Monolith to Microservice: pitchforks not included

Learn how GitLab turned it's omnibus into cloud native Helm charts by way of containerization and orchestration. This talk aims to help practitioners already running large scale, successful products make decisions on how to move to microservices while maintaining product development cadence and serving customers on legacy software everyday. It's like driving a race car and fixing it as you are competing in a race, without pit stops.

We will cover:

- How we made the application stack capable of scaling via containerization, through many changes to stateful behaviors.
- Why we made the changes from an architectural view.
- How on earth we accrued the technical debts we had to fix in the first place.
- Most importantly, we'll demonstrate why the monolith concept was the right place to start, but Kubernetes is our future.

GitLab

Monolith to Microservice:
Pitchforks not included

$ whoami

Jason Plum
Senior Distribution Engineer

@WarheadsSE
gitlab.com/warheadsse
linkedin.com/in/jplum

- Develop and maintain install methods
  - Omnibus GitLab
  - GitLab Helm charts
- History in containerization
  - Pushing Docker forward since 2013
  - Docker on ARM
  - Does anyone know what '-bip' does?
- More than 1 year building one of the most complex Helm charts available

Overview

TL;DR: Here's the gist, but you'll miss the rest.

- Reality of GitLab as a complete solution
- Evolution of a Monolith
- Outgrowing tradition
  - Scaling
  - Sharding
- New approaches
  - Gitaly
  - Containerization
  - Object storage
- Understanding new challenges
  - Requirements
  - Scaling
  - Resilience

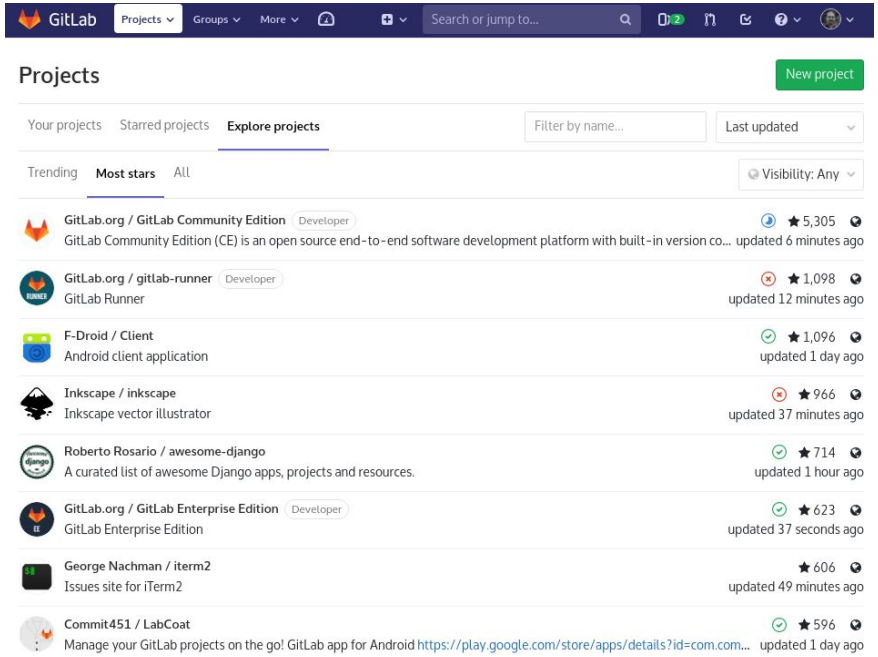
What is GitLab?

