CS 291 Guest Lecture

Andrew Mutz
Chief Technology Officer, Property Management
November 14, 2017

Outline for today

- Appfolio Overview
- Technology Overview
- Engineering Overview
- Scalability Lessons Learned
- Challenges Ahead

The story starts with the company and the product. (Why?)

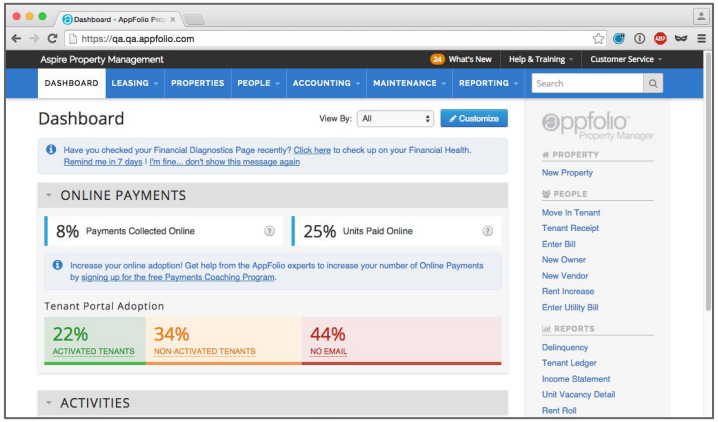
Appfolio makes

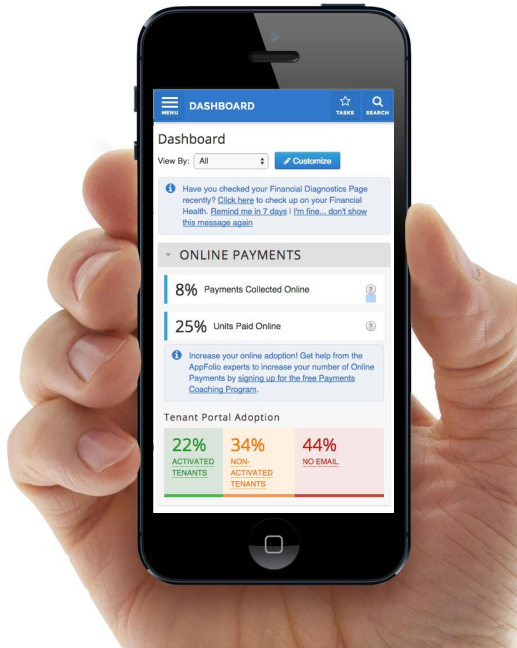
- Business software, sold as a service
- For small and medium sized businesses
- In specific industries

“Business software, sold as a service”

- Software is referred to as “a service” (SaaS) when the software is managed and run by an outside company
  - Think Outlook vs. GMail
  - Business software used to be primarily on the premises of the customer (and still is to a great deal)
  - Customer data is stored in data centers, managed by experts
  - Users interact with the system through web browsers and mobile apps
  - Updates are frequent
  - Users pay per month

- Software as a Service used to be the exception, but it is increasingly the norm
  - Consumer adoption of SaaS is very high: GMail, Facebook, Snapchat, YouTube, etc
  - Business adoption also high, but transition is slower





“For small and medium sized businesses”

- Employees: Tens or hundreds
- Appfolio Property Manager doesn’t serve either end of the spectrum:
  - If someone manages less than 50 units, they might not be a good fit
  - If someone manages many tens of thousands of units, they might not be a good fit
- SMB is not just about the number of users, it is an approach to building software
  - Customers are large enough that they are experts in their business
  - Customers are small enough that they do not need their software customized
    - Large companies tend to prefer software that is customized to their particular business practices
  - Appfolio does not build features or customize the product for a single company.
    - Caveat: we can turn features on and off for customers

“In specific industries”

