

# LISTEN NOTES

Trending: "Steven Levy"

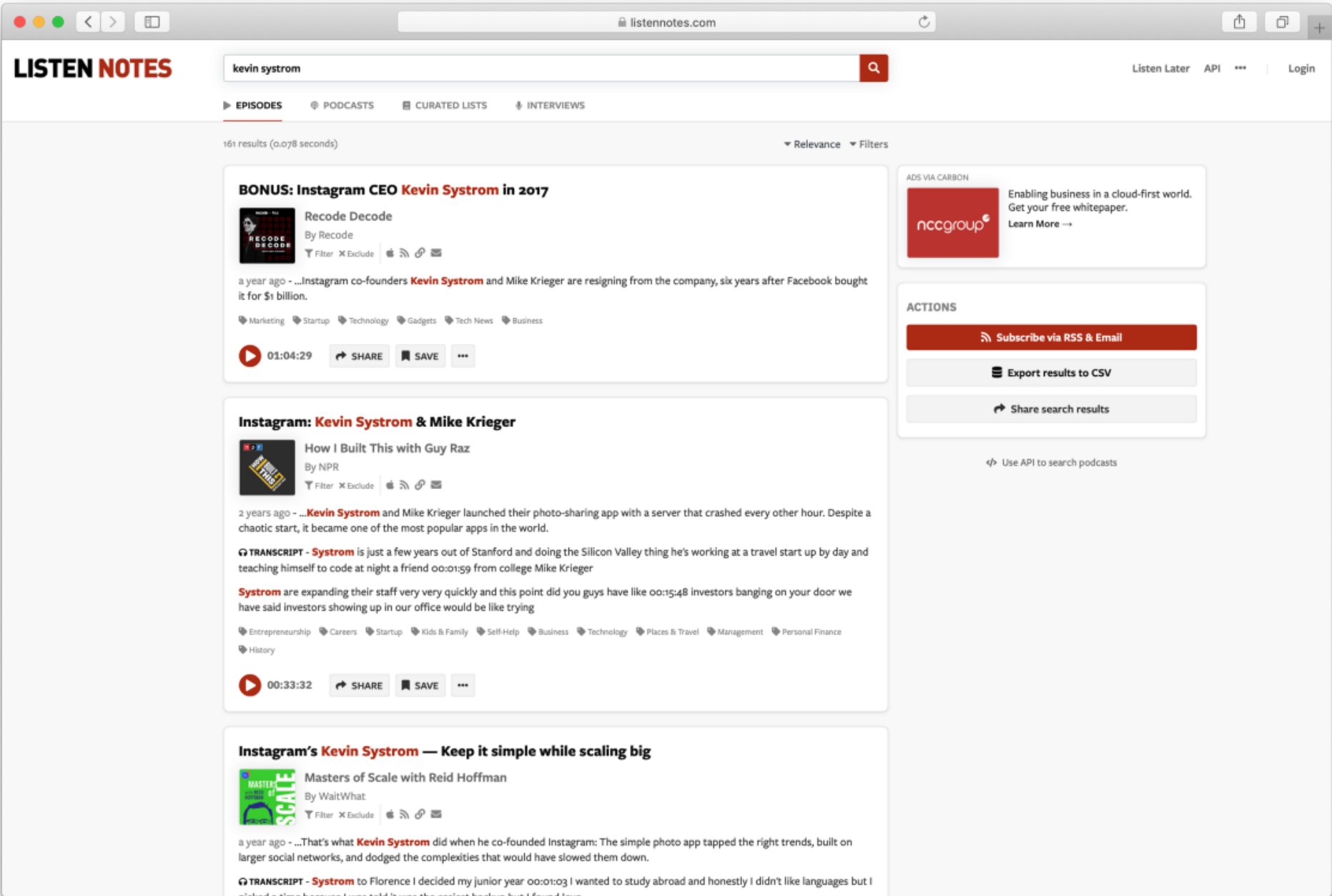
[Listen Later](#) [API](#) [...](#) | [Login](#)

The Official Listen Notes Blog

## The boring technology behind a one-person Internet company

By **Wenbin Fang** · Jan. 24, 2018

SHARE



The search result page on [ListenNotes.com](#)

[Updated @ Dec, 2019] I found that this old blog post got shared & discussed a lot recently on Hacker News and Reddit. I'd like to clarify a few things:

- This post is not very up-to-date now. The tech stack keeps evolving... and is becoming a bit complex over the full-time work in the past 2 years. Initially, Listen Notes was running on 3 DigitalOcean droplets (~\$30/month) in Jan 2017. "Boring" doesn't mean "simple" or "using very old & very enterprise-y tech stack". "Boring" here means I just use the tech stack that I'm familiar with so I can quickly launch the project & focus more on the business end.

