A Kubernetes Primer

Recent years have seen tremendous growth in the container technologies market. From being a non-existent category just a few years ago, last year the market generated $762 million in revenue, according to 451 Research. This growth doesn't seem to be slowing: by 2020, the firm projects annual container revenue to hit $2.7 billion. This movement and growth suggests a significant shift in how businesses are planning for the future, and that application infrastructure is undergoing a significant transformation.

In fact, the entire application delivery supply chain is changing as the age of truly reusable code and abstract application creation is upon us. This change is fueled by the adoption of a few key technologies, including shared code repositories, continuous integration/development, and cloud computing. However, the ultimate driver of this movement is a software delivery mechanism: containers.

Containers change the way we think about application architecture and the speed at which we can deliver on business requirements. They provide consistency and portability across environments and allow developers to focus on building a great product, without the distraction of underlying execution details.

Though containers aid developers and simplify software development by abstracting away application execution details, it is becoming ever more critical to get operations right in this new world. Modular containerized deployments reduce operational complexity and present drastically reduced overhead compared to virtual machines. However, managing the ensemble of containers that compose a deployment presents some unique logistical challenges.

To address these challenges, many organizations are turning to Kubernetes to help orchestrate and manage their containers in production, development and test

environments. Kubernetes is leading this movement because it provides a reliable platform to build on, ensures workload portability, and allows system architects to design truly hybrid environments (across cloud providers and on-prem). It is an open source project that has served as the foundation for many cloud-native companies, and is now helping every organization reach web scale and expand the future of their applications.

Kubernetes comprises a rich set of complex features, and understanding this new platform is critical to the future of your organization. To complete your introduction to Kubernetes, this paper will provide clarity around its governing concepts and functionalities. This paper will also prepare you for a more productive discussion as you begin your organization's shift to containers.

What is Kubernetes?

Kubernetes is an open source container orchestration platform that helps manage distributed, containerized applications at massive scale. You tell Kubernetes that you want your software to run, and the platform takes care of virtually everything else.

Kubernetes provides a unified API to deploy web applications, batch jobs and databases. Containers—how apps are packaged and deployed in Kubernetes—cleanly decouple applications from their environment. Kubernetes automates the configuration of your applications, manages their lifecycles, and maintains and tracks resource allocation in a cluster of servers.

Borg, Omega and the Origin of Kubernetes

While widespread interest in and adoption of containers is relatively new, containers have been around for decades. Google was one of the first organizations to run containers at massive scale, and they did so well before they open sourced Kubernetes in 2014.

Kubernetes is the product of over 15 years of Google's experience running one of the most demanding production environments in the history of computing. It began with Borg, an internal Google project built to manage long-running services and batch jobs. As Google developed more applications to run on top of Borg, they

created a broad and intelligent ecosystem of tools and services for:

- Auto-scaling
- Self-healing infrastructure
- Configuration and updating of batch jobs
- Service discovery and load balancing
- Application lifecycle management
- Quota management

While Borg was highly successful and continues to run the majority of internal Google workloads, various limitations with the system motivated Google to work on a second-generation system called Omega.

Omega was developed with the desire to improve upon the engineering of the Borg ecosystem. It focused on a more intelligent scheduler that was needed to handle an increasing number of diverse jobs. With Omega, the scheduler was broken up into two separate schedulers with a central shared-state repository. This solution worked, but was complex in itself; a better system was needed.

Kubernetes is Google's third container management system. While it was developed as an entirely new codebase, it represents a combination of the approaches used and lessons learned in the monolithic Borg controller and the more flexible Omega controller. Kubernetes was designed to remove complexity and simplify the management of massive infrastructures. Since its launch, the Kubernetes project has fostered a diverse and highly dedicated developer community—now with a majority of contributors from outside Google—that continues to support and contribute to its growth.

Containers and why they drive modern application development

Let's take a step back from orchestration to talk about containers and why they are so popular.

