size of images.
- Twelve folks with significant experience working on and with CDNs provide their perspectives in this piece: CDN experts on CDNs.

Static content learning checklist

1. Identify a content delivery network to offload serving static content files from your local web server. I recommend using Amazon S3 with CloudFront as it's easy to set up and will scale to high bandwidth demands.
2. Update your web application deployment process so updated static files are uploaded to the CDN.
3. Move static content serving from the www subdomain to a static (or similarly named) subdomain so browsers will load static content in parallel to www HTTP requests.

Task queues

Task queues manage background work that must be executed outside the usual HTTP request-response cycle.

Why are task queues necessary?

Tasks are handled asynchronously either because they are not initiated by an HTTP request or because they are long-running jobs that would dramatically reduce the performance of an HTTP response.

For example, a web application could poll the GitHub API every 10 minutes to collect the names of the top 100 starred repositories. A task queue would handle invoking code to call the GitHub API, process the results and store them in a persistent database for later use.

Another example is when a database query would take too long during the HTTP request-response cycle. The query could be performed in the background on a fixed interval with the results stored in the database. When an HTTP request comes in that needs those results a query would simply fetch the precalculated result instead of re-executing the longer query. This precalculation scenario is a form of caching enabled by task queues.

Other types of jobs for task queues include

- spreading out large numbers of independent database inserts over time instead of inserting everything at once
- aggregating collected data values on a fixed interval, such as every 15 minutes
- scheduling periodic jobs such as batch processes

Task queue projects

The defacto standard Python task queue is Celery. The other task queue projects that

arise tend to come from the perspective that Celery is overly complicated for simple use cases. My recommendation is to put the effort into Celery's reasonable learning curve as it is worth the time it takes to understand how to use the project.

- The Celery distributed task queue is the most commonly used Python library for handling asynchronous tasks and scheduling.

- The RQ (Redis Queue) is a simple Python library for queueing jobs and processing them in the background with workers. RQ is backed by Redis and is designed to have a low barrier to entry. The intro post contains information on design decisions and how to use RQ.

- Taskmaster is a lightweight simple distributed queue for handling large volumes of one-off tasks.

- Huey is a simple task queue that uses Redis on the backend but otherwise does not depend on other libraries. The project was previously known as Invoker and the author changed the name.

- Huey is a Redis-based task queue that aims to provide a simple, yet flexible framework for executing tasks. Huey supports task scheduling, crontab-like repeating tasks, result storage and automatic retry in the event of failure.

Hosted message and task queue services

Task queue third party services aim to solve the complexity issues that arise when scaling out a large deployment of distributed task queues.

- Iron.io is a distributed messaging service platform that works with many types of task queues such as Celery. It also is built to work with other IaaS and PaaS environments such as Amazon Web Services and Heroku.

- Amazon Simple Queue Service (SQS) is a set of five APIs for creating, sending, receiving, modifying and deleting messages.

- CloudAMQP is at its core managed servers with RabbitMQ installed and configured. This service is an option if you are using RabbitMQ and do not want to maintain RabbitMQ installations on your own servers.

Open source examples that use task queues

- Take a look at the code in this open source Flask application and this Django application for examples of how to use and deploy Celery with a Redis broker to send text messages with these frameworks.

- flask-celery-example is a simple Flask application with Celery as a task queue and Redis as the broker.

Task queue resources

- Getting Started Scheduling Tasks with Celery is a detailed walkthrough for setting up Celery with Django (although Celery can also be used without a problem with other frameworks).

- Distributing work without Celery provides a scenario in which Celery and RabbitMQ are not the right tool for scheduling asynchronous jobs.

- International Space Station notifications with Python and Redis Queue (RQ) shows how to combine the RQ task queue library with Flask to send text message notifications every time a condition is met - in this blog post's case that the ISS is currently flying over your location on Earth.

- Evaluating persistent, replicated message queues is a detailed comparison of Amazon SQS, MongoDB, RabbitMQ, HornetQ and Kafka's designs and performance.

- Queues.io is a collection of task queue systems with short summaries for each one. The task queues are not all compatible with Python but ones that work with it are tagged with the "Python" keyword.

- Why Task Queues is a presentation for what task queues are and why they are needed.

- Flask by Example Implementing a Redis Task Queue provides a detailed walkthrough of setting up workers to use RQ with Redis.

- How to use Celery with RabbitMQ is a detailed walkthrough for using these tools on

an Ubuntu VPS.

- Heroku has a clear walkthrough for using RQ for background tasks.
- Introducing Celery for Python+Django provides an introduction to the Celery task queue.
- Celery - Best Practices explains things you should not do with Celery and shows some underused features for making task queues easier to work with.
- The "Django in Production" series by Rob Golding contains a post specifically on Background Tasks.
- Asynchronous Processing in Web Applications Part One and Part Two are great reads for understanding the difference between a task queue and why you shouldn't use your database as one.
- Celery in Production on the Caktus Group blog contains good practices from their experience using Celery with RabbitMQ, monitoring tools and other aspects not often discussed in existing documentation.
- A 4 Minute Intro to Celery is a short introductory task queue screencast.
- Heroku wrote about how to secure Celery when tasks are otherwise sent over unencrypted networks.
- Miguel Grinberg wrote a nice post on using the task queue Celery with Flask. He gives an overview of Celery followed by specific code to set up the task queue and integrate it with Flask.
- 3 Gotchas for Working with Celery are things to keep in mind when you're new to the Celery task queue implementation.
- Deferred Tasks and Scheduled Jobs with Celery 3.1, Django 1.7 and Redis is a video along with code that shows how to set up Celery with Redis as the broker in a Django application.
- Setting up an asynchronous task queue for Django using Celery and Redis is a straightforward tutorial for setting up the Celery task queue for Django web applications using the Redis broker on the back end.

