Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components*

*Inspired by Chad Fowler, http://chadfowler.com/

Agenda

What is immutable infrastructure? Why? Why not?

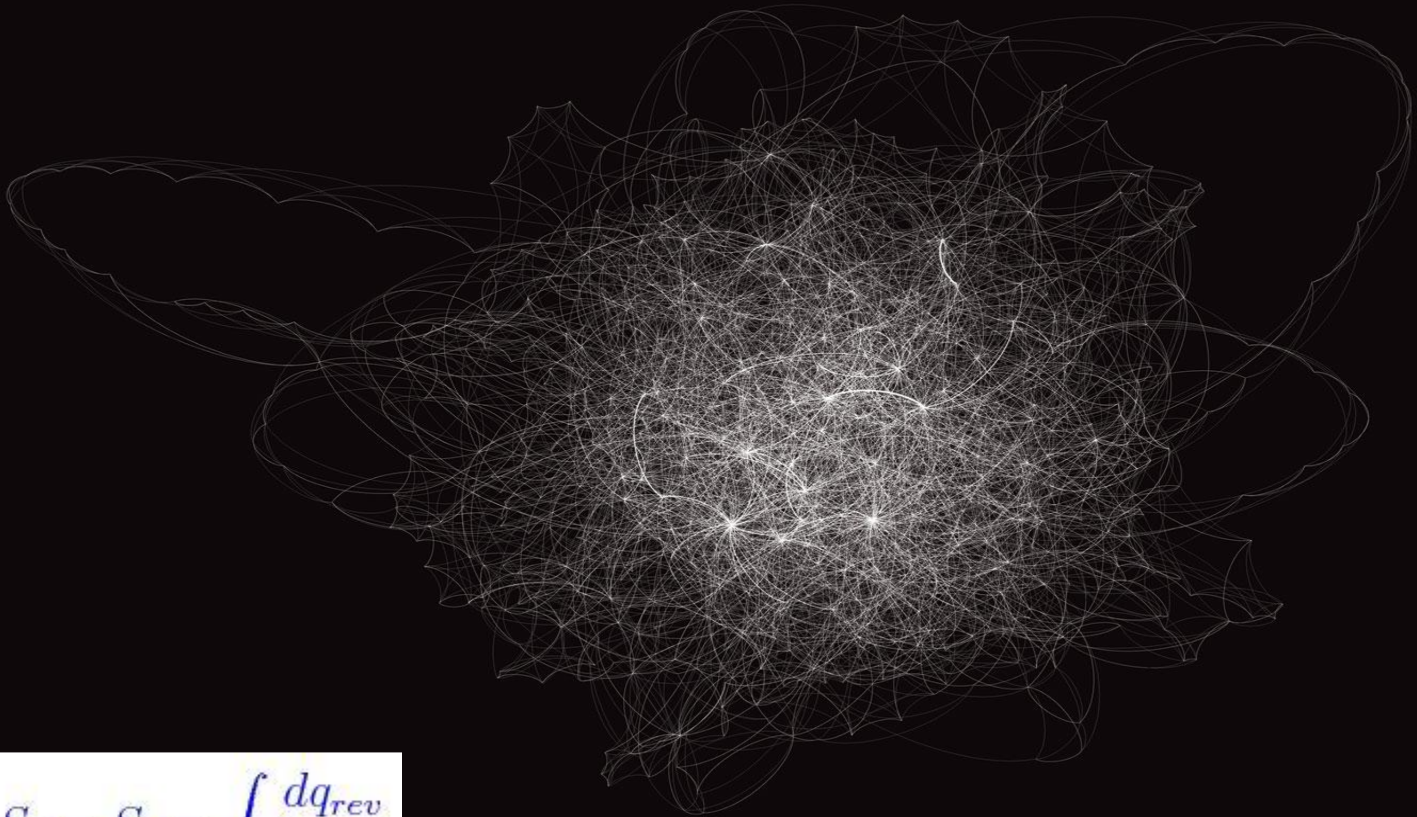
Building blocks for immutable infrastructures

Red Hat technologies involved

Managing immutable infrastructures?

Demos

The Problem



$$\Delta S = S_f - S_i = \int \frac{dq_{rev}}{T}$$

In plain English

“For an isolated system, the natural course of events takes the system to a more disordered state.”

In even plainer language?

ATOM

The Solution

Phoenix

In Greek mythology, a **phoenix** or **phenix** (Greek: φοῖνιξ *phoinix*) is a long-lived bird that is cyclically regenerated or reborn. A phoenix obtains new life by arising from the ashes of its predecessor. According to some sources, the phoenix dies in a show of flames and combustion, although there are other sources that claim that the legendary bird dies and simply decomposes before being born again.

(https://en.wikipedia.org/wiki/Phoenix_mythology)

What & Why?

What is immutable infrastructure?

1. Never change any part of your system once it is deployed. If you need to change it, deploy a new system.

2. Automate the setup and deployment for every part and every layer of your infrastructure.

Benefits of immutable infrastructures

Absolute certainty on the state of a server once it has been provisioned

Simplified testing and deployment. No difference between upgraded and new environments. No difference between staging environments.

Simpler roll-forward. Problematic servers will be destroyed.

Simpler roll-back. Just use last image.

Wait! Isn’t that what Ansible/Chef/Puppet promised?

Only the state of objects within these tools’ control can be guaranteed.

Writing and maintaining playbooks/recipes/manifests is time consuming, so most people tend to focus their efforts on automating the most important areas of the system, leaving fairly large gaps. (Pareto's Principle or 80-20 rule)

Challenges with immutable infrastructures

Practicing CI/CD will result in many new and parallel systems and therefore additional compute costs.

→ Cloud-like cost models should alleviate (but not neutralize) this

System upgrades are slower

→ Practice your automation kung-fu!

→ Clouds & Containers to the rescue!

Loss of local data.

→ Carve out data into persistent layer

Dependency problems due to name and address changes

→ Build service discovery layer

Challenges with immutable infrastructures

Replacement can impact/disrupt other systems

→ Proxies and queues can mitigate

Replacing traditional databases might be hard to impossible

→ noSQL alternatives? Offload to public cloud?

→ Break data into smaller junks

→ This is not a cure-all!

Fixing problems might be slower. Can’t SSH and fix. Need to redeploy.

→ Well this is why we doing this in the first place. Practice your kung-fu!

Possible loss of troubleshooting skills

→ Do you really care? Server+OS being heavily commoditized

How?

“An EC2 instance is not a server—it’s a building block.”

(Werner Vogels, CTO Amazon)

Building Immutable Infrastructures

While this could be done on bare metal, cloud technologies, linux containers, and the tooling around them really have allowed this new concept

Red Hat Technologies Mix

RHEL + Docker + Kubernetes : adding a lot of DIY a customer could build II on pure RHEL + Kube

RHEL Atomic Host : a true immutable atomic OS, either stand alone or as nodes for OSE / AEP

Openstack / RHEV : IaaS layer for container platform

OpenShift / Atomic Enterprise Platform

Satellite / Ansible : to manage Container infrastructure (OSE masters & nodes), to manage build process (build host consuming satellite channels)

Managing Immutable Infrastructure?

The future of configuration management systems is in deploying cloud infrastructure that will later run systems via an API level.

Manage ‘Build’ not ‘Run’

Demo

Immutable Infrastructures with Red Hat OpenShift

Demo video goes here ...