HIERARCHY OF NEEDS:

A HUMAN-CENTRIC LOOK AT WHAT USERS ACTUALLY NEED FROM A CONTAINER PLATFORM

BUILD THE PLATFORM AROUND THE WORK NOT THE ORG CHART.

Selecting a container management platform isn't just about features, integrations, or logos on a vendor slide deck. It's about one question: will this platform help your people do their jobs?

That may sound obvious, but it's rarely how platform evaluations start. Too often, teams begin with a shopping list of features drawn from analyst reports or what their peers use; without grounding those features in the day-to-day tasks of the people who will rely on the platform. A robust RBAC model might be on the checklist, but has anyone asked who actually needs access and to what? Logging integrations may look great in a demo, but are they usable by the people who need to triage production issues at 2am?

This whitepaper takes a different approach. It outlines the core “jobs to be done” across the primary personas who interact with container platforms every day. It focuses on the real needs of those users, drawn from hundreds of platform rollouts across enterprises, government, and industry.

The goal is simple: to help you, as an IT leader or platform owner, better assess whether a given platform meets the needs of your own organization, not in theory, but in practice.

LET’S BEGIN WITH THE PEOPLE AT THE COALFACE.

The Operations Engineer, keeping the business online

In most organizations, particularly those that don’t build all of their own software, the Operations team is the first and most consistent user of the container platform. Their job isn’t to write code. It’s to make sure that workloads (whether commercial off-the-shelf applications or internal tools) run reliably, securely, and predictably.

When something breaks, they’re the first line of defense. And when it’s time to scale, patch, or upgrade an app, they’re often the ones doing the work.

The platform must support this by making operational visibility and control straightforward. Can an Ops engineer see what’s running, understand how it’s behaving, and react quickly when something goes wrong? Can they deploy a vendor-supplied Helm chart or Compose stack without needing elevated permissions or deep Kubernetes expertise? Can they compare production and staging environments side by side to identify drift?

They also carry the burden of compliance. Whether it’s enforcing CVE scans on container images, proving that privileged workloads are blocked, or confirming that resource limits are in place, Ops is often the team tasked with supplying the evidence.

When evaluating a container platform, ask: does it empower the Ops team to do all of this directly? Or does it require them to file tickets, escalate to platform engineering, or maintain custom YAML just to deploy a third-party app?

In many organizations, especially those deploying enterprise software, the Ops team is the single largest user of the container platform. Treat them as such.

The Developer, moving from code to runtime without drama

While not every organization builds all their own applications, nearly every one has developers working on something, whether internal tools, integrations, or customer-facing systems. When developers interact with the container platform, their needs are usually tied to two critical phases: validating that their code works in a production-like environment, and supporting it once it’s live.

In the development phase, they want to run code in containers that mimic production without needing to become a Kubernetes expert. That means clear starting points, sane defaults, and the ability to get feedback quickly. Does the platform support local development environments or ephemeral dev clusters? Can developers deploy code into a sandbox with minimal friction? Can they test changes without needing to learn Helm or edit complex manifests?

In the support phase, the key question becomes visibility. Developers aren’t managing infrastructure, but when their app misbehaves in test or production, they need insight into what went wrong. That might be container logs, startup events, network traffic between pods, or configuration differences between environments.

Importantly, the platform must respect boundaries. Developers don’t need to be cluster admins. But they do need enough access to observe and debug their applications safely. A good container management platform makes this clear. A bad one leaves them stuck waiting for someone else to run `kubectl` .

The Platform Engineer, architecting for scale without becoming the bottleneck

While developers and ops teams are the most visible consumers of the platform, the Platform Engineering team is responsible for designing, building, and maintaining it. Their mission isn't to ship applications. It's to ensure that the platform enables everyone else to do so, without compromising stability or security.

For this team, the evaluation lens shifts. It's not just about what the platform does, it's about how reliably and repeatably it can do it across clusters, environments, and teams.

A container management platform must support lifecycle management at scale. That includes provisioning new environments, upgrading clusters, and maintaining consistency across them all. Is there a single place to apply global policies? Can changes be rolled out without downtime? Does the platform offer templated configurations for repeatable cluster creation?

Then there's policy enforcement. The platform engineer needs to set the rules: who can deploy where, which registries are trusted, what happens when a pod exceeds its limits. But the enforcement needs to be quiet, reliable, and automatic, not something that requires manual policing.

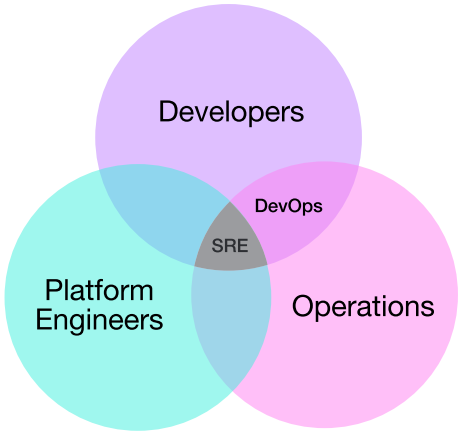
Platform teams also maintain the "glue" services: logging, metrics, ingress, secret management, network policies. These aren't just infrastructure, they're shared services used by Ops and Dev. If the container management platform bundles these services, are they usable? If it doesn't, does it integrate cleanly with third-party tools?

And perhaps most importantly, a good platform prevents the platform team from becoming a blocker. If the only way to onboard a new team, provision a namespace, or roll back a deployment is through a Jira ticket to the Platform team, the platform has failed. Self-service isn't a luxury. It's survival.

The overlap is the point, understanding shared responsibilities

It's tempting to categorize these three roles Ops, Dev, and Platform) as cleanly separated personas. But in reality, the boundaries blur. The same person might be deploying commercial software one day, debugging a containerized app the next, and then patching a base image the day after that.

To reflect this, it's useful to visualize the overlaps, not just the distinctions.



In the overlap between Developers and Operations, you often find DevOps engineers; those responsible for CI/CD pipelines, deployment automation, and release management. They're developers who also care deeply about how code reaches production.

At the intersection of all three roles sits the SRE; the Site Reliability Engineer. Equal parts system thinker, incident responder, and performance tuner, the SRE bridges infrastructure, application behavior, and user impact.

The point here isn't to obsess over job titles. It's to recognize that container platforms don't serve personas, they serve jobs.
Those jobs are often shared. So the platform must be usable across boundaries.
If a tool only works for the Platform team but confuses the Ops team, it fails.
If it's great for developers but impossible to operate securely, it fails.
If it's powerful but unusable for anyone outside of engineering, it fails.

The overlaps are the point. Your platform must support them.

Oversight Teams, proving that the rules are followed

Outside the core engineering functions sit oversight teams; Security, Compliance, Governance, and Finance. These teams aren't using the platform to deploy applications. But they are deeply invested in what gets deployed, how it's configured, and whether policies are followed.

Security teams want to know whether CVEs are being scanned, whether workloads are running with elevated privileges, and whether image sources are trusted. Governance wants audit logs; who deployed what, when, and how. Finance wants to know who's using which environments, how much capacity they're consuming, and what the cost implications are.

Most of these needs are met not by extra tooling, but by the platform surfacing what it already knows. Does the platform expose audit trails? Can it show image provenance? Are usage metrics broken down by team, namespace, or application?

A platform that hides this information (or makes it impossible to extract) puts the burden back on the platform team to create custom scripts, dashboards, and reports. A platform that treats these as first-class concerns becomes a partner in accountability.

Evaluating a Platform: A checklist built around real work

When selecting a container management platform, the checklist shouldn't start with vendor logos or CNCF alignment. It should start with real user needs.

- Can your Ops team deploy and manage commercial applications without writing Kubernetes manifests from scratch?
- Can your developers observe and debug their applications without asking for elevated permissions?
- Can your platform engineers enforce policies, manage clusters, and expose self-service capabilities without building everything from scratch?
- Can your security team validate controls without waiting weeks for someone to assemble an audit report?
- And perhaps most importantly: can all of these people use the platform without needing to learn Kubernetes internals just to get their work done?

A good platform makes the right things easy and the dangerous things hard. It removes friction without removing control. And it enables the people who build, run, and support software to focus on delivering value, not deciphering YAML.

So, in order to operate a Platform that users love, design for the jobs, not the titles

This framework is not prescriptive. Your organization may not have separate Platform Engineers. Your developers may never touch a container image. Your SREs might live in the Ops team. That's fine. What matters is understanding what work needs to be done, and evaluating platforms through that lens.

Container platforms are powerful. But that power is only useful when it helps your people move faster, respond to issues sooner, and operate systems more safely.

Don't just build or buy a platform. Deliver a platform that fits your people.

HIERARCHY OF NEEDS: CONTAINER OPERATIONS MIND MAP

Scan the QR code to download the full res mind map.

USER STORIES: SUPPORTING THE MIND-MAP

Operations Engineer

As an Operations Engineer, I want to:

- see what applications are currently deployed across all environments, so that I can maintain awareness of the production landscape.
- view container logs, events, and status in real time, so that I can diagnose and resolve issues quickly.
- compare application configurations between environments, so that I can identify drift that may be causing instability.
- check whether an image has passed a vulnerability scan, so that I can enforce security standards before deployment.
- verify that containers are running without privileged access, so that I can maintain workload isolation and hardening compliance.
- inspect network traffic between services (east-west), so that I can troubleshoot application-to-application communication failures.
- see which nodes are overloaded or underutilized, so that I can make decisions about scheduling, scaling, or resourcing.
- monitor container restarts and crash loops, so that I can intervene before customer-facing issues arise.
- roll back a deployment if application health declines post-release, so that I can restore system stability without delay.
- deploy third-party or commercial workloads using a standard method Helm, Compose, Operator), so that I can support the business with minimal friction.
- promote a working deployment from staging to production, so that I can accelerate safe releases.
- expose operational metrics and logs to my observability stack, so that I can correlate platform data with business service health.

Developer

As a Developer, I want to:

- deploy my application into a sandbox environment that closely matches production, so that I can test new functionality early.
- run my application locally in a container identical to what runs in production, so that I can develop confidently without surprises.
- access logs and environment variables for my running container, so that I can debug application-level issues myself.
- inspect startup behavior, probes, and resource usage of my app, so that I can tune performance before go-live.
- view the application version and deployment status in each environment, so that I can validate successful promotion or rollback.
- view service-to-service communication paths during runtime, so that I can understand how my application interacts with others in the system.
- observe how my app behaves under simulated production traffic, so that I can test for latency and error handling.
- be notified when my deployment causes failures, so that I can respond and fix it quickly.
- safely redeploy my application without touching infrastructure, so that I can ship fixes and features independently.
- validate that configuration values (e.g., secrets, environment variables, volume mounts) are applied correctly at runtime, so that I can troubleshoot without involving the Ops team.
- triage issues in production without needing cluster admin permissions, so that I can act quickly without compromising security.

Platform Engineer

As a Platform Engineer, I want to:

- provision clusters using a repeatable template, so that I can ensure consistency across environments.
- enforce baseline policies across all clusters RBAC, PSPs, registry restrictions), so that I can control risk and maintain governance.
- define resource limits and quotas for teams, so that I can prevent noisy neighbor problems.
- configure ingress controllers, DNS, and TLS termination centrally, so that I can provide secure and consistent access to services.
- define safe defaults for app deployments, so that teams can deploy without knowing all the details.
- expose platform observability (nodes, clusters, events) in a centralized control plane, so that I can monitor system health at scale.
- integrate my preferred logging, monitoring, and alerting stack with the container platform, so that I can standardize observability across teams.
- roll out cluster upgrades with minimal disruption, so that I can maintain security posture without breaking workloads.
- enable teams to self-service the creation of namespaces or projects with scoped access, so that I can reduce support overhead.
- manage multiple clusters from a single interface, so that I can reduce operational complexity.
- connect to edge or remote clusters securely, so that I can extend the platform to constrained or distributed environments.
- validate that workloads adhere to platform policy (e.g. registry usage, labels, security contexts), so that I can identify and remediate violations.
- separate concerns between platform and application layers, so that teams know what they own and what the platform controls.
- see usage patterns over time (by user, namespace, or workload), so that I can make scaling and cost decisions.

Oversight Roles (Security, Compliance, Governance, Finance)

As a Security Officer, I want to:

- receive alerts when a container image contains critical CVEs, so that I can triage and block deployments that introduce risk.
- verify that image sources are from trusted registries, so that I can prevent supply chain compromises.
- review access logs and deployment actions, so that I can investigate incidents or policy breaches.
- export an audit trail showing who deployed what and when, so that I can demonstrate controls to auditors.
- verify that RBAC is consistently applied across environments, so that I can meet regulatory access controls.
- know which teams own which workloads and resources, so that I can ensure accountability across business units.
- I want to see resource usage broken down by team, so that I can attribute spend and inform chargebacks or budgeting.
- validate that workloads are running with appropriate privileges and configurations, so that I can report posture to leadership.

PORTAINER.iO