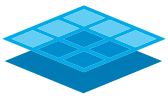
WHITE PAPER

Why Kubernetes Matters

We are on the cusp of a new era of software: instead of bolting on operations as an afterthought to the software development process, Kubernetes promises to bring development and operations together by design. With declarative, infrastructure-agnostic constructs to describe how applications are composed, interact and managed, Kubernetes enables an order of magnitude increase in operability of modern software systems. This document describes why Kubernetes matters, and why it represents a significant step forward for DevOps organizations.



PLATFORM9

Introduction

Over the last 2-3 years, Docker containers have made it very easy to run cloud-native applications on physical or virtual infrastructure. They are lighter weight compared to VMs and make more efficient use of the underlying infrastructure. Containers are meant to make it easy to turn apps on and off to meet fluctuating demand and move applications seamlessly between different environments e.g. Dev/Test/Production, or even across clouds. While the container runtime APIs meet the needs of managing an individual container on one host, they are not suited to managing applications running in multiple containers deployed across multiple hosts. This is where container orchestration tools, such as Kubernetes, are required.

According to the Kubernetes website – "Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications." Kubernetes was built by Google based on their experience running containers in production over the last decade.

Kubernetes Benefits

Write once, run anywhere (any cloud)

DevOps teams today must work across a wide range of infrastructure choices: on-premises dedicated servers, on-premises private cloud (IaaS), public clouds such as AWS, GCP and Azure, or a hybrid cloud across these. Historically, applications and DevOps tooling for operations ended up depending in many ways on how the underlying infrastructure was implemented. This "coupling" made it very expensive to support other infrastructure choices, despite compelling economic or strategic reasons.

Here are just a few ways in which organizations can end up with such "infrastructure lock-in:"

- Performance dependencies due to network architecture, routing or load-balancing
- Coupling to cloud provider specific constructs such as auto-scaling-groups or orchestration techniques (e.g. AWS Auto Scaling Groups and CloudFormation templates)
- Assumptions around specific storage backends and how they are available to compute nodes (e.g. EBS or RDS)

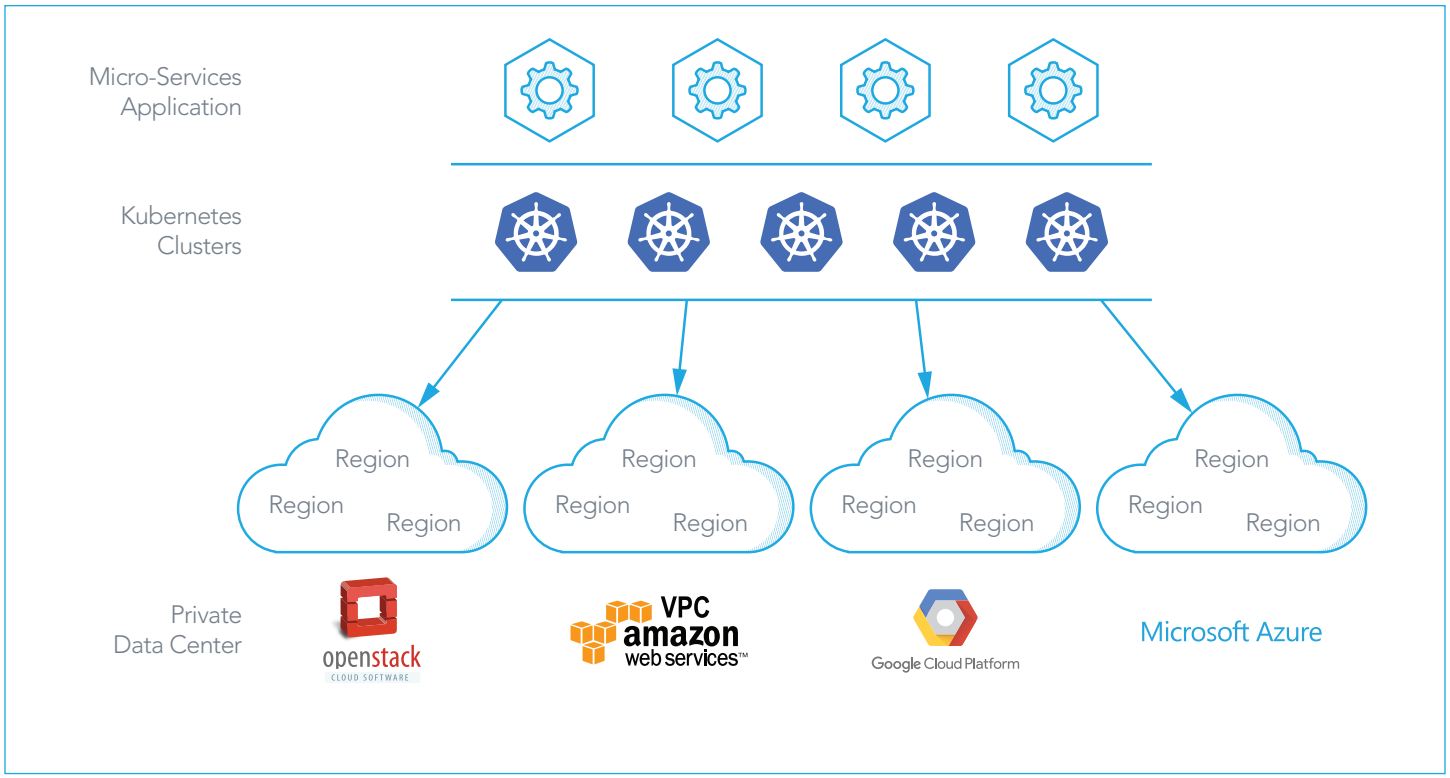
Some solutions, such as 1st generation Platform-as-a-Service, try to address these problems. However, they do so while being very restrictive and opinionated around programming languages and application frameworks, which makes them very limiting to most software development teams.

