portainer.io

Containerization Operational Maturity

Self-Assessment

How to use this guide..

1 Using the bullet point lists, determine your current level of maturity for each section, based on your ability to answer “yes” to the bullets stated.. If you cannot answer Yes to all bullets in a section, then you have not reached that level of maturity. **Remove a tick or cross when determining.** ✓ ✗

2 Map the green ticks onto the matrix as green squares so you can visually see maturity level.

3 Then define your required level of maturity based on your business need. Map this onto the maturity matrix as a horizontal line at the top of the maturity level you need.

4 You can now see the gaps you need to address (in inversely where you are over investing)

The Maturity Matrix

Understanding the Maturity Model – Indicators of maturity

Maturity Level	Personal Readiness	Organizational Readiness	Application Readiness	Technology Readiness
Advanced	Containers are the deployment method for applications that form critical elements of the business, specialist team responsible for SLAs, and deep tooling deployed			
Capable	Container adoption now mainstream for business apps, central team responsible for supporting the platform, end to end tooling deployed.			
Emerging	Containers now accepted as a deployment choice for non-core applications, support is widened to the team that deploys them, supports them. Basic tooling deployed			
Opportunistic	Initial Deployment of Containers for Business purposes, person that deploys them, supports them, limited “bespoke” tooling deployed			
Ad-Hoc / Experimentation	Select few individuals beginning to learn, no official company-wide investment into tooling or supporting org structures			
Traditional	VM-Centric, no exposure to Containers			

Understanding the Maturity Model

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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

Example – The Model, with current maturity mapped

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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

Example – The Model, with desired maturity mapped

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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

Example – The Model, with gaps identified

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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

Example – The Model, showing over investment in advance of needs

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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

Personal Readiness

How to scorecard “Personal Readiness”

In the detailed list that follows, you need to scorecard the technical understanding of Linux, infrastructure, containerization, and its surrounding tooling, that your team currently possess.

The goal is to understand what their basis level of knowledge is, and how it can be mapped to organization needs, and where training may be required.

Note that the knowledge needs to be both theoretical and applied.

Personal Readiness (Map for those responsible for managing the container platforms)	Present?
Advanced Kubernetes Knowledge: • Architectural understanding of Kubernetes: control-plane, workers, API server, kubelet, networking, storage, deployments, service discovery, clustering, security, monitoring. • Deep understanding of the technicalities of Kubernetes Container Orchestration • Kubernetes Extensibility: CNI, CSI, ServiceMesh, Operators, Admission Controllers, Policy Engines • Ability to write comprehensive Kubernetes RBAC Roles and Cluster Roles • Secret and Configuration Management • Knowledge of Autoscaling (Pod and Cluster) and how to set per-namespace resource allocations (Quotas) • Extensive knowledge of the full array of Kubernetes CLI Commands: deployment, troubleshooting, lifecycle • How to write Advanced Kubernetes Manifests and HELM charts • Deep understanding of GitOps and Policies	✓ ✗
Basic Kubernetes Knowledge: • Conceptual understand of Kubernetes Orchestration • Kubernetes constructs: deployments, pods, containers, services • Resource overhead of running Kubernetes • Kubernetes Cluster build using “bootstrap” tooling • Kubernetes Cluster lifecycle using “bootstrap” tooling • Kubernetes CLI, basic understanding (top 10 commands)	✓ ✗
Advanced Docker Knowledge: • Distributed Micro-Services, and the architectural impacts (performance, troubleshooting) • Simple Container Orchestration (Docker Swarm, or Simple Kubernetes) • Maintaining Data Persistence across Nodes • Cross-Node Networking (Overlay Networking, Tunnelling) • Reverse Proxies (Ingress Controllers) • DNS (Service Discovery)	✓ ✗
Basic Docker Knowledge: • Architecture of a container and how a container runtime works • Differences between Stateless vs Stateful Containers; understand how to persist data, and what happens if you don’t. • Differences between VMs and Containers • How Docker Volumes Work, and differences from bind mounts • How to write Docker Compose files and use Compose to start and stop groups of containers that comprise an application stack • Docker Images composition, how these are built and distributed	✓ ✗
Advanced Linux and Infrastructure Knowledge: • Data storage methodologies, NFS/CIFS, Block Storage, and the difference between each, understanding of RWO/RWX file systems. • Understanding of Networking concepts, NAT, VLAN, Overlay Networking (VXLAN, Encapsulation/Tunnelling)	✓ ✗
Basic Linux Knowledge: • SSH Server Configuration (with SSH keys) • Installing components / apps using Apt • SSL/TLS • Iptables / IPFW • Standard Linux commands	✓ ✗

Organizational Readiness

How to scorecard “Organizational Readiness”

In the detailed list that follows, you need to scorecard your IT organizational structure against the known models for supporting Containerized platforms.

Note that simply calling your team “platform engineering” doesn’t mean they operate that way (as an example). You need to look at the indicators/ways of working to validate the team structure you have in place accurately matches the maturity levels indicated.

Also note that you may run a combination of DevOps, Platform Engineering, and Site Reliability Engineering, as the more advanced levels of maturity compliment (but can also replace) lower levels.

SRE engineering is a very advanced level of engineering maturity, normally reserved for mission critical deployments, and almost exclusively the realm of senior engineering folk.

Organizational Readiness (Answering depending on the IT Operational support structure you have for Containers)	Present?
SRE • Dedicated full-stack engineering team • Focussed exclusively on preventative engineering, spanning development through platforms, with a focus on ensuring uptime/resilience, automation, and incremental improvements. • Proactive monitoring and remediation	✓ ✗
Platform Engineering: • Transition from DevOps model to Platform Engineering • Realization that the Container platforms are business critical • Central Policy/Control/Security/Operations • Dev Self Service Platform / Devs “are the customer”	✓ ✗
DevOps Adoption: • “You Build it, you support it” approach to the adoption of Containerization by the Dev team. • No Centralized management of Containerization • Discrete pockets of skills • High Risk due to Dev Team lack of exposure to Infrastructure fundamentals • Containerization underpins certain discrete production services	✓ ✗
“Champion” based initial adoption of Containerization: • Single Developer or Lead Adopting Containers in their workflow • “Accidental” transition to a production service • Insufficient operational controls/visibly • Historically known as “Shadow IT”; Adoption of the technology by an individual/team due to central IT reluctance to officially support	✓ ✗
Traditional Platform Support Team: • Central Platform Team responsible for common platform components (Cloud, Virtualization, DB’s, Web Services, API Gateways) • Offer IT as a Service from a Service Catalogue • Simple Self-Service offerings (auto-provisioning) • No exposure to Containerization in this team	✓ ✗
Traditional IT Operations: • Infrastructure Support Team, • Application Support Team, • Development Team (if software developed internally), • “Ticket” based service delivery	✓ ✗

Application Readiness

How to scorecard “Application Readiness”

In the detailed list that follows, you need to scorecard the application portfolio that you plan to migrate to containerization. Not every application can run in containers, and for “purchased” software, you need to be sure your vendor supports container platforms.

If you do not produce your own software (no internal developers), then scorecard against your ISV provided software through questioning your vendors.

If you do develop your own software, your development team will be able to share insights on the potential/supportability to migrate to containers.

Application Readiness (Map based on the applications you want to deploy/migrate)	Present?
Container Native 12 Factor Applications • Horizontally Scalable • Micro-Service Architecture • Stateless	✓ ✗
Container Native three-Tier Applications: • ISV provided container images and deployment manifests • Internal Developer provided container images and deployment manifests	✓ ✗
Containerizable, Modern Three-Tier Applications: • Self-Developed three tier (Web/Middleware/DB) applications that can be separated into an application stack that run distributed across disparate servers with no impact to application performance • COTS software that can be repackaged to run within discrete containers distributed across a cluster of physical servers	✓ ✗
Containerizable Traditional "LAMP" Three-Tier Applications: • Self-Developed three tier (Web/Middleware/DB) applications that can be separated into an application stack that run on a single physical server • COTS software that can be repackaged to run within discrete containers on a single physical server	✓ ✗
Containerizable Monolith Applications: • Self-Developed Applications based off common frameworks / components with ready-available container image bases • COTS software that can be repackaged to run within a container	✓ ✗
Traditional Installable Software • Purchased Software provided by a Software Vendor • Self-Developed, complied with an installer • Windows UI centric Applications	✓ ✗

Technology Readiness

How to scorecard “Technology Readiness”

In the detailed list that follows, you need to scorecard your Container and tooling deployment against the Container Management Platform* standard. As your adoption of containerized applications increases, it is critical to have the platform tooling to adequately support it.

Tooling can either be discrete tools, or a consolidated platform tool, but regardless, you need the capabilities listed in the following.

* See Gartner Description of Container Management Reference Architecture

Technology Readiness (Map based on the level of Platform Build you are considering or have already deployed)	Present?
Advanced Container Service Platform • Service Mesh • Geo-Distributed Applications • Advanced Observability, Security, and Compliance Tooling	✓ ✗
“Containers as a Service” Platform • 100% Self-Service Developer Portal • Golden Path’s • Multi-Cluster/Multi-Cloud/Hybrid-Cloud • Rapid Lifecycling of Clusters • Infrastructure as Code • GitOps as the preferred deployment Model • Full Observability Suite; Realtime and historical monitoring for a full array of metrics	✓ ✗
Centralized Container Operations Platform: • Migration from discrete deployments to central deployment • Migration of Operations from Dev to a Platform Engineering team • Centralized Access/Authorization/Policy/Security • Single Deployment Model (cloud, On-prem) • Often a very small number of shared environments (likely one, sometimes more) • Clusters as “Cattle” • GitOps and ClickOps combined • Basic Observability; real-time monitoring for a limited range of metrics	✓ ✗
Discrete Container Operations: • “Developer Lead” deployment of Containerization in production – cloud based primarily • Discrete per-project deployments, not a common/shared platform • Bootstrap tools to create environment; k3s, RKE, AKS/GKE/EKS centric deployments • No central management and control • Environments as “pets” • “ClickOps” / UI Centric Management	✓ ✗
Container Experimentation • Dev Environments / Dev Laptops • Docker Desktop/Podman Desktop/MiniKube • Non-Critical Deployment • Distributed, no central control • Discrete Monitoring Tools • CLI / UI Centric Management	✓ ✗
Traditional IT Monitoring and Alerting Tooling • VM Monitoring • Application Performance Monitoring • Service Availability Monitoring • Threshold based alerting • SYSLOG servers	✓ ✗

Determine your required level of maturity

How to scorecard “Required Maturity”

In order to determine how you should invest in your containerization build-out, you first need to understand what your goals/aspirations are. If you are not looking to leverage containers for any serious business functionality then your required level of maturity will be significantly lower than if your business 100% depends on container-based applications.

You should scorecard based on your short term and longer-term aspirations, as you need to be aware that maturity changes over time.

Be careful of setting too high a maturity level too soon, as this may lead to over-investment far in advance of needs, which leads to a negative ROI / high TCO.

Required Maturity Level	Business Use of Containers / Why are you adopting Containers
Advanced	Mission critical use, vast majority of applications will be (or are) running in containers. Absolute requirement for “five 9’s” of availability.
Capable	Multiple business critical applications will be (or are) deployed in containers. Need to deliver high levels of SLA.
Emerging	Multiple business important applications will be (or are) deployed in containers. Customer facing, or business internal services.
Opportunistic	One or more non-critical applications will be (or are already) deployed as containers to validate the value and operational impact
Ad-Hoc / Experimentation	Early experimentation into the business value of containers. No customer or internal facing apps. Trial use.
Traditional	Beginning to think about Containerization, no current adoption

Maturity Level	Personal Readiness	Organizational Readiness	Application Readiness	Technology Rediness
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 Team	Containerizable Monolith Applications	Container Experimentation
Traditional	Basic Linux Knowledge	Traditional IT Operations	Traditional Installable Software	Traditional IT Monitoring and Alerting Tooling

How can Portainer help me?

Portainer is a Container Management platform that helps you achieve higher levels of operational maturity at a significantly lower costs / faster time than other methods.

Portainer removes operational complexity, allowing you to operate with one or two levels of personal readiness lower than you would be able to without Portainer.

Portainer is an end-to-end container management platform tool, and so a deployment of Portainer in your environment immediately propels you to "capable" level in technology readiness. This is a very fast/efficient way of obtained technical maturity.

portainer.io