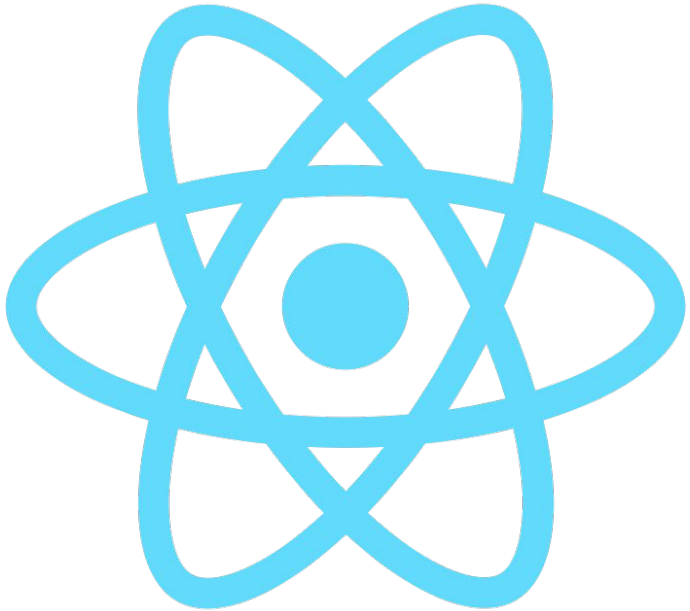
- Software that is targeted at a specific industry is referred to as “Vertical” software
- Instead of picking a specific type of work that is universal to all companies, we build exactly what is needed by a specific type of company
- Vertical software emphasizes integration between different functional areas
    - The CRM system is integrated with the Accounting system, Calendaring functionality is integrated with the leasing system, etc
- Building vertical products generally provides a UX that is easier to use and of higher value to the user.
- Appfolio’s mission is to provide a portfolio of applications to a variety of verticals
    - Today we serve Property Management and Legal
    - Tomorrow many more

Appfolio Company Highlights

- Founded in 2006 by Klaus Schauser and Jon Walker
- About 600 employees, of those about 100 are engineers
- Headquartered in Santa Barbara with offices in San Diego and Dallas
- Number of customers
  - Property Management: ~10,000 companies
  - Legal: ~10,000 companies
- Number of “units” managed by Appfolio: ~3 Million
- Electronically transact ~$20 billion dollars in rent annually
- Facilitate background check on ~2 million people annually
- ~20 million HTTP requests a day
- Publicly traded, market cap ~$1.5 billion

