

# Choosing a Container-Based Platform as a Service

## With Special Considerations for DoD Implementations

15 August 2019

### Introduction

Kubernetes [3] has become a *de facto* industry standard framework for large-scale orchestration of containers. Several commercial companies and open source communities have leveraged Kubernetes to build container-based platform as a service (PaaS) products. While there are certainly non-Kubernetes-based approaches, a large number of DoD programs are choosing Kubernetes-based products for their container infrastructure. There are also several hosted / managed options for Kubernetes (Amazon EKS, Google GKE, etc.) as well. While these options can greatly simplify infrastructure deployment, many DoD programs handle classified data and/or must deploy to standalone (or tactical) networks that prohibit the use of a hosted, cloud-based PaaS. Thus, the focus herein is on locally deployed implementations.

This article aims to describe a few of the differences (at the time of writing) between implementing a stock Kubernetes system or leveraging a full-blown PaaS based on Kubernetes, as well as some of the critical considerations when deciding which option might be best for a given project.

### Kubernetes

Kubernetes is the open source engine that powers many container-based PaaS systems. It is a distributed system focused on scheduling containers across a cluster of nodes. It will allocate the requisite compute, storage, and networking resources required to run containerized workloads defined by declarative deployment descriptors. Kubernetes also provides the necessary APIs to manage the cluster along with command line utilities that can be used by administrators or other processes. Figure 1 illustrates the high-level architecture of Kubernetes.

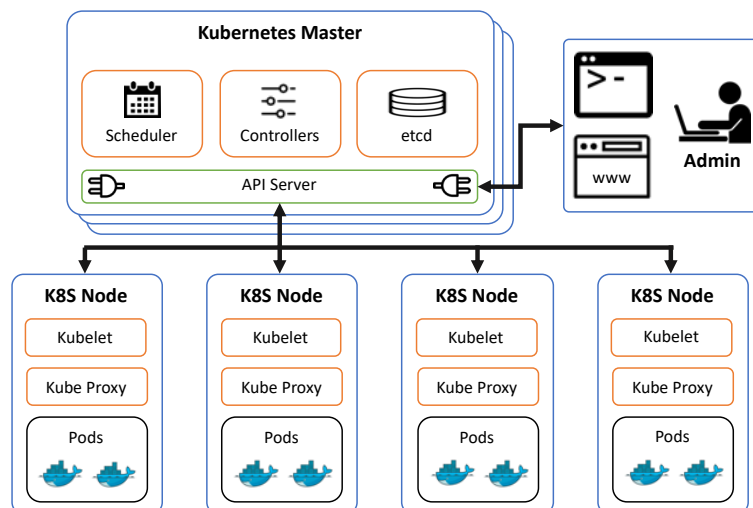


Figure 1. Kubernetes Architecture

Kubernetes itself is essentially a container orchestration framework, with a highly extensible architecture and a large ecosystem of extensions and plugins. It focuses mainly on container orchestration (e.g. scheduling, resource limitations, etc.) and cluster management (managing many nodes). Kubernetes itself does not provide a web-based administration console, an installer, software defined networking, nor a host of other features. However, there are a large number of related projects built to work with Kubernetes that add these functions.

Kubernetes was originally tied very closely to the Docker [2] implementation of containers. In recent months the Open Container Initiative [5] and the Cloud Native Foundation have created open specifications and a new generation of tools that have freed Kubernetes from tight coupling to Docker. Projects like CRI-O, runC, and containerD have decoupled Kubernetes from Docker and allow it to interact with any number of container runtimes that support the Open Container Initiative specifications.

## Kubernetes-Based Platform as a Service (PaaS)

In Figure 2 below, the Kubernetes project itself focuses mainly on only the Container Orchestration and Cluster Management components of a PaaS. A full-blown PaaS project / product brings together a broader set of functionalities typically needed to run containerized workloads in a production setting. Software Defined Networking (SDN), virtualized storage, logging and monitoring, a container registry, security and access controls are some of the common features added to Kubernetes by PaaS providers. In addition, it is common for a PaaS to include automated or semi-automated installers as well as provide an aesthetically-pleasing web-based management console. Together, the installers and management consoles greatly reduce the effort required to deploy and manage a Kubernetes based cluster.

