Containers – To Use It or Not to Use It

Understanding New Technologies to Evaluate Need for Containers

A Technical Paper prepared for SCTE•ISBE by

Avinash Raghavendra
Solution Architecture, NFVi and Cloud
Ericsson North America
6300 Legacy Dr, Plano, TX 75024
469-266-3077
avinash.raghavendra@ericsson.com

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

Introduction

With the proliferation of Cloud Computing into all major segments of the society, be it private, public or hybrid, Network Function and Application owners now have a multitude of choices to deploy their Network Functions/Applications. Previously it was deployed directly on a bare metal, now the applications can be virtualized or containerized.

One of the most popular and rapidly growing technology is Containers. The containers with their cloud friendly features and functionality have rapidly made strides to be in the forefront of the next big wave of deployments be it Edge, 5G or any Network Function.

But is it a panacea for all that ails a virtual or physical world? In this paper, we shall look to answer that important question by evaluating the needs of the network function to identify the best technology fit, be it a Virtual Machine or a Container for the Network Function/Application.

There are many different aspects that are considered as part of this evaluation. These include Infrastructure, Networking, Application Design & Deployment. Other Operational considerations including scalability, availability, monitoring, and life cycle management also play a very important role in the evaluation. Finally, one must not ignore the security considerations that play an important role in this evaluation.

Content

The dawn of general-purpose computing spurred the application revolution. Application were developed and over time optimized to run on various compute architectures. As applications became more specific in purpose, the resource utilization climbed higher. The resource utilization also became harder to predict for application placement. In medium and large enterprises, this brought an unexpected result, underutilization of compute resources.

The first step taken by the industry to solve this problem was virtualization. Applications were either uplifted or rewritten to be virtualized. Virtualization meant the applications had well defined boundary and were clearly quantifiable at definition and at runtime. This method caught on rapidly and with more adoption we started to see the benefit of virtualization, which were better utilization of hardware and network while providing better orchestration of deployment and life-cycle.

Specialized applications that were CPU, RAM and Network bound were virtualized quickly, saw the limitations of virtualizations. The limitation was of wide range, from virtual layer over head for CPU and Network to more static deployments. These issues resulted in development of customization for dedication of CPU, RAM and Network to virtual workloads which in turn caused more static deployments. Static deployments resulted again in underutilized compute resources and islands in workloads.

Solving these issues meant bringing the workload of application back to bare metals. Containers provide a packaging for application with access and resource reservation while sharing the underlying kernel. Containers eliminates the need to have guest operating system there by removing the overhead caused by virtualization. By contrast, containers require container runtimes that does not add to the continuous bulk overhead like in virtualization but just facilitate the initiation of the container and lightweight runtime linkage to the host OS (Operating System) and its common libraries.

