Fast, Reliable, Secure

June 17, 2015
About me

• Partner @ Scale Venture Partners

• Operations & Cloud Engineering @ Netflix

• Founder CTOWorks

• Engineering / Products / Technology @ Startups (5-150 people)

• Engineering @ Siebel Systems / Oracle Corp

• UC Berkeley CS / Wharton MBA (Finance & Marketing)
About Scale Venture Partners

• >$1B total under management

• Spring 2013, ScaleVP IV, L.P., $300M

• Early-in-revenue enterprise software companies

• SaaS and Cloud Infrastructure

• Aviso, Agari, Box, BrightRoll, Chef, Crittercism, DataStax, Demandbase, DocuSign, ExactTarget, HubSpot, PubNub and RingCentral
Who cares about DevOps?
ESSAY

Why Software Is Eating The World

By MARC ANDREessen
August 20, 2011

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoning its struggling PC business in favor of investing more heavily in software, where it sees better potential for growth. Meanwhile, Google plans to buy up the cellphone handset maker Motorola Mobility. Both moves surprised the tech world. But both moves are also in line with a trend I’ve observed, one that makes me optimistic about the future growth of the American and world economies, despite the recent turmoil in the stock market.

In short, software is eating the world.

More than 10 years after the peak of the 1990s dot-com bubble, a dozen or so new Internet companies like Facebook and Twitter are sparking controversy in Silicon Valley, due to their rapidly growing private market valuations, and even the occasional successful IPO. With scars from the heyday of Webvan and Pets.com still fresh in the investor psyche, people are asking, "Isn't this just a dangerous new bubble?"
Everything-as-a-Service

Alphabet soup

• IaaS
• PaaS
• DaaS
• mBaaS
• SBS
• …

All started with SaaS

With great power comes great responsibility

(Consumer web companies figured this out long ago)
Competition

Software is a at the core of every business, not just software companies

Ability to deliver software to market quickly is a competitive advantage
Accelerating Pace of Innovation

- Quality
- DevOps
- Cloud

Rate of Innovation
What is DevOps?
DevOps Core Components

1. Self-service / Decentralization
2. Automation
3. Collaboration
Software Lifecycle

Develop

Operate

Test

Deploy
Develop

Self-provisioning & self-service

Decentralized, loosely coupled & tightly aligned

Operational platform that provides
- Load balancing
- Service discovery
- Logging, metrics, alerting, and notifications
- Resiliency design patterns (bulk heads & fallbacks)
- Security
- Data access (caching, persistence)
Test

Unit / functional / regression / integration

Chaos (failure injection / simulation)

Performance / Squeeze

Security

Automate as much as possible (all!)
Deploy

Continuous Delivery

Automated configuration (bake or bootstrap)

Zero-downtime (red/black or rolling)

Canary Analysis

Redundancy
Deploy: Continuous Delivery

Replace manual release processes with automation

Assembly line

Forcing function for continuous improvement
Example Continuous Delivery pipeline

* Sangeeta Narayanan, Move Fast; Stay Safe: Developing & Deploying the Netflix API
Deploy: Automated Configuration

“Servers Aren’t Pets; They’re Cattle”

- The old: curated, hand-created infrastructure
- The new: disposable, elastic, self-configuring
Deploy: Automated Configuration

Two approaches

1. Bake virtual images or containers
2. Bootstrap at launch
Deploy: Zero downtime

Two options

1. Rolling upgrade

2. Red/black deployment

Must be able to rollback quickly!
Deploy: Canary Analysis

Always, always stage deployments and compare them to baseline before fully deploying

Launch 2 instances, one with new code one with old. Compare them. If good, continue deploying new

Lots (183 slides!) of detailed tips & tricks from QConNY at: http://www.slideshare.net/royrapoport/20140612-q-con-canary-analysis
Deploy: Redundancy

Think in 3s

Multiple data centers (AZs)

Multiple regions

Active-active
Operate

Monitoring
Alerting
Notifications
Dynamic auto-scaling
Failure
Operate: Monitoring

Collect everything

Prefer structured time series over log data

Build base health metrics into the platform (e.g. 2xx, 3xx, 4xx, 5xx counts, latency percentiles)
Operate: Alerting

Self-service, based on meaningful service metrics

Error rates often a better metric than success
Operate: Notifications

Don’t rely on people to watch screens. Alerts should trigger notifications

On-call rotations should include engineers that built the service

Alerts should be granular down to service
Operate: Dynamic auto-scaling

Save costs

Increase availability
Operate: Resiliency through failure
Failure is all around us

- Disks fail
- Power goes out. And your generator fails
- Software bugs introduced
- People make mistakes

*Failure is unavoidable*
We design around failure

• Exception handling

• Clusters

• Redundancy

• Fault tolerance

• Fall-back or degraded experience

• All to insulate our users from failure

Is that enough?
It’s not enough

• How do we know if we’ve succeeded?
• Does the system work as designed?
• Is it as resilient as we believe?
• How do we prevent drifting into failure?

*The typical answer is...*
More testing!

- Unit testing
- Integration testing
- Stress testing
- Exhaustive test suites to simulate and test all failure mode

Can we effectively simulate a large-scale distributed system?
Building distributed systems is hard

Testing them exhaustively is even harder
• Massive data sets and changing shape

• Internet-scale traffic

• Complex interaction and information flow

• Asynchronous nature

• 3rd party services

• All while innovating and building features

Prohibitively expensive, if not impossible, for most large-scale systems
What if we could reduce variability of failures?
There is another way

- Cause failure to validate resiliency

- Test design assumption by stressing them

- Don’t wait for random failure. Remove its uncertainty by forcing it periodically
And that’s exactly what we created
Instances fail
Chaos Monkey taught us…

• State is bad
• Clusters are good
• Surviving single instance failure is not enough
Lots of instances fail
Chaos Gorilla
Chaos Gorilla taught us…

- Hidden assumptions on deployment topology
- Infrastructure control plane can be a bottleneck
- Large scale events are hard to simulate
- Rapidly shifting traffic is error prone
- Smooth recovery is a challenge
- Cassandra works as expected
What about larger catastrophes?

Anyone remember Sandy?
Chaos Kong
The Sick and Wounded
Latency Monkey
HYSTRIX
Defend Your App
http://github.com/Netflix/Hystrix
Hystrix, RxJava
http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html
Latency Monkey taught us

- Startup resiliency is often missed

- An *ongoing unified* approach to runtime dependency management is important (visibility & transparency gets missed otherwise)

- Know thy neighbor (unknown dependencies)

- Fallbacks can fail too

- It’s hard to be ready for Latency Monkey, especially with no customer impact
Entropy
Clutter accumulates

- Complexity
- Cruft
- Vulnerabilities
- Cost
Janitor Monkey
Janitor Monkey taught us…

- Label everything
- Clutter builds up
Ranks of the Simian Army

- Chaos Monkey
- Chaos Gorilla
- Chaos Kong
- Latency Monkey
- Janitor Monkey
- Conformity Monkey
- Circus Monkey
- Howler Monkey
- Security Monkey
- Efficiency Monkey
A recent community addition: Infected Chaos Monkey

The Infected Chaos Monkey

We believe that modern datacenters should deploy security mechanisms that do not rely on the fact that no host past the perimeter ever gets compromised. Rather, have a built in “immune system” in place which can handle breaches as they happen.

Posted April 14, 2015 by pavel

A couple of years ago Netflix introduced a concept they called a "Simian Army". The idea was to have a bunch of automated processes that checked their cloud’s resilience to various failure scenarios. A prime example was a “Chaos Monkey” which randomly shuts down servers in their infrastructure to test the application’s ability to withstand server failures. When you know that a Chaos Monkey is running free in your infrastructure and your service stays up you know that you are can handle server failure effectively. We think that a similar approach applies to securing cloud infrastructure.
Observability is key

• Don’t exacerbate real customer issues with failure exercises

• Deep system visibility is key to root-cause failures and understand the system
Organizational elements

- Every engineer is an operator of the service
- Each failure is an opportunity to learn
- Blameless culture

Goal is to create a learning organization
The Good & The Bad
Pros & Cons

- Increased engineering velocity
- Faster rate of innovation
- Improved security
- Increased availability

- Investment into tooling and automation
- Change in culture & org
  
  Higher risk


What is Risk?
Modeling the future
More formally

\[ S = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n - 1}} \]
Another definition

A weighted probability distribution of negative events

Risk = \sum \text{Likelihood of adverse event} \times \text{Impact of event}
Sources of risk in software

- Software bugs
- Infrastructure failure
- Dependency failure
- Capacity
- Operator error
- Malicious attack
DevOps Reduces Risk!

Likelihood of adverse event \times \text{Impact of event}
Risk reduces because…

Likelihood decreases

- Testing frequency increases to every checkin
- Greater coverage with automation
- Instant process improvement through automation, auditable and repeatable
- More automation = less operator error
- Security & stress testing part of code delivery

Impact decreases

- Higher velocity = faster recovery
- Higher velocity = faster patches
- Fewer changes per deployments
- Less time between deployments (time is the greatest enemy of causality)
- More context in operations
- Resilient design reduces impact of dependency and infrastructure failures
Summary

Software is becoming ubiquitous,

delivered as an always-on service,

and is at the heart of enabling business innovation.

Every organization is looking for ways to accelerate innovation.

DevOps accelerates innovation while reducing risk and improving security.
Q & A

Questions?

You can reach me at
⇒ ariel@scalevp.com
⇒ www.linkedin.com/in/atseitlin
⇒ @atseitlin