Whitepaper

TIGERA

Building an Enterprise Container-as-a-Service Platform with Kubernetes

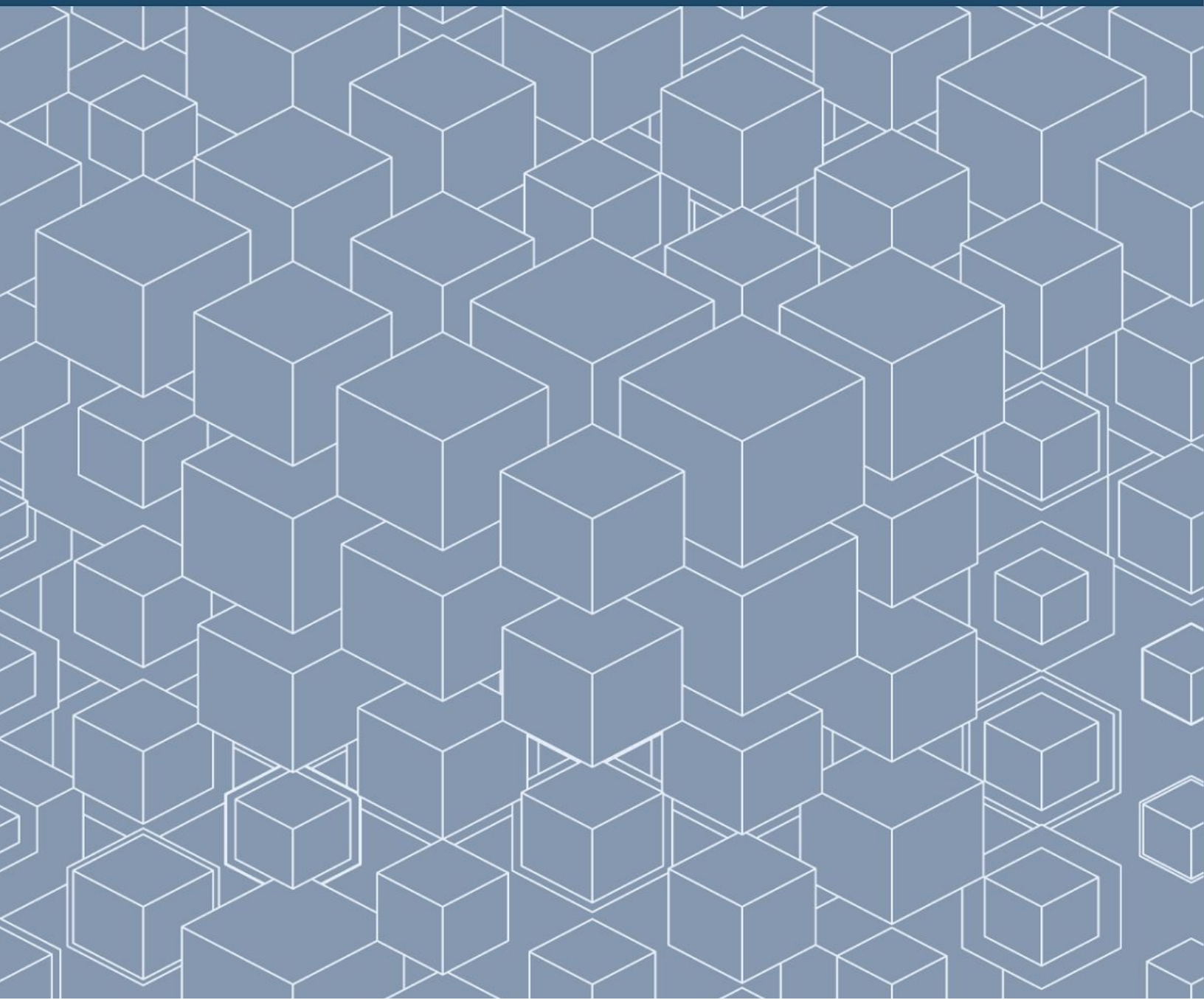


Table of Contents

Overview

Kubernetes has become the de facto platform to run cloud-native workloads. Businesses are considering investing in a consistent container platform to enable internal teams to run modern line-of-business applications designed as microservices. Enterprise IT is tasked with the responsibility of building a Container as a Service (CaaS) platform to host internal applications.

