Portainer.io: Gaining Operational Maturity for Effective Docker/Kubernetes Implementation

Neil Cresswell, CEO, Portainer.io

Why Operational Maturity Matters

Containers and Kubernetes are transformative but not magic bullets.

Adopting Kubernetes is a “lifestyle choice” are you prepared to make that choice?

Operational maturity is essential to unlock the full potential.

Lack of maturity can lead to security breaches, downtime, and loss of trust.

Worse, the technology can be deemed a “failure” and rollout stopped, hindering your ability to compete

The Allure of Containers: Aspirations vs. Reality

Promises:

- Speed
- Agility
- Scalability
- Portability

Overlooked Demands:

- Complexity
- New skills/new hires
- Operational overhead
- Day 2 ops

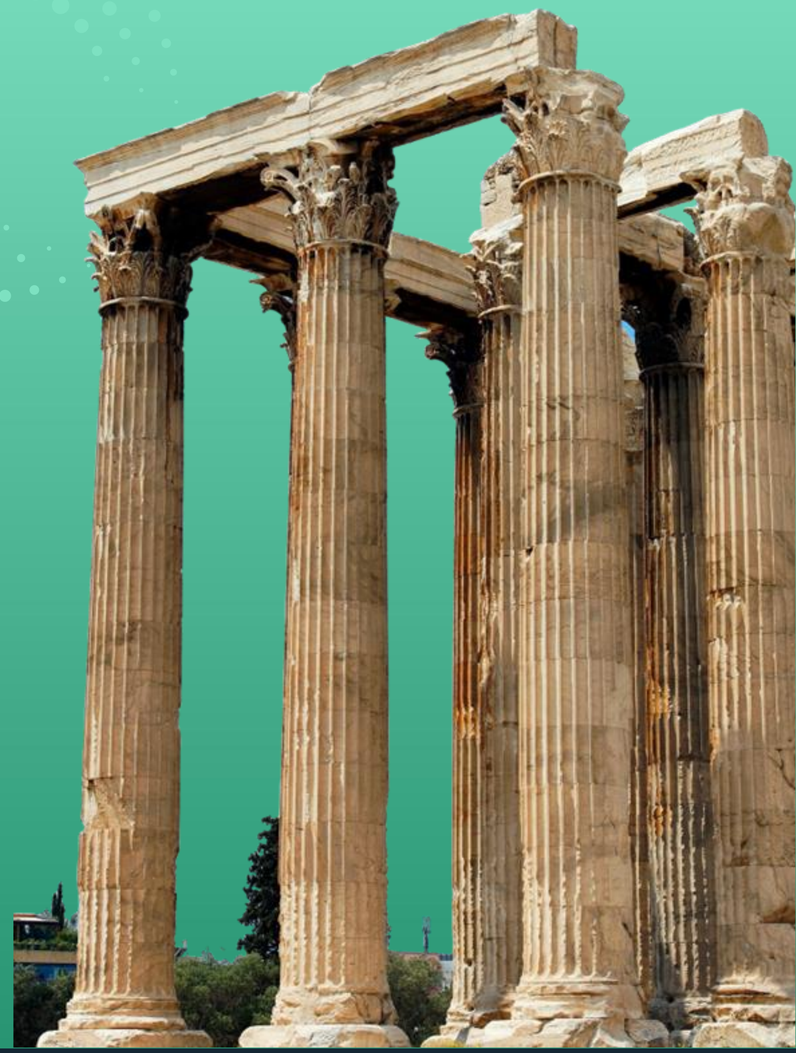
Are we genuinely prepared for these demands?

Defining Operational Maturity

Four Pillars

- Skills
- Structure
- Tooling
- Processes

Weakness in one affects all.



Self-Assessment: Where does your organization stand vs your needs?

Challenge 1

Insufficient Platform Engineering & DevOps Skills

Containers are VERY different from VMs. As a result, they require specialized skills beyond traditional IT

Assuming existing teams can manage without upskilling is risky

Consequences include misconfigurations, security compromises, and operational failures

The Skills Gap: Risks and Consequences

Operational failures and increased incidents

Productivity losses due to inefficient problem-solving

Competitive disadvantage against better-prepared organizations

HashiCorp Survey* found that 64% of container projects are stalled or compromised due to skill shortages.

