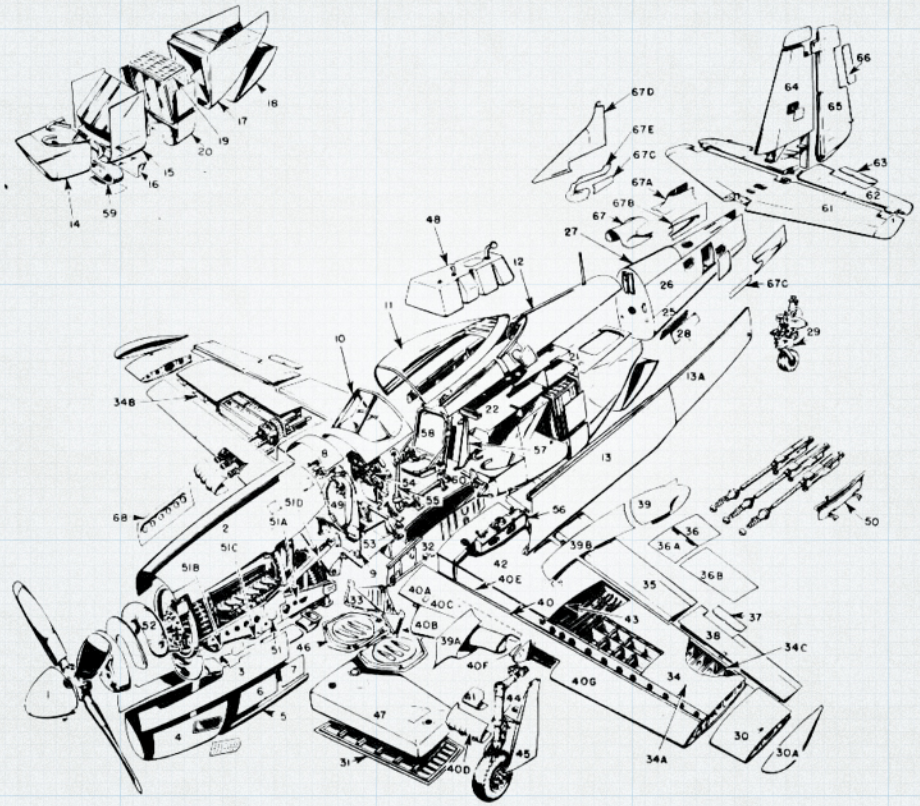


FIXING THE FLYING PLANE

Major SAAS Upgrades by a Production DevOps Team

Introduction

Calvin Domenico
Director

Jesse Campbell
Sr. Software Engineer, Lead of Development

Marie Hetrick
Manager of Hosting

Alastair Firth
Software Engineer

Elijah Aydnwylde
Sr. Sysadmin, Lead of Operations

Brandon Arsenault
Project Manager

Patrick McAndrew
Sr. Sysadmin, Lead of Infrastructure

The “Before” Environment

- ~20 custom-developed services accessed by 10,000+ school districts nationwide
- Software not designed for SaaS
- Virtualized environment in Managed Hosting datacenter limited visibility and prevented admin access to infrastructure

The “Before” Environment

Problem Scenario

- Customers reporting networking issues
- Troubleshooting isolates load balancer
- MSP says it can't be

Solution

- Bypass the load balancer

Cost

- Lost customers
- Man-weeks of troubleshooting and workarounds (attempts to work with MSP almost doubled this)

OPERATORS

can't

OPERATE

if they can't

SEE

The Project

- **SOLVE** the Managed Services problem without incurring the business and man-hour costs of colocating
- **DESIGN** a datacenter for the purpose of serving this specific software as SaaS
- **PLAN** up to 5x growth within 2 years, as well as upcoming changes to the software (i.e. clustering)
- **PROOF** the new datacenter in a local virtualized environment so that as much of it as possible can be "ported" directly to the new hardware

The Challenge:

DON'T LAND THE PLANE

The Challenge

- **One week of total downtime** for all operations
- **Six months maximum** limit for datacenter design, code development & implementation
- **Design**, **Build**, **Code**, **Upgrade**, and **Migrate** all at once!

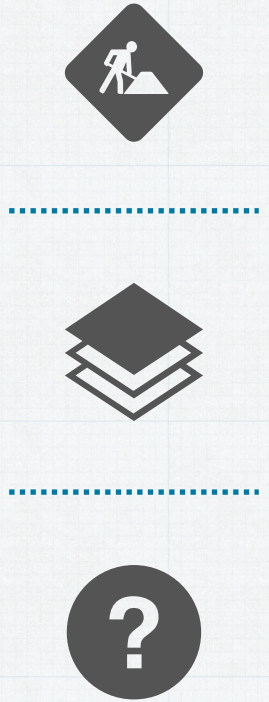
The

DEVELOPMENT

The Development

Requirements

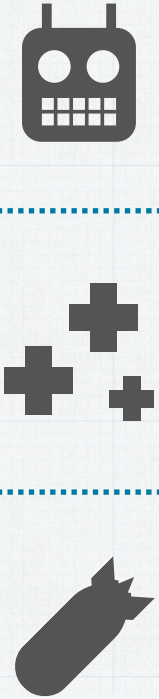
- What to build?
  - Manage multiple layers
    - Virtual Infrastructure
    - Machine
    - Application
    - Data
- Why should we build it?



The Development

What Did We Build?

- Automated Control engine for existing technologies
  - NFS, Git, Puppet, VSphere, bash, perl
- Unified control front-end
- Extensible framework
- No recovery: destroy and rebuild
- Easy to pick up and create a new complete stack



The Development

The Team

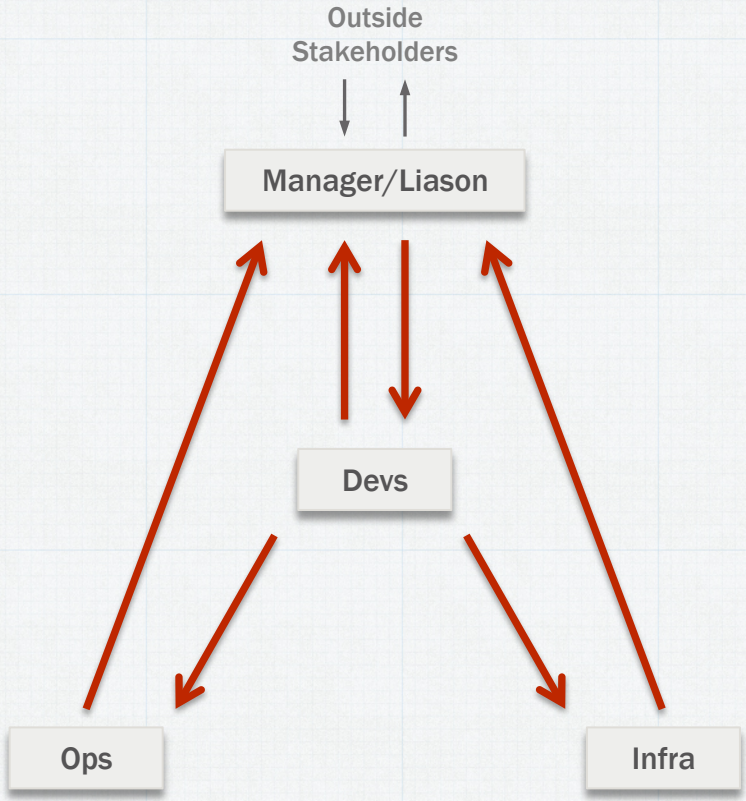
- Methodology
- Mentality
- Motivation
- Personality

- Ownership?
- Who writes the spec?

The Development

The Dev Environment

- Tight schedule
  - Fast iterations
    - Design, Develop, Deploy, Destroy
  - Feature driven design
- Communication
  - Oversight / insight
    - Single point of contact
  - Open access for devs
  - Appeasing stakeholders
    - Legitimate concerns



THEN

and

NOW

Then and Now

Time to Create and Deploy a Site

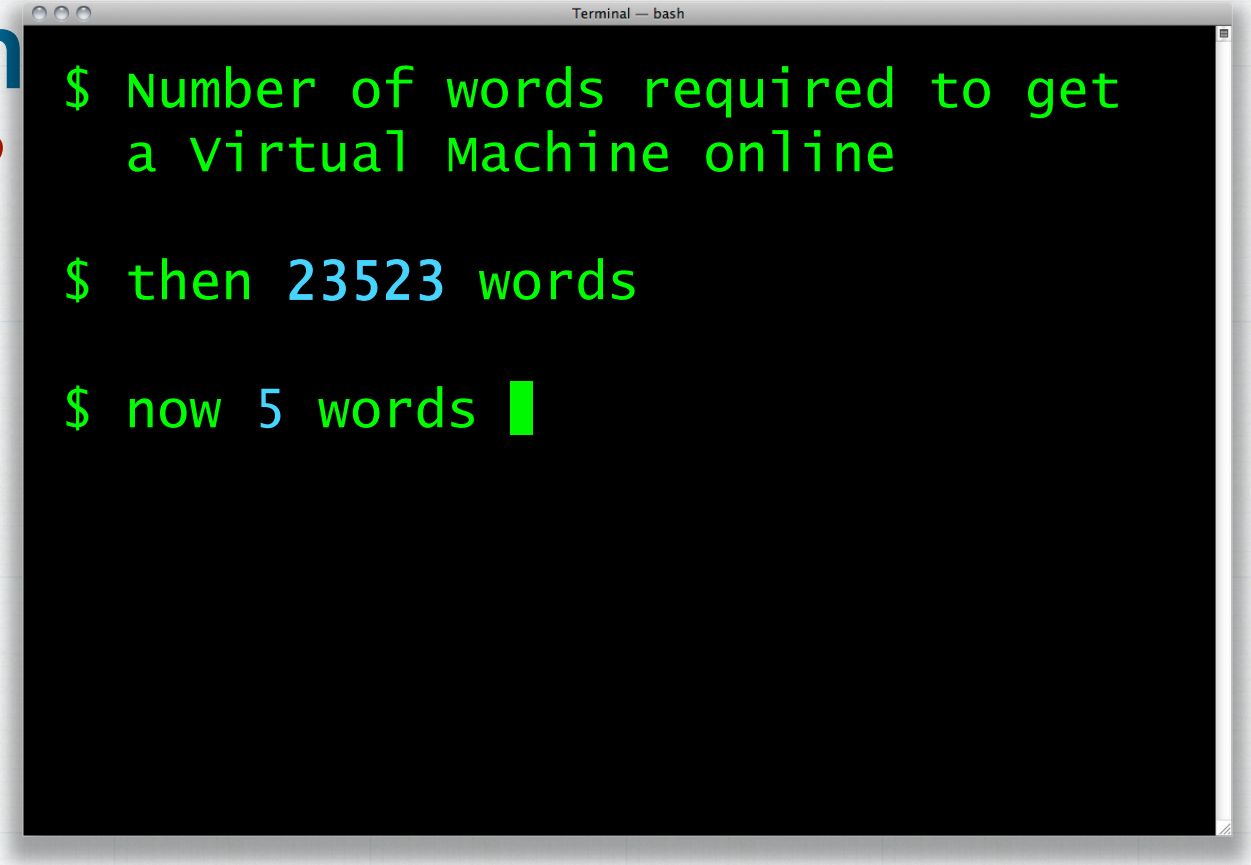
3–5 DAYS

Vs.

24 HOURS

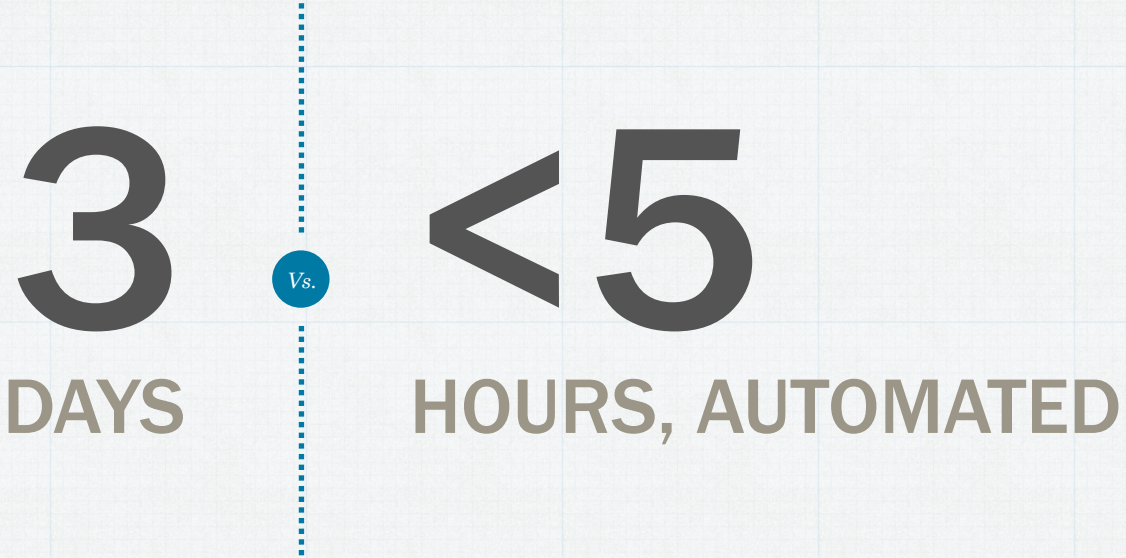
Then

Time to


Terminal — bash
$ Number of words required to get
a Virtual Machine online
$ then 23523 words
$ now 5 words

Then and Now

Time to Configure an Application Server



Then and Now

Time to Configure a Database Server

1 WEEK

Vs.

<5 HOURS, AUTOMATED

Then and Now

Time to Deploy a Patch (Hours)

Then and Now

Time to Re-balance Database Layer

1.5 *Vs.* **4/4**

MONTHS OF OVERTIME

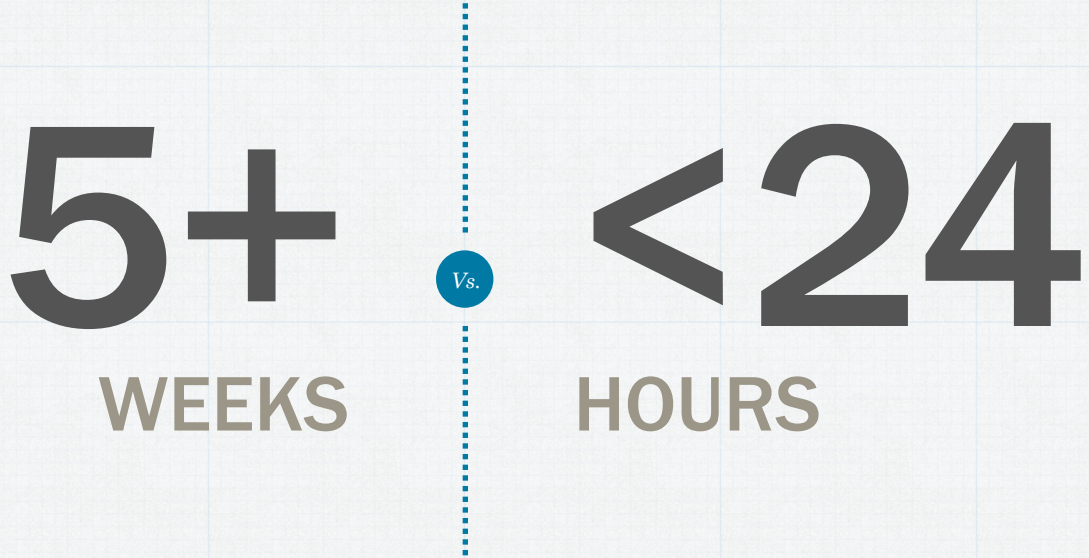
2 People

DECISION-MAKING/4 HOURS REVIEW

Automated

Then and Now

Time to Recover Our Entire Environment



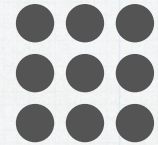
how did it all

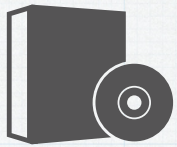
COME TOGETHER?

How Did it All Come Together?

Abstracting Enterprise Components

- Abstracting System and Software Components
  - What are our Software Components?
    - Application Agents
    - Customer Databases
  - What are our System Components?
    - Application Servers
    - Database Servers

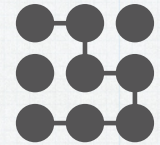




How Did it All Come Together?

Abstracting Harder

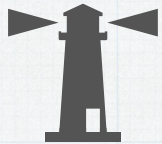
- What are the relationships between these components?
- How can they be abstracted?
  - **Cluster**
    - A selection of Customers grouped together and handled by a single Agent
  - **Node**
    - An instance of a cluster running on an Application Server
- What do these abstractions allow us to infer by relation?



How Did it All Come Together?

Agile Development

- Adaptable to
  - Unknown Performance and Needs
  - Changing Requirements
- High Visibility provides
  - Decreased Risk
  - Increased Business Value
- Collaborative Design promotes
  - Diverse Viewpoints
  - Shared Experience




end