Even though Kubernetes is the foundation of the CaaS, there are other building blocks that make the platform reliable, secure, available, and resilient. This platform has to deliver consistent experience and workflow to developers dealing with Kubernetes internally (enterprise data center) and externally (public cloud).

This white paper discusses the key attributes of a production CaaS environment. It then maps the building blocks of the platform to various open source and commercial offerings from the cloud-native ecosystem.

Business decision makers and technology decision makers will find this guide useful in implementing Container as a Service within their organizations.

Why Build a CaaS?

Organizations are considering the microservices design pattern for building modern, greenfield applications. Microservices are fine-grained services that are most commonly packaged and deployed as containers. Multiple services are assembled together to form an application. Modern applications may have few hundreds of services working in tandem to deliver expected functionality. It is also common that many/most microservices will be used in multiple applications, creating a kind of service mesh or graph where most connectivity in the data center (or cloud) is internal (otherwise called 'east-west') rather than external (commonly called 'north-south').

Containers are one of the most disruptive trends affecting modern infrastructure. The rise of Docker followed by Kubernetes has accelerated the transition from VM-based monolith applications to containerized microservices.

As organizations start to adopt containers they face two challenges:

1. The complexity involved in deploying and scaling containerized workloads in production, and
2. The lack of integrated toolchain to manage distributed, container infrastructure.

To manage the deployment, discovery, and scaling of services, containerized applications need an orchestration engine. Kubernetes has emerged as the de facto container orchestration and management platform. There are other platforms that also orchestrate containers, but the de-facto orchestrator is Kubernetes. Every major cloud and container platform provider (Google, Amazon, IBM, Microsoft, Docker, Pivotal, RedHat, etc. have made substantial investments in Kubernetes, and have commercial Kubernetes offerings., Kubernetes manages multiple containers running across a collection of hosts, which is referred to a cluster.

Container runtimes like Docker and container orchestration engines like Kubernetes form the foundation of modern infrastructure. To take advantage of the microservices paradigm, organizations need a platform built on the foundation laid by Kubernetes. Through an integrated tool-chain, the platform manages the end-to-end lifecycle of applications - from source code to scaling. It bridges the gap between developers, operators, and users by enabling the rapid delivery of software.

This integrated container management platform created a new category of application delivery model - Containers as a Service (CaaS). Public cloud platforms such as Google Cloud Platform, Amazon Web Services, and Microsoft Azure offer CaaS alongside Infrastructure as a service (IaaS) and Platform as a Service (PaaS). This new delivery model is one of the fastest growing services in the public cloud. Cloud providers are extending existing building blocks - compute, storage, and network - to deliver a highly available, robust, resilient CaaS platform.

Like how a private cloud mimics the delivery model of the public cloud within an enterprise data center, CaaS can be built and deployed in the data center. Enterprises can extend their physical or virtual infrastructure to deliver CaaS to internal development teams. With Kubernetes as the level-playing field, CaaS becomes a consistent platform across the data center and the public cloud. Developers and DevOps can adopt a consistent workflow to develop, deploy, and manage containerized applications no matter where they run.

With CaaS as the common fabric across the data center and the public cloud, organizations can build hybrid applications that securely connect internal assets to the public cloud. CaaS is fast becoming an enabler of the hybrid cloud and multi-cloud deployments. Developers and operators can easily move applications across disparate environments.