- Background jobs with Django and Celery shows the code and a simple explanation of how to use Celery with Django.
- Asynchronous Tasks With Django and Celery shows how to integrate Celery with Django and create Periodic Tasks.
- Three quick tips from two years with Celery provides some solid advice on retry delays, the -Ofair flag and global task timeouts for Celery.

Task queue learning checklist

1. Pick a slow function in your project that is called during an HTTP request.
2. Determine if you can precompute the results on a fixed interval instead of during the HTTP request. If so, create a separate function you can call from elsewhere then store the precomputed value in the database.
3. Read the Celery documentation and the links in the resources section below to understand how the project works.
4. Install a message broker such as RabbitMQ or Redis and then add Celery to your project. Configure Celery to work with the installed message broker.
5. Use Celery to invoke the function from step one on a regular basis.
6. Have the HTTP request function use the precomputed value instead of the slow running code it originally relied upon.

Configuration Management

Configuration management involves modifying servers from an existing state to a desired state and automating how an application is deployed.

Configuration management tools

Numerous tools exist to modify server state in a controlled way, including Puppet, Chef, SaltStack, and Ansible. Puppet and Chef are written in Ruby, while SaltStack and Ansible are written in Python.

Ad hoc tasks

Configuration management tools such as Chef, Puppet, Ansible, and SaltStack are not useful for performing ad hoc tasks that require interactive responses. Fabric and Invoke are used for interactive operations, such as querying the database from the Django manage.py shell.

Configuration management tool comparisons

- Moving away from Puppet: SaltStack or Ansible? is an openly biased but detailed post on why to choose SaltStack over Ansible in certain situations.
- Ansible vs. Shell Scripts provides some perspective on why a configuration management tool is better than old venerable shell scripts.
- Ansible vs. Chef is a comparsion of Ansible with the Chef configuration management tool.
- This post on Ansible and Salt: A Detailed Comparison shows the differences between these two Python-powered tools.

Ansible

Ansible is an open source configuration management and application deployment tool built in Python.

Ansible Resources

- An Ansible tutorial is a fantastically detailed introduction to using Ansible to set up servers.
- Ansible Text Message Notifications with Twilio SMS is my blog post with a detailed example for using the Twilio module in core Ansible 1.6+.
- Python for Configuration Management with Ansible slides from PyCon UK 2013
- First Steps with Ansible
- Red Badger on Ansible
- Getting Started with Ansible
- An introduction to Ansible is a tutorial on the basics of getting started with the tool.
- Ansible and Linode
- Multi-factor SSH authentication with Ansible and Duo Security
- Introducing Ansible into Legacy Projects
- Automating your development environment with Ansible
- Post-install steps with Ansible
- First Five (and a half) Minutes on a Server with Ansible
- 6 practices for super smooth Ansible experience
- Shippable + Ansible + Docker + Loggly for awesome deployments is a well written detailed post about using Docker and Ansible together with a few other pieces.

- Create a Couchbase Cluster with Ansible
- Idempotence, convergence, and other silly fancy words we often use
- How to Write an Ansible Role for Ansible Galaxy
- Testing with Jenkins, Docker and Ansible

Application dependencies learning checklist

1. Learn about configuration management in the context of deployment automation and infrastructure-as-code.
2. Pick a configuration management tool and stick with it. My recommendation is Ansible because it is by far the easiest tool to learn and use.
3. Read your configuration management tool's documentation and, when necessary, the source code.
4. Automate the configuration management and deployment for your project. Note that this is by far the most time consuming step in this checklist but will pay dividends every time you deploy your project.
5. Hook the automated deployment tool into your existing deployment process.

Continuous Integration

Continuous integration automates the building, testing and deploying of applications. Software projects, whether created by a single individual or entire teams, typically use continuous integration as a hub to ensure important steps such as unit testing are automated rather than manual processes.

Why is continuous integration important?

When continuous integration (CI) is established as a step in a software project's development process it can dramatically reduce deployment times by minimizing steps that require human intervention. The only minor downside to using CI is that it takes some initial time by a developer to set up and then there is some ongoing maintainence if a project is broken into multiple parts, such as going from a monolith architecture to microservices.