A commercial PaaS based on Kubernetes will usually integrate the base Kubernetes project with several other commercial products or open source project to provide a turn-key solution. Many of these additional components are sourced from the open source community around Kubernetes. For example, there are more than half a dozen open-source software defined networking (SDN) systems available for Kubernetes. PaaS vendors will often select and integrate one of these projects into the PaaS, thoroughly test it, and integrate the SDN capability with other functions within the PaaS, such as security and access control, load balancing, auto-scaling, etc.

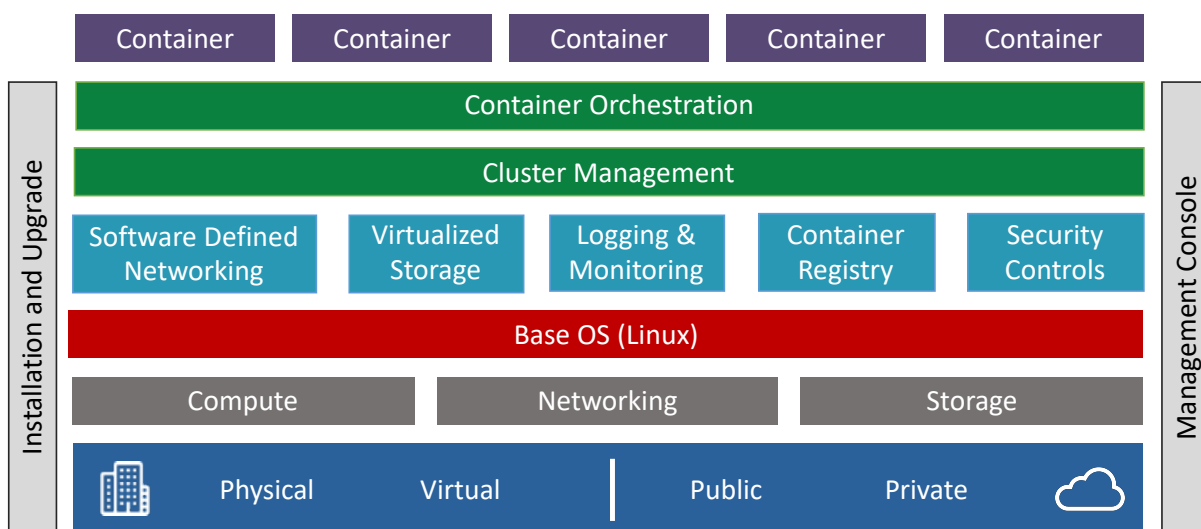












Figure 2. Container-Based PaaS Architecture

## Conceptual Model of the Kubernetes Ecosystem

While it is a somewhat loose analogy, a container-based PaaS is often referred to as a “cloud operating system”. A normal operating system (OS) is responsible for securely managing the resources of and scheduling processes on a single server. Similarly, a PaaS is responsible for securely managing the resources of and scheduling processes (containers) across a large cluster of servers. This analogy can be further extended by comparing the relationship of the Linux Kernel to Operating Systems, Package Managers, and Application Lifecycle Managers. The resulting conceptual model can help developers and system administrators evaluate the Kubernetes ecosystem by relating it to Linux, another well-known ecosystem that has been around for decades [6].

Table 1 below draws a parallel between the Linux Ecosystem and the Kubernetes Ecosystem using Red Hat products. Red Hat was used as an example simply because they have prominent products in both ecosystems. The same logical comparison could be done with other vendor products as well.

Table 1. Kubernetes vs. Linux Ecosystems

Kernel	Upstream OS (Community)	Enterprise OS (Commercial)	Package Management	Lifecycle Manager
				
Linux Kernel	Fedora Linux	Red Hat Enterprise Linux	RPMs	Red Hat Satellite
				
Kubernetes	OKD	OpenShift Container Platform	Helm Charts	Operator Lifecycle Manager



Kubernetes can be thought of as the “Kernel” of a container-based PaaS. It provides container scheduling and resource management across a cluster of servers. By itself, the Linux Kernel does not have the usual commands and services that users have come to expect from a full-fledged Linux OS. While extremely powerful, without things like bash, cron, vim, network / filesystem drivers, installers etc. the Linux Kernel, on its own, is of limited utility for most use cases. Kubernetes is similarly powerful and similarly lacks many of the capabilities of a full-fledge PaaS, including several key features generally required to deploy and manage a robust containerized application infrastructure.

Open source projects like Fedora transform the Linux Kernel into a full-fledged Linux Operating System Distribution by integrating additional functionality, such as the aforementioned tools and services. Part of the Fedora project’s role is to find a set of drivers, utilities, services, etc. that are all compatible with each other and with a particular version of the Linux Kernel. Fedora also provides a convenient installer that

installs the Linux Kernel along with these other components, so that an end user does not have to do this themselves. Similarly, the OKD project takes the base Kubernetes distribution and integrates SDN, storage, security components, a web-based management console, an installer, and other capabilities to deliver an open source PaaS Distribution.

Fedora serves as the upstream project for Red Hat Enterprise Linux (RHEL). RHEL is a heavily tested and commercially supported Linux Distribution. Red Hat takes Fedora and hardens it for use in production and enterprise scenarios. Red Hat publishes a lifecycle schedule for RHEL which includes a long-term support strategy. RHEL customers get priority support, bug fixes, and cyber security patches on a longer-term and quicker schedule than Fedora. Similarly, Red Hat takes OKD and produces the Open Shift Container Platform (OCP), which is a hardened and commercially supported PaaS Distribution. Much like RHEL, users of OCP will get long-term support and priority bug fixes.

Moving up the stack, the Red Hat Package Manager (RPM) format is an open standard way to package applications for installation in Red Hat flavor Linux distributions. The RPM format defines how packages are installed / removed from a single server. Likewise, Helm is an open source package manager and packaging format which describes how containerized workflows can be installed / uninstalled to a Kubernetes based PaaS. Finally, Red Hat Satellite provides application lifecycle management (upgrades, compliance, etc.). The Operator Lifecycle Manager (OLM) framework for Kubernetes provides lifecycle management for containerized workloads in a PaaS.

### A Key Trade-Off

In either ecosystem, the amount of work to deploy and maintain a production-ready system increases the further “left” you go, but the flexibility also increases. If building a Linux-based application, starting with the raw Linux Kernel would require a lot of work to first build enough of an OS to host your application. Similarly, if starting with Kubernetes, there is a lot of work to do to build a production-ready PaaS to host your containerized workloads. That said, with the increased work comes a greater amount of flexibility and control of exactly how the system is built. This will be a key theme for many of the tradeoffs between Kubernetes and a full PaaS.

## Kubernetes vs. PaaS Considerations

There are several considerations to evaluate when choosing between leveraging Kubernetes or a commercial or Open Source PaaS solution. Several of them are discussed below:

### Supported Infrastructures

It is possible, although complicated, to get Kubernetes running on almost any Linux-based operating system. This is largely because the installation process is manual. There are a few projects (kubeadm, kops, etc.) that aim to streamline installation of Kubernetes on several cloud providers. PaaS vendors typically build installers for their offering. These installers greatly simplify the installation and updating of the cluster. However, the set of infrastructures they support is sometimes limited.

### Hardening and Stability

PaaS projects generally lag behind Kubernetes versions. This is because they are taking care to check compatibility between components, develop security policy, create installation and upgrade procedures, generate documentation, and generally test the platform. This results in a greater degree of stability in the platform. The increased stability does come at the cost of slower adoption of new features and functionality. Early adopters often prefer Kubernetes or Open Source PaaS over commercial PaaS solutions. Some PaaS vendors attempt a more rapid integration of upstream features than others.

## API Stability

A non-trivial portion of the Kubernetes API is in an alpha or beta state. There are often breaking API changes between releases, and occasionally experimental APIs are removed altogether if the community decides to go in another direction. While the community does their best, and generally follows semantic versioning principles, there is no guarantee that the API that exists today will be there in a subsequent version. That said, generally once part of the API is marked stable and has been in place for a long time, it can be considered safe.

## Do-It-Yourself Functionality

Generally, when implementing raw Kubernetes, the project must select and integrate several other components that are not available by default. This includes:

- **Identity Management:** Kubernetes does not come with user and identity management built in. Recently the role-base-access-control features have greatly improved. However, if the project requires integration to LDAP or Active Directory for user authorization, this will need to be implemented by the project.
- **DNS Integration:** Kubernetes requires a functioning DNS infrastructure to route traffic to the appropriate containers. Projects will need to configure and integrate DNS.
- **Software Defined Networking:** Kubernetes has over half a dozen options for networking. Projects will need to integrate one to allow containers to communicate with each other.
- **Storage Infrastructure:** Projects will need to integrate a method for provisioning storage in the cluster.
- **Ingress and Load Balancing:** Projects will need to implement and integrate an ingress controller into Kubernetes to allow external consumers to leverage services in the cluster.
- **Service Mesh Implementation:** If a service mesh is required (e.g. Istio), projects will need to integrate it themselves.
- **Log Aggregation:** Kubernetes does not provide a log aggregation capability out of the box. Projects typically integrate FluentD, Elasticserach, and Kibana. These must then be secured, as they are not out of the box.
- **Metrics and Monitoring:** Kubernetes does not provide a metrics and monitoring capability out of the box. Projects typically integrate Prometheus and Grafana. These must then be secured, as they are not out of the box.
- **Management Console:** Kubernetes does not have a robust web-based management console by default. There is an open source Kubernetes Dashboard project that can be integrated, though it is not as robust as a PaaS.

It should be noted, that while each of these is not terribly complex, the project is now responsible for:

1. Researching the plethora of options out there, and selecting the best tools based on project requirements.
2. Choosing between open source and supported versions of all of these projects.
3. Ensuring that all of these integrated projects are compatible with the specific version of Kubernetes they have.
4. Tracking all cyber vulnerabilities across all these projects and upgrading / patching them as necessary.

The potential trap is that the full magnitude of work required to deploy and maintain Kubernetes in production is often not discovered until a project is twelve months down the road. At this point, the project may be past the point of no return and stuck with an implementation that is very hard to manage.

## Installation and Upgrades

Most PaaS products supply installers and provide “managed upgrades”. These installers install and upgrade all of the constituent components (as mentioned previously) in one fell swoop and are heavily tested by the vendor. Conversely, the project is often on their own when attempting to install and upgrade the cluster. Furthermore, the cluster cannot be upgraded as a whole. The project must usually upgrade each individual component. Care must be taken to ensure that when upgrading one component, compatibility with Kubernetes and each of the interdependent services is considered.

## Patching

The Kubernetes community does not provide long-term support. The community is fairly responsive to mission critical bugs and severe cyber vulnerabilities. However, the community does not “backport” fixes very far. Kubernetes will generally fix bugs in the latest release, and perhaps a point release one or two minor versions back. If your project is on a version that is more than 6 months old, it is likely that you will not receive a patch. Most commercial PaaS products provide long-term support and will patch much older versions of their products, in essence doing the “backport” for you.

## Cyber Security Accreditation

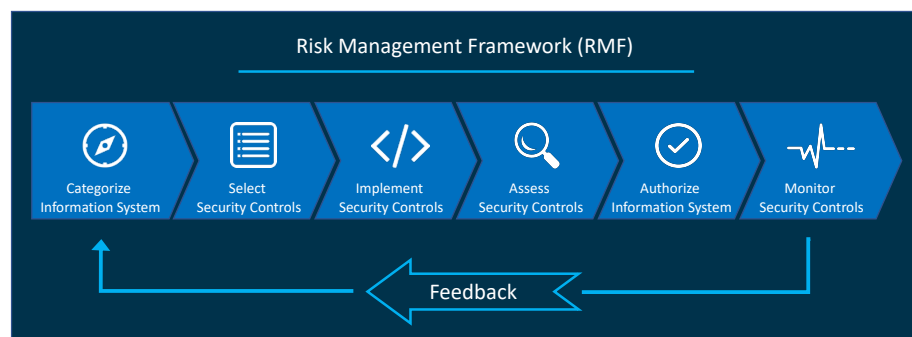
When building systems in the DoD, Cyber Security Accreditation is often a bottleneck. Many commercial vendors see the DoD as a viable market for their software and take steps to address cyber security accreditation. Most open source projects do not do this. When building from Kubernetes it is likely that a team will develop a highly custom collection of tools and services. Even if the common tools and services are integrated into Kubernetes, the project will likely have integrated them in a custom way. Cyber Security Accreditors will be looking at something they have not seen before. This can be problematic.

For Linux based systems, the base OS (RHEL, Ubuntu, etc.) has historically been a key focus of Cyber Security Accreditation. Standard Technical Implementation Guides (STIGs) are developed to secure the OS and automated vulnerability scanners have been developed to validate the security posture of the OS. The reason being that if the base OS is not secure, then the security mechanisms in the individual applications deployed in the OS are of little use. The PaaS is now viewed as the OS of the distributed cluster. Hardening of container infrastructure is as equally important to the security of the distributed system as the hardening Linux OS was to the individual server.

DoD guidance generally requires major open source infrastructure components be supported by a commercial company that can provide both long-term support and cyber vulnerability remediation. While the security community is just now contemplating the impacts of PaaS, it is likely that the same type of policy for the base OS, will be coming for the container infrastructure. Even prior to formal policy, accreditation organizations are already starting to require commercial support of the PaaS layer. If commercial support is not provided, the program will have to demonstrate they have a sufficient ability to self-support the various technologies involved.

Leveraging popular open source or commercial PaaS products may result in easier accreditation since

accreditors will have seen “prior art” and because vendors will work with the DoD in providing the requisite artifacts such as STIGs, OpenSCAP profiles, automation Scripts, etc.





## When to Choose a Commercial PaaS?

Leveraging commercial PaaS will be the least amount of work but will provide the least flexibility. If the product fits your use case, then the flexibility consideration is generally not an issue. It will take some time (several months) for new features and functionality in the open source community to become available. Projects should consider using a commercial PaaS if:

- The project has “production-like” reliability requirements.
- Long-term support is required.
- License fees are not prohibitive.
- License fees are offset by the large reduction in labor required to manage the PaaS.
- The project desires commercial support and priority bug fixing.
- The project team does not have the ability to patch or backport bug fixes themselves.
- The project values stability and security over being on the bleeding edge.

## When to Choose an Open Source PaaS?

Leveraging an open source PaaS is somewhat of a middle ground. This option will be less work and does not impose license costs. New features and functionality will arrive slower than using Kubernetes, but generally faster than using a commercial PaaS. Projects should consider using an Open Source PaaS if:

- Long-term support is not required.
- The project team has the ability to patch and backport bug fixes themselves.
- There is not a need to be on the bleeding edge, but the project values quicker access to new features and functionality than will typically be available in a Commercial PaaS.
- The project is comfortable with some level of self-support.
- License fees of a commercial PaaS are prohibitive\*.

## When to Choose Kubernetes?

Leveraging Kubernetes directly will require the most work but will also be the most flexible. Projects will have the quickest access to new features and functionality coming out of the Kubernetes ecosystem. Projects should consider using Kubernetes if:

- Long-term support is not required.
- The project team has the ability to patch and backport bug fixes themselves.
- There is no commercial or open source PaaS distribution that is installable in the target environment (e.g. exotic hardware).
- The project requires heavy customization of Kubernetes.
- The project requires tight control of the exact modules installed.
- The team has deep expertise in Kubernetes and the ability to troubleshoot and remediate issues.
- The primary goal is to deeply learn and understand the internals of Kubernetes.
- The project does not have “production-like” reliability requirements.
- There is a desire to be on the bleeding edge, potentially due to a newly developed, but required, feature that has not made its way into commercial or open source PaaS distributions.
- License fees of a commercial PaaS are prohibitive\*.