Skills

https://www.hashicorp.com/state-of-the-cloud

Challenge 2

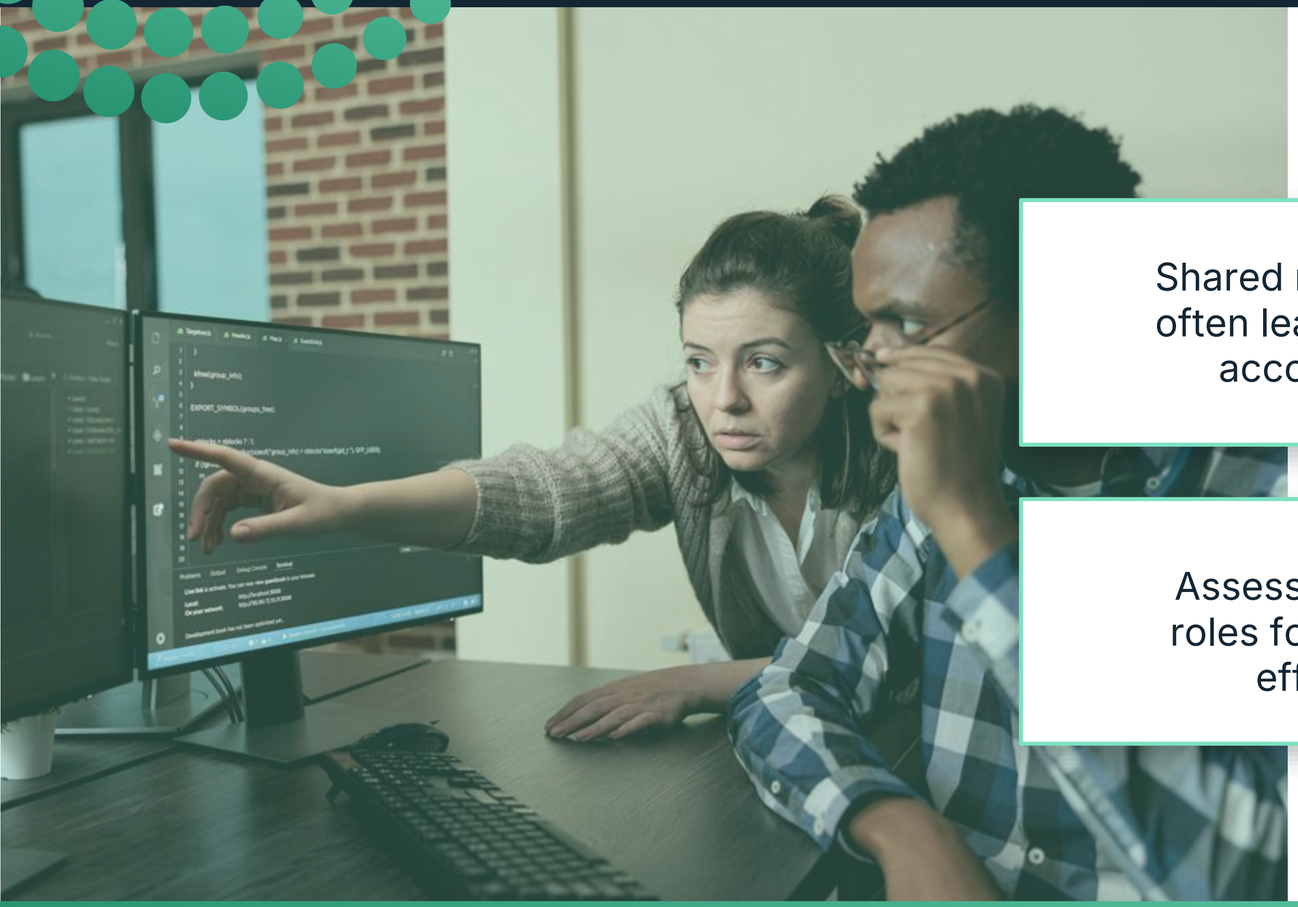
Organizational Structure Constraints

Fragmented responsibilities blur accountability

Siloed departments hinder communication and efficiency

Unclear ownership leads to decision-making bottlenecks

Shared Responsibility vs. Clear Ownership



Shared responsibility often leads to lack of accountability

Dedicated teams provide focused expertise and faster resolutions

Assess and realign roles for clarity and efficiency

Be careful of engineering leaders engineering in a vacuum.. Its very common

Structure

Challenge 3

Incorrect Tooling

Effective container management requires specialized tools

Relying on legacy tools can hinder operations

Equally, adoption of disparate tools lead to integration challenges. The CNCF landscape is a mess of over 2000 disparate tools.. Which do you need?

Are your tools empowering your team or holding them back? From who's perspective?