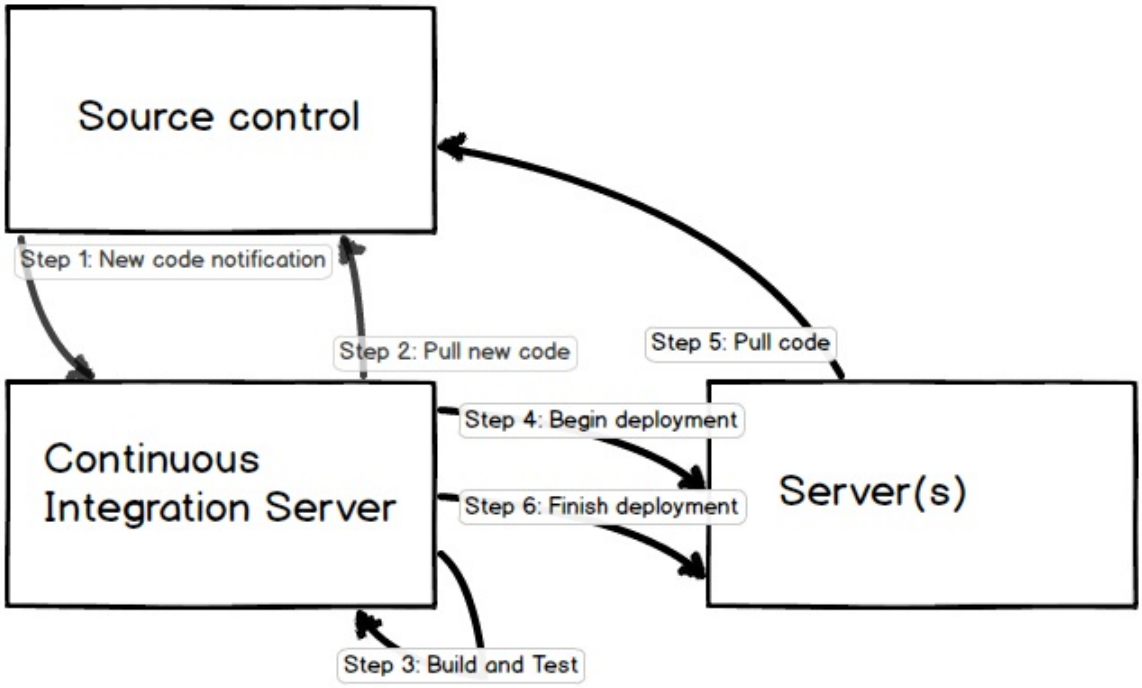
Automated testing

Another major advantage with CI is that testing can be an automated step in the deployment process. Broken deployments can be prevented by running a comprehensive test suite of unit and integration tests when developers check in code to a source code repository. Any bugs accidentally introduced during a check-in that are caught by the test suite are reported and prevent the deployment from proceeding.

The automated testing on checked in source code can be thought of like the bumper guards in bowling that prevent code quality from going too far off track. CI combined with unit and integration tests check that any code modifications do not break existing tests to ensure the software works as intended.

Continuous integration example

The following picture represents a high level perspective on how continuous integration and deployment can work.



In the above diagram, when new code is committed to a source repository there is a hook that notifies the continuous integration server that new code needs to be built (the continuous integration server could also poll the source code repository if a notification is not possible).

The continuous integration server pulls the code to build and test it. If all tests pass, the continuous integration server begins the deployment process. The new code is pulled down to the server where the deployment is taking place. Finally the deployment process is completed via restarting services and related deployment activities.

There are many other ways a continuous integration server and its deployments can be structured. The above was just one example of a relatively simple set up.

Open source CI projects

There are a variety of free and open source continuous integration servers that are configurable based on a project's needs.

Note that many of these servers are not written in Python but work just fine for Python applications. Polyglot organizations (ones that use more than a single language and ecosystem) often use a single CI server for all of their projects regardless of the

programming language the application was written in.

- Jenkins is a common CI server for building and deploying to test and production servers. Jenkins source code is on GitHub.
- Go CD is a CI server by ThoughtWorks that was designed with best practices for the build and test & release cycles in mind. Go CD source code is on GitHub.
- Strider is a CI server written in node.js. Strider source code is on GitHub.
- BuildBot is a continuous integration **framework** with a set of components for creating your own CI server. It's written in Python and intended for development teams that want more control over their build and deployment pipeline. BuildBot source code is on GitHub.
- TeamCity is JetBrains' closed source CI server that requires a license to use.

Jenkins CI resources

- Assembling a continuous integration service for a Django project on Jenkins shows how to set up a Ubuntu instance with a Jenkins server that'll build a Django project.
- My book on deploying Python web applications walks through every step of setting up a Jenkins project with a WSGI application to enable continuous delivery. Take a look if you're not grokking all of the steps provided in these other blog posts.
- Setting up Jenkins as a continuous integration server for Django is another solid tutorial that also shows how to send email notifications as part of the build process.
- If you're running into difficulty adding an SSH key to your Jenkins system account so you can connect to another server or Git repository this blog post on connecting Jenkins with Git to get the steps to solve that problem.
- Automated Servers and Deployments with Ansible & Jenkins
- Running Jenkins in Docker Containers is a short tutorial showing how to use the official Jenkins container on the Docker hub.
- Securing Jenkins is the landing page for Jenkins security. If you're deploying your

own instance, you'll need to lock it down against unauthorized users.

- Can we use Jenkins for that? looks at how one team uses Jenkins for more than typical continuous integration situations - they also use it as an administrative interface, cron jobs, data analytics pipelines and long-running scripts.

General continuous integration resources

- What is continuous integration? is a classic detailed article by Martin Fowler on the concepts behind CI and how to implement it.

- Continuous Deployment For Practical People is not specific to Python but a great read on what it entails.

- Continuous Integration & Delivery - Illustrated uses well done drawings to show how continuous integration and delivery works for testing and managing data.

- Diving into continuous integration as a newbie is a retrospective on learning CI from a Rackspace intern on how she learned the subject.

- StackShare's Continuous Integration tag lists a slew of hosted CI services roughly ranked by user upvotes.

- Good practices for continuous integration includes advice on checking in code, commit tests and reverting to previous revisions.

- Deploying to AWS using Ansible, Docker and Teamcity is an example walking through one potential way to use the Teamcity CI server for automated deployments.

Logging

Logging saves output such as errors, warnings and event information to persistent storage for debugging purposes.

Why is logging important?

Runtime exceptions that prevent code from running are important to log to investigate and fix the source of the problems. Informational and debugging logging also helps to understand how the application is performing even if code is working as intended.