Technology Overview

- Front end
- Application Layer
- Data Layer
- Operating System
- Hardware

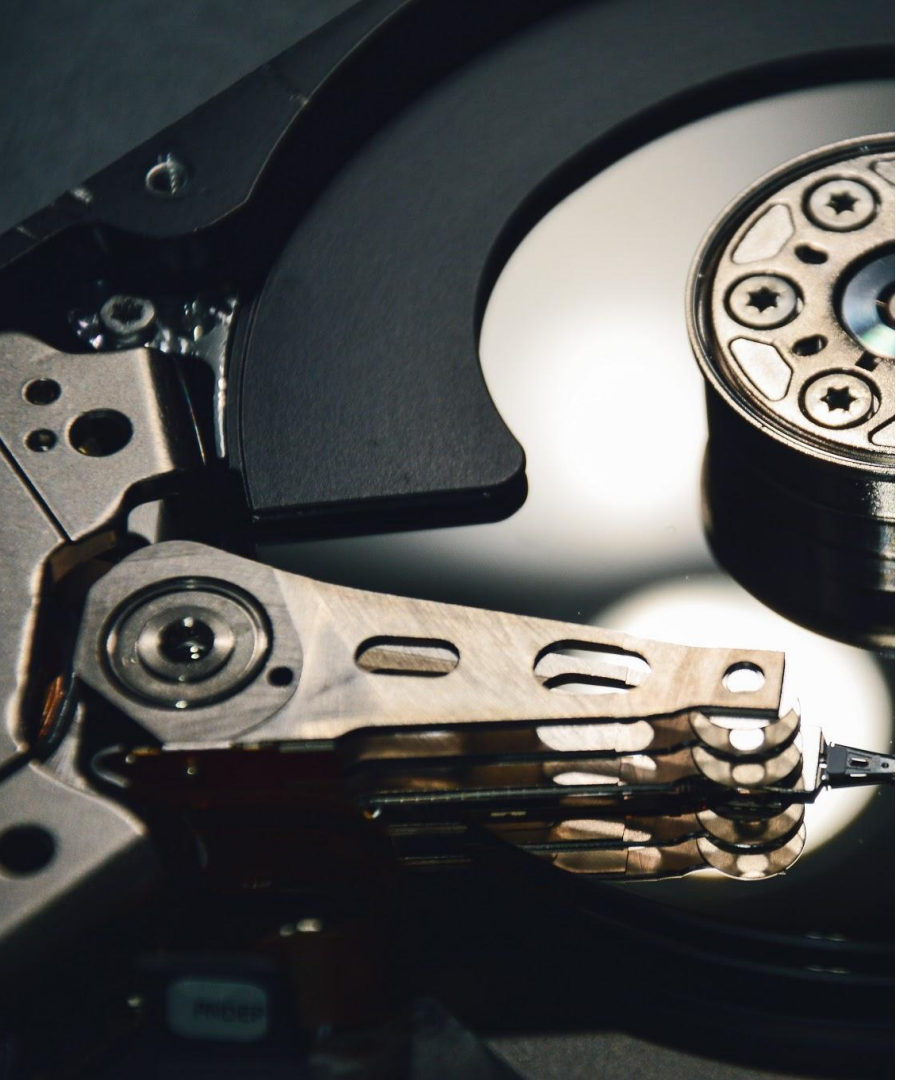


Technology Overview: Front End

- Traditionally markup is generated on the server, but increasingly on the client
  - Client-side UIs built in React
  - We use MobX and Redux to manage React state
  - REST/JSON to communicate to back end
  - ES2015, ECMA stage 2
- Mobile apps built with React Native
  - Allows devs to write native apps using javascript and the same framework as client-side web
- Mobile apps use web views and responsive web design for many parts of the UI
  - This allows our developers to write a feature once and have a high quality experience across traditional web, mobile web, iOS app and Android app
  - Web views aren't perfect, but this solution is perfect for our product and business
- Styling is done using a bootstrap-style shared component library

Technology Overview: Application Layer

- All of our applications are built using Ruby on Rails
- Why Ruby?
  - Languages can be thought of as on a spectrum, productivity vs. performance
  - The right place to be on this spectrum is determined by your product and company
  - Appfolio users are high revenue-per-request, so the cost of operating servers is a tiny fraction of the revenue they generate
- Why Rails?
  - High developer productivity: convention over configuration allows us to go fast
- App servers run phusion passenger
  - Passenger supports multi-threaded, but we do multi-process. Why?
- Nginx sits in front of passenger, but passenger speaks HTTP. Why do we need both?

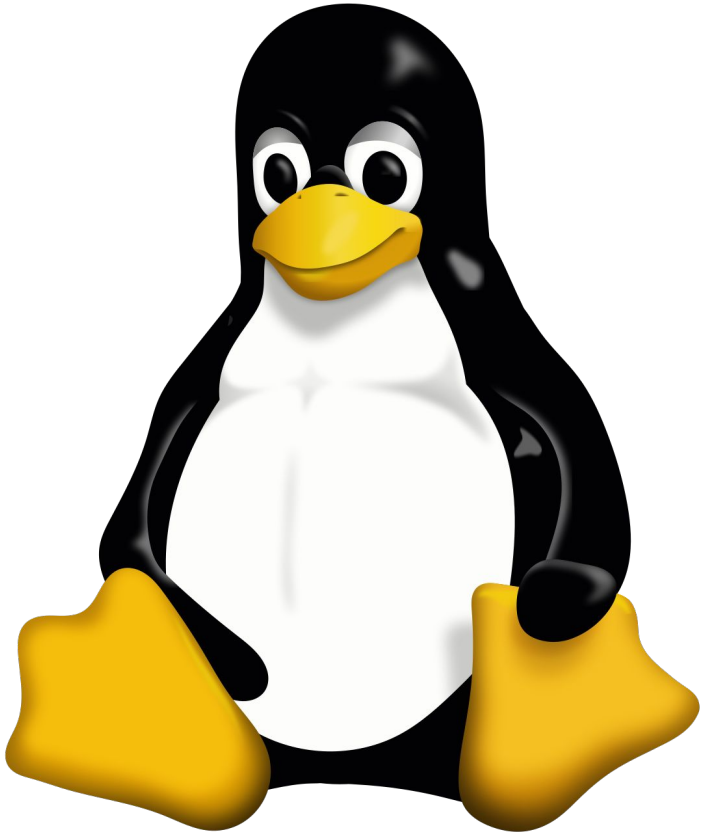


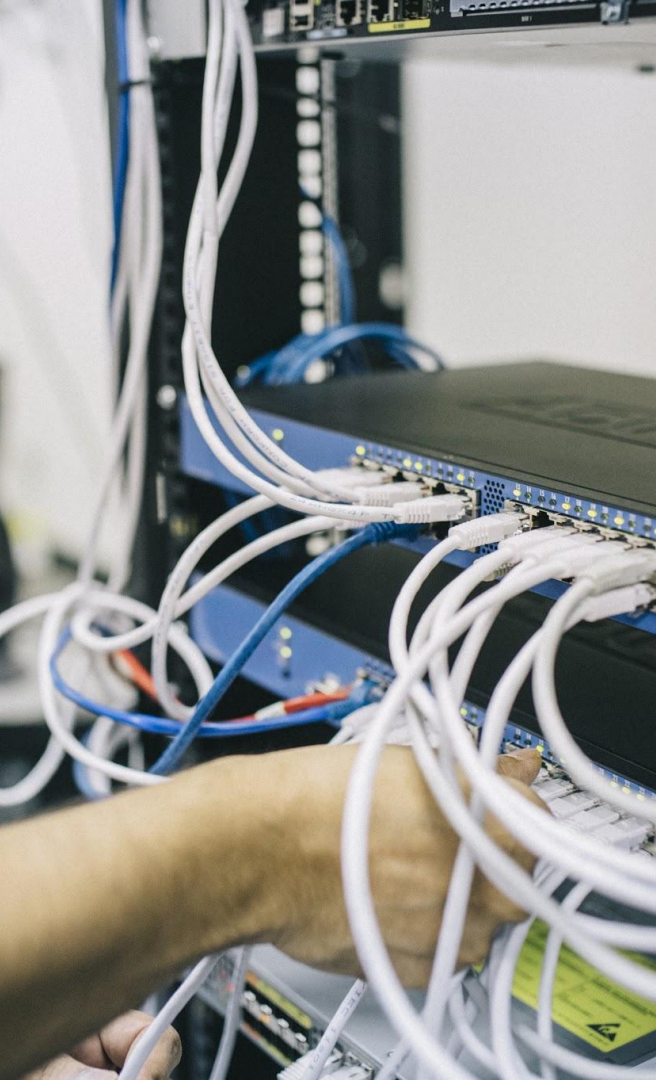
Technology Overview: Data Layer

- MySQL for relational data
  - Each customer's data is stored in a separate relational database
- Memcached for caching
  - Caching used extensively to reduce trips to the database
  - Permissions, frequently used accounting data, etc
- RabbitMQ as a message bus
  - Our asynchronous messaging is primarily used for asynchronous jobs
  - We don't generally rely on asynchronous messaging between services
- S3 for large file storage
  - No customer limit on storage
  - Users can attach arbitrary files to entities
  - As system of record, this gets used extensively
- SOLR for search
  - Application uses text search extensively (global search, entity selection)
  - SOLR uses Lucene under the hood
  - Creative use of ORM & SQL to get high performance SOLR-MySQL sync

Technology Overview: Operating System

- All our servers run CentOS Linux
- Transitioning to run our processes inside containers.
  - What are containers?
    - Filesystem and network isolation for unix processes
  - Why containers?
    - Increased isolation between apps
    - Developers can build in any language
    - Increased reproducibility between prod and dev
  - Docker vs. Rocket
  - Orchestration
    - Kubernetes vs. docker compose vs. nothing