A container provides two key features: a packaging mechanism and a runtime environment. At the runtime level, the container allows an application to run as an isolated process with its own view of the operating system. While VMs provide isolation via virtualized hardware, containers leverage the ability of the Linux kernel

to provide isolated namespaces for individual processes. This lightweight nature means each application gets its own container, preventing dependency conflicts. As a packaging mechanism, a container is typically just a tarball: a way to bundle the code, configuration and dependencies of an application into a single file. This eliminates the problem of “It worked on my environment, why doesn’t it work on yours,” because everything necessary to run the application consistently is transported with the container. Ideally, applications produce the same output regardless of environment, and containerization makes that ideal a lot easier to reach. The result is a containerized application that will start, stop, make requests and log the same way.

For any business, containers represent a large opportunity.

- Developers will spend less time debugging environment issues and more time writing code.
- Server bills will shrink, because more applications can fit on a server using containers than in traditional deployments.
- Containers can run anywhere, increasing the available deployment options.

For complex applications consisting of multiple components, containers vastly simplify updates. Placing each component in a container makes it simple to make changes without having to worry about unintended interactions with other components. This is the holy grail of decoupling.

This has led to containers becoming a great way to develop and deploy microservices. Microservices—applications with a single function—naturally benefit from containers, as they provide a clean separation between components and services.

Container management and orchestration

Containers allow us to ultimately break down an application into discrete functional parts. This has obvious advantages, but also means there are more parts to manage, increasing the workload as your business scales. This introduces complexities for configuration, service discovery, load balancing, resource scaling, and discovering and fixing failures.

Managing this complexity is very difficult when attempted manually. Clusters run

anywhere from tens, hundreds and even upwards of 1,000 containers; this is why sophisticated automation is necessary.

Kubernetes delivers production-grade container orchestration, automating container configuration, simplifying scaling, and managing resource allocation. Kubernetes can run anywhere. Whether you want your infrastructure to run on-premise, on a public cloud, or a hybrid configuration of both, Kubernetes delivers at massive scale.

Kubernetes as container orchestration

Kubernetes is container orchestration. This means Kubernetes figures out 'how' to run your containers. This allows developers to easily scale your applications without getting mired in the underlying details of scheduling, services and resource management. More explicitly, it provides three primary functions:

1. **Schedulers and scheduling**
   Schedulers are über managers. They intelligently compare the needs of a container with the resource availability of your cluster, and suggest where new containers might fit. Think of a scheduler as an assistant who manages a very large calendar. A controller, which actually assigns work, consults the scheduler and then does the work of assigning and then monitoring the container.

   The cruise control in your car and your household thermostat are declarative systems, just like Kubernetes. When you set your thermostat to 72 degrees, you expect it to work and to maintain your declared climate. It constantly checks the temperature and resolves it against a setting—72 degrees—on a regular basis.

   In the same way, Kubernetes seeks to maintain desired cluster state on your behalf. Freed from minding the proverbial temperature, your developers are able to focus on work that provides actual value for your business.

2. **Service discovery and load balancing**
   In any system, service discovery can be a challenge, and container orchestration systems are no exception: the more services that make up your app, the more difficult it all is to track.

   Thankfully, Kubernetes automatically manages service discovery. In typical use, we might ask Kubernetes to run a service like a database or a REST API.

Kubernetes takes notes about these services—including where they are located in the cluster—and is able to return a list if we ask about them later.

Service health is also important. Kubernetes may send client requests to a service in the cluster, but that's only useful if the service actually works. If Kubernetes detects a crash in your service, it will automatically attempt to restart it. In addition to these basic checks, Kubernetes also allows you to add more subtle health checks. For example, perhaps your database hasn't crashed. But, what if it's very slow? We can ask Kubernetes to track this and, if it detects slowness, to direct traffic to a backup.

For any complex system, load balancing is a must. Modern services scale horizontally: they scale by running replicas of the service pods. If you need more power, you simply run more replicas. The load balancer is the key piece that distributes and coordinates traffic across these replicas.

Kubernetes handles load balancing with aplomb. If we want a custom load balancing solution, like HAProxy, Kubernetes runs load balancing that way. If we want a cloud provided load balancer, like AWS Elastic Load Balancer, Kubernetes also makes it simple to run load balancing that way.