Logging levels

Logging is often grouped into several categories:

1. Information
2. Debug
3. Warning
4. Error

Logging errors that occur while a web framework is running is crucial to understanding how your application is performing.

Logging aggregators

When you are running your application on several servers, it is helpful to have a monitoring tool called a "logging aggregator". You can configure your application to forward your system and application logs to one location that provides tools for viewing, searching, and monitoring logging events across your cluster.

Another advantage of log aggregation tools is they allow you to set up custom alerts and alarms so you can get notified when error rates breach a certain threshold.

Open source log aggregators

- Sentry started as a Django-only exception handling service but now has separate logging clients to cover almost all major languages and frameworks. It still works really well for Python-powered web applications and is often used in conjunction with other monitoring tools. Raven is open source Python client for Sentry.

- Graylog2 provides a central server for log aggregation as well as a GUI for browsing and searching through log events. There are libraries for most major languages, including python. Saves data in Elasticache.

- Logstash Similar to Graylog2, logstash offers features to programmatically configure log data workflows.

- Scribe A project written by Facebook to aggregate logs. It's designed to run on multiple servers and scale with the rest of your cluster. Uses the Thrift messaging format so it can be used with any language.

Hosted logging services

- Loggly is a third party cloud based application that aggregates logs. They have instructions for every major language, including python. It includes email alerting on custom searches.

- Splunk offers third party cloud and self hosted solutions for event aggregation. It excels at searching and data mining any text based data.

- Papertrail is similar to both Loggly and Splunk and provides integration with S3 for long term storage.

- Raygun logs errors and provides immediate notification when issues arise.

- Scalyr provides log aggregation, dashboards, alerts and search in a user interface on top of standard logs.

- There is a hosted version of Sentry in case you do not have the time to set up the open source project yourself.

Logging resources

- This intro to logging presents the Python logging module and how to use it.
- Logging as Storytelling is a multi-part series working the analogy that logs should read like a story so you can better understand what's taking place in your web application. Part 2 describes actions and part 3 talks about types.
- A Brief Digression About Logging is a short post that gets Python logging up and running quickly.
- Taking the pain out of Python logging shows a logging set up with uWSGI.
- Django's 1.3 release brought unified logging into project configurations. This post shows how to set up logging in a project's settings.py file. Caktus Group also has a nice tutorial on central logging with graypy and Graylog2.
- Django Logging Configuration: How the Default Settings Interfere with Yours explains a problem with the default Django logging configuration and what to do about in your project.
- Exceptional Logging of Exceptions in Python shows how to log errors more accurately to pinpoint the problem instead of receiving generic exceptions in your logs.

Logging learning checklist

1. Read how to integrate logging into your web application framework.
2. Ensure errors and anomalous results are logged. While these logs can be stored in monitoring solutions, it's best to have your own log storage location to debug issues as they arise to complement other monitoring systems.
3. Integrate logging for system events you may need to use for debugging purposes later. For example, you may want to know the return values on functions when they are above a certain threshold.

Monitoring

Monitoring tools capture, analyze and display information for a web application's execution. Every application has issues arise throughout all levels of the web stack. Monitoring tools provide transparency so developers and operations teams can respond and fix problems.

Why is monitoring necessary?

Capturing and analyzing data about your production environment is critical to proactively deal with stability, performance, and errors in a web application.

Difference between monitoring and logging

Monitoring and logging are very similar in their purpose of helping to diagnose issues with an application and aid the debugging process. One way to think about the difference is that logging happens based on explicit events while monitoring is a passive background collection of data.

For example, when an error occurs, that event is explicitly logged through code in an exception handler. Meanwhile, a monitoring agent instruments the code and gathers data not only about the logged exception but also the performance of the functions.

This distinction between logging and monitoring is vague and not necessarily the only way to look at it. Pragmatically, both are useful for maintaining a production web application.

Monitoring layers

There are several important resources to monitor on the operating system and network level of a web stack.

1. CPU utilization
2. Memory utilization

3. Persistence storage consumed versus free
4. Network bandwidth and latency

Application level monitoring encompasses several aspects. The amount of time and resources dedicated to each aspect will vary based on whether an application is read-heavy, write-heavy, or subject to rapid swings in traffic.

1. Application warnings and errors (500-level HTTP errors)
2. Application code performance
3. Template rendering time
4. Browser rendering time for the application
5. Database querying performance

Open source monitoring projects

- statsd is a node.js network daemon that listens for metrics and aggregates them for transfer into another service such as Graphite.

- Graphite stores time-series data and displays them in graphs through a Django web application.

- Bucky measures the performance of a web application from end user's browsers and sends that data back to the server for collection.

- Sensu is an open source monitoring framework written in Ruby but applicable to any programming language web application.

- Graph Explorer by Vimeo is a Graphite-based dashboard with added features and a slick design.

- PacketBeat sniffs protocol packets. Elasticsearch then allows developers to search the collected data and visualize what's happening inside their web application using the Kibana user interface.

- Munin is a client plugin-based monitoring system that sends monitoring traffic to the Munin node where the data can be analyzed and visualized. Note this project is written in Perl so Perl 5 must be installed on the node collecting the data.

Hosted monitoring services

- New Relic. Application and database monitoring as well as plug ins for capturing and analyzing additional data about tools in your stack.
- CopperEgg is lower-level monitoring on server and infrastructure. It's popular with DevOps shops that are making changes to their production environments and want immediate feedback on the results of those modifications.
- Status.io focuses on uptime and response metrics transparency for web applications.
- StatusPage.io (yes, there's both a Status and StatusPage.io) provides easy set up status pages for monitoring application up time.
- PagerDuty alerts a designated person or group if there are stability, performance, or uptime issues with an application.
- App Enlight provides performance, exception and error monitoring and is currently specific to Python web applications.

Monitoring resources

- The Virtues of Monitoring
- Effortless Monitoring with collectd, Graphite, and Docker
- Practical Guide to StatsD/Graphite Monitoring is a detailed guide with code examples for monitoring infrastructure.
- Bit.ly describes the "10 Things They Forgot to Monitor" beyond the standard metrics such as disk & memory usage.
- Four Linux server monitoring tools
- How to design useful monitoring and graphing visualizations
- 5 years of metrics and monitoring is a great presentation highlighting that

visualization so humans can understand measurements is a hard problem. Line graphs are often not the best solution and they are overused.

- The Collector Highlight Series has an article on StatsD that explains how to install it and how it works.
- This survey on monitoring tools has some nice data and graphs on what developers and operations folks use in their environments.
- Ryan Frantz wrote a nice post on Solving Monitoring with a new definition of what monitoring means based on today's complex systems and how the practice should evolve going forward.

Monitoring learning checklist

1. Review the software-as-a-service and open source monitoring tools below. Third party services tend to be easier to set up and host the data for you. Open source projects give you more control but you'll need to have additional servers ready for the monitoring.
2. My recommendation is to install New Relic's free option with the trial period to see how it works with your app. It'll give you a good idea of the capabilities for application-level monitoring tools.
3. As your app scales take a look at setting up one of the the open source monitoring projects such as StatsD with Graphite. The combination of those two projects will give you fine-grained control over the system metrics you're collecting and visualizing.

Web analytics

Web analytics involves collecting, processing, visualizing web data to enable critical thinking about how users interact with a web application.

Why is web analytics important?

User clients, especially web browsers, generate significant data while users read and interact with webpages. The data provides insight into how visitors use the site and why they stay or leave. The key concept to analytics is *learning* about your users so you can improve your web application to better suit their needs.

Web analytics concepts

It's easy to get overwhelmed at both the number of analytics services and the numerous types of data points collected. Focus on just a handful of metrics when you're just starting out. As your application scales and you understand more about your users add additional analytics services to gain further insight into their behavior with advanced visualizations such as heatmaps and action funnels.

User funnels

If your application is selling a product or service you can ultimately build a user funnel (often called "sales funnel" prior to a user becoming a customer) to better understand why people buy or don't buy what you're selling. With a funnel you can visualize drop-off points where visitors leave your application before taking some action, such as purchasing your service.

Open source web analytics projects

- Piwik is a web analytics platform you can host yourself. Piwik is a solid choice if you cannot use Google Analytics or want to customize your own web analytics platform.

- Open Web Analytics is another self-hosted platform that integrates through a

JavaScript snippet that tracks users' interactions with the webpage.

Hosted web analytics services

- Google Analytics is a widely used free analytics tool for website traffic.
- Clicky provides real-time analytics comparable to Google Analytics' real-time dashboard.
- MixPanel's analytics platform focuses on mobile and sales funnel metrics. A developer builds what data points need to be collected into the server side or client side code. MixPanel captures that data and provides metrics and visualizations based on the data.
- KISSmetrics' analytics provides context for who is visiting a website and what actions they are taking while on the site.
- Heap is a recently founded analytics service with a free introductory tier to get started.
- CrazyEgg is tool for understanding a user's focus while using a website based on heatmaps generated from mouse movements.

Python-specific web analytics resources

- Building an Analytics App with Flask is a detailed walkthrough for collecting and analyzing webpage analytics with your own Flask app.
- Pandas and Google Analytics shows how to use pandas for data analysis with Google Analytics' API to perform calculations not available in the tool itself.
- Build your own Google Analytics Dashboard in Excel show how to extract your Google Analytics data via their web API and Python helper library so it can be used in other tools such as Excel.

General web analytics resources

- Google Analytics for Developers
- This beginner's guide to math and stats behind web analytics provides some context for understanding and reasoning about web traffic.
- An Analytics Primer for Developers by Mozilla explains what to track, choosing an analytics platform and how to serve up the analytics JavaScript asynchronously.
- This post provides context for determining if a given metric is "vanity" or actionable.

Web analytics learning checklist

1. Add Google Analytics or Piwik to your application. Both are free and while Piwik is not as powerful as Google Analytics you can self-host the application which is the only option in many environments.
2. Think critically about the factors that will make your application successful. These factors will vary based on whether it's an internal enterprise app, an e-commerce site or an information-based application.
3. Add metrics generated from your web traffic based on the factors that drive your application's success. You can add these metrics with either some custom code or with a hosted web analytics service.
4. Continuously reevaluate whether the metrics you've chosen are still the appropriate ones defining your application's success. Improve and refine the metrics generated by the web analytics as necessary.

Docker

Docker is an open source infrastructure management platform for running and deploying software. The Docker platform is constantly evolving so an exact definition is currently a moving target.

Why is Docker important?

Docker can package up applications along with their necessary operating system dependencies for easier deployment across environments. In the long run it has the potential to be the abstraction layer that easily manages containers running on top of any type of server, regardless of whether that server is on Amazon Web Services, Google Compute Engine, Linode, Rackspace or elsewhere.

Python projects within Docker images

- This Docker image contains a Flask application configured to run with uWSGI and Nginx. You can also see the image on Docker hub.
- minimal-docker-python-setup contains an image with Nginx, uWSGI, Redis and Flask.