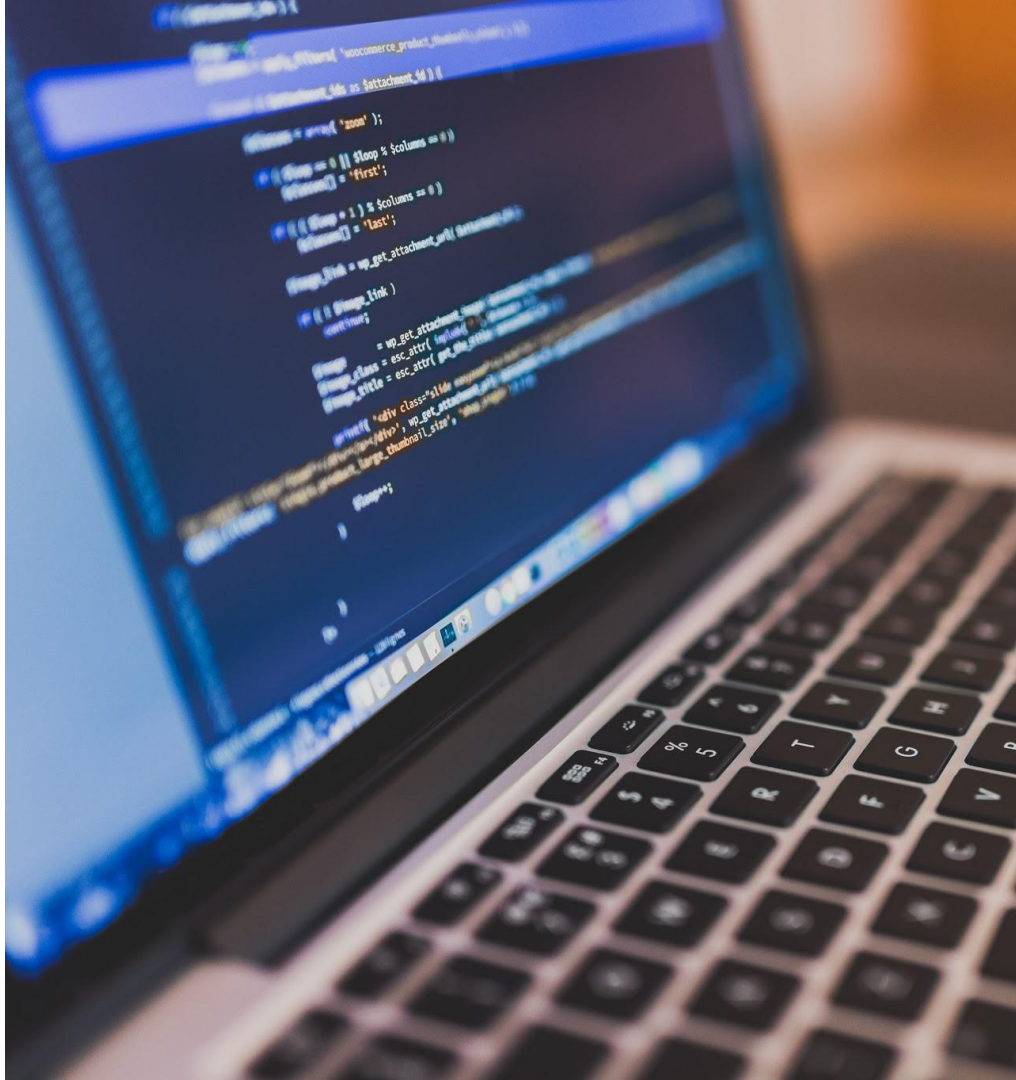
Technology Overview: Hardware

- APM runs in server colocation facilities
  - Dallas and Denver
  - ~200 servers
  - For load balancing and request routing we use Citrix Netscaler
  - For quick (but not cheap) durability, we use NetApp Filer
  - For load balancers, routers, switches, and filer we have two of everything
    - If any one of these devices fails, we can run on the backup until we can physically replace the failed hardware
  - For application servers and databases, we have two possible failover solutions
    - We can bring up the app servers and databases on another machine in the local datacenter
    - Or we can fail over to the twin machine in the remote data center
    - If a datacenter fails, we failover all machines to the other datacenter
    - How do clients know to route requests to Denver after a failover instead of to Dallas?
- Dell PowerEdge R630
  - Intel Xeon E5-2650 2.2GHz, Hyper-threading, 12 Cores, 24 Threads
  - RAM: 128GB
  - Disks: 3 hot-swappable 1.2GB, 10k rpm

- MyCase is in AWS.
  - Why is it different?

