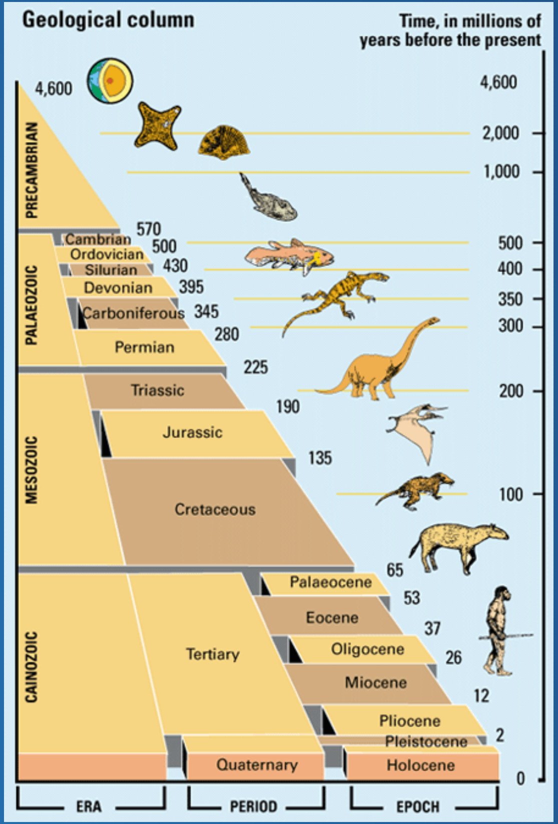
A Tale of Automation

Chris Chalfant
Andrew Kaczorek

5 Years of Apps on AWS

- The Age of Chef
- The Age of CloudFormation
- The Immutable Architecture Diversion
- The Age of Elastic Beanstalk
- What Have We Learned?
- The Future?



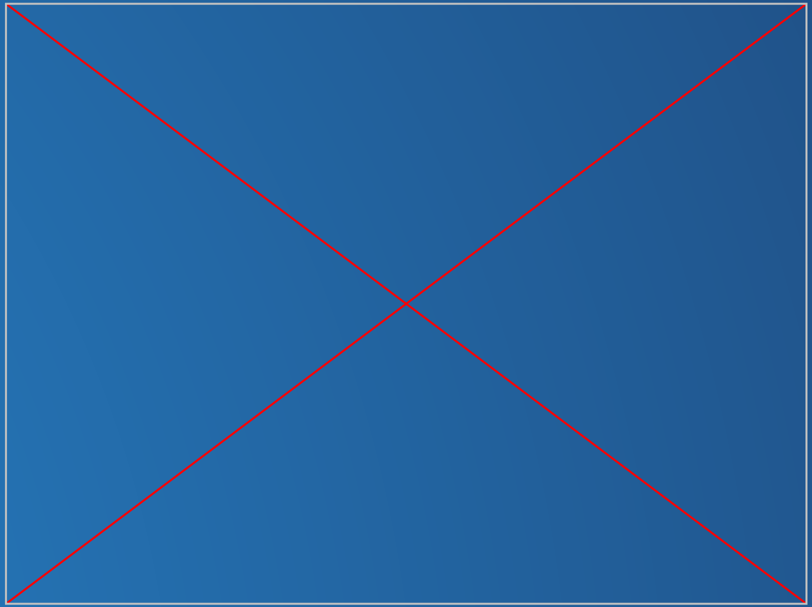
Age of Chef: Use Case

Clusters of 10s-1000s of worker machines for HPC.

Should start in <30 minutes regardless of size.