Kubernetes eliminates infrastructure lock-in by providing core capabilities for containers that enable DevOps, but does not impose any restrictions beyond those capabilities:

- Applications can be modeled incrementally using Kubernetes Services. Services can be used to abstract legacy systems that aren't yet ready for micro-services
- Any piece of code that can run in a container can be managed and run using Kubernetes Pods
- Kubernetes Services can be easily configured for discoverability, observability, horizontal scaling and load balancing

All of these mechanisms are infrastructure, cloud and application independent.

Run cloud-native applications on any cloud



Modularizing application design using containers

Containers today present an opportunity to decompose applications into smaller, more focused units with clear separation of concerns. The abstraction layer provided by an individual container image makes it possible to fundamentally rethink how distributed applications are built. Modular containers provide several benefits:

- Faster development with smaller, more focused teams that are responsible for specific containers (modules) with clear contracts between containers
- Isolation of dependencies and path-complexity to software deployment and testing
- Greater reuse of expertly-tuned, smaller components

Containers alone are not enough to realize these benefits; a system that integrates and orchestrates them is also required. Kubernetes facilitates modular containers using Pods: Pods are co-located containers that share resources like filesystems, kernel namespaces and an IP address. By making it easy to build focused containers that can be easily co-located as needed, Kubernetes removes the tendency to cram different functionality into the same container image.

Pods can be used to build applications using some powerful design patterns[1]:

- **Sidecar containers:** Sidecars extend a "main" container and make it better in a certain way. They are co-located with the existing, main container, but can be tested and reused independently. Examples of sidecars include data change watchers, monitors, event publishers and updaters.

- **Ambassador containers:** Ambassadors proxy a local connection to external systems. By abstracting the external system, they separate ownership and testing of these systems; and also enable use of different implementations in different contexts. Examples of ambassadors include data loaders, cache managers and proxies.

- **Adapter containers:** Adapters standardize and normalize output between containers. For instance, a monitoring application can use adapters to integrate with different application components that report data differently.

Modularity in application design can grow over time. Teams can start with a monolithic application and refactor different parts over time, to end up with well structured micro-services. Kubernetes helps modularize applications at every stage of the journey.