What is GitLab

Manage	Plan	Create	Verify	Package	Release	Configure	Monitor	Secure
Since 2016 GitLab added:	**Since 2011 GitLab added:**	**Since 2011 GitLab added:**	**Since 2012 GitLab added:**	**Since 2016 GitLab added:**	**Since 2016 GitLab added:**	**Since 2018 GitLab added:**	**Since 2016 GitLab added:**	**Since 2017 GitLab added:**
Cycle Analytics	Kanban Boards	Source Code Management	Continuous Integration (CI)	Container Registry	Continuous Delivery (CD)	Auto DevOps	Metrics	SAST
DevOps Score	Project Management	Code Review	Code Quality	Maven Repository	Release Orchestration	Kubernetes Configuration	Logging	DAST
Audit Management	Agile Portfolio Management	Wiki	Performance Testing	NPM Registry	Pages	ChatOps	Cluster Monitoring	Dependency Scanning
Authentication and Authorization	Service Desk	Snippets	**Coming soon:**	**Coming soon:**	Review apps	Runbook Configuration	**Coming soon:**	Container Scanning
Coming soon:	**Coming soon:**	Web IDE	System Testing	Rubygem Registry	Incremental Rollout	**Coming soon:**	Tracing	License Management
Code Analytics	Value Stream Management	**Coming soon:**	Usability Testing	Linux Package Registry	Feature Flags	Serverless	Error Tracking	**Coming soon:**
Workflow Policies	Requirements Management	Design Management	Accessibility Testing	Helm Chart Registry	**Coming soon:**	PaaS	Production Monitoring	Runtime Application Security
	Quality Management	Live Coding	Compatibility Testing	Dependency Proxy	Binary Authorization	Chaos Engineering	Incident Management	
						Cluster Cost Optimization	Status Page	

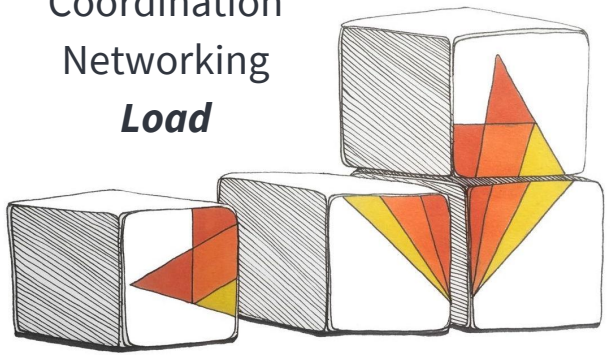
What is GitLab

End User



Engineering

File systems
Databases(s)
Memory stores
Containers
Automation
Coordination
Networking
Load



In the beginning ...

Monoliths make sense, while viable

- Clear focus for Minimum Viable Product (MVP)
- Adding features is simple
- Everything in one bundle

Advantages of Omnibus

- Full-stack bundle provides all components necessary to use every feature of GitLab
- Simple to install
- Components can be individually enabled/disabled
- Easy to distribute
- Highly controlled, version-locked components
- Guaranteed configuration stability

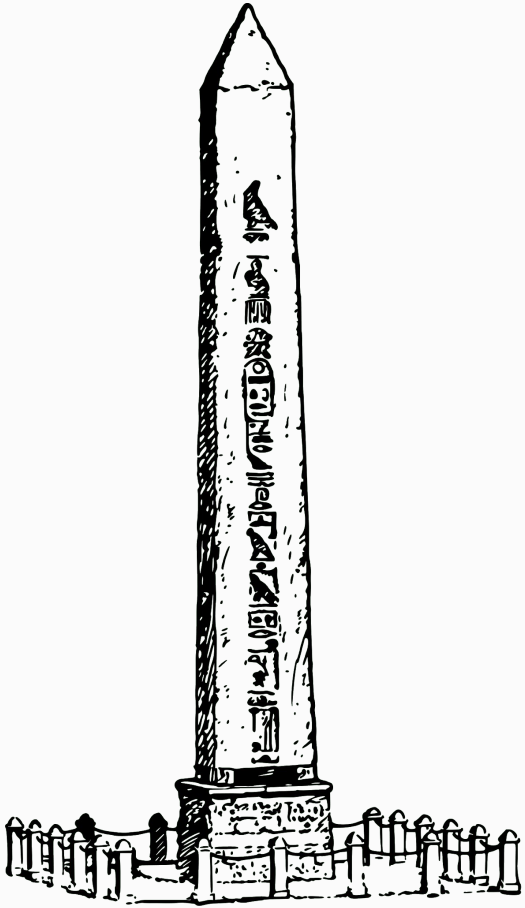
Monoliths

Massive, singular, unwieldy

Omnibus GitLab provides a single source of truth and configuration, for everything about GitLab.

We use it. Our customers use it.

It is massive.



Let me tell you a story ...

Git is hard

Key Concepts

- **Snapshot based:** stores **complete copy** of **every version** of a file
- **Number of files:** Indexes, pointers, pack files.
- **Scale:** Bigger = Slower

Example Case

- **Clone** torvalds/linux.git
- **Checkout** a branch (any)
- **Diff** master

How many files were read?

Branch and Merge Request

- **Change files, stage commits.**
- **Push these to your remote.**
- **Now view this in a 'diff' view in the GitLab UI**

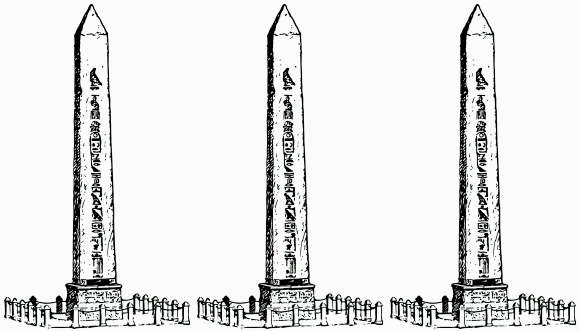
Now multiply by 10,000

The hard parts

Solving Disk

Spread the load

Faster! Faster! Faster!

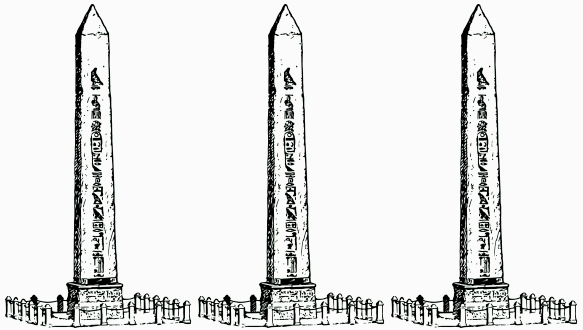


So many widgets!

The hard parts

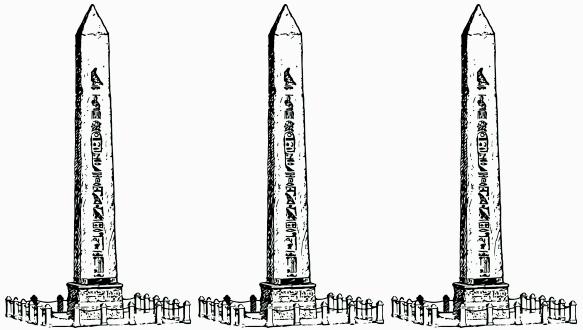
Solve one, cause another

- Sharding disk with NFS
- Off the disk, onto the network



Only two problems:

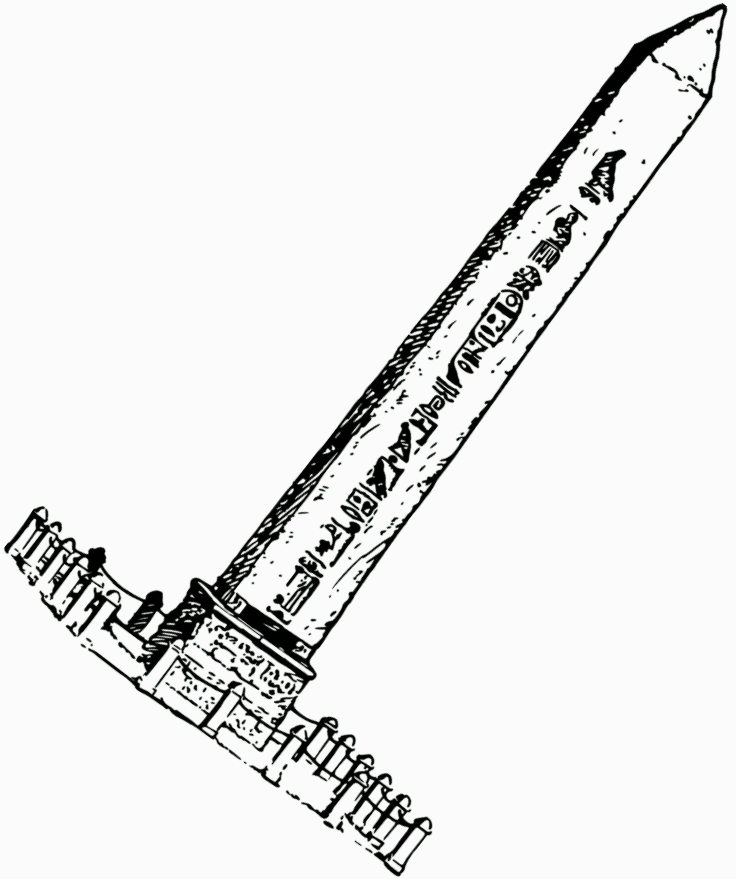
1. Disk IO
2. Network Throughput
3. NFS



Monoliths have limits

Massive, singular, unwieldy

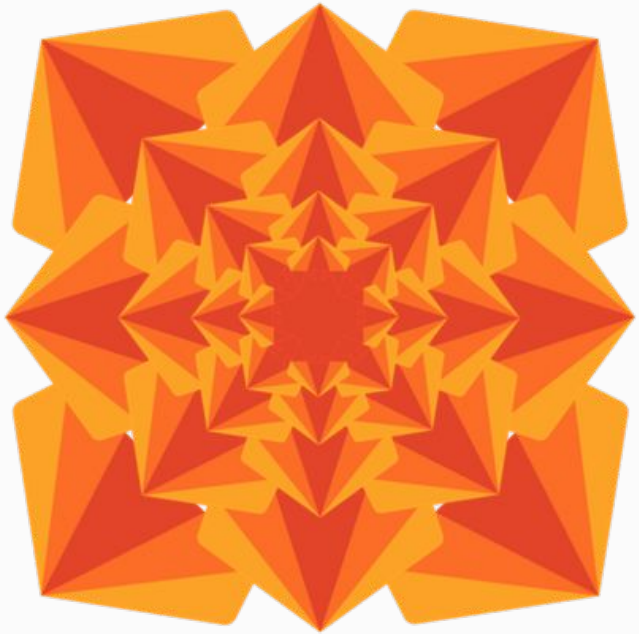
At a certain scale, they start to tip over



Old problem, New answer

Gitaly

gRPC based network service for Git

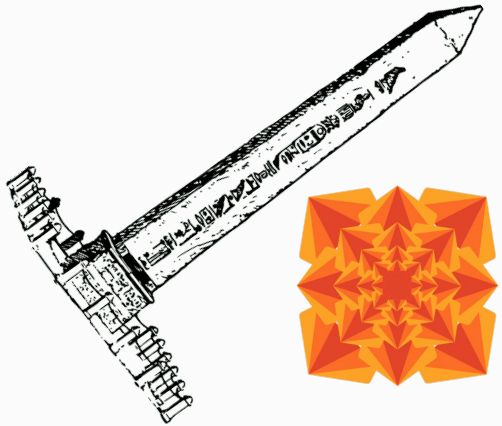


Significant gains

- Throughput requirement significantly reduced
- Service nodes don’t need disk access
- Optimize for the *specific problem*

We’ve propped the Monolith up!

Now we can focus on other bottlenecks



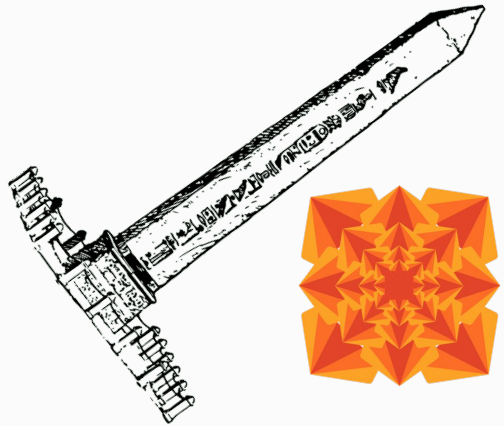
Forward!

Integrating Gitaly exposes less urgent bottlenecks

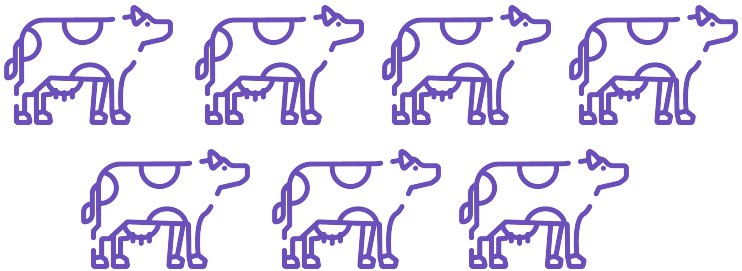
- NFS shards still used for traditional files
- Does every node need to have NFS??

Solution:

- Object Storage



Separate Pets & Cattle



Pets

Configured Omnibus at scale

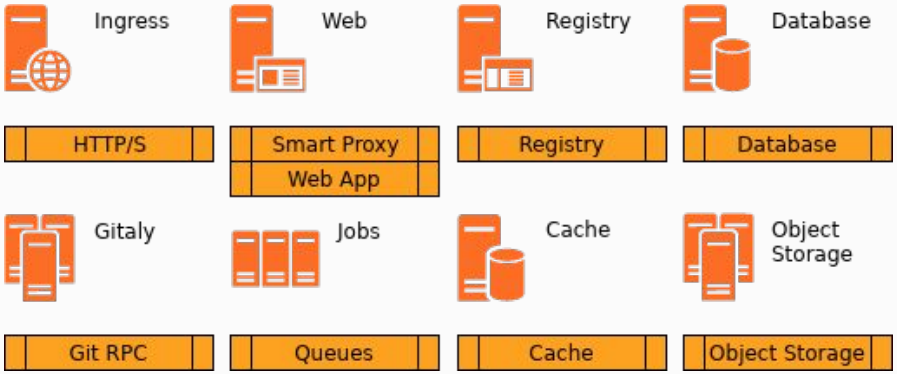
When deployed at scale, each VM has all roles available, but only small portions activated.

Cattle

Containerized Services

Component Docker images provide lower resource requirements.

Each component is directly configured, resulting in startup as short as 5 seconds.



Legacy Debt

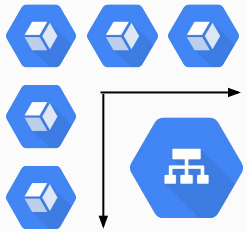
Some services remain coupled

- Sharing is caring
- Speak <UNIX> sockets to me

< time constraints >

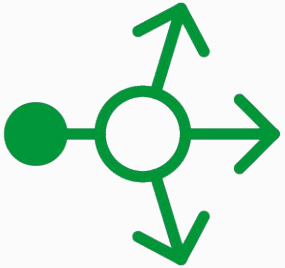
Can we define individual component requirements?

- Resources:
  - CPU?
  - Memory?
- Network:
  - Throughput?
  - Services?



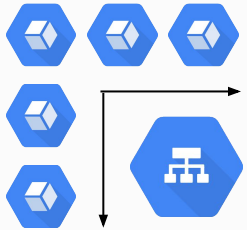
What do we use for load balancing?

- Which services?
- Which providers?



Scaling:

- Horizontal or vertical?
- Automatic or manual?



Resilience:

- What happens when things go boom?
- How to recover?
- How to plan!

Summary

1. GitLab's beginnings as a monolithic project provided the means for focused acceleration and innovation.

2. The need to scale better and faster than traditional models caused us to reflect on our choices, as we needed to grow beyond the current architecture to keep up.

3. New ways of doing things require new ways of looking at them. Be open minded, and remember your correct choices in the past could not see the future you live in.

Questions?

THANK YOU!

This story is based on
gitlab.com/charts/gitlab

Jason Plum
Senior Distribution Engineer

@WarheadsSE
gitlab.com/warheadsse
linkedin.com/in/jplum

Comes see GitLab

Booth S44

GitLab

Resources

Cloud Native GitLab

- gitlab.com/charts/gitlab/
- docs.gitlab.com/ee/install/kubernetes
- about.gitlab.com/kubernetes/

Gitaly

- The road to Gitaly v1.0
- gitlab.com/gitlab-org/gitaly