- I only spend 10%~20% of my time (likely close to 10%) on engineering nowadays—80% of engineering time is used to iterate on existing features/infra/internal tools, while 20% is to experiment on new things. Most of my time is spent on talking to other human beings, replying emails (30%~40% of my time), and thinking (!!!), which is not considered as “real work” by engineers :) I usually stay in office for 4~8 hours/day, but I work from iPhone (e.g., reply emails, ChatOps from Slack...) and think about Listen Notes a lot when I’m not in office.
- I have to apologize to those who feel offended by this post, because I didn’t praise your favorite technology, or I didn’t write things that you agree to, or I didn’t answer all of your questions in this post—to be fair, I can’t predict what questions you may have when writing this post! It’s impossible to make everyone happy :(
- This blog post is to show you **one way** of doing engineering for a very specific type of online business. It’s not the only way. It’s certainly not the best way. I hope this blog post provides one useful data point to the tech world. You can find more data points from places like Hacker News or Indie Hackers :)



Wenbin  
@wenbinf



Your overthinking is my opportunity.

3:48 AM · Jan 9, 2019



109



35 people are talkin...

[Listen Notes](#) is a podcast search engine and database. The technology behind Listen Notes is actually very very boring. No AI, no deep learning, no blockchain. [“Any man who must say I am using AI is not using True AI”](#) :)

After reading this post, you should be able to replicate what I build for Listen Notes or easily do something similar. You don’t need to hire a lot of engineers. Remember, [when Instagram raised \\$57.5M and got acquired by Facebook for \\$1B](#), they had only [13 employees](#)—not all of them were engineers. The Instagram story happened in early 2012. It’s 2019 now, it’s more possible than ever to build something meaningful with a tiny engineering team—even one person.

If you haven’t used Listen Notes yet , try it now: <https://www.listennotes.com/>

## Overview

Let’s start with requirements or features of this Listen Notes project.

Listen Notes provides two things to end users:

- A website [ListenNotes.com](#) for podcast listeners. It provides a search engine, a podcast database, [Listen Later](#) playlists, [Listen Clips](#) that allows you to cut a segment of any podcast episode, and [Listen Alerts](#) that notifies you when a specified keyword is mentioned in new podcasts on the Internet.
- [Podcast Search & Directory APIs](#) for developers. We need to track the API usage, get money from paid users, do customer support, and more.

I run everything on AWS. There are 20 production servers (as of May 5, 2019):

Hostname	Status	CPU	IOWait	Load 15
production-db1	UP	23% <div></div>	2% <div></div>	0.83
production-db2	UP	14% <div></div>	0.8% <div></div>	0.18
production-es1	UP	11% <div></div>	1% <div></div>	0.28
production-es2	UP	9% <div></div>	0.6% <div></div>	0.28
production-es3	UP	11% <div></div>	1% <div></div>	0.26
production-lb1	UP	5% <div></div>	0.3% <div></div>	0.1
production-pangu1	UP	2% <div></div>	1% <div></div>	0.07
production-v1api1	UP	0.3% <div></div>	0% <div></div>	0
production-v2api1	UP	1% <div></div>	0.01% <div></div>	0
production-v2api2	UP	1% <div></div>	0% <div></div>	0.04
production-web1	UP	10% <div></div>	0% <div></div>	0.39
production-web2	UP	11% <div></div>	0% <div></div>	0.2
production-worker1	UP	100% <div></div>	0% <div></div>	2.55
production-worker2	UP	94% <div></div>	0% <div></div>	2.52
production-worker3	UP	100% <div></div>	0% <div></div>	6.01
production-worker4	UP	82% <div></div>	0.01% <div></div>	1.88
production-worker5	UP	97% <div></div>	0% <div></div>	2.34
production-worker6	UP	85% <div></div>	0% <div></div>	2.33
production-worker7	UP	88% <div></div>	0% <div></div>	0.6
production-worker8	UP	16% <div></div>	0% <div></div>	0.8

The servers that run Listen Notes. This is a dashboard using [Datadog](#).

You can easily guess what does each server do from the hostname.

- **production-web** serves web traffics for [ListenNotes.com](#).
- **production-api** serves api traffics. We run two versions of API (as of May 4, 2019), thus v1api (the legacy version) and v2api (the new version).
- **production-db** runs PostgreSQL (master & slave)
- **production-es** runs an Elasticsearch cluster.
- **production-worker** runs offline processing tasks to keep the podcast database always up-to-date and to provide some magical things (e.g., search result ranking, episode/podcast recommendations...).
- **production-lb** is the load balancer. I also run Redis & RabbitMQ on this server, for convenience. I know this is not ideal. But I'm not a perfect person :)
- **production-pangu** is the production-like server that I sometimes run one-off scripts and test changes. What's the meaning of "pangu"?

Most of these servers can be horizontally scaled. That's why I name them *production-something1*, *production-something2*... It could be very easy to add *production-something3* and *production-something4* to the fleet.

## Backend

The entire backend is written in Django / Python3. The operating system of choice is Ubuntu.

I use [uWSGI](#) to serve web traffics. I put [NGINX](#) in front of uWSGI processes, which also serves as load balancer.

The main data store is [PostgreSQL](#), which I've got a lot of development & operational experience over many years—battle tested technology is good, so I can sleep well at night. [Redis](#) is used for various purposes (e.g., caching, stats,...). It's not hard to guess that [Elasticsearch](#) is used somewhere. Yes, I use Elasticsearch to index podcasts & episodes and to serve search queries, just like [most boring companies](#).