Fault-tolerant by design

A best practice for some cloud-native applications is that the application must be deployable across ephemeral (non-persistent) infrastructure elements. However, doing so requires integrating many different facets: provisioning infrastructure and applications, configuring load balancers, discovering services, lifecycle management and so forth.

A Kubernetes Service represents an abstract micro-service. Kubernetes makes distributed computing easy using micro-services and other features e.g. Controllers, to easily administer that micro-service across collections of ephemeral containers:

- A Service is exposed using a virtual-IP and collection of ports
- Services can be visible internal to a Kubernetes cluster, or visible publicly
- Services can be discovered easily - anywhere - using DNS: *<service-name>.<kubernetes-namespace>*
- Services are automatically load-balanced out-of-the-box, using an external load balancer or using Kubernetes' simple built-in load balancing mechanism
- Services can be used to abstract and interface with external elements, including external databases or application components that aren't yet ported to Kubernetes. This feature makes it easy to adopt Kubernetes incrementally

Manage software deployments, not just infrastructure

The process of deploying and redeploying (updating) software is the ultimate litmus test for most DevOps teams: this is where any gaps in design, scalability, version support and quality of the software show up, often in stressful ways.

Historically, operations teams spent a lot of their time managing infrastructure. Increasingly, as the world adopts DevOps, the focus is moving from infrastructure management to managing how software is deployed and updated at scale. Unfortunately, this is something that most infrastructure frameworks, including public and private clouds, don't provide direct support for. In the absence of such support, DevOps teams often script their own software deployment, scaling and update workflows. Some of the world's largest services are run with large software teams building custom ways to orchestrate software deployment[2]. This is clearly inefficient, and creates a lot of custom automation for a need that is felt by every DevOps organization across the world.

What is needed today is a way for DevOps teams to focus on managing software deployments, not just infrastructure. Kubernetes enables this focus by separating infrastructure elements (Nodes, Clusters) from Controllers. Kubernetes Controllers make it easy to use infrastructure to manage the application lifecycle.

The Deployment Controller, which is recommended for most use cases, enables:

- Deploying software for the first time in a scale-out manner across Pods
- Querying status to identify completed, in-process and failing deployments
- Updating deployed Pods using newer versions of application images
- Rolling back to an earlier deployment if the current deployment isn't stable
- Pausing a deployment mid-flight, and resuming it at a later time
- Scaling-out and scaling-in software deployments

Among other possibilities, there are a few specific deployment operations that are especially valuable to modern applications:

- **Horizontal auto-scaling:** Kubernetes auto-scalers automatically size a deployment's number of Pods based on the usage of specified resources (within defined limits). Kubernetes can optionally take advantage of externally-controlled auto-scaling by programmatically supporting the external scaler.

- **Rolling updates:** Updates to a Kubernetes deployment are orchestrated in "rolling fashion," across the deployment's Pods. These rolling updates are orchestrated while working with optional pre-defined limits on the number of Pods that can be unavailable, and the number of spare Pods that may exist temporarily.

- **Canary deployments:** A useful pattern when deploying a new version of a deployment is to first test the new deployment in production, in parallel with the previous version, and scale up the new deployment while simultaneously scaling down the previous deployment. This pattern, also known as "Canary deployments" is very easy with Kubernetes.

Container integrations for production

It is common for applications to require specific policies and controls when running in production. Kubernetes makes it easy to integrate such common policies with containers:

- **Distributing credentials:** Kubernetes Secrets facilitate delivery of sensitive credentials to applications without compromising security. The credentials are not stored in container images or environment variables

- **Resource management:** Containers can be provisioned by the Kubernetes scheduler with adequate resource QoS by providing limits and requests for compute and memory

- **Liveness and readiness probes (health checks):** Applications can sometimes go into a broken or unresponsive state, and require corrective action such as a restart. Kubernetes provides liveness probes to detect and remedy such situations

- **Lifecycle hooks:** While nodes and applications may fail at any time, Kubernetes provides signals and pre-stop lifecycle hooks to enable graceful shutdown of applications

- **Termination message:** Kubernetes speeds debugging by surfacing the cause of fatal errors, which are collected by the termination message facility

Why Other Alternatives Fall Short

Traditional PaaS is too restrictive

Kubernetes is not a traditional, all-inclusive PaaS (Platform-as-a-Service) system. It preserves user choice where it is important.