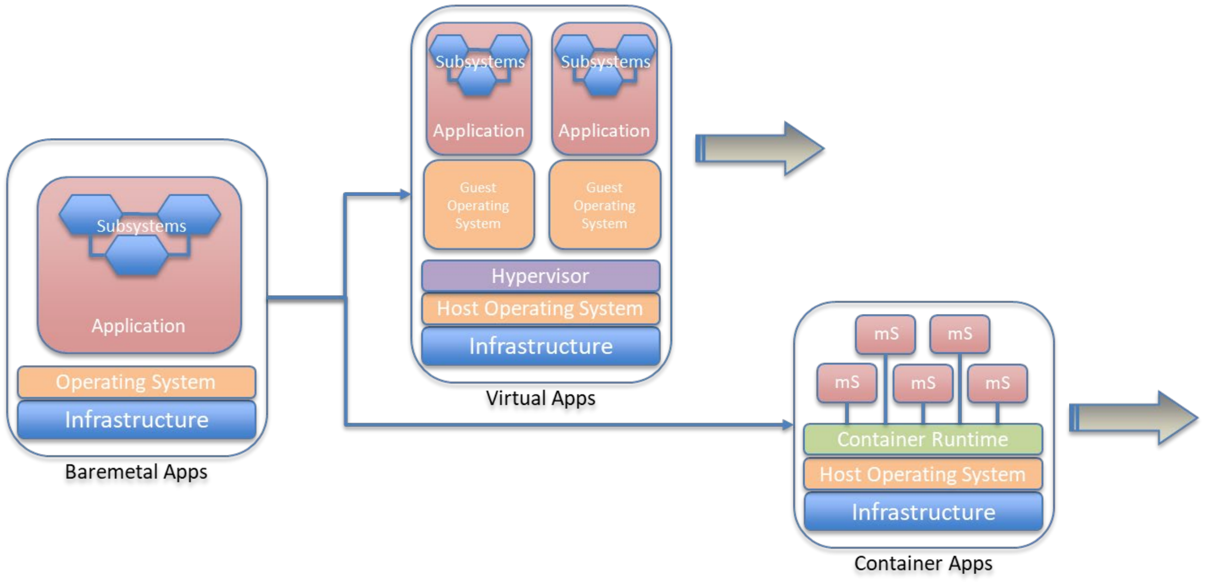

Figure 1 – Deployment Model

A Case for Containers

A declarative model is where the definition and configurations of an application is specified ahead of its deployment in a structured way. An application that is modeled on declarative way is the best fit to move to containers.

Historically applications have been designed and built to be monolithic, but this is changing. Application that are designed to be its smallest unit of work that interact using REST (Representational State Transfer) interfaces are called microservices. Microservices isolate a single application and its dependencies, all the external software libraries the app requires to run both from the underlying operating system and from other containers. Applications that are designed as microservices are best suited to run and managed as containers. Another benefit of microservices are that they are extremely portable since all its dependencies are packages together without the common libraries and operating system. Applications that are microservices should also callout their individual CPU, memory and storage requirements.

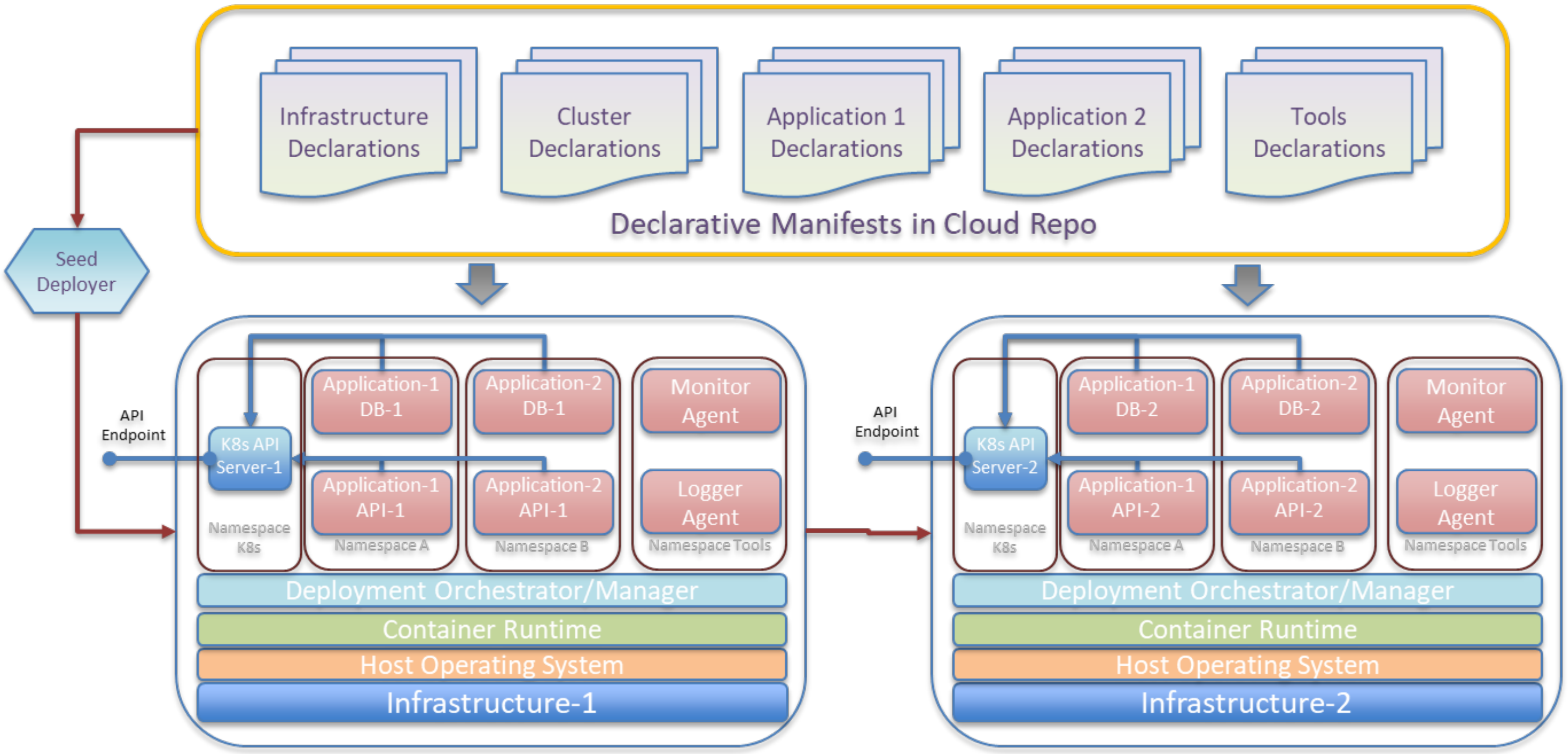

Figure 2 – Container Model Expanded

Existing and planned infrastructure also plays important role in deciding for containers. Containers work well on small but distributed deployments of infrastructure. This is because microservices that interact closely to deliver a solution are usually deployed with multiple instances over a geo-redundant infrastructure to load balance and to provide resiliency. The benefit of such deployments is on-demand scalability, when the need for those services increases, more application instances can be quickly deployed to meet the demand just at the required region and later scaled down.

The benefit of the portability of well-defined and de-coupled containers is high resiliency and scalability. Since it is well defined, it can be automated for runtime deployment. With successful and repeatable automation, multiple instance of the same piece of software can be quickly deployed on demand, geographically with little change.

Networking is another aspect that requires consideration for container technology. Containers work well with simple networking layout and standard network protocols. Container, though in use in IT enterprises for many years are still nascent in advanced networking. New developments are being made at fast pace, but it is best suited for application that need or have requirements with simple networking model TCP(Transmission Control Protocol), HTTP (HyperText Transfer Protocol)and HTTPS (HTTP Secure) and technologies like OVS (Open vSwitch), SR-IOV (Single Root I/O Virtualization) and DPDK (Data Plane Development Kit).

Life-cycle management and Operation is critical for any application. The operations and the team behind it must be very well versed on container technologies like Docker, Containerd, Kubernetes etc. to take advantage of the benefits of containers. DevOPS model is where the development and operations are part of the same process chain that seamlessly delivers and operates the application and solutions. Application as containers requires their features and capabilities to be developed and deployed iteratively. This combined with multiple instances of the same microservice means little or no impact when new features are delivered. This methodology is called rolling deployment.

Monitoring and alerting mechanisms will change with container as well. It is no more a single entity alerting on its various subsystems. The monitoring is now on each microservice that consumes the resources independently to operate while sharing the common infrastructure. This means the monitoring thresholds must be declared again and configured at deploy time. The monitoring software must also understand and differentiate each unit of software and their dependencies on each other. Clear visualization of alerts and dependencies helps in better life cycle management of the whole solution.

Each container or microservice logs independently. This means the log capturing mechanisms must now be capable of handling the several pieces of logs, roll and archive them. It should also provide mechanisms to search and visualize the various logs that are generated by each microservice.

A Case against Containers

Security is the most important yet least understood components of applications. Applications that handle single channel security policy enforcements expect policies are applied to the front-end services while the rest of the internal applications are hidden behind façade. This is true in case of virtualized workload where frontend application that accept ingress traffic are usually secured with TLS and firewall etc. while internal applications like database etc. do not need such policies. Such application is a least fit for container or microservices methodology. Each container deployed on an infrastructure introduces a small but definite surface area that can be compromised or used to compromise the infrastructure. The DevOPS (Development-Operations) model that is not equipped to scan all container images that need to be deployed also weakens the security of the solution. In a virtual world, a privilege of an application is contained within a virtual machine, so a rouge application at worst can bring down a VM (Virtual Machine), but a rouge container can now access the bare metal and compromise the whole infrastructure or the solution.

The design and architecture of software and application is the foundational in deciding deployment and runtime model used. An application that is designed as top down with relatively full analysis requirements sees less benefit from the container model. Containers benefit from iterative introduction of small features seamlessly rather than massive changes that disrupt the solution. Not all applications perform better in container model, applications that are tightly coupled to provide a specific service in efficient way may not be suitable to be broken down in to containers. Such applications are better suited either as virtual machine or application specific hardware.

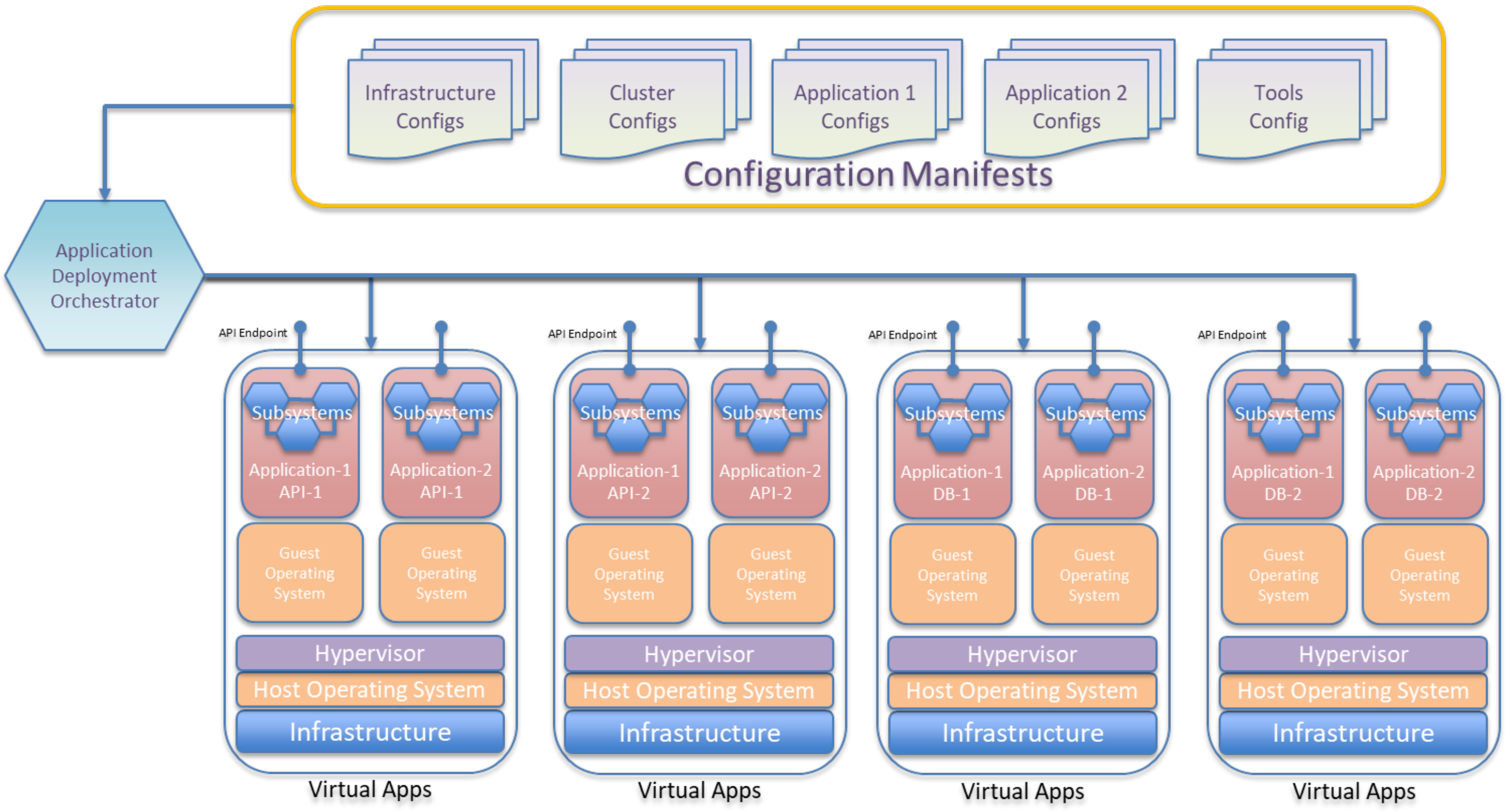

Figure 3 – Virtual Model Expanded

The CI (Continuous Integration)/CD (Continuous Deployment) model for traditional software tend to look at application as a black box that has overall input code and overall output application and test based on that. Such models cannot fit in container model as each container now represents a unit of software providing its services decoupled from other containers. This means the CI/CD model should not cater to individual container, its code and its dependencies