Engineering Overview

- TDD
- Agile/Scrum
- Release Cycle



Engineering Overview: Test-driven Design

- Test-driven development:
  - Write the test (watch it fail)
  - Write the code (watch it pass)
  - Refactor (tests should pass)

- Goal: by only writing code that has a corresponding test, you are guaranteed good test coverage.

- Third step (refactor) is very important and frequently forgotten
  - Without it, you get a very organic design.

- AppFolio has an extremely large rails app (~300K lines of code)
  - Heavily tested code is extremely valuable (we have more test code than actual code)
  - Currently releasing on a cadence that would be impossible without large test harness

Engineering Overview: Test-driven design

- TDD advocates can be dogmatic about test-first, we are more pragmatic
  - Since the goal is high test coverage, the order we do this is not important as long as we get high coverage

- AppFolio tests at every level:
  - Unit tests: Is a given object or model class working as designed?
  - Functional Tests: Is a web request working as designed?
  - Selenium Tests: Is a sequence of browser actions across the app working as designed?
    - Page objects keep this manageable
  - JavaScript Tests: Are our Javascript methods behaving correctly
  - RPC integration tests: Are our separately deployed services interacting correctly?
  - Saved report changes: Is this version of the application compatible with previously saved data?

Engineering Overview: TDD

- Integrating disparate changes is hard if you do it rarely
- Basic idea: check in often and run entire test suite each time
    - Goal: developers instantly know when they are introducing bugs into the system and can fix immediately

- All features are built on feature branches. CI evaluates these branches before merging.
    - Developers merge their work to master as frequently as possible.

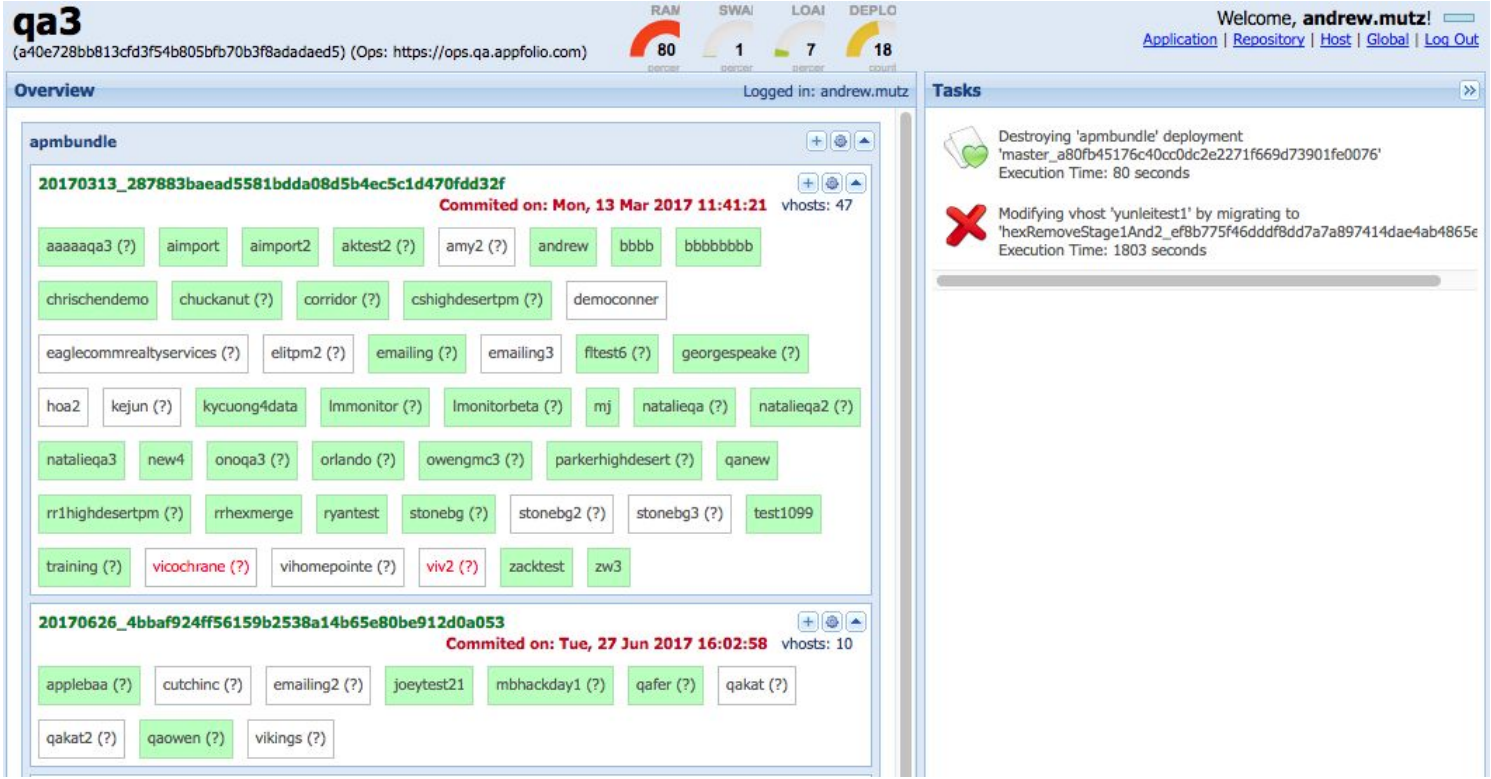
- Continuous Integration is the backbone of our development process
    - Spend a lot of time and money to make it fast and easy
    - Automated tests are so important, we have more test code than production code
    - Automated tests are so important, we have about as many CI servers as production servers