Should start doing work as fast as possible (reduce cost overhead)

Clusters should allow the installation of customer-specific worker software



The Age of Chef

- Image creation from empty disk and bash scripts
- Ran Chef Server(s) to support simultaneous launches of 1000s of instances
- All AWS resource management was custom python/boto/django code in a sophisticated state machine
- Custom bootstrapping using encrypted user data

Pros

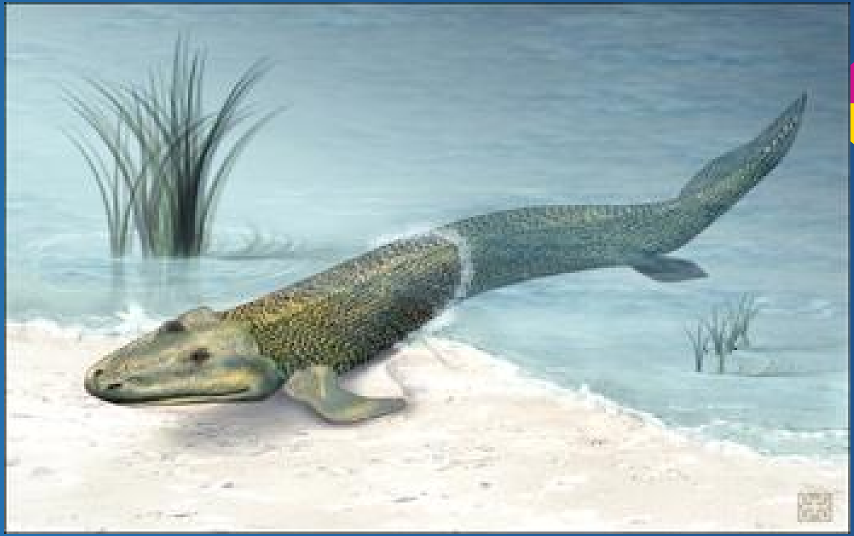
- Simple environment w/ few “roles”
- Chef replaced manual labor and shell scripts

Cons

- AMI creation slow, relies on good documentation/scripts
- Chef server scale-out was a PITA
- State machine creaky, scales poorly

We Evolved a Little Bit

- Got very good at Chef.
- Allowed the same AMI to grow up to be many different instances.
- Moved to chef-solo. No more Chef servers. Cookbooks served from S3.
- Leveraging the scale and reliability of S3 made everything better.



The Age of CloudFormation: Use Case

Green field development driven by our new “Cloud Manifesto”

SOA: Stateless, horizontally-scaled, load-balanced web servers on Ubuntu

The Age of CloudFormation

- Define a huge environment with a single json document
- Not just VMs: network, ACLs, security groups, databases, ALL THE THINGS
- Early Cloudformation created resources serially: so we split templates to run in parallel
- The killer combination was adding another piece of magic...

Pros

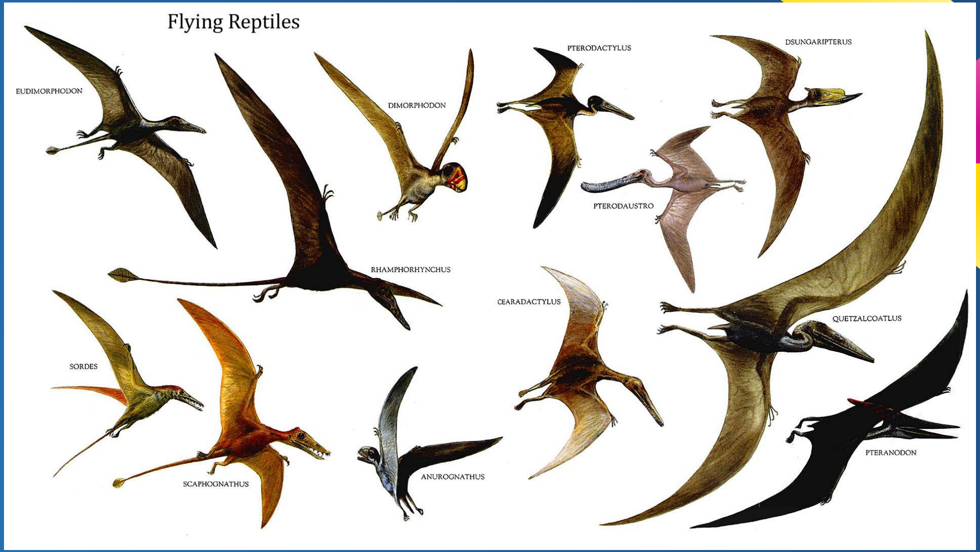
- Everything is defined in code!
- We launch the whole environment with one script!
- We are true to the Manifesto!

Cons

- Massive json documents with common code
- Complicated bootstrap: how do we get from new instance to running chef?

CloudInit Appears

- No longer did we need to write our own custom instance bootstrapping
- Came standard on several OS images (we like Ubuntu)
- This was the perfect way to give a machine personality and then let chef-solo take over.



Radical Evolution

Have a Manifesto and stick to it.

Define your philosophy and best practices. Hold them dear.

Automate everything. All resources should be managed by code.

Tear everything down and rebuild it. Every day. To prove you can.

IAM Instance Roles means never having to pass in AWS Credentials

Then...

Interlude: Some Notes About Chef

We don’t feel strongly about config management holy wars. So just pick one.

Many community cookbooks are outstanding, but some get extremely complicated and brittle

Chef Dev workflow still seems clunky, though its gotten much better

Complicated machines can take a long time to converge. So...

A Brief Affair with Immutable Architecture

Immutable Servers: Never make changes to machines. Destroy and replace them.

Our use case: Single .NET web server, single database. Dead simple.

- We used a combination of CloudFormation and custom scripting to stand up machines, configure them, and bake images in a build
- Why? faster launch, no external dependencies, treating machines like disposable units

Netflix does it!

The Affair: It’s Not You, It’s Me

- The builds take a long time. Developers become restless.
- Hotfix? New image. Really?
- The instance launch time is dominated by OS boot time (thanks, Windows)
- Bottom line: premature optimization. We ultimately ripped it out.

This is still a viable option for other use cases.

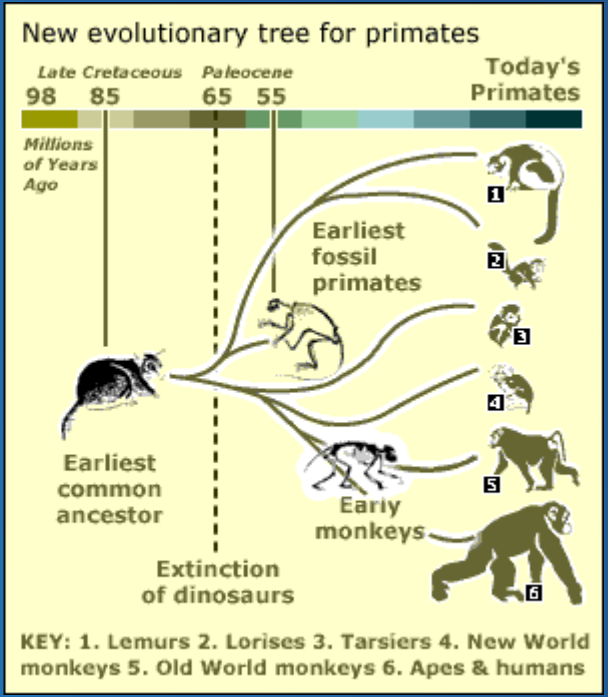
The Age of Elastic Beanstalk: Use Case

.NET application, built up of several services running on IIS.

Central SQL Server database.

Desire for Continuous Deployment.

We have all these ruby scripts laying around from the last gig...



The Age of Elastic Beanstalk

Beanstalk abstracts away traditional load-balanced, autoscaling web server cluster.

But it still uses CloudFormation.

We combined our scripts into an open-source gem called automan.

We layer on site-specific configuration with a separate private gem

Pros:

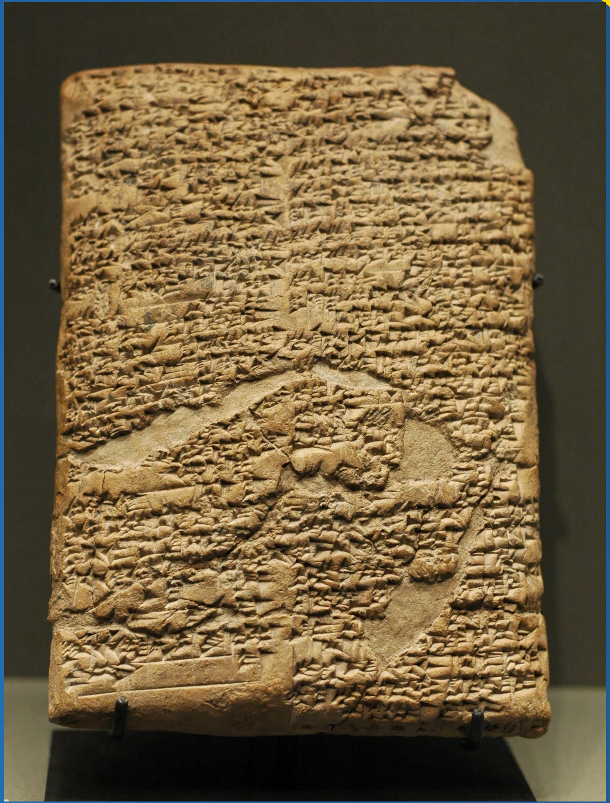
- Hides complexity of standard EC2 web server architecture
- Decent console support
- Baking our opinions into a gem means easier re-use

Cons:

- Simplification means a loss of control
- One app per cluster

What have we learned?

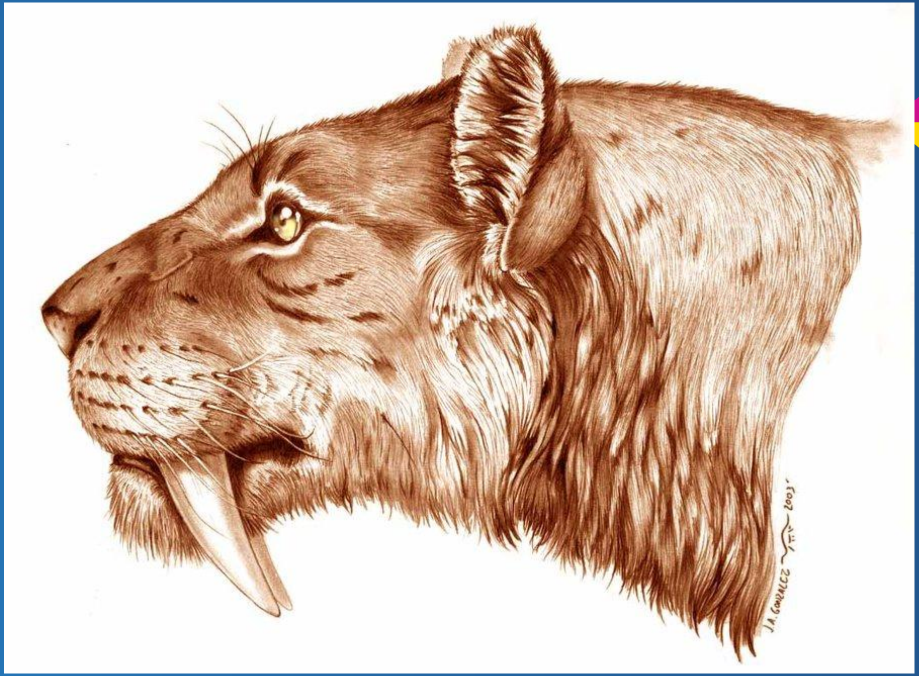
- Learn from, but do not copy The Big Guys
- Write your own Manifesto and follow it (until you write a new one)
- Build the simplest, most easily supportable environment for the current problem
- Automate all the things
- Monitor all the things
- Don't change more than One Big Thing at a time.



Evolutionary Dead Ends

For us anyway…

- DIY Orchestration Engines
- Server-based Config Management
- AWS OpsWorks
- Our version of Immutable Servers
- Service Auto-discovery
- DIY Monitoring
- DIY Logging



Use the right tool for the job!

The Future?

- Determine how Docker fits into our repertoire
- Elastic Container Service, Google Container Engine?
- Serverless architecture with one-page apps and/or Lambda
- Standardize practice of software VPN into VPC
- Other clouds: Azure, Rackspace, etc
- .NET apps on Linux?