- Kubernetes does not limit the types of applications supported. It does not dictate application frameworks (e.g. Wildfly), restrict the set of supported language runtimes (e.g. Java, Python, Ruby), cater to only 12-factor applications, or distinguish "apps" from "services". Kubernetes aims to support an extremely diverse variety of workloads, including stateless, stateful, and data-processing workloads. If an application can run in a container, it should run great on Kubernetes.
- Kubernetes does not provide middleware (e.g., message buses), data-processing frameworks (e.g. Spark), databases (e.g. mysql), or cluster storage systems (e.g. Ceph) as built-in services. Such applications run on top of Kubernetes.
- Kubernetes does not have a click-to-deploy service marketplace.
- Kubernetes is unopinionated in the source-to-image space. It does not deploy source code and does not build your application. Continuous Integration (CI) workflow is an area where different users and projects have their own requirements and preferences. Kubernetes supports layering on CI workflows, but doesn't dictate how it should work.
- Kubernetes allows users to choose the logging, monitoring, and alerting systems of their choice.
- Kubernetes does not provide or mandate a comprehensive application configuration language/system (e.g. jsonnet).
- Kubernetes does not provide or adopt any comprehensive machine configuration, maintenance or management systems.

Other container orchestration frameworks

There has been an explosion of container orchestration tools in the recent past. Some of them preceed Kubernetes e.g. Apache Mesos, and some of them are more recent e.g. Docker Swarm. While each orchestration framework has its strengths and weaknesses, some of the frameworks that have become popular are Apache Mesos with Marathon, Docker Swarm, AWS EC2 Container Service (ECS), HashiCorp's Nomad etc.

Docker Swarm chose to maintain compatibility with Docker APIs so that users could easily transition from Docker to Swarm. AWS ECS may be preferred by current users of the AWS ecosystem. However, Kubernetes clusters can run on EC2 and integrate with AWS services such as EBS, ELB, ASG etc. Since it has been around longer, Mesos has been used in production environments more than other frameworks and also supports a larger scale of deployments. It works great when teams want to deploy other frameworks, like Hadoop or even Kubernetes, onto an abstracted view of infrastructure.

Most of these frameworks are catching up to each other in terms of features and functionality. However, Kubernetes has become immensely popular both due to its architecture and the large, active community around it.

Conclusion

Kubernetes represents a breakthrough for DevOps because it lets teams focus on the application architecture and deployment, instead of worrying about the underlying infrastructure plumbing. Given the multi-cloud world that DevOps teams are operating in, Kubernetes abstracts away the cloud provider and truly enables the goal of creating cloud-native applications that can be easily run anywhere.

About Platform9

Platform9 is a Silicon Valley startup building a business offering Open source-as-a-Service to enterprise customers. Currently, we offer two very popular projects, OpenStack and Kubernetes, as managed services. Since we function on a software-as-a-Service model, our solutions are very easy to deploy. Unlike other vendors, it doesn't require dozens of engineers to get these complex framework deployed in an enterprise-ready configuration.This extreme ease of use, coupled with production-ready SLAs, gives our customers:

- Production-ready environments in minutes, not months.
- Zero-touch upgrades and bug fixes, ongoing monitoring and troubleshooting
- Functionality right out of the box. No need to hire a team of experts. Platform9 "just works"

See how easy it is to deploy open source frameworks for your cloud infrastructure. Start a Free Trial.

References

1. The Distributed System ToolKit: Patterns for Composite Containers - Brendan Burns
   http://blog.kubernetes.io/2015/06/the-distributed-system-toolkit-patterns.html
2. A behind-the-scenes look at Facebook release engineering
   http://arstechnica.com/business/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering