Should I Implement Kubernetes in My Datacenter Cloud?

The Evolution of Kubernetes in the DevOps Environment

Improvement is essential in any DevOps environment. The IT professional continuously looks for new ways to implement efficiency, reduce downtime, and save money. One way the IT profession executed improvement in the application environment with web-scale needs was with Linux containers. Linux containers are an attractive solution when compared to traditional server virtualization because they are independent execution environments, resulting in the feel of a virtual machine without the weight and overhead of a guest operating system. Prior to container technology, deploying and rolling out applications involved more overhead, time, and complexity. Because of the ease of debugging, management, and the quick remediation of issues, IT professionals became enormous fans of container technology.

Container Technology to the Rescue

With its efficiencies of virtualizing the operating system, container technology allows for larger scale applications in virtualized environments and has the added benefit of building and testing applications more quickly. Efficiency and enhancement is what any IT professional strives for, so it is no surprise managing Linux containers would be the next improvement to container technology. *Docker* containers is one such platform for management. *Docker* containers are a fast and effective way to transport software within the cloud and are fast becoming the standard unit of application deployment.

Docker is excellent for managing the lifecycle of one machine, one container, but running a server cluster on a set of Docker containers on a single Docker host is vulnerable to a single point of failure. Docker containers are no longer sufficient enough - tools for managing the containerized datacenter infrastructure was required - one with a special emphasis on treating the datacenter as one large server. *Kubernetes* is one such management tool for proficiently managing containers across clusters.

The Proficiency of Kubernetes

Kubernetes is the framework for building distributed platforms. It provides the foundation to build tools that are needed to run the infrastructure. Kubernetes schedules, runs, and manages containers in a cluster of virtual or physical machines, specifically it:

- Manages the lifecycle of containerized applications running in production.
- Automates the distribution and configuration of applications.
- Ensures higher levels of utilization and efficiency.
- Maintains and tracks the global view of the cluster.

Kubernetes was originally developed as an open source project by Google for managing containers and container clusters of Google's datacenter. *Kubernetes* is lightweight, portable, and extremely scalable. An essential feature of *Kubernetes* is it takes any available machines in the network (virtual or physical) and makes them homogeneous: the datacenter is no longer a collection of computers, it *is* the computer.

Kubernetes is an advanced type of monolithic scheduler, improving on other scheduler design limitations by decoupling:

- The application from the node. Apps are not assigned to any specific machines. Decoupling facilitates team growth - as the infrastructure grows, decoupling eases this growth.
- Various scheduling and monitoring components, improving system availability.

What Makes Kubernetes Technology Ideal in a Datacenter Cloud?

Kubernetes is more than a scheduler for containers because its design neatly solves the many difficult problems associated with clustering microservices and containerized applications. In a Kubernetes DevOps environment, the basic features used to run applications are:

- Pods: Units of deployment, which are a small group of tightly coupled homogeneous containers. Pods use a shared network and data volumes communicating via local host. Pods use a routable IP address.
- Replication controllers: Ensures availability and scalability. There are two states in the Kubernetes world, the *desired state* and the *actual state*. There are health checks on the system verifying that the actual state and the desired state are equivalent. For example, when deploying a pod, *x* copies (replicas) are run. The system starts or kills pods and handles pod failures.
- Labels: Key-value pairs for identification. Labels contain the declarations of the application:
    - Containers (image/tag).
    - Environment variables (CPU, RAM, etc.).
    - Data volumes.
- Services: Collection of pods exposed as an end point. Pods can die, so services provide a permanent *virtual* IP address and DNS name. If a pod dies, the IP address and DNS stay the same, despite the actual node not being the same. Services provides simple load balancing, including session affinity.

Why should you implement Kubernetes?

Efficiency, reduced downtime, and money saved. As with most datacenter systems, the collective CPU utilization is around five percent which is an inefficient use of resources. Systems may be automating, but they are wasting a ton of money. *Kubernetes* makes better use of the datacenter system resources and eliminates waste. *Kubernetes* not only deploys (creates the world), it maintains (heals the world). If any node or service breaks, Kubernetes fixes the problem without anyone being paged in the middle of the night, and that is always a good thing. The primary goal of *Kubernetes* is to use all available resources as the production

workloads come in and adjusting the resources accordingly. This not only increases efficiency, it reduces the amount of money spent on processors that are doing absolutely nothing, along with the amount of time and energy wasted in any network downtime – which is the goal of any DevOps team.