3. **Resource management**
   Every computer has a limited amount of CPU and memory, and how applications consume these resources varies significantly. An application that transcodes video might be CPU bound, and an application that parses text might be memory bound. This means the video application will run out of CPU first, while the text parsing application will run out of memory first. Completely running out of either resource leads to system instability and crashes.

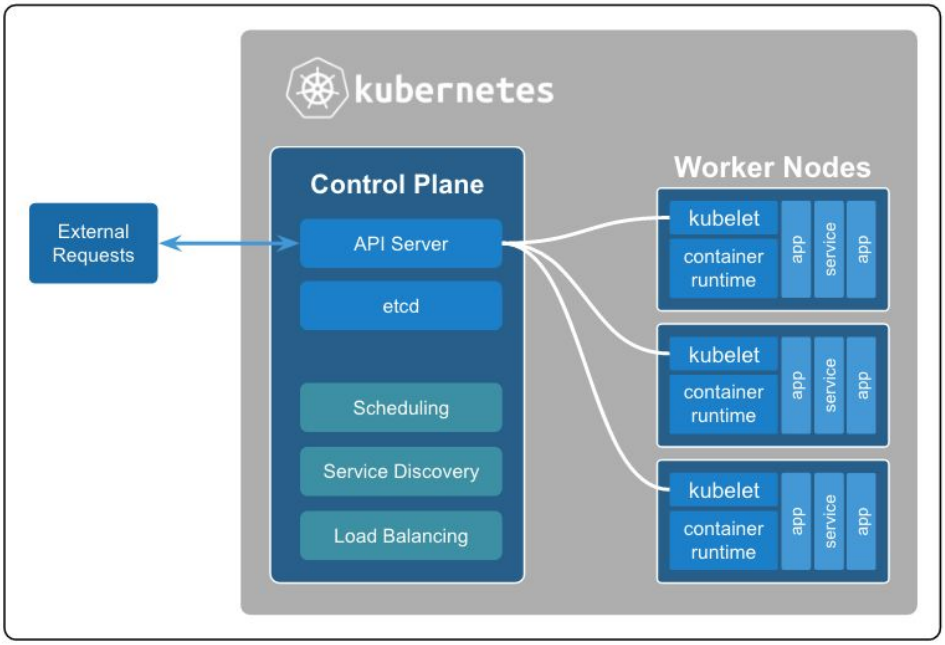
   Proper resource management is the result of intelligent scheduling. Kubernetes' advanced scheduler plays this delicate game of Tetris for you, scheduling applications so that resources like CPU and memory are well-utilized while staying cautious about the kind of overutilization that leads to system instability.

Basic Kubernetes architecture

Kubernetes is a mature platform that delivers an amazing set of features. Understanding its inner workings may be important for its operators, but for the

purpose of this paper, we present a simplified version of the architecture. This will establish a good model for reasoning about Kubernetes.

Some of these key concepts are depicted below.



Kubernetes is software for managing containerized applications on a cluster of servers. These servers are either controller or worker nodes. Together they run applications or services.

- **External requests**
  Users interact with Kubernetes through a declarative API. Here, they tell Kubernetes what their requirements are, describing their applications and services, and Kubernetes then does the hard work managing the cluster and implementing that declaration for them.

- **Control plane**
  The control plane is roughly equivalent to the concept of a controller. It acts as the brain of any Kubernetes cluster. Scheduling, service discovery, load balancing and resource management capabilities are all provided by the control plane. For this high-level architecture discussion, we will not get into the details of these functions. Rather, we will present them as parts of the control plane.

- **API server**
  Kubernetes' API server is the point of contact for any application or service. Any internal or external request goes to the API server. The server determines if the request is valid and if the requester has the right level of access, and then it forwards the request.

- **etcd**
  If the control plane is the brain, then etcd is where memories are stored. A Kubernetes server without etcd is like a brain that's unable to make memories. As a fault-tolerant, inherently distributed key-value store, etcd is a critical component of Kubernetes. It acts as the ultimate source of truth for any cluster.