Expectations from an Enterprise CaaS

CaaS goes beyond the integration of infrastructure and containers. It delivers the essential components required to manage containerized workloads in production. CaaS enhances security through a private registry to store and retrieve images. It provides centralized monitoring tools to track the health of production workloads. CaaS increases collaboration between developers and IT operations by providing a consistent and unified environment to build, package, and

deploy applications. A mature CaaS offering plays a critical role in accelerating DevOps adoption among enterprises.

Developers using CaaS in the public cloud expect the same experience when using internal implementations. They want the same availability, stability, security, and reliability of a public cloud-based CaaS.

Enterprise IT is expected to deliver a CaaS platform with the following attributes:

Consistent platform - Users of CaaS expect a consistent experience running in the public cloud and on-premises environments.

DevOps processes - DevOps pipelines that ensure rapid delivery of software should work consistently across the public cloud and on-premises environments.

Security - CaaS platforms ensure the security of infrastructure and applications. They detect vulnerabilities before deploying applications while constantly monitoring the infrastructure for potential violations.

Infrastructure reliability - Internal users of CaaS expect an SLA-driven service delivery model that delivers maximum uptime of the infrastructure and platform.

High availability of workloads - Apart from infrastructure, business applications deployed in CaaS need to be highly available.

Multi-tenancy & policy-driven management - CaaS has to provide strong isolation among the tenants utilizing the platform. Application deployment and management should be governed by policies.

These expectations put pressure on enterprise IT to deliver a robust CaaS platform to internal users. The IT teams are looking for the right components to build a stable, reliable, and secure CaaS platform.