Docker resources

- What is Docker and When to Use It clearly delineates what Docker is and what it isn't. This is a good article for when you're first wrapping your head around Docker conceptually.
- Andrew Baker presented a fantastic tutorial at PyOhio on beginner and advanced Docker usage. Andrew also wrote the article what containers can do for you and created the O'Reilly Introduction to Docker video that's well worth buying.
- Docker curriculum is a detailed tutorial created by a developer to show the exact steps for deploying an application that relies on Elasticsearch.

- How to install Docker and get started provides a walkthrough for Ubuntu 13.04 for installing and beginning to use Docker for development.
- It Really is the Future discusses Docker and containers in the context of whether it's all just a bunch of hype or if this is a real trend for infrastructure automation. This is a great read to set the context for why these tools are important.
- Docker Jumpstart is a comprehensive introduction to what Docker is and how to get started with using the tool.
- If you want to quickly use Docker on Mac OS X, check out these concise instructions for setting up your Docker workflow on OS X in 60 seconds.
- What the Bleep is Docker? is a plain English explanation with examples of what Docker provides and what it can be used for in your environment.
- Docker in Practice - A Guide for Engineers is an explanation of the concepts and philosophy by the authors of the new Manning Docker book in early access format.
- Eight Docker Development Patterns shares lessons learned and explains how to work with the containers so you get more use out of them during development.
- The marriage of Ansible and Docker is a detailed look at how Docker and Ansible complement each other as deployment tools.
- Building Docker containers from scratch is a short tutorial for creating a Docker container with a specific configuration.
- 10 things to avoid in Docker containers provides a lot of "don'ts" that you'll want to consider before bumping up against the limitations of how containers should be used.

Python-specific Docker resources

- Hosting Python WSGI applications using Docker shows how to use Docker in WSGI application deployments specifically using mod_wsgi.
- How to Containerize Python Web Applications is an extensive tutorial that uses a Flask application and deploys it using a Docker container.

- The Docker is awesome miniseries explains how to get a Django + AngularJS application running under Docker. Part 2 continues the tutorial.
- Docker in Action - Fitter, Happier, More Productive is a killer tutorial that shows how to combine Docker with CircleCI to continuously deploy a Flask application.
- Deploying Django Applications in Docker explains some of the concepts behind using Docker for Python deployments and shows how to specifically use it for deploying Django.
- A Docker primer – from zero to a running Django app provides specific commands and expected output for running Django apps with Docker and Vagrant.
- Using Docker and Docker Compose to replace virtualenv is a tutorial for using Docker instead of virtualenv for dependency isolation.
- Lincoln Loop wrote up a closer look at Docker from the perspective of Python developers handling deployments.
- Curious how pip and Docker can be used together? Read this article on Efficient management Python projects dependencies with Docker.
- Python virtual environments and Docker goes into detail on whether virtual environments should be used with Docker and how system packages can generally be a safer route to go.

Caching

Caching can reduce the load on servers by storing the results of common operations and serving the precomputed answers to clients.

For example, instead of retrieving data from database tables that rarely change, you can store the values in-memory. Retrieving values from an in-memory location is far faster than retrieving them from a database (which stores them on a persistent disk like a hard drive.) When the cached values change the system can invalidate the cache and re-retrieve the updated values for future requests.

A cache can be created for multiple layers of the stack.

Caching backends

- memcached is a common in-memory caching system.
- Redis is a key-value in-memory data store that can easily be configured for caching with libraries such as django-redis-cache.

Caching resources

- "Caching: Varnish or Nginx?" reviews some considerations such as SSL and SPDY support when choosing reverse proxy Nginx or Varnish.
- Caching is Hard, Draw me a Picture has diagrams of how web request caching layers work. The post is relevant reading even though the author is describing his Microsoft code as the impetus for writing the content.
- While caching is a useful technique in many situations, it's important to also note that there are downsides to caching that many developers fail to take into consideration.

Caching learning checklist

1. Analyze your web application for the slowest parts. It's likely there are complex database queries that can be precomputed and stored in an in-memory data store.

2. Leverage your existing in-memory data store already used for session data to cache the results of those complex database queries. A task queue can often be used to precompute the results on a regular basis and save them in the data store.

3. Incorporate a cache invalidation scheme so the precomputed results remain accurate when served up to the user.

Microservices

Microservices are an application architecture style where independent, self-contained programs with a single purpose each can communicate with each other over a network. Typically, these microservices are able to be deployed independently because they have strong separation of responsibilities via a well-defined specification with significant backwards compatibility to avoid sudden dependency breakage.

Why are microservices getting so much buzz?

Microservices follow in a long trend of software architecture patterns that become all the rage. Previously, CORBA and (mostly XML-based) service-oriented architectures (SOA) were the hip buzzword among ivory tower architects.

However, microservices have more substance because they are typically based on RESTful APIs that are far easier for actual software developers to use compared with the previous complicated XML-based schemas thrown around by enterprise software companies. In addition, successful applications begin with a monolith-first approach using a single, shared application codebase and deployment. Only after the application proves its usefulness is it then broken down into microservice components to ease further development and deployment. This approach is called the "monolith-first" or "MonolithFirst" pattern.

Microservice resources

- Martin Fowler's microservices article is one of the best in-depth explanations for what microservices are and why to consider them as an architectural pattern.
- On monoliths and microservices provides some advice on using microservices in a fairly early stage of a software project's lifecycle.
- Developing a RESTful microservice in Python is a good story of how an aging Java project was replaced with a microservice built with Python and Flask.

- Microservices: The essential practices first goes over what a monolith application looks like then dives into what operations you need to support potential microservices. For example, you really need to have continuous integration and deployment already set up. This is a good high-level overview of the topics many developers aren't aware of when they embark on converting a monolith to microservices.

- How Microservices have changed and why they matter is a high level overview of the topic with some quotes from various developers around the industry.

- The State of Microservices Today provides some general trends and broad data showing the increasing popularity of microservices heading into 2016. This is more of an overview of the term than a tutorial but useful context for both developers and non-developers.

DevOps

DevOps is the combination of application development and operations, which minimizes or eliminates the disconnect between software developers who build applications and systems administrators who keep infrastructure running.

Why is DevOps important?

When the Agile methodology is properly used to develop software a new bottleneck often appears during the deployment and operations phases. New updates and fixes are produced so fast in each sprint that infrastructure teams can be overwhelmed with deployments and push back on the pace of delivery. To allievate some of these issues, application developers are asked to work closely with operations folks to automate the delivery from development to production.

General DevOps resources

- DevOps vs. Platform Engineering considers DevOps an ad hoc approach to developing software while building a platform is a strict contract. I see this as "DevOps is a process", while a "platform is code". Running code is better than any organizational process.

- DevOps: Python tools to get started is a presentation slideshow that explains that while DevOps is a culture, it can be supported by tools such as Fabric, Jenkins, BuildBot and Git which when used properly can enable continuous software delivery.

- Why are we racing to DevOps? is a very high level summary of the benefits of DevOps to IT organizations. It's not specific to Python and doesn't dive into the details, but it's a decent start for figuring out why IT organizations consider DevOps the hot new topic after adopting an Agile development methodology.

Testing & Debugging

Testing

Testing determines whether software runs correctly based on specific inputs and identifies defects that need to be fixed.

Why is testing important?

As software scales in codebase size, it's impossible for a person or even a large team to keep up with all of the changes and the interactions between the changes. Automated testing is the only proven method for building reliable software once they grow past the point of a simple prototype. Many major software program development failures can be traced back to inadequate or a complete lack of testing.

It's impossible to know whether software works properly unless it is tested. While testing can be done manually, by a user clicking buttons or typing in input, it should be performed automatically by writing software programs that test the application under test.

There are many forms of testing and they should all be used together. When a single function of a program is isolated for testing, that is called unit testing. Testing more than a single function in an application at the same time is known as integration testing. *User interface testing* ensures the correctness of how a user would interact with the software. There are even more forms of testing that large programs need, such as *load testing*, *database testing*, and *browser testing* (for web applications).

Testing in Python

Python software development culture is heavy on software testing. Because Python is a dynamically-typed language as opposed to a statically-typed language, testing takes on even greater importance for ensuring program correctness.

Testing resources

- The Minimum Viable Test Suite shows how to set unit tests and integration tests for

a Flask example application.

- Good test, bad test explains the difference between a "good" test case and one that is not as useful. Along the way the post breaks down some myths about common testing subjects such as code coverage, assertions and mocking.

- Python Testing is a site devoted to testing in - you guessed it - the Python programming language.

- The case for test-driven development by Michael DeHaan explains how automation is the only way to build software at a large scale.

- Google has a testing blog where they write about various aspects of testing software at scale.

- Still confused about the difference between unit, functional and integration tests? Check out this top answer on Stack Overflow to that very question.

- Using pytest with Django shows how to get a basic pytest test running for a Django project and explains why the author prefers pytest over standard unittest testing.

Unit testing

Unit testing is a method of determining the correctness of a single function isolated from a larger codebase. The idea is that if all the atomic units of an application work as intended in isolation, then integrating them together as intended is much easier.

Why is unit testing important?

Unit testing is just one form of testing that works in combination with other testing approaches to wring out the bugs from a piece of software being developed. When several functions and classes are put together it's often difficult to determine the source of a problem if several bugs are occurring at the same time. Unit testing helps eliminate as many of the individual bugs as possible so when the application comes together as a whole the separate parts work as correct as possible. Then when issues arise they can often be tracked down as unintended consequences of the disparate pieces not fitting together properly.

Unit testing tools

There are many tools for creating tests in Python. Some of these tools, such as pytest, replace the built-in unittest framework. Other tools, such as nose, are extensions that ease test case creation. Note that many of these tools are also used for integration testing by writing the test cases to exercise multiple parts of code at once.

- unittest is the built-in standard library tool for testing Python code.
- pytest is a complete testing tool that emphasizes backwards-compatibility and minimizing boilerplate code.
- nose is an extension to unittest that makes it easier to create and execute test cases.
- testify was a testing framework meant to replace the common unittest+nose combination. However, the team behind testify is transitioning to pytest so it's recommended you do not use testify for new projects.

Unit testing resources

- Dive into Python 3's chapter on unit testing has a complete example with code and a detailed explanation for creating unit testing with the unittest module.
- Unit Testing Your Twilio App Using Python's Flask and Nose is a detailed tutorial for using the nose test runner for ensuring a Flask application is working properly.
- Understanding unit testing explains why testing is important and shows how to do it effectively in your applications.
- Unit testing with Python provides a high-level overview of testing and has diagrams to demonstrate what's going on in the testing cycle.
- The Python wiki has a page with a list of Python testing tools and extensions.
- Working Effectively with Unit Tests is an interview with the author of a book by the title where he shares some of the insight he's learned on the topic.
- Generate your tests shows how to write a test generator that works with the `unittest` framework.
- An Extended Introduction to the nose Unit Testing Framework shows how this test runner can be used to write basic test suites. While the article is from 2006, it remains relevant today for learning how to use nose with your projects.

Integration Testing

Integration testing exercises two or more parts of an application at once, including the interactions between the parts, to determine if they function as intended. This type of testing identifies defects in the interfaces between disparate parts of a codebase as they invoke each other and pass data between themselves.

How is integration testing different from unit testing?

While unit testing is used to find bugs in individual functions, integration testing tests the system as a whole. These two approaches should be used together, instead of doing just one approach over the other. When a system is comprehensively unit tested, it makes integration testing far easier because many of the bugs in the individual components will have already been found and fixed.

As a codebase scales up, both unit and integration testing allow developers to quickly identify breaking changes in their code. Many times these breaking changes are unintended and wouldn't be known about until later in the development cycle, potentially when an end user discovers the issue while using the software. Automated unit and integration tests greatly increase the likelihood that bugs will be found as soon as possible during development so they can be addressed immediately.

Integration testing resources

- Integration testing with Context Managers gives an example of a system that needs integration tests and shows how context managers can be used to address the problem. There are a couple other useful posts in this series on testing including thoughts on integration testing and processes vs. threads for integration testing.

- Pytest has a page on integration good practices that you'll likely want to follow when testing your application.

- Integration testing, or how to sleep well at night explains what integration tests are and gives an example. The example is coded in Java but still relevant when you're

learning about integration testing.

Code Metrics

Code metrics can be produced by static code analysis tools to determine complexity and non-standard practices.

Why are code metrics important?

Code metrics allow developers to find problematic codebase areas that may need refactoring. In addition, some metrics such as technical debt assist developers in communicating to non-technical audiences why issues with a system are occurring.

Open source code metrics projects

- Radon is a tool for obtaining raw metrics on line counts, Cyclomatic Complexity, Halstead metrics and maintainability metrics.
- Pylint contains checkers for PEP8 code style compliance, design, exceptions and many other source code analysis tools.
- PyFlakes parses source files for errors and reports on them.
- Pyntch is a static code analyzer that attempts to detect runtime errors. It does not perform code style checking.

Hosted code metrics services

- Coveralls shows code coverage from test suites and other metrics to help developers improve the quality of their code.

Code metrics resources

- Static Code Analizers for Python is an older article but goes over the basics of what Python static code analyzers do.

- This Stack Overflow question on Python static code analysis tools contains comparison discussions of PyLint, PyChecker and PyFlakes.
- Getting Started with Pylint goes over setting up Pylint, generating the .pylintrc file and what's in the configuration.
- This /r/Python poll on what linters the community uses provides some input on using PyCharm just for its linting features as well as some other approaches.

Debugging

Developers often find themselves in situations where the code they've written is not working quite right. When that happens, a developer debugs their code by instrumenting, executing and inspecting the code to determine what state of the application does not match the assumptions of how the code should be correctly running.

Why is debugging important?

There are bugs in every modest sized or larger application. Every developer has to learn how to debug code in order to write programs that work as correctly as time and budget allow.

Debugging resources

- Debugging your Python code walks through a scenario where pdb can be used to find a defect in a block of Python code.
- pdb - Interactive Debugger is featured on the Python Module of the Week blog and has some great detail on using the program effectively.
- Python debugging tools provides a list of tools such as pdb and its derivatives ipdb, pudb and pdb++ along with how they can be used in the hunt for defects.
- Debugging in Python elaborates on what pdb does and how it can be used.

Meta

What "full stack" means

The terms "full stack" and "Full Stack Python" are ambiguous but I am using a specific definition here on this site. These term can be defined for a web stack either to mean

1. Every layer, from the machine code up to the browser, are written in Python

2. Python code interacts with code written in other languages such as C and JavaScript to provide a complete web stack

I named this site specifically for the second definition: Python is one programming language among many that are used to build your application stack.

Some folks took the title of the site to mean Python runs everything from the web browser on down. That's simply not practical or possible. While Python is an amazing programming language, there are many tasks it does not do well.

Python is just one language among many that allows our computers to execute software and communicate with each other.

For beginners, learning the syntax and libraries in Python necessary to build a web application or web API is a major undertaking. Even intermediate and advanced Python software developers need to constantly program and learn to keep up with our ever evolving ecosystem. I created Full Stack Python to be just one of many resources that help Python developers build and maintain their programming skills.

Future Directions

Full Stack Python has completely blown away my expectations for what I could accomplish with a side project. I really appreciate all the in-person feedback, emails and pull requests I've received from the community. Keep them coming!

For 2016 I'm building out the scope of the site beyond web development into core Python concepts, data analysis and visualization and some hardware hacking such as with the Raspberry Pi and Arduino Yun.

The biggest update though will come with the release of The Full Stack Python Guide to Deployments, a step-by-step tutorial book for learning how to deploy Python web applications.

Note that these plans can change based on pull requests from the community. I work to integrate PRs within a day or two so please submit one when you see a fix or improvement that needs to be made!

About the Author

This website was coded and written by Matt Makai (@mattmakai), currently a Developer Evangelist at Twilio in San Francisco, California.

Other projects by Matt include The Full Stack Python Guide to Deployments book, Coding Across America, Underwear and Choose Your Own Adventure Presentations. You can reach him by email at matthew.makai@gmail.com, Twitter @mattmakai or on GitHub at makaimc.

Read my thoughts on the "full stack" trend in a post I wrote for O'Reilly Radar.

Typos, inaccurate statements or general areas for improvement can be handled through an issue ticket or pull request on GitHub.