- **Worker nodes**
  A worker node in Kubernetes runs an application or service. There are many worker nodes in a cluster, and adding new nodes is how you scale Kubernetes.

- **Kubelet**
  A kubelet is a tiny application that lives on every worker node. It opens a line of communication (listens) with the API server, waits to be told what to do, and then it carries out those actions. Under special circumstances, the kubelet will listen for events from the control plane, prompting it to carry out the requested actions on the worker node.

- **Container runtime engine**
  The container runtime (typically rkt or docker) runs containerized applications. It is the conduit between a portable container and the underlying Linux kernel.

So, you want to run an application on Kubernetes?
Create a manifest, and share it with Kubernetes. Manifests are simple YAML files which declare the type of application to run, and how many replicas are required to run an adequately resilient system.

Sharing the manifest with Kubernetes will cause it to pass through the API server, and ultimately get stored on etcd. The control plane will read your manifest from etcd, and place the application on a node.

Are you ready to scale?
Whether it's an application or cluster, Kubernetes makes it easy to scale. Scaling an

application is one of the best demonstrations of Kubernetes' simplicity. Change a value in your manifest, and Kubernetes will begin deploying more containers until it has reached your newly desired state.

Cluster scaling may occur if you use Kubernetes on a virtualized provider with a compute API. Kubernetes can be set to detect high usage and add more nodes to the cluster to improve performance by reaching out to the compute API and allocating an additional worker node.

What happens when a pod crashes?
Anything deployed on Kubernetes has a desired state. For example, if I deploy a web server, I can set a desired state of "three replicas". In other words, I expect three copies of the web server to be running. If one of those pods crashes, Kubernetes will automatically reconcile the difference and re-schedule the third pod. The scheduler knows we want three pods, but can see we only have two; so, it will add one more to satisfy the desired state.

What happens when a worker node crashes?
Kubernetes is a self-healing environment, always reconciling the observed cluster state with the desired state declared by the user. If a worker node crashes, your pods will automatically be rescheduled onto available nodes.

The benefits of Kubernetes

Containers are changing the way modern applications are built. Shifting from VMs to containers can seem daunting. The learning curve may be steep, but the benefits are real. Now is the perfect time to begin your containerized future. Here's why:

- **Scalability**
  Kubernetes automatically and immediately (depending on your cloud provider) scales up your cluster when you need it. It scales back down when you don't, saving you resources and money.

  Imagine you run an online event ticketing service where customers can purchase tickets 24 hours a day, 7 days a week with a load that is variable in time. You have a rough idea of what load (traffic) looks like when tickets go on sale during the day, and you know that evenings are rarely ever a strain on your system. However, there are a few very popular events that cause exponential spikes in

traffic. In this scenario, you would want your cluster to dynamically scale in order to meet the demand. With Kubernetes' Cluster Autoscaling and Horizontal Pod Autoscaler features, all of those adjustments happen automatically.

The Cluster Autoscaler is responsible for making sure the combined resources of the nodes are adequate given the observed resource demand on the cluster. The Pod Autoscaler takes care of tuning the number of pod replicas to control what size slice each application can take from the cluster resource pie.

- **Portability**
  Kubernetes can run anywhere. It runs on-premise in your own datacenter, or in a public cloud, such as Google Container Engine (GKE) or Amazon Web Services (AWS). It will also run in a hybrid configuration of both public and private instances.

  You may prefer to run some of your workloads on-premise. Other workloads you might prefer to run on the public cloud. With Kubernetes, it doesn't matter if you're on AWS or on-premise. The same commands can be used anywhere.

- **Consistent deployment**
  Kubernetes' ability to run anywhere also means deployments are consistent. This is true across clouds, bare metal and local development environments. Why? Containers embody the concept of immutable infrastructure. In other words, all of the dependencies and setup instructions required to run an application are bundled with the container.

  Here's a helpful analogy: Fine China vs Paper Plates. Fine china is special, often a gift, and often unique. For instance, you might recognize a server behaving strangely, and, in keeping with the analogy, remember who gave it to you. You spend time patching it, creating a customized environment to run your production. You might be scared to kill this server, and replace it with another one. This is usually because it's difficult to remember how it was set up, or what changed after the initial setup.