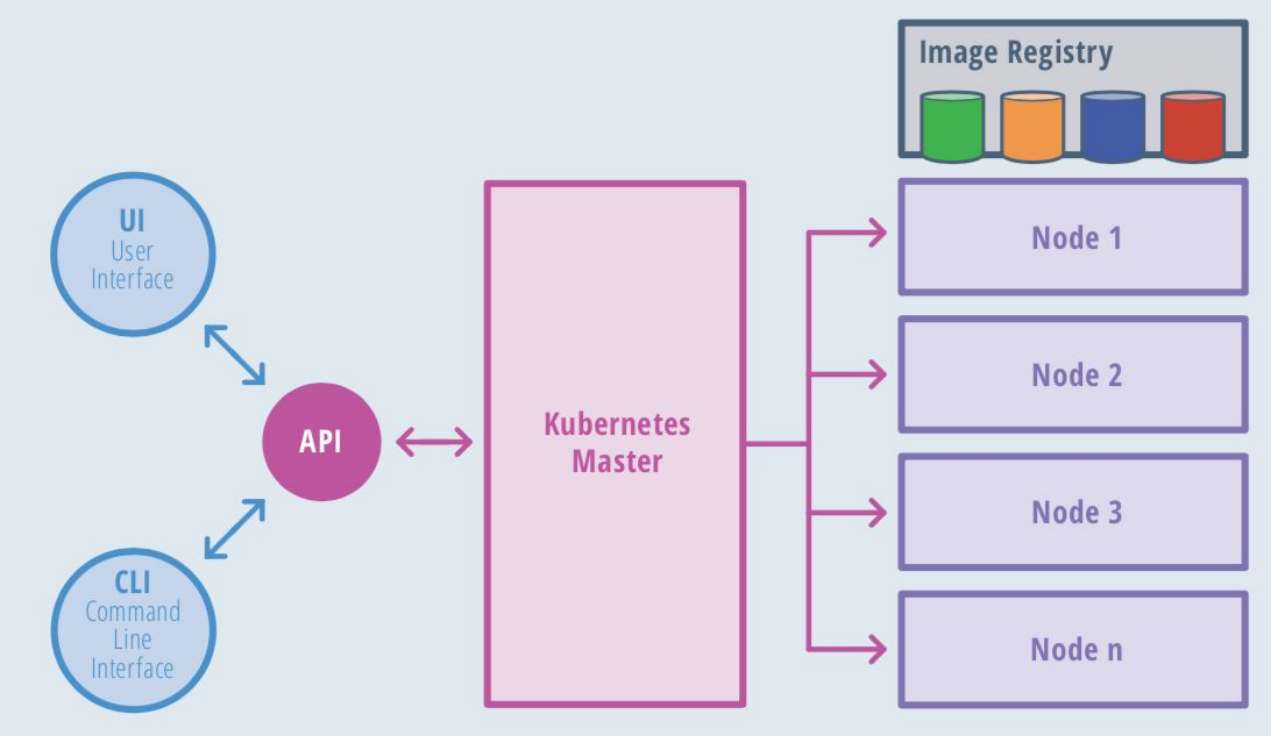
Kubernetes as the Foundation of CaaS

The most common CaaS is a highly distributed environment based on Kubernetes (which can also be known as Kubernetes as a Service, or KaaS). The cluster that is responsible to run the workloads has two node types - master and worker. The master nodes host the control and management components and manage the rest of the cluster.

Like most distributed computing platforms, a Kubernetes cluster consists of at least one (but usually at least three) master and multiple compute nodes. The components that (normally) run on the master nodes act as API endpoints, schedulers, controllers, state key/value store, etc.

Each Kubernetes node runs a container runtime, such as Docker, along with an agent that communicates with the master. The node also runs additional components for logging, monitoring, service discovery and optional add-ons. Nodes are the workhorses of a Kubernetes cluster. They expose compute, networking and storage resources to applications. Nodes can be virtual machines (VMs) in a cloud or bare metal servers in a data center.

DevOps tools that deal with deployment and scaling microservices talk to the Kubernetes control plane. When a workload is deployed, its characteristics such as the number of instances of each service, CPU, memory, and storage requirements are submitted to the master server. Based on the available resources, Kubernetes control plane decides the placement of microservices among the nodes of the cluster.

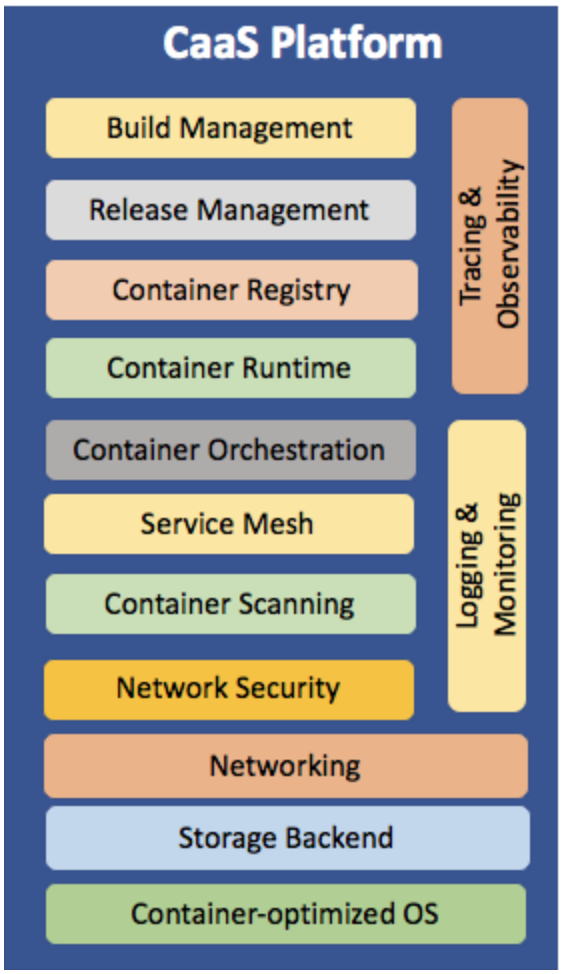


Apart from Kubernetes, CaaS platforms have other building blocks that are essential to managing the end-to-end lifecycle of applications. The next section takes a closer look a the building blocks and maps them to prominent open source and commercial implementations.