[Celery](#) is used for offline processing. And [Celery Beat](#) is for scheduling tasks, which is like Cron jobs but a bit nicer. If in the future Listen Notes gains traction and Celery & Beat cause some scaling issues, I probably will switch to the two projects I did for my previous employer: [ndkale](#) and [ndscheduler](#).

[Supervisord](#) is used for process management on every server.

Wait, how about Docker / Kubernetes / serverless? Nope. As you gain experience, you know when not to over-engineer. I actually did some early Docker work for my previous employer back in 2014, which was good for a mid-sized billion-dollar startup but may be overkill for a one-person tiny startup.

## Frontend


The web frontend is primarily built with [React](#) + [Redux](#) + [Webpack](#) + [ES](#). This is pretty standard nowadays. When deploying to production, JS bundles would be uploaded to [Amazon S3](#) and served via [CloudFront](#).

On [ListenNotes.com](#), most web pages are half server-side rendered ([Django template](#)) and half client-side rendered ([React](#)). The server-side rendered part provides a boilerplate of a web page, and the client-side rendered part is basically an interactive web app. But a few web pages are rendered entirely via server side, because of my laziness to make things perfect & some potential SEO goodies.

## Audio player

I use a heavily modified version of [react-media-player](#) to build the audio player on ListenNotes.com, which is used in several places, including [Listen Notes Website](#), [Twitter embedded player](#), and embedded player on 3rd party websites:

**Preview:**



Nerd Fight - Episode #34

15

30

1.0X

21:14

1:11:15

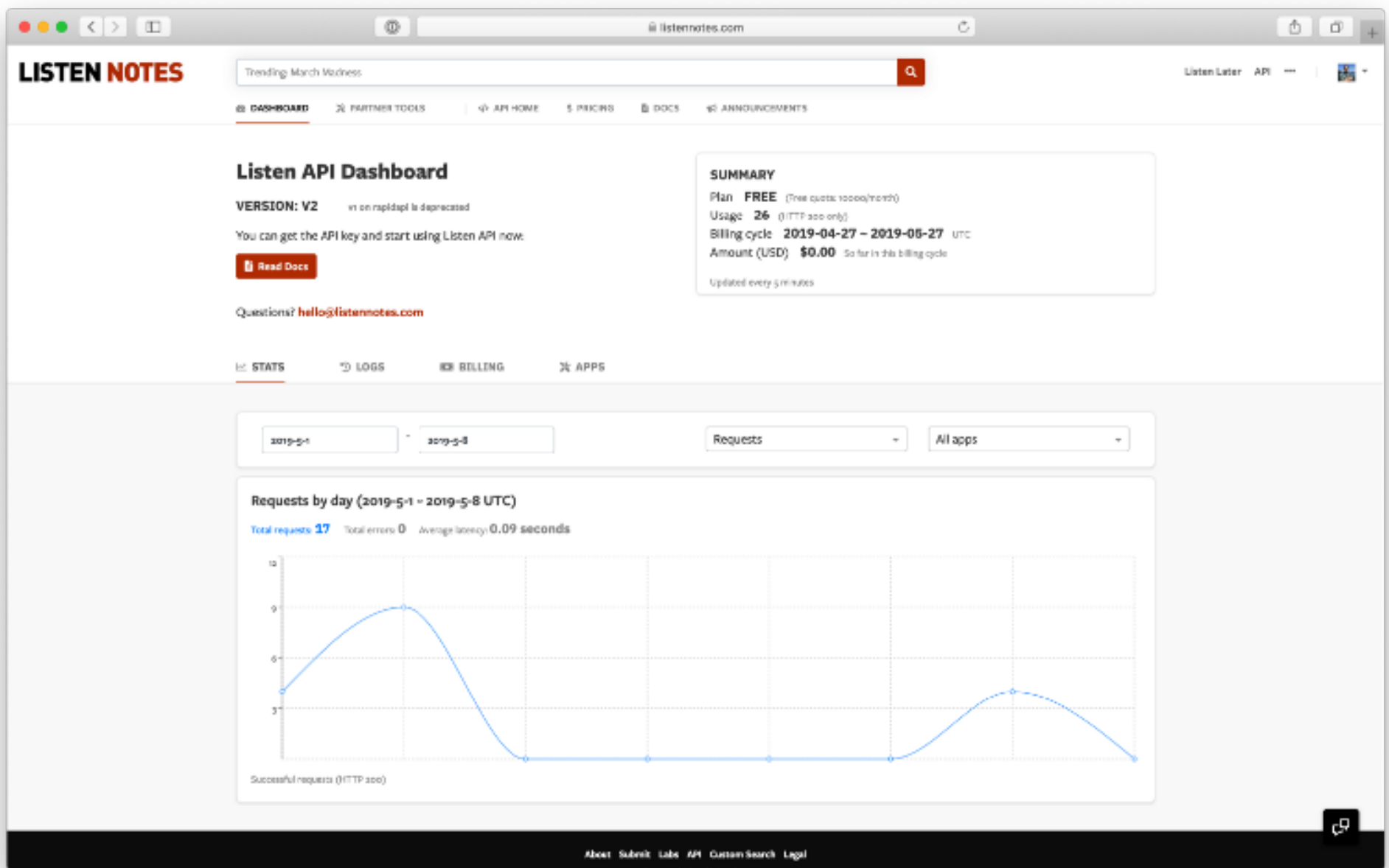
LISTEN NOTES

**Code:**

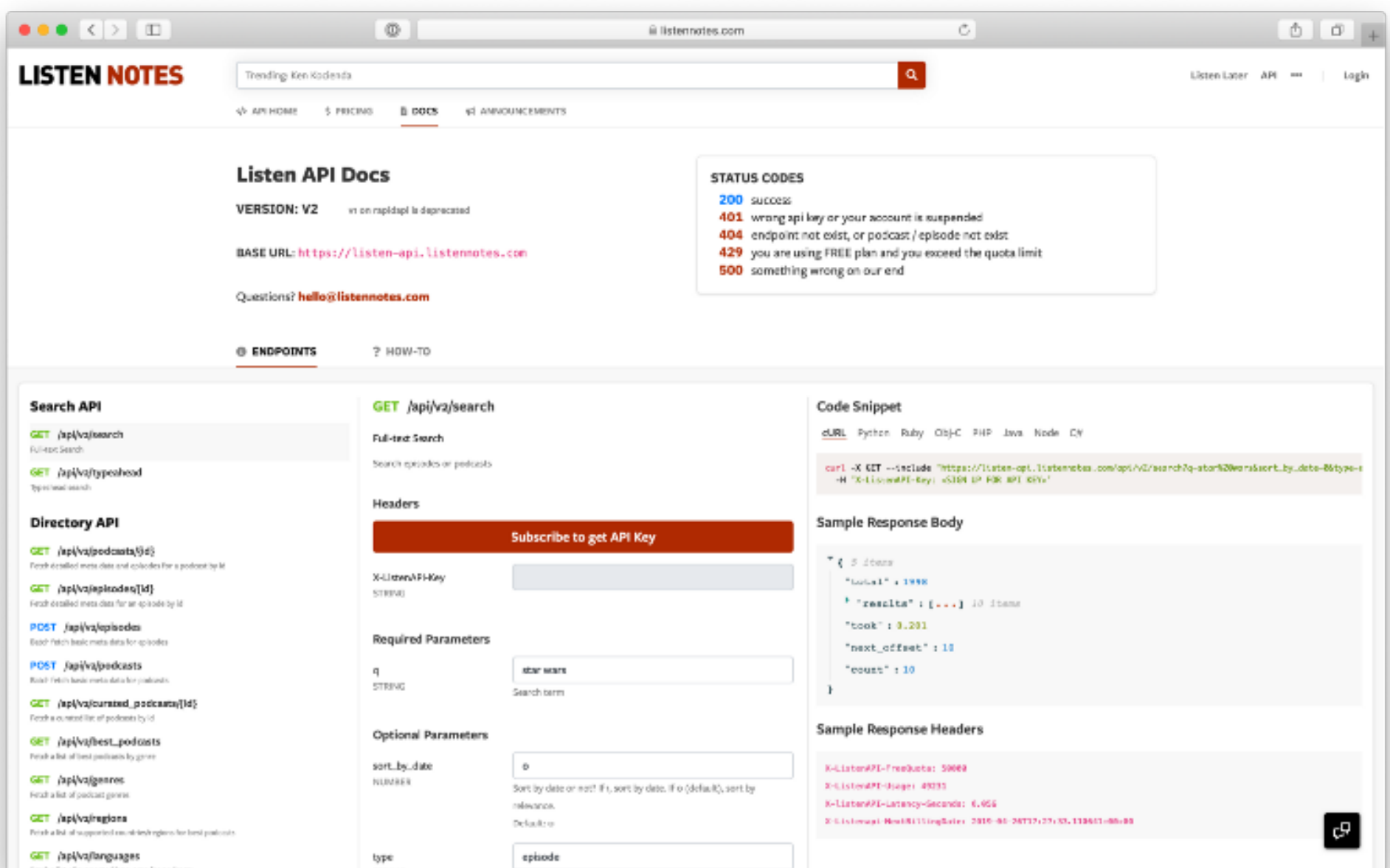
```
<iframe
src="https://www.listennotes.com/embedded/e/391b06f633224bdb84220492e31098f9/"
height="170px" width="100%" style="width: 1px; min-width: 100%;" frameborder="0"
scrolling="no"></iframe>
```

## Podcast API

We provide a simple and reliable [podcast API](#) to developers. Building the API is similar to building [the website](#). I use the same Django/Python stack for the backend, and ReactJs for the frontend (e.g., API dashboard, documentation...).



Listen API dashboard



[Listen API documentation](#)

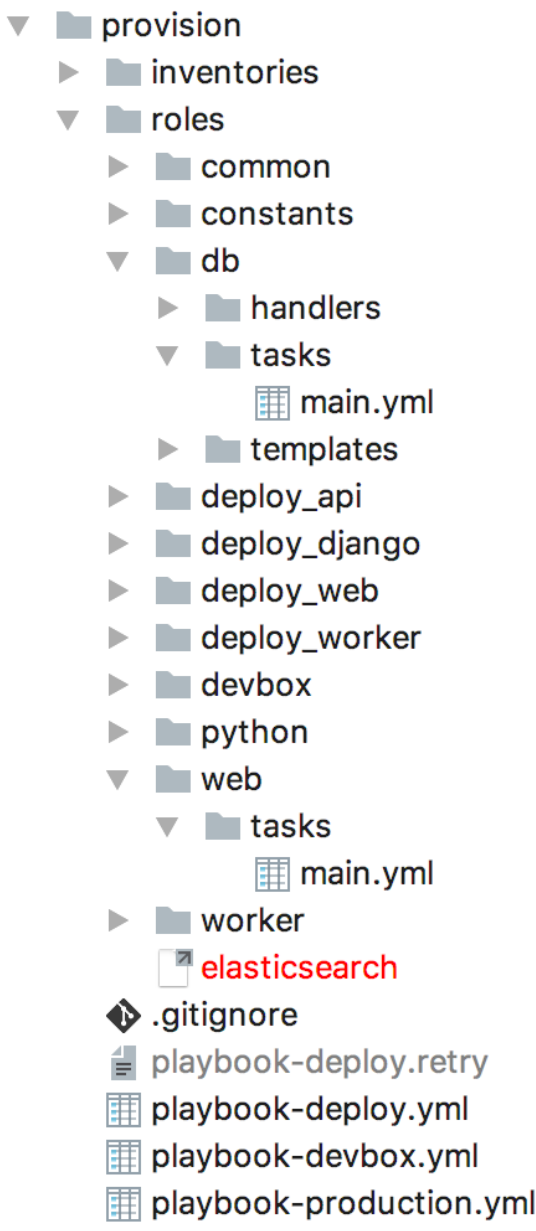
For the API, we need to track how many requests a user uses in current billing cycle, and charge \$\$\$ at the end of a billing cycle. It's not hard to imagine that Redis is heavily used here :)



# DevOps

## Machine provisioning & code deployment

I use [Ansible](#) for machine provisioning. Basically, I wrote a bunch of yaml files to specify what type of servers need to have what configuration files & what software. I can spin up a server with all correct configuration files & all software installed with one button push. This is the directory structure of those Ansible yaml files:



I could’ve done a better job in naming things. But again, it’s good enough for now.

I also use Ansible to deploy code to production. Basically, I have a wrapper script *deploy.sh* that is run on macOS:

```
./deploy.sh production HEAD web
```

The *deploy.sh* script takes three arguments:

- **Environment:** production or staging.
- **Version of the listennotes repo:** HEAD means “just deploy the latest version”. If a SHA of a git commit is specified, then it’ll deploy a specific version of code—this is particularly useful when I need to rollback from a bad deployment.
- **What kind of servers:** web, worker, api, or all. I don’t have to deploy to all servers all at once. Sometimes I make changes on Javascript code, then I just need to deploy to web, without touching api or worker.