  Containers, on the other hand, are designed like paper plates: mass produced and identical. They set themselves up the same way, every time. Because they are immutable, additional configuration changes are prohibited. (You won't be adding a gold rim to your paper plates.) There's no fear in killing a container and starting another. The consistent experience means developers can spend less

time debugging and more time delivering business value.

- **Separated and automated operations and development**
  It's common for operations and development teams to be in contention. Operations values stability, which means being conservative about change. Development values innovation, which means valuing the ability to make and test changes with minimal overhead.

  Kubernetes resolves this conflict. Thanks to the intelligence and automation of Kubernetes, operations can feel confident in the stability of the system. Conversely, containerized deployments save both developers and operations time, allowing rapid iteration cycles and resiliency to coexist.

What to consider when deploying Kubernetes

Once you see the value of Kubernetes and what it means for the future of your company, the only thing left to do is deploy it. Before doing that, consider these questions:

1. **How do you develop and manage applications today?**
   Adopting a container-centric view of development is quickly becoming a necessity. For you and your organization, this might be new. The first step is to ask if your developers are already using containers.

   If they are, you're in great shape. You have an application that's ready to run on Kubernetes.

   If they aren't using containers, the first step is to get an application containerized. This may sound challenging, but containerizing an application can usually be done in an afternoon.

2. **Can you handle being tied to a vendor?**
   Many cloud deployments rely on provider-specific features to run an application. The result is vendor lock-in to that specific cloud provider.

   Kubernetes is open source, and designed to run on any kind of infrastructure. As a bonus, Kubernetes federation provides a unified interface for balancing workloads between multiple providers. As conditions such as price, service level or personal preference change Kubernetes gives you the power to react

accordingly.

3. **Can you build and manage your containers without the latest updates, features and functionality of Kubernetes?**
   If you decide that running your own instance of Kubernetes is not an option—you would not be alone. There are a wide variety of managed solutions. It's worth noting that not all of these alternatives are equivalent.

   As is common in software, companies will sometimes take an open source product, insert proprietary elements and give it a new name. In the case of Kubernetes, this happens too.

   This can be problematic when you consider the pace of Kubernetes development. Updates to Kubernetes provide new features, the latest fixes, and security patches. A proprietary solution will naturally lag behind this release cycle.

   When you select a solution that is pure upstream Kubernetes, it means you will receive updates as they are released. Running an out of date version of Kubernetes from a proprietary solution is a major security concern.

Secure and simplify deployment and management of containers with CoreOS

Ready to start your journey with containers? CoreOS is a leader and active builder of major container technology (including Kubernetes). We are the creator of essential tools for securing, simplifying and automatically updating your container infrastructure.

If you are entirely new to containers, start by exploring CoreOS Container Linux to run a fully-supported Kubernetes cluster. Container Linux is an automatically updating, minimal, and secure OS, making it the ideal foundation on which to build your Kubernetes environment.

Then, when the time comes to evaluate container orchestration platforms and running containers in production, try CoreOS Tectonic. Tectonic is our enterprise-ready and self-driving version of Kubernetes. Tectonic will:

- Ensure your cluster is always up to date with the most recent patches/fixes
- Simplify the installation and ongoing management of your Kubernetes cluster
- Add web dashboards, monitoring, and other essentials to your cluster

If that's not reason enough, Tectonic is free for users running up to 10 nodes or less. CoreOS builds the additional functionality of Tectonic around pure upstream Kubernetes. The result is an easy-to-use, enterprise ready solution, with all the advantages of Kubernetes.

Sources

- https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf
- https://research.google.com/pubs/pub43438.html
- https://www.youtube.com/watch?v=XsXlm4wmB6o
- http://queue.acm.org/detail.cfm?id=2898444
- https://coreos.com/blog/kubernetes-cluster-federation.html
- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture.md

---

Begin your Kubernetes journey today at coreos.com!

Contact: sales@coreos.com or call +1 (800) 774-3507
