ISSN 2320 - 2602
Volume 12 No. 4, April 2023
International Journal of Advances in Computer Science and Technology
Available Online at http://www.warse.org/IJACST/static/pdf/file/ijacst011242023.pdf
https://doi.org/10.30534/ijacst/2023/011242023

Cloud Container Service Orchestrated with Kubernetes: a State-of-the-Art Technology Review and Application Proposal

Nikolas Naydenov[1], **Assoc. Prof. Stela Ruseva**[2]
[1]Sofia University, Bulgaria, nikolann@uni-sofia.bg
[2]Sofia University, Bulgaria, stela@fmi.uni-sofia.bg

Received Date : February 15, 2023 Accepted Date : March 18, 2023 Published Date : April 07, 2023

ABSTRACT

This paper discusses specific technology related issues with the latest release of the container orchestrator Kubernetes and combining it with certain container runtimes. We provide a technology-focused state-of-the-art review, addressing identified research gaps in this area. Following the depreciation of Dockershim in the latest K8s release 1.26, we've tested the migration to another container management tool, that enables using the latest functions or the Containerd runtime (Nerdctl). At the same time this migration also allows running containers in a secure and resource effective way and managing them together with container images. The application we study is orchestrating a container cluster-based cloud NIDS service by using the latest Snort 3 system. We've also outlined directions for future research.

Key words: cloud computing, container orchestration, container orchestrator, Kubernetes, container runtime, Containerd, Nerdctl, Docker, network intrusion detection system (NIDS), intrusion prevention system (IPS), Snort.

1. INTRODUCTION - CONTAINER ORCHESTRATION IN THE CLOUD

The role of container orchestration in enabling the cloud service model is well knows in the IT industry. It aids main characteristics of cloud computing like resource pooling and rapid elasticity, while leveraging optimal compute resource usage. It is appropriate to look at what major IT companies have to say about container orchestration, especially those that apply it in critical production use-cases. Currently containerized microservices have become the foundation for cloud-native applications [1]. Container orchestration becomes necessary to deal with the operational complexity of containerized microservice environments. This automated form of management and coordination of container cluster components also gives added security [2].

One among the container orchestrators – Kubernetes (K8s), "has become the industry standard." Some of its advantages are extensive capabilities, portability, being highly declarative [2]. But for all its advantages, K8s has become quite complex and managing this platform is not a trivial task.

With this work, we've tried to apply it to create Network Intrusion Detection System (NIDS), that runs as a containerized service in a cluster. Snort is a scalable parallel intrusion detection (IDS) and intrusion prevention system (IPS). It is considered the standard for a NIDS.

2. RELATED WORK / STATE-OF-THE-ART

Kubernetes is the most used container orchestration platforms among researchers. Many researchers use it as a base platform for their proposed custom container orchestration solutions, as we found out in our systematic mapping study of scientific papers dedicated to container orchestration which were published in 2022 [3]. We also discovered that the Docker container engine is coupled with Kubernetes, which is not advisable for several reasons, explained in section 3. Dockershim support was deprecated since Kubernetes 1.20, in 2020 [4].

We intended to use container orchestration to provide a state-of-the art NIDS cloud service – we found several papers where researchers explain the need of such a service, the benefits of the cloud and the role of container orchestration to enforce the multi-tenant flexible cloud model in an executive way.

We've applied the following search string in Scopus:

```
TITLE-ABS-KEY("container*" AND "orchestrat*" (ids OR ips OR "intrusion detection" OR "intrusion prevention")) AND (LIMIT-TO(PUBSTAGE, "final")) AND (LIMIT-TO(DOCTYPE, "cp") OR LIMIT-TO(DOCTYPE, "ar")) AND (LIMIT-TO(LANGUAGE, "English"))
```

We were looking for scientific journal and conference papers that employed container orchestration and applied it to create an intrusion detection or intrusion prevention systems. The specified search query returned 5 results.

One study proposes a cloud-based dynamic and scalable parallel Network Intrusion Detection System [6]. Another study proposes a cloud-based containerized Snort NIDS that works with big data. However, none of the two papers [6], [7] makes use of a fully-fledged container orchestrator. They only use Docker to containerize Snort workloads, and the Docker Compose container management software, which does not have the full functionality of container orchestrators. Nevertheless, the two publications explain the need of a NIDS in cloud environments and the benefits of containerization to implement such cloud solutions. They lay the foundations for further research.

One of the studies made performance and energy consumption evaluation of container orchestrators on resource constrained IoT gateways (Raspberry Pi devices). The physical infrastructure and architecture of the test environment and evaluated metrics (creation time of a container, CPU utilization, memory usage, energy consumption) were explained clearly. The containerized services tested were two types: 1) IDS services: Snort and Firewall containers and 2) Data analytics services: containers running Pytorch, Tensorflow and Keras machine leaning algorithms. Kubernetes and Docker Swarm are the compared orchestrators. However, we couldn't find a clear account of what version of Kubernetes was used for the tests, nor the container runtime. We didn't find details about the configuration of the container clusters and the services running inside them which would help us reproduce the experiments and check the results.

One of the papers was a secondary study on "Network Function Virtualization and Service Function Chaining Frameworks" that we couldn't apply in the context of our study. We could not get free access to the study on the topic of "a programmable threat intelligence framework for containerized clouds", so it was excluded from our research.

Snort is dubbed by other researchers as the "de facto standard open-source intrusion detection and prevention system" [6].

For these reasons, we decided that the technologies we should explode in our research should be Kubernetes and Snort. We searched several well-known databases for previous research combining these two technologies. Below are our search strings and the results:

1. Google Scholar: Kubernetes+snort (abstracts only) - 0 results
2. Scopus: TITLE-ABS-KEY (kubernetes AND snort) AND (LIMIT-TO (LANGUAGE , "English")) - 0 results
3. IEEE Explore: ("All Metadata":kubernetes) AND ("All Metadata":snort) - 0 results

We didn't find any previous research.

3. MOTIVATION

Container orchestration in the cloud is an evolving dynamic area of research. As already explained, the standard container orchestrator is Kubernetes, but it is combined with the Docker engine too often. We'll try to explain why this is no longer a good practice.

The removal of Dockershim – the interface between K8s and the Docker engine - enables utilizing features incompatible with