International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 04 Issue: 05 | May -2020 ISSN: 2582-3930

Server Scalability Review: Technical & Operational Feasibility

Dhunav Dhuray

Prof. Shailendra Tiwari, HOD CSE, Astral Institute of Technology and Research.

Abstract – A survey on Kubernetes with scaling both horizontal and vertical can result in cheap and high speed response time. Mainly managing microservices and maintaining thousands of containers are difficult but with Kubernetes, it is automated and made hassle free to work with. Scalability helps in increasing the response time saving the money by scaling down when not needed.

Key Words: Kubernetes, scalability, microservices, container.

1. INTRODUCTION

Cloud computing is an emerging general purpose technology (GPT) that could provide a fundamental contribution to efficiency in the private and public sectors, as well as promote growth, competition, and business creation. It is an Internet-based technology through which information is stored in servers and provided as an on-demand service to clients [1].

Server consolidation is an option widely used to maximize the effective use of server resources to decrease the number of servers required by a company. Generally, servers in many enterprises run at 15-20% of their capacity. Needless to say, this is not a sustainable ratio in the current economic environment. In order to reduce unnecessary costs and increase the return on investment in the data center, companies are now increasingly turning to server consolidation. Even though consolidation can considerably improve the efficient use of server resources, it may also bring about complex configurations of data, applications, and servers that can be confusing for the average user to deal with. To mitigate this issue, companies use server virtualization to mask the details of server resources from users while optimizing resource sharing [2].

In server virtualization, single server executes the task of multiple servers by sharing out the resources of an individual server across multi-environments. The server virtualization and consolidation (SVC) benefits result from a reduction in the overall number of systems and related recurring costs (peripheral, cooling, power, rack space, etc.)[3].

In the real world, I expect most of us are going to be running both containers and VMs on our clouds and data-centers. The economy of containers at scale makes too much financial sense for anyone to ignore. In addition, containers tend to lock you into a particular operating system version. That can be a good thing: You don't have to worry about dependencies once you have the application running properly in a container [4].

Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. Containers are often referred to as "lightweight," meaning they share the machine's operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs [5].

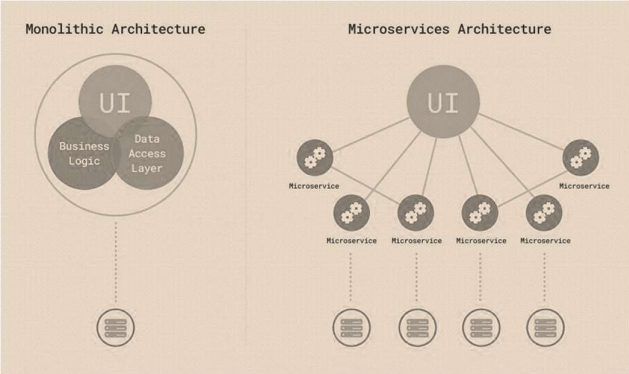
Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. Horizontal scaling, Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage [6].

Micro services system architecture is the process of splitting up core services into their own ecosystems. A key part of your application may be an image processing service that can save, delete, and cache and manipulate images. This service could be set up as its own infrastructure which means that it would be separated from the other application services. You'll often hear the term separation of concerns when referring to micro services. Although each core service having its own infrastructure can make scalability easier, it can still add a lot of complexity to your application. You'll now have to manage multiple servers but also change your application code to handle these changes. Vertical scaling is often thought of as the "easier" of the two methods. When scaling a system vertically, you add more power to an existing instance. This can mean more memory (RAM), faster storage such as Solid State Drives (SSDs), or more powerful processors (CPUs).The reason this is thought to be the easier option is that hardware is often trivial to upgrade on cloud platforms like AWS, where servers are already virtualized. There is also very little (if any) additional configuration you are required to do at the software level. Horizontal scaling is slightly more complex. When scaling your systems horizontally, you generally add more servers to spread the load across multiple machines. With this, however, comes added complexity to your system. You now have multiple servers that require the general administration tasks such as updates, security and monitoring but you must also now sync your application, data and backups across many instances [7].