Building Blocks of CaaS

To build an enterprise CaaS offering, IT has to choose the best of the breed software from the cloud native ecosystem. From choosing the right operating system to implementing a CI/CD pipeline, the choice made at every building block plays a key role in building a production-grade container services platform.

The diagram below depicts the logical layers of the container as a service technology stack.



The bottom-most layer represents the physical infrastructure of the cluster in the form of CPU, RAM, and storage. The CaaS platform adds various layers of abstraction to optimally utilize the underlying physical infrastructure.

Let's take a closer look at each of these layers.

Container-optimized Operating System

Containers redefined the role of an operating system. With much of the heavy lifting moving to container runtime, an OS has become a thin layer that provides access to physical resources. This shift has resulted in a new breed of operating systems called container-optimized OS (COS).

When compared to a traditional OS, COS a lightweight OS with a much smaller footprint. It contains the most essential components that are required to run the container runtime. Choosing the right COS goes a long way in maintaining the CaaS deployment.

Customers can choose from Red Hat Atomic Hosts, Container Linux also from Red Hat (via the CoreOS acquisition), Ubuntu Core from Canonical, RancherOS from Rancher Labs to deploy the COS.

Most of the vendors offer an optional commercial subscription plan that includes regular updates, patches, and professional support.

Container Runtime

The container runtime is responsible for managing the lifecycle of a container. It provides the execution environment for containers. Container runtime acts as an interface between the workload and the host operating system.

Docker Engine is one of the most popular container runtimes implemented by CaaS providers. Apart from Docker, there are other choices such as Kata Containers, runC, and CRI-O.

Docker Engine Enterprise comes with enterprise-class support and professional services.

Container Orchestration Engine

As discussed above, the most common container orchestration engine for CaaS platforms in Kubernetes (KaaS). Kubernetes is responsible for scheduling the containers packaged as Pods in one of the Nodes. A Pod may contain one or more containers packaged and deployed as a unit. Kubernetes can maintain the desired state of Pods based on predefined policies. For example, a frontend may be configured and deployed to run at least 3 instances of the Pod. If one of the Pods gets killed or dies an untimely death, Kubernetes can automatically launch another Pod to maintain the desired count of 3. The same technique may be applied to run databases and other stateful workloads through StatefulSets.

Kubernetes comes with service discovery that makes it easy for pods to discover services that they, in turn, rely on to provide their service. This pattern makes Kubernetes the preferred platform to run microservices. To expose services to the outside world, Kubernetes can be easily integrated with either traditional hardware or software L4 and L7 load balancers or more modern, container-native load balancers and ingress controllers.

Even though Kubernetes can be deployed from the upstream open source distribution available on Github, customers may want to invest in a commercial distribution that comes with maintenance and support packages. Red Hat OpenShift (which can also be used as a PaaS), Canonical Distribution of Kubernetes, Mirantis Cloud Platform, Mesosphere Kubernetes Engine, Cisco Container Platform, Rancher Kubernetes, and Pivotal Kubernetes Service are some of the commercial distributions in the market.

Customers with investment in public cloud CaaS platforms based on Google Kubernetes Engine or IBM Kubernetes Service can choose GKE On-Prem or IBM Cloud Private, which are tightly integrated with the control plane running in the cloud.

Container Registry

Microservices are packaged as container images before they are deployed and scaled by the orchestration engine. The container registry acts as the central repository that stores container images. It is integrated with build management tools for automatically generating images each time a new version of the service is built. Release management tools pick the latest version of the image from the registry and deploy them.

Docker Trusted Registry, Red Hat Quay, and JFrog Artifactory are some of the commercially available container registries. Apart from storing the container images, they also perform vulnerability scanning on images, authentication, and authorization of users. Some enterprises require that repositories exist on premise (they disallow access to Internet-based registries). In those cases, some of the commercial CaaS/KaaS offerings offer internal registries. Otherwise, it is possible to build an internal registry with many of the same features.

Container-native Storage

Storage is one of the most critical components of a CaaS platform. Container-native storage exposes the underlying storage services to containers and microservices. Like software-defined-storage, they aggregate and pool storage resources from disparate mediums.

Container-native storage enables stateful workloads to run within containers. Combined with Kubernetes primitives such as StatefulSets, it delivers the reliability and stability to run mission-critical workloads in production environments.

Even though CaaS works with traditional, distributed filesystems such as NFS and Gluster, it is highly recommended to use container-aware storage fabric which is designed to address the requirements of stateful workloads running in production. Customers can choose from a variety of open source projects and commercial implementations.

Rook, Vitess, and OpenEBS are some of the open source projects while Portworx, StorageOS, and Red Hat OpenShift Container Storage offer commercial support.

Networking

Similar to container-native storage, the container-native network abstracts the physical network infrastructure to expose a flat network to containers. It is tightly integrated with Kubernetes to tackle the challenges involved in container-to-container, pod-to-pod, node-to-node, pod-to-service, and external communication.

Container-native networks go beyond basic connectivity. They provide dynamic enforcement of network security rules. Through pre-defined policy, it is possible to configure fine-grained control over communications between containers, pods, and nodes.

Choosing the right networking stack is critical to maintain and secure the CaaS platform. Customers can select the stack from open source projects including Flannel, Project Calico, OpenContrail, and Contiv. Vendors such as Tigera and Weaveworks offer commercial networking software.

Network Security

Network Security is an important part of hardening the platform. Enterprise IT should invest in a zero trust network layer to meet security and compliance requirements. These networks enforce fine-grained policy control at multiple points in the infrastructure, analyze for anomalies, and encrypt and authorize the traffic flowing between microservices using mutual secure protocols like Transport Layer Security (mTLS). Tigera is one of the leading zero trust network providers for Kubernetes and CaaS platforms.

Service Mesh

Service mesh has evolved as a core component of a CaaS platform. It enables fine-grained control of traffic among various microservices while providing insights into the health of each microservice.

Service mesh complements container-native networking. It focuses on east-west traffic while the north-south traffic is handled by the core networking layer. Service mesh adds a proxy to each of the microservice which intercepts the inbound and outbound traffic. Since it is placed close to each microservice, it can track the health of the service.

Envoy, Linkerd, Istio are some of the popular open source service mesh projects. Customers can choose from commercial implementations of service mesh offered by vendors such as Tetrate and Solo.io.

Static Analysis and Container Scanning

Security is critical to the success of a CaaS platform. Like other IT projects, security plays an important role in the implementation of container platforms.

A secure CaaS uses trusted images that are signed and verified. This feature is tightly integrated with the container registry component that stores container images. It implements a vault to securely store the secrets including usernames, passwords, and other sensitive data. CaaS must be integrated with existing LDAP and Active Directory deployment to configure integrated role-based access control (RBAC).

Open source projects such as Clair deliver static analysis of vulnerabilities in application containers. Aqua, Twistlock, and StackRox provide commercial solutions for securing container and Kubernetes infrastructure.

Tracing and Observability

Observability is an important building block of an enterprise CaaS. It has three elements:

- Monitoring
- Logging
- Tracing

Monitoring collects metrics from the infrastructure and application stack of the CaaS. A robust monitoring platform monitors the state of the Kubernetes cluster as well as the deployed applications. The agents installed in each node collect detailed information about the resource utilization, cluster health, node health, storage and memory availability, number of containers running on each node, along with detailed metrics related to Pods. Customers can deploy open source monitoring tools such as Prometheus, as well as either an ELK or EFK stack, or invest in commercial platforms such as Sysdig Monitor or DataDog.

Logging collects and aggregates the information, warnings, and errors raised by various components of the CaaS platform. Almost every component of Kubernetes generates logs that

provide detailed insight into the current state of the cluster. As a best practice, developers are encouraged to integrate logging into microservices. Various agents are deployed within the cluster to collect and stream the logs to a central repository. Service mesh software such as Istio and Linkerd are tightly integrated with the logging platforms. Monitoring platforms discussed above double up as repositories to store and analyze logs.

Since microservices are assembled from disparate, autonomous services, it is important to track the chain of communication and the time it takes for each service to respond. This mechanism is critical to monitoring and analyzing application performance. Open source tools based OpenTracing can deliver application performance monitoring (APM) capabilities to microservices. Commercial offerings include LightStep and AppDynamics that provide end-to-end tracing and monitoring features for microservices.

Build Management

For rapid delivery of microservices, each time code is committed to the repository, a new container image is built and pushed into the container registry. These images may be deployed to staging or test environments within CaaS for automated and manual testing. Build management involves converting the latest version of code into various artifacts including container images, libraries, and executables. It forms an important stage of continuous integration and continuous deployment pipeline.

The source code management system based on internal or external git repositories trigger an automated and secure build process which will result in a deployment artifact. The outcome from this stage may include container images, Helm charts, and Kubernetes artifacts such as Pods, Deployments, and ReplicaSets.

Jenkins is a popular build management software available in both open source and commercial versions. CloudBees offers a commercial version of Jenkins build management server. JFrog, SemaohoreCI, CircleCI, TravisCI, GitLabCI, and Xebia Labs have commercial CI/CD solutions for microservices.

Release Management

Release management represents the final stage of a CI/CD pipelines. It involves deploying a fully-tested version of microservices in the production environment. Advanced deployment strategies such as canary releases, blue/green deployments, rollbacks, rollfowards are a part of release management.

Spinnaker is one of the most popular release management tools for microservices. It complements Jenkins and other build management tools by automating the management and

deployment of artifacts. Armory, OpsMX, and Kenzan have a commercial implementation of Spinnaker.

Below is a summary of various tools and technologies discussed in the previous sections.

CaaS Building Block	OSS Reference Project	Commercial Implementation
Container-optimized OS	CoreOS Container Linux	Ubuntu Server
Container Runtime	CRI-O, dockerd, runc	NA
Kubernetes Distribution	Upstream Kubernetes	Red Hat OpenShift
Container Registry	Harbor	JFrog Artifactory, quay, docker
Storage Backend	Rook	Portworx
Network, Microsegmentation	Project Calico	Tigera
Service Mesh	Istio	Tetrate
Logging & Monitoring	Prometheus, EFK/ELK	Sysdig Monitor, DataDog
Tracing & Observability	OpenTracing	LightStep
Static Analysis and Container Scanning	Clair	Anchore
Build Management	Jenkins, CircleCI, GitLab, Semiphore	CloudBees Core
Release Management	Spinnaker	Armory

Best Practices for deploying and managing a CaaS Platform

There are a set of best practices that enterprises may follow to get the best out of the investment made in building the CaaS. Some of them are discussed below:

Ensuring high availability of the control plane

Since CaaS is a mission-critical platform to run line-of-business applications, enterprise IT should make sure that it is highly available. While the worker nodes can be easily added and removed to the pool, maintaining uptime of the control plane is important. The master nodes should be configured for redundancy and high availability to avoid a single point of failure. It's highly recommended that each component of the control plane - etcd, API server, controller, and scheduler run in multiple independent failure domains for resiliency. The storage layer used for etcd KV database should be highly available. Refer to the official documentation on configuring a highly available Kubernetes deployment.

Maintaining and managing the cluster

The Kubernetes community follows a quarterly upgrade cadence. Enterprises may plan to perform upgrades at least twice in a year.

Production Kubernetes clusters should be backed up frequently. During the upgrades, a new node pool running a newer version of Kubernetes should be created followed by draining and cordoning the nodes running an older version. If you invest in a commercial distribution such as Red Hat OpenShift, work with your vendor on the recommended upgrade process.

Velero (formerly Heptio Ark) offers the tools to back up and restore Kubernetes cluster resources and persistent volumes.

Integration with legacy infrastructure

Microservices deployed in Kubernetes may need to integrate and interoperate with existing applications such as databases. The best practice to expose legacy resources such as database clusters and ERPs is to configure a headless service within Kubernetes. A headless service acts as a proxy to the remote resource by creating a discoverable endpoint within Kubernetes Namespace. For more details on configuring a headless service, refer to the official Kubernetes documentation.

Implementing Authentication and RBAC

Enterprises have a corporate directory in the form of LDAP or Active Directory. Users, Roles, and permissions are centrally stored in the corporate directory. Kubernetes can be integrated with an existing LDAP directory for integrated authentication. This enables a fine-grained access mechanism to various Kubernetes resources and associated actions.

Kubernetes documentation has detailed steps for both authentication and RBAC.

Enabling hybrid cloud and multi-cloud integration

Customers may want to evaluate Open Service Broker API to integrate with existing public cloud platforms. OSBA exposes various managed services available in the public cloud through simple API. This architecture enables hybrid cloud and multi-cloud integration with an enterprise CaaS running on-premises.

Summary

Containers as a Service is one of the emerging trends of modern infrastructure. It is fast becoming an alternative to PaaS. Deploying a Container Platform in the enterprise data center involves choosing the right set of open source and commercial software.

Key takeaways from this report include the following:

- Choose an OS that's optimized for containers and Kubernetes
- Invest in a commercial Kubernetes offering that comes with support and maintenance
- Cloud-native storage and network layers play an important role in enterprise CaaS
- Service mesh is emerging as a key component for managing the east-west traffic among microservices deployed in CaaS
- Securing the images, performing static analysis, and implementing a zero-trust network are critical for enterprise CaaS
- DevOps and SRE need an observability platform to monitor the state of the CaaS and the deployed applications
- Modern build and release management tools enable rapid iteration of software development and deployment
- Adopting the best practices of deploying and managing the CaaS will result in higher availability, scalability, and reliability of the platform.

About Tigera

Tigera provides Zero Trust network security and continuous compliance for Kubernetes platforms. Tigera Secure Enterprise Edition extends enterprise security and compliance controls to Kubernetes environments with support for on-premises, multi-cloud, and legacy environments. Tigera Secure Cloud Edition is available on the AWS marketplace and enables fine-grained security and compliance controls for Kubernetes on AWS and Amazon EKS. Tigera powers all of the major Hosted Kubernetes environments including Amazon EKS, Azure AKS, Google GKE, and IBM Container Service. Tigera is also integrated with the major on-premises Kubernetes deployments and is shipped "batteries included" in Docker EE and fully integrated with Red Hat OpenShift.

Contact Tigera

To request a live demo of our solutions, visit: www.tigera.io/demo

For all other inquiries please visit www.tigera.io/contact.

About the Whitepaper Contributors

For more information about this study or related topics please contact us directly:

Janakiram MSV
jani@janakiram.com
Principal Analyst, Janakiram & Associates

Janakiram MSV is the Principal Analyst at Janakiram & Associates. He is an Ambassador for the Cloud Native Computing Foundation, and also one of the first Certified Kubernetes Administrators and Certified Kubernetes Application Developers. Janakiram is also a Google Certified Professional Cloud Architect, an Amazon Certified Solution Architect, an Amazon Certified Developer, an Amazon Certified SysOps Administrator, and a Microsoft Certified Azure Professional. His previous experience includes Microsoft, AWS, Gigaom Research, and Alcatel-Lucent. He works with platform companies on their product strategy and roadmap.

Vince Lau
Director of Product Marketing
vince@tigera.io

Vince has over a decade of experience with fraud and security, in addition to driving go to market (GTM) strategy for security products. Prior to Tigera, Vince worked for other notable fraud and security vendors including ThreatMetrix and Imperva. Vince holds an MBA from Santa Clara University, BS in Computer Engineering from Cal Poly San Luis Obispo, and a CISSP.