*\* Teams often underestimate the cost of the labor required to build and support a production-grade PaaS. System testing, backporting patches, version compatibility, integrating other tools, development of training and documentation, etc. can all be very expensive. A detailed cost analysis should be performed.*

## Kubernetes Based PaaS Examples

Below are several examples of commercial and open source PaaS products. This is not an exhaustive list nor an endorsement of any product. The list simply illustrates that there are many viable options.

- **Commercial PaaS**
  - Red Hat OpenShift
  - Docker Enterprise
  - Pivotal Container Services (PKS)
  - Kontena Pharos
  - VMWare PKS
  - Gravitational Gravity
- **Open Source PaaS**
  - OKD
  - Rancher (Support Options Available)
  - Apache Mesosphere Kubernetes Engine
  - Canonical Charmed Kubernetes (Support Options Available)
  - Zenko

## Product Lock-In

Most PaaS products provide additional, product specific functionality on top of the Kubernetes API. These features often greatly simplify application deployment and management. However, if a project leverages these features, porting to another PaaS may require rework. In many cases, it is fine to select a product and deal with the rework **IF** it is ever required. Proper design (encapsulation, abstraction, etc.) can also limit effort of moving to another PaaS. If a project desires to remaining agnostic, most PaaS products directly expose the raw Kubernetes API. A valid risk mitigation strategy is to limit interaction with the PaaS to the standard Kubernetes API. This will make the project maximally portable between PaaS products.

## Conclusion

Kubernetes is an immensely valuable open source project and is powering a large number of next generation systems in both the commercial world and the DoD. There exists a spectrum of options to deploy a Kubernetes-based infrastructure ranging from a raw Kubernetes implementation to a productized, commercially supported PaaS. Choosing the right option can have a large and profound impact on the long-term viability of a project. Hopefully this article helps projects properly frame the decision, allowing them to make the best possible choice and pave the way to successful deployment and sustainment of containerized workloads.

## About SOLUTE, Inc.

SOLUTE is a premier DoD engineering firm specializing in system modernization using the latest advances in Software Engineering, Cyber Security, Cloud Architectures, and DevSecOps. SOLUTE has a talented workforce with tremendous expertise in building, deploying, and managing containerized applications deployed to public / private cloud infrastructures. SOLUTE is leading the charge across multiple large and complex Navy, Army, and Air Force systems and is actively collaborating with DoD leadership on engineering best practices for mission critical PaaS deployments and DevSecOps best practices.

SOLUTE, Inc. is a Service-Disabled Veteran Owned Small Business, founded in 2002, headquartered in San Diego, CA and with offices in Denver, CO; Washington, DC; and Lexington Park, MD. SOLUTE has over 150 cleared engineers ready to bring innovation the DoD's most complex, mission critical systems.



## References

1. Cholewa, Tomasz. *10 most important differences between OpenShift and Kubernetes*. 10 June 2019. <https://cloudowski.com/articles/10-differences-between-openshift-and-kubernetes/>. Accessed 11 August 2019.
2. "Docker (software)." Wikipedia. Wikipedia.Org, 2 August 2019, [https://en.wikipedia.org/w/index.php?title=Docker\\_\(software\)](https://en.wikipedia.org/w/index.php?title=Docker_(software)). Accessed 11 August 2019.
3. Kubernetes. <https://kubernetes.io/>. Accessed 11 Aug 2019
4. Maayan, Gilad D. *Kubernetes vs OpenShift: What Is the Difference?* 11 July 2019. <https://dzone.com/articles/kubernetes-vs-openshift-what-is-the-difference>. Accessed 11 August 2019.
5. Open Container Initiative. <http://www.opencontainers.org>. Accessed 11 August 2019.
6. Swift, Greg. *K8S is the Kernel*. 12 March 2019. <https://logdna.com/blog/k8s-is-the-kernel/>. Accessed 11 Aug 2019.
7. Yegulalp, Serdar. *10 Kubernetes distributions leading the container revolution*. 15 May 2019. <https://www.infoworld.com/article/3265059/10-kubernetes-distributions-leading-the-container-revolution.html>. Accessed 11 August 2019.