Re-imagining management methods for distributed and clustered systems

Kraken + Layercake

J. Lowell Wofford
Kevin Pelzel
Travis Cotton

Los Alamos National Laboratory

Video: LA-UR-21-24586
Slides: LA-UR-21-24563
Who We Are

❖ J. Lowell Wofford (presenter)
  ❖ Scientist, LANL HPC-Design.
❖ Kevin Pelzel (co-author)
  ❖ Scientist, LANL HPC-Environments.
❖ Travis Cotton (co-author)
  ❖ Scientist, LANL HPC-Systems.
The Problem
People are building lots of clusters...

- High-performance computing
- VM clusters & Private (or Public) Clouds
- Container Orchestration (kube, shift, swarm, oh my!)
- Storage & data clusters (ceph, mongo, etcd,...)
- ML/AI & Data analytics
- Application & Service clusters
People are building clusters out of lots of things...

The hardware landscape is evolving:

❖ Different kinds of processors
❖ Different kinds of baseboard management
❖ Different kinds of boot processes
❖ Different kinds of networks
❖ ...

Los Alamos Takes New HPE Apollo 80 System on a Test Drive
September 1, 2020

LOS ALAMOS, N.M., Sept. 1, 2020 — Los Alamos National Laboratory recently began using HPE Apollo 80 Systems to conduct its largest ever Enterprise (HPE) that features some of the most demanding...
But how we manage them hasn’t changed.

- Maintain some configuration store (probably in a database).
- Use configuration store to create misc. configuration files.
- Use configuration files to control classic, monolithic services.
- Power everything on and hope for the best!
Imagine a world...

- Make a configuration change in git
- Automatic tests run
- System images built
- "Canary" test run
- Magical Mystery Management Machinery
- Changes quickly, reliably, safely deployed across 1000s of nodes.
A Re-Think in Three Inspirations
Inspiration #1: Container Orchestration

❖ **Continuous Enforcement:** System state should be enforced continuously.

❖ Example: If a service gets out-of-line, detect it and fix it.

❖ **Hardware Abstraction:** Components should be abstracted from the services they provide.

❖ Example: A pod should care what it provides, not where it runs.
Inspiration #2: Configuration Management

- **Centralized Configuration:** Centralize and organize system configuration.
  - Example: CM states should enforce local system states.

- **Declarative Administration:** Say what you want, not how to get it.
  - Example: Describe the functions a system should have (e.g. "roles"), not how to create those functions.

- Bonus: naturally makes systems idempotent.
Inspiration #3: API-Driven Design

❖ **Layer Abstractions:** Build systems in layers.
   ❖ Example: Define clear boundaries in functionality, abstract outside details.

❖ **Well-defined Interactions:** Prescribe interactions, not actions.
   ❖ Example: One component shouldn't tell another how to do its job, only how to interact.
A rough sketch

- Centralize where system state lives.
- Configure systems declaratively.
- Enforce states continuously.
- Abstract hardware from its uses.
- Clearly delineate layers of functionality.
- Provide simple APIs to allow components to interact.
Kraken + Layercake
The Kraken Framework

- Kraken is a framework for building automation tools that are:
  - **Distributed**: Able to manage automation workflows across many systems.
  - **Scalable**: Able to scale to the demands of workloads like HPC.
  - **Declarative**: Say what, not how.
  - **Modular**: Can create tools for a variety of use-cases based on included modules.
  - **Always-on**: Continuously enforces states across a distributed system.

https://kraken-hpc.io
Layercake: The basics

- Written in Go; Based on Kraken.
- Inherits from Kraken:
  - Declarative
  - Continuous state enforcement
  - 100% Modular
- Adds:
  - System abstraction layers (hence the name)
  - Centralized state; stateless nodes.
Think of systems in layers.

Changing a higher layer affect a lower layer.

E.g. Change Applications without changing System Images.

E.g. Change System Images without changing Software control plane.

Layers and their interactions should be modular.
Application & Users

❖ Focused on Applications, users, and their needs, not systems.
❖ Keep applications containerized & abstracted from system images as much as possible.
❖ Particulars vary significantly based on the system type.
System Images & Personalities

- System Images determine available functionality for Layer 2.
- An HPC cluster might run Slurm.
- A Container Orchestration might run k3s.
- Don't be picky about image formats & specifications.
- Just tell us where to find it and how to run it.
The Minimal OS (minOS)

- Software abstraction layer between system images (personalities) & hardware.
- The system image should have to worry about how it was provisioned.
- Changing a system image/personalities shouldn't require hardware interaction.
- In general: Try not to re-provision.
Provisioning

❖ We don't get to dictate hardware & firmware.
❖ (well, maybe firmware...)
❖ Try not to make assumptions about how to get to Layer 0.
❖ Allow for many different paths to reach the minOS.
Keeping things modular
Keeping things modular

[Diagram showing the process from Off to On to PXE to minOS]
Keeping things modular

- Off
- BMC Config
- On
- HTTPBoot
- PXE
- minOS
Keeping things modular

- Off
- On
- PXE
- minOS
- BMC Config
- HTTPBoot
- Custom Firmware Init
Take the path you need...

- BMC Config
- Custom Firmware Init
- minOS

Options:
- Off
- On
Reference implementation

- Power control through multiple mechanisms for many physical & virtual systems.
- Standard, integrated PXE stack.
  - All required services in one binary.
  - Several alt-stacks in dev.
- All-Golang minOS (u-root* based).
  - Tiny (~10M!), yet powerful.
- System image building via special tooling + Dockerfile.
  - But we can handle many different image formats.
- System image loading via ImageAPI service & Ceph RBD.
  - Loads system images privileged containers.
  - Supports everything from microservice containers to systemd.

*https://u-root.org*
Demo time!
Demo

Zero to minOS

- Installing Layercake
- Running Layercake
- Node definitions
- Boot a single node to minOS
Video Link: http://bit.ly/lisa21_video_wofford_demo1
Demo

Zero to Cluster

- Layercake in k3s
- The Kraken Dashboard
- Layercake + Ansible
- Boot entire Cluster to system images
- Self-healing
Demo

System Images to HPC

- Build a Slurm image
- Roll an update
- See that the update is stateful
- Run a (very simple) Slurm job
But wait...
...we don’t all want HPC!

- Assign a new image to specific nodes
- Roll nodes into k3s agents
Let’s recap.

- We argued we need better tooling for distributed and clustered systems.
- We derived some guiding principles.
- We talked a little about the theory of Kraken/Layercake.
- Then we actually played with it a bit.
- Is it the "Magical Mystery Management Machinery" in this diagram?

![Diagram showing the process of making a config change in git, running automatic tests and system images, followed by a "Canary" test run and deployment to thousands of nodes.]
Let’s recap.

- Is it the "Magical Mystery Management Machinery" in this diagram?
- We think we're headed there.
  - It meets the requirements in theory.
  - And already powerful in practice.
- It's still got some "rough edges."
- We hope you'll join us in getting it the rest of the way...
Thanks for listening!
References

❖ Design Inspiration

❖ Kraken HPC Project
- https://kraken-hpc.io/ - Kraken & related projects
- https://github.com/kraken-hpc/kraken - The Kraken framework
- https://github.com/kraken-hpc/kraken-layercake - The Kraken/Layercake tools

❖ Kraken Theory & Architecture