and package it, scan it and test it based on the container specific interfaces. The CI/CD model also must tie all the individual containers together in the sequence represented by the solution to test the end-to-end flow and be able to deploy it. The conversion of traditional CI/CD model to true DevOPS model is in most cases not feasible if the traditional model is already successful.

Existing operational and support model plays a crucial role too. Enterprises and operators who have a working and successful operational tooling and support process in traditional software and application either on bare metal or as virtualized would experience an immense surface area to cater in a container model. Each container requires individual logging, monitoring and altering capabilities and appropriate seamless action or pro-action from the operations. Monolithic applications that define, operate, monitor and alert its subsystems efficiently does not see a benefit by moving to container model, on the contrary the software might become more complex and less efficient. This is also true for application that does not rely on declarative model.

The birth of containers was in IT and Enterprises which had little or no specializations in terms of workloads like NUMA (Non-Uniform Memory Access) awareness, usage of advanced networking like SR-IOV and DPDK. So, application that do not demand specializations work well in the container model. On the other side most TELCOs and Content Providers have applications that requires very specific technologies to function and meet the required efficiency and performance. It would be better for such applications to wait till advanced technologies are integrated, evaluated and mature before adoption.

Conclusion

Containers and its ecosystem offer a tremendous advantage to runtime capabilities and efficient utilization of underlying infrastructure. Every person or organization planning to create a piece of software must look at these emerging technologies and evaluate themselves to adopt to it. Operators of existing software, application and deployments must consider ways to incrementally add tooling, knowledge and process to convert to container model if their evaluations can conform container model.
A high-level flow chart represented in figure 4 can be used to evaluate a specific application based on factors discussed in this paper.