2. Discussion

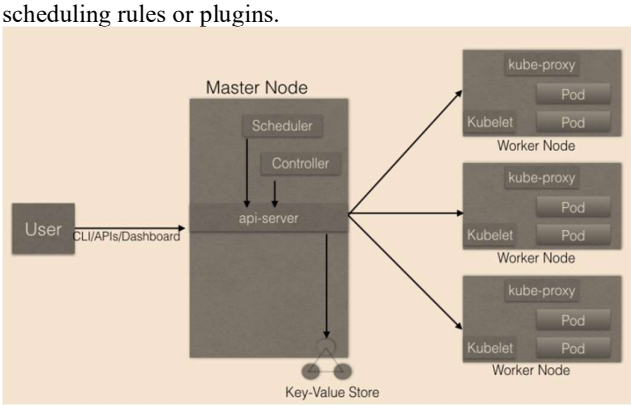
A monolith has a rather luxurious taste in hardware. Being a huge, single section of software, which nonstop grows, it must run on a solo system which has to mollify its compute, memory, storage, and networking requirements. The hardware of such size is both complex and expensive. They are carved out of the monolith, unglued from one another, becoming distributed components each labeled by a set of specific features. Once weighed all collected. These are loosely coupled microservices,

each performing specific occupational function. All the purposes grouped together form the overall functionality of the novel monolithic application.

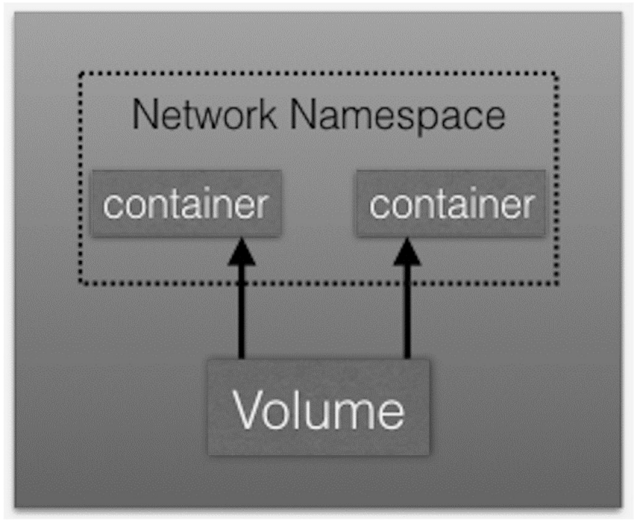


With container images, we confine the application code, its runtime, and all of its dependencies in a pre-defined format. And, with container runtimes like runC, containerd, or rkt we can use those pre-packaged images, to create one or more containers. All of these runtimes are moral at running containers on a solo host. But, in repetition, we would like to have a fault-tolerant and scalable resolution, which can be accomplished by creating a single controller/management unit, after connecting multiple nodes together. This controller/management unit is generally referred to as a container orchestrator.

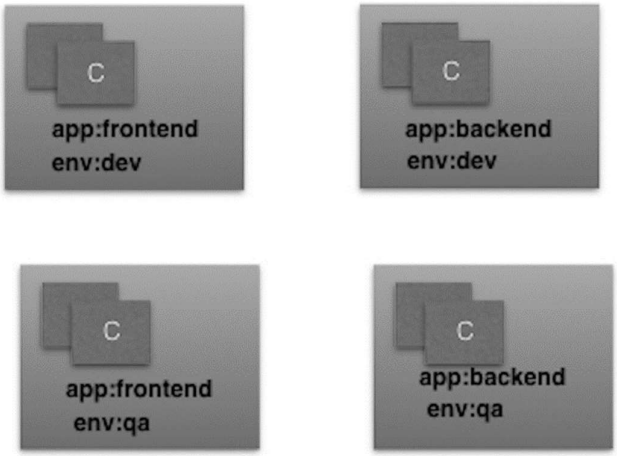
Kubernetes' architecture is modular and pluggable. Not only that it orchestrates modular, decoupled microservices type applications, but also its architecture trails decoupled microservices patterns. Kubernetes' functionality can be extended by writing custom resources, operators, custom APIs, scheduling rules or plugins.



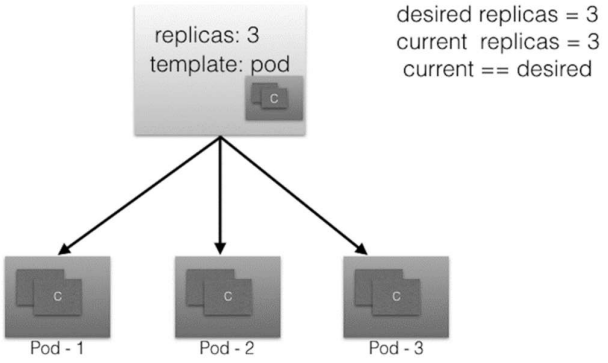
A Pod is the smallest and simplest Kubernetes object. It is the unit of deployment in Kubernetes, which represents a single instance of the application.



Labels are key-value pairs attached to Kubernetes objects (e.g. Pods, ReplicaSets). Labels are used to organize and select a subset of objects, based on the requirements in place.

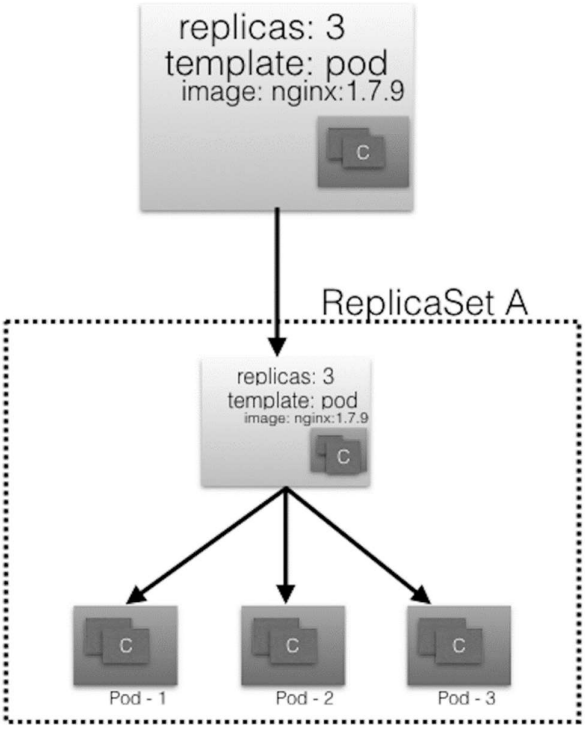


A ReplicaSet is the next-generation Replication Controller. ReplicaSets support both equality- and set-based selectors, whereas ReplicationControllers only support equality-based Selectors. Currently, this is the only difference.



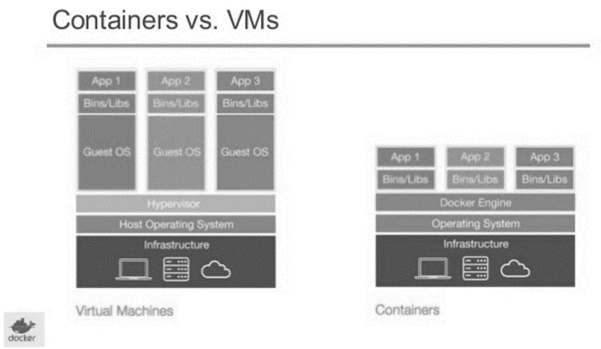
Deployment objects provide declarative updates to Pods and ReplicaSets. The DeploymentController is part of the master node's controller manager, and it ensures that the current state

always matches the desired state. It allows for seamless application updates and downgrades through rollouts and rollbacks, and it directly manages its ReplicaSets for application scaling.



3. CONCLUSIONS

A container that runs as privileged inside a virtual machine, without resource limitations, security profiles and so on, it's kind of a smart tarball and nothing more. But if you put together all the capabilities provided by containers you can reach a good isolation, plus a light and easy ecosystem to run, distribute and manage your application. Starting a virtual machine is more expensive in terms of time than starting a container. The same goes for the distribution and building part. How munch a Dockerfile looks easier than other solution related virtual machine provisioning. They are similar in that they both provide isolated environments – they can both be used to package up and distribute software. However, containers are typically much smaller and faster, which makes them a much better fit for fast development cycles and microservices. The trade-off is that containers don't do true virtualization; you can't run a windows container on a Linux host for example. It's also worth pointing out that several companies are trying to create tooling around slimmed down VMs to try to get the best of both worlds e.g. hyper.sh, Intel Clear Containers and vSphere Integrated Containers.



VMs allow users to manage hosts by APIs and offer infrastructure elasticity. Docker allows users to define software as small lego blocks to assemble, so they embrace modern architectures: immutable infrastructures, microservices, distributed software, and more.

REFERENCES

1. S. Dutta and I. Mia, "The global information technology report 2009–2010," in World Economic Forum and INSEAD, SRO-Kundig Geneva, Switzerland, 2010.
2. Soaring Eagle Database Consulting, "What is server consolidation and when is it necessary?." 2018. [Online]. Available: https://soaringeagle.biz/what-is-server-consolidation-and-when-is-it-necessary/.
3. C.Chuan-Fu and C.Shiuann-Shuoh, "Implement server virtualization and consolidation using 2P-cloud architecture," Journal of Applied Science and Engineering, vol. 20, no. 1, pp. 121-130, 2017.
4. S J. Vaughan-Nichols, "Containers vs. virtual machines: How to tell which is the right choice for your enterprise." 2016. [Online]. Available:https://www.networkworld.com/article/3068392/containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterprise.html
5. Kubernetes, "Production-Grade Container Orchestration." 2019. [Online]. Available: https://kubernetes.io/.
6. IBM Cloud Education, "Containerization." 2019. [Online]. Available:https://www.ibm.com/cloud/learn/containerization.
7. Linux Academy, "Scaling". [Online] Available: https://linuxacademy.com/blog/cloud/scalability-cloud-computing/