Tooling Gaps and their Impact

Efficiency losses from manual processes

Increased risks due to inadequate monitoring

Financial implications often exceed the cost of proper tooling

Don’t forget your internal users and their experiences.. Complicated tools that are misunderstood are as bad as no tools at all

Tooling

Challenge 4

Containerizing Legacy Applications

Legacy apps often aren't suited for containers without modification

Complex dependencies can cause unexpected issues

Migrating legacy apps to containers is valuable, however be prepared for the time investment to do so

Maybe consider lower criticality apps first, to cut your teeth

The Risks of Containerizing Legacy Apps

Potential performance issues affect user experience

Quick fixes increase technical debt

Consider re-architecting or replacing instead

Not all apps are worth containerizing—choose wisely

Processes

The Cost of Low Operational Maturity

- Frequent Application or Platform Outages
- Extended outages when they occur
- Outages blamed on “the technology”
- Fear to perform (required) platform/system upgrades due to the unknown impact of such
- Security vulnerabilities, or an unknown security posture across the platform
- Unpredictable application performance
- Engineers spending considerable time learning vs doing
- Engineers spending a large proportion of their time fixing unexpected issues vs running the system
- Projects taking substantially longer than expected to implement

- Technical understanding inaccuracies such as “namespaces give network/security isolation between apps” - they dont!
- Monitoring tools costs spiraling due to poorly optimized logging/monitoring tooling capturing “noise”
- Container/Host “Sprawl” to unconstrained access to resources, leading to cost blowouts
- Engineers forever wanting to automate everything before the platform is ready
- Disengagement between Devs and Ops
- No centralized management of decentralized platforms
- No central visibility into the security / governance of the environment

And the outcome...

We're leaving Kubernetes

Kubernetes seems like the obvious choice for building out remote, standardized and automated development environments. We thought so too and have spent six years invested in making the most popular cloud development environment platform at internet scale. That's 1.5 million users, where we regularly see thousands of development environments per day. **In that time, we've found that Kubernetes is not the right choice for building development environments.**

This is our journey of experiments, failures and dead-ends building development environments on Kubernetes. Over the years, we experimented with many ideas involving SSDs, PVCs, eBPF, seccomp notify, TC and io_uring, shiftfs, FUSE and idmapped mounts, ranging from microVMs, kubevirt to vCluster.

In pursuit of the most optimal infrastructure to balance security, performance and interoperability. All while wrestling with the unique challenges of building a system to scale up, remain secure as it's handling arbitrary code execution, and be stable enough for developers to work in.

This is not a story of whether or not to use Kubernetes for production workloads that's a whole separate conversation. As is the topic of how to build a comprehensive soup-to-nuts developer experience for shipping applications on Kubernetes.

This is the story of how (not) to build development environments in the cloud.

I Didn't Need Kubernetes, and You Probably Don't Either

By Ben Houston, 2024-11-05

(This blog post was discussed on YCombinator's Hacker News here. A response from Romaric Philogène to this essay is also here)

Kubernetes often represents the ultimate solution for container orchestration, but my experience has led me to leave it behind in favor of a simpler, cost-effective solution using Google Cloud Run. This transition has made my infrastructure projects easier to manage, more scalable, and significantly cheaper. Here's why I made this choice and how Cloud Run offers a better fit for my needs going forward.

How I ended up on Kubernetes

First, let's quickly look at how we ended up on Kubernetes. We has launched Clara.io (now sunset), an online 3D editor and rendering platform, in 2013. We cost optimized the platform by using bare metal machines from OVH for both its primary servers, DBs and job workers. While it worked, bare metal machines were a source of potential failures and we did have some over the years. Luckily we had a redundant setup so our users never noticed. But it was a massive amount of work to provision, monitor and maintain.

I Stopped Using Kubernetes. Our DevOps Team Is Happier Than Ever

Why Letting Go of Kubernetes Worked for Us

Crafting-Code · Follow
Published in Stackademic · 10 min read · Nov 19, 2024

Introducing the Operational Maturity Self-Assessment

Maturity Level	Personal Readiness	Organizational Readiness	Application Readiness	Technology Readiness
Advanced	Advanced Kubernetes Knowledge	SRE Team	Container Native 12 Factor Applications	Advanced Container Service Platform
Capable	Basic Kubernetes Knowledge	Platform Engineering	Container Native Three-Tier Applications	"Containers as a Service" Platform
Emerging	Advanced Docker Knowledge	DevOps Adoption	Containerizable Modern Three-Tier Applications	Centralized Container Operations Platform
Opportunistic	Basic Docker Knowledge	"Champion" based initial adoption of Containerization	Containerizable Traditional "LAMP" Three-Tier Applications	Discrete Container Operations
Ad-Hoc / Experimentation	Advanced Linux and Infrastructure Knowledge	Traditional Platform Support Teams	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

A tool to evaluate your organization's readiness

Covers skills, structure, tooling, and processes

Identifies gaps and provides a roadmap for improvement

Empowers your organization by understanding where you stand

Embrace Operational Maturity

1. Complete the self-assessment to gain critical insights
2. Use findings to prioritize actions and investments
3. Operational maturity is essential for sustained success
4. Don't let assumptions hinder progress—take action today

Assessment Download Link