- Each time a developer checks in, tests are run
    - Automatically deployed
    - Testing servers automatically migrated

Engineering Overview: Release cycle

- We release daily to a small group of customers
- We release every week to all customers
- Technically, we don't do continuous delivery, but all changes that are merged to master and pass CI can get released.

- So if we are releasing daily, how do we build features that are large and naturally take weeks or months to build?
  - Application has notion of "Experiments"
  - Developer creates a feature toggle that changes application behavior
  - toggle is set per-customer
  - Experiments are integrated with our deployment tools
    - We do slow rollout of features to get the right amount of feedback at the right time
  - Experiments are integrated in development workflow
  - Experiments are not an AB testing framework, but sometimes are used as such



Engineering Overview: Agile/Scrum

At Appfolio we work in agile teams

- 2-4 developers
- 1 QA Engineer
- 1 product manager
- 1 UX expert

Teams are feature teams, not component teams

- Teams are tasked with solving a problem
- Teams have high degree of freedom to solve problems as they see fit
- No code ownership at Appfolio, teams can make changes to any part of our application

Teams are organized into colleges

- A college has 2-5 teams

Appfolio Property Manager has 5 colleges and 20 teams

Process is heterogeneous

Scaling lessons from industry

#1 Don’t solve the general case
#2 Keep it simple
#3 Never forget the customer

Challenges Ahead

- Technology is moving beyond mobile apps and web browsers
  - Package Management
  - Smart Door Locks
  - Voice Telephony
  - Alexa
  - These have different techn problems
    - Vastly different user interface
    - Uptime guarantees need to be much stronger

Challenges Ahead

- Data and Machine Learning
  - Tools are becoming commodi
    not
    - Our business produces lots of valuable structured data
  - Examples
    - Screening Tenants
    - Understanding Paper Documents
    - Rent Optimization

Challenges Ahead

- Scaling load is sometimes easier than scaling humans
  - Conways law
  - Service Oriented Architectures, Microservices
  - How do you make thir maintainable?
  - Does SOA introduce m problems than it solves?

We are hiring full time and interns!

appfolio.com/jobs

Thank you!

Questions?