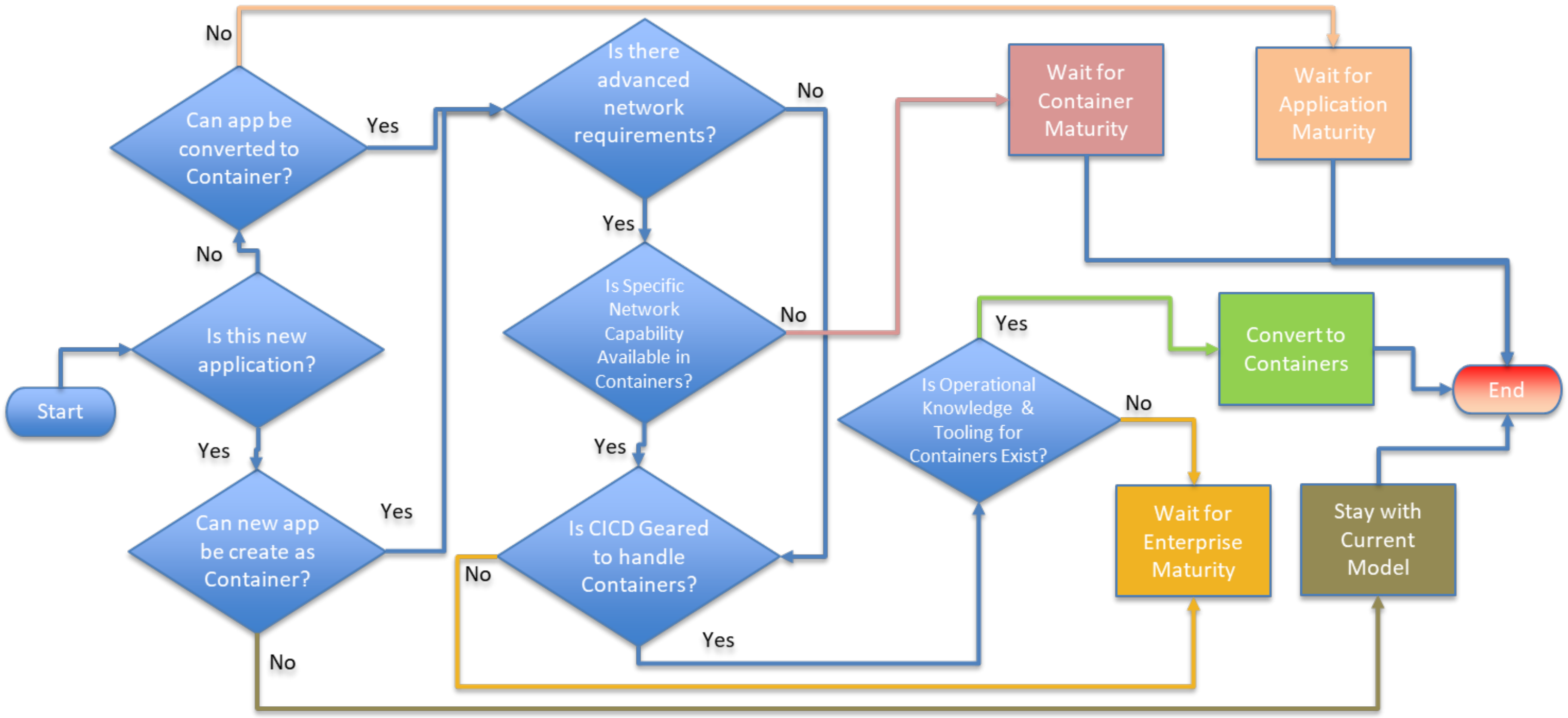
Start
Is this new application?
Yes
No
Can app be converted to Container?
No
Yes
Can new app be create as Container?
Yes
No
Is there advanced network requirements?
No
Yes
Is Specific Network Capability Available in Containers?
No
Yes
Is CICD Geared to handle Containers?
No
Yes
Is Operational Knowledge & Tooling for Containers Exist?
Yes
No
Wait for Container Maturity
Wait for Application Maturity
Convert to Containers
Wait for Enterprise Maturity
Stay with Current Model
End

Figure 4 – Evaluation Flow Chart

Abbreviations

NFV	Network Function Virtualization
NFVi	NFV Infrastructure
Edge	Closest to consumer
NUMA	Non-Uniform Memory Access
REST	Representational State Transfer
TCP	Transmission Control Protocol
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
OVS	Open vSwitch
SR-IOV	Single Root I/O Virtualization
DPDK	Data Plane Development Kit
DevOPS	Development-Operations
TLS	Transport Layer Security
VM	Virtual Machines
CI	Continuous Integration
CD	Continuous Deployment
API	Application Programming Interface
E2E	End-To-End
OSS	Operations Support System
VNF	Virtual Network Function
QoS	Quality of Service

Bibliography & References

OCI: Open Container Initiative https://www.opencontainers.org/

Kubernetes Projects in Special Interest Groups: https://github.com/kubernetes-sigs

Kubernetes: https://kubernetes.io/

Docker: https://www.docker.com/

Containerd: https://containerd.io