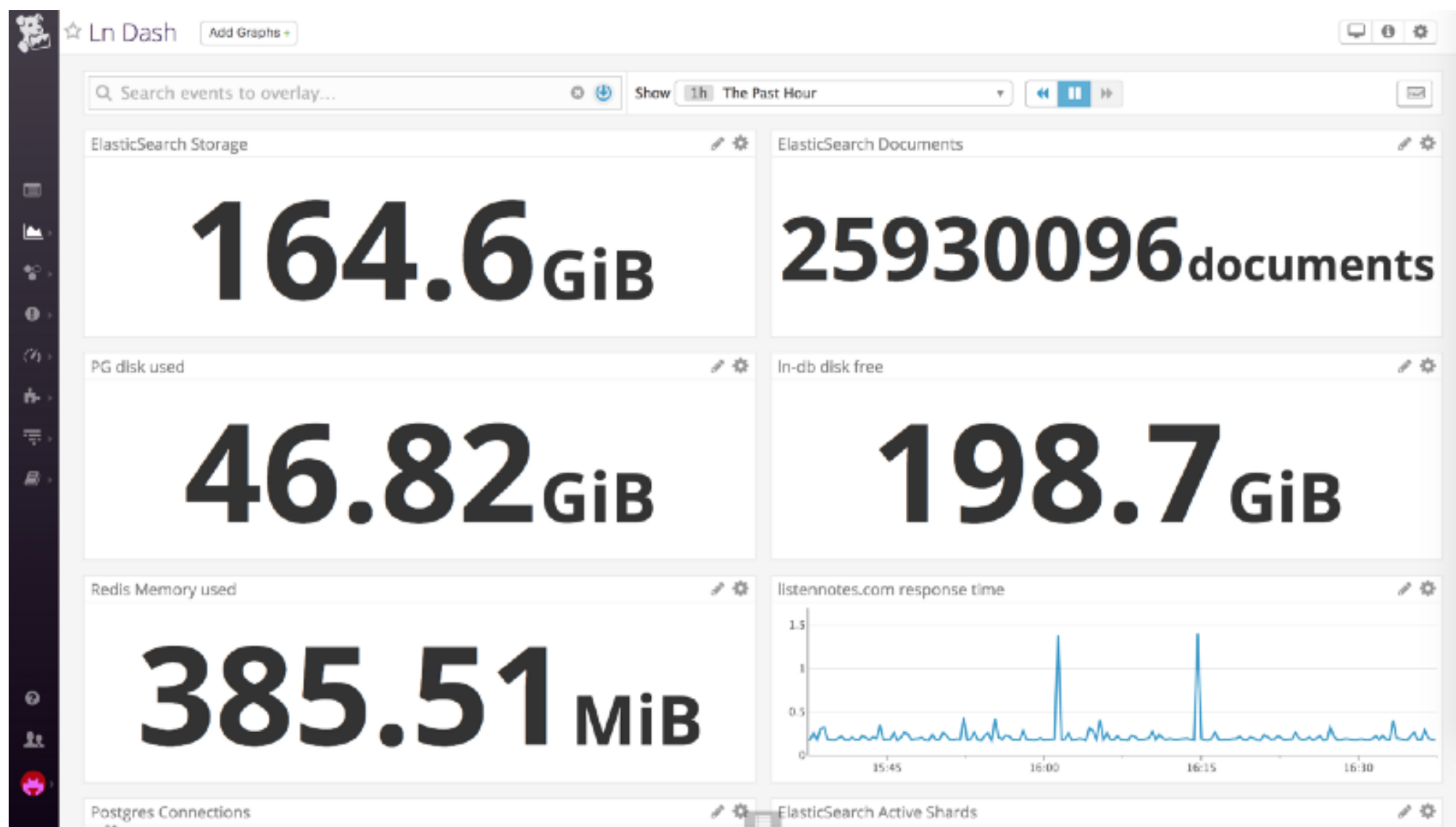
The deployment process is mostly orchestrated by Ansible yaml files, and of course, it’s dead simple:

- **On my Macbook Pro**, if it’s to deploy to web servers, then build Javascript bundles and upload to S3.
- **On the target servers**, git clone the listennotes repo to a timestamp-named folder, check out the specific version, and pip install new Python dependencies if any.
- **On the target servers**, switch symlink to the above timestamp-named folder and restart servers via supervisorctl.

As you can see, I don’t use those fancy CI tools. Just dead simple things that actually work.

## Monitoring & alerting

I use [Datadog](#) for monitoring & alerting. I’ve got some high level metrics in a simple dashboard. Whatever I do here is to boost my confidence when I am messing around the production servers.



Datadog dashboard for Listen Notes, as of Dec 2017.

I connect [Datadog](#) to PagerDuty. If something goes wrong, [PagerDuty](#) will send me alerts via phone call & SMS.

I also use [Rollbar](#) to keep an eye on the health of Django code, which will catch unexpected exceptions and notify me via email & Slack as well.

I use [Slack](#) a lot. Yes, this is a one-person company, so I don't use Slack for communicating with human beings. I use Slack to monitor interesting application-level events. In addition to integrating Datadog and Rollbar with Slack, I also use [Slack incoming webhooks](#) in Listen Notes backend code to notify me whenever a user signs up or performs some interesting actions (e.g., adding or deleting things). This is a very common practice in tech companies. When you read some books about Amazon or PayPal's early history, you'll know that both companies had similar notification mechanism: whenever a user signed up, there would be a "ding" sound to notify everyone in the office.

Since launched in early 2017, Listen Notes hasn't got any big outage (> 5 minutes) except for [this one](#). I'm always very careful & practical in these operational stuffs. The web servers are significantly over-provisioned, just in case there's some huge spike due to press events or whatever.

## Development

I work in a [WeWork](#) coworking space in San Francisco. Some people may wonder why not just work from home or from some random coffee shops. Well, I value productivity a lot and I'm willing to invest money in productivity. I don't believe piling time helps software development (or any sort of knowledge/creativity work). It's rare that I work over 8 hours in a day (Sorry, [996 people](#)). I want to make every minute count. Thus, a nice & relatively expensive private office is what I need :) Instead of optimizing for spending more time & saving money, I optimize for spending less time & making money :)



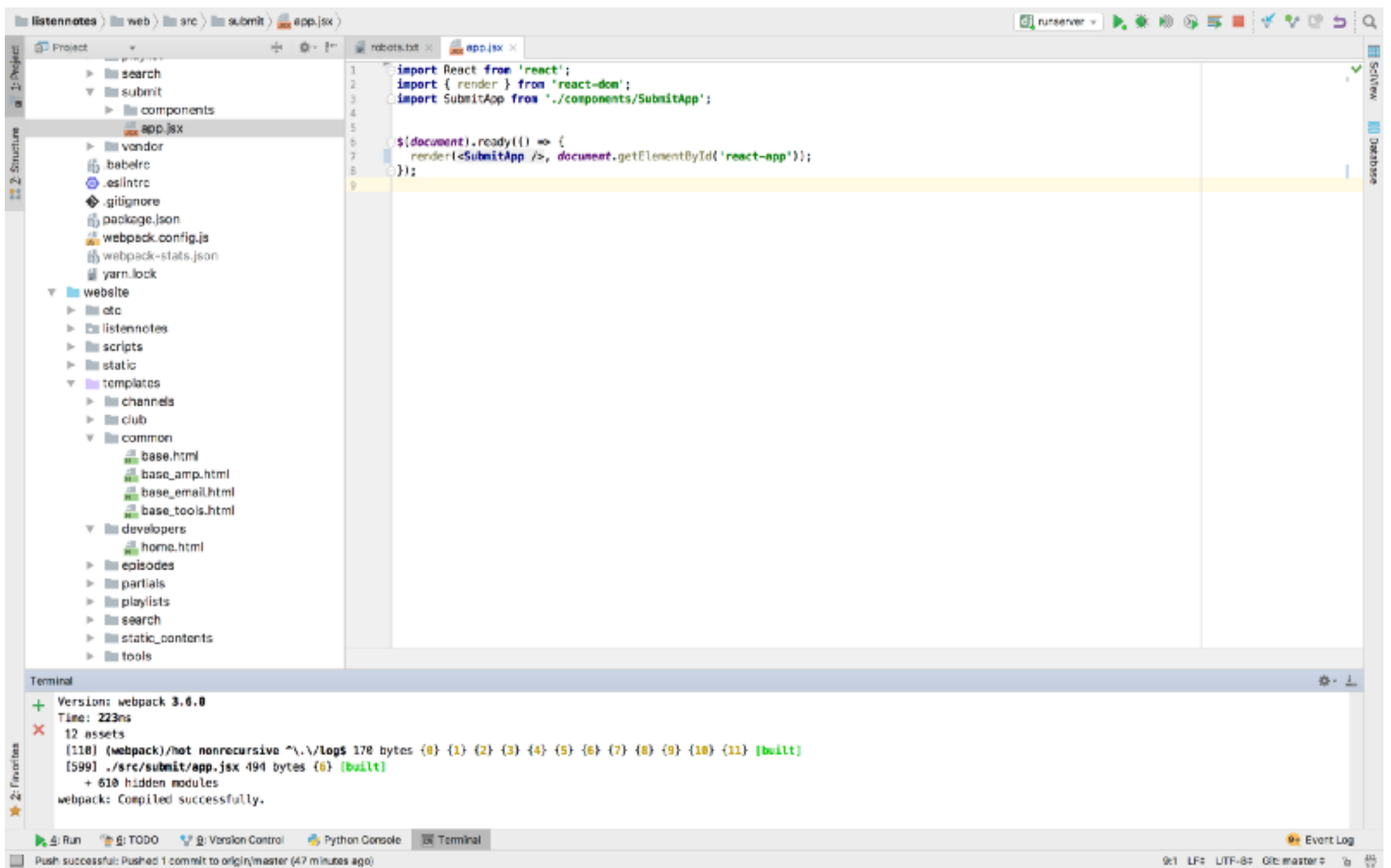
My office at WeWork

I'm using a MacBook Pro. I run the (almost) identical infrastructure inside [Vagrant](#) + [VirtualBox](#). I use the same set of Ansible yaml files as described above to provision the development environment inside Vagrant.

I subscribe to the [monolithic repo](#) philosophy. So there's one and only one listennotes repo, containing DevOps scripts, frontend & backend code. This listennotes repo is hosted as a GitHub private repo. I do all development work on the master branch. I rarely use feature branches.

I write code and run the dev servers (Django runserver & webpack dev server) by using [PyCharm](#). Yea, I know, it's boring. After all, it's not Visual Studio Code or Atom or whatever cool IDEs. But PyCharm works just fine for me. I'm old school.





My PyCharm

## Miscellaneous

There are a bunch of useful tools & services that I use to build Listen Notes as a product and a company:

- [iTerm2](#) and [tmux](#) for the terminal stuffs.
- [Notion](#) for TODO lists, wiki, taking notes, design documents...
- [G Suite](#) for @listennotes.com email account, calendar, and other Google services.
- [MailChimp](#) for sending the [monthly email newsletter](#).
- [Amazon SES](#) for sending transactional & some marketing emails.
- [Gusto](#) to pay myself and contractors who are not from Upwork.
- [Upwork](#) to find contractors.
- [Google Ads Manager](#) to manage direct sales ads and track performance.
- [Carbon Ads](#) and [BuySellAds](#) for fallback ads.
- [Cloudflare](#) for DNS management, CDN, and firewall.
- [Zapier](#) and [Trello](#) to streamline the [podcaster interview](#) workflow.
- [Medium](#) for the company blog (obviously).
- [Godaddy](#) and [Namecheap](#) for domain names.
- [Stripe](#) for getting money from users (primarily for [API](#)).
- [Google speech-to-text API](#) to transcribe episodes.
- [Kaiser Permanente](#) for health insurance.
- [Stripe Atlas](#) to incorporate Listen Notes, Inc.
- [Clerky](#) to generate legal documents for fund raising (SAFE) and hiring contractors who are not from Upwork.
- [Quickbooks](#) for bookkeeping.
- [1password](#) to manage login credentials for tons of services
- [Brex](#) for charge card—you can get incremental \$5000 AWS credits, which can be applied on top of the AWS credits from WeWork or Stripe Atlas.
- [Bonvoy Business Amex Card](#)—You can earn Marriott Bonvoy points for luxury hotels and flights. It's the best credit card points for traveling :)
- [Capital One Spark](#) for checking account.

Keep calm and carry on...

As you can see, we are living in a wonderful age to start a company. There are so many off-the-shelf tools and services that save us time & money and increase our productivity. It's more possible than ever to build something useful to the world with a tiny team (or just one person), using simple & boring technology.


As time goes, companies become smaller and smaller. You don't need to hire tons of full-time employees. You can hire services (SaaS) and on-demand contractors to get things done.

Most of time, the biggest obstacle of building & shipping things is over thinking. What if this, what if that. Boy, you are not important at all. Everyone is busy in their own life. No one cares about you and the things you build, until you prove that you are worth other people's attention. Even you screw up the initial product launch, few people will notice. [Think big, start small, act fast](#). It's absolutely okay to use the boring technology and start something simple (even ugly), as long as you actually solve problems.




Wenbin

@wenbinf




Your overthinking is my opportunity.


3:48 AM · Jan 9, 2019



109



35 people are talkin...



There are so many [cargo-cult](#)-type people now. Ignore the noises. Keep calm and carry on.

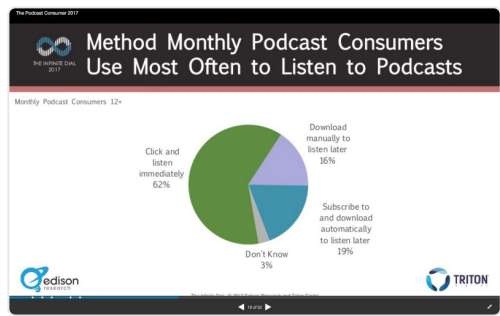
PREVIOUS ARTICLE:

[My Y Combinator interview experience \(W18\).](#)

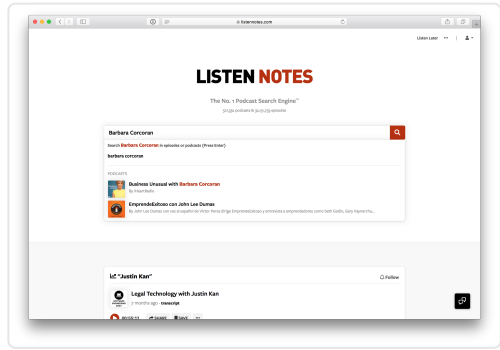
NEXT ARTICLE:

[Postmortem on Apr 22, 2018 outage](#)

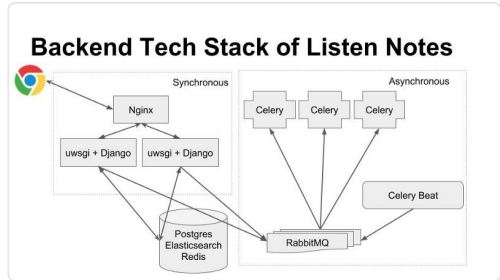
FEATURED BLOG POSTS >>



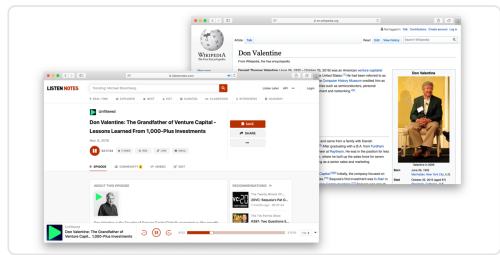
[How do people use Listen Notes?](#)



[Why a standalone podcast search engine website?](#)



[Good enough engineering to start an Internet company.](#)



[Why Podcasts Are My New Wikipedia—the Perfect Informal Learning Resource](#)

PLAYLISTS FROM COMMUNITY >>



Cultura, società e politica

1 episodes and 78 podcasts

Updated 3 hours ago



Jeanne's audio diary\_playlist

4 episodes and 2 podcasts

Updated 5 hours ago



Jeanne's Covid 19\_playlist


9 episodes and 4 podcasts

Updated 5 hours ago

 Listen Notes for Chrome

The fastest way to find podcasts!

[About](#) [Submit](#) [Labs](#) [Business](#) [Integrations](#) [Legal](#) [Help](#) [Stats](#) [Blog](#)



 [hello@listennotes.com](mailto:hello@listennotes.com)

© 2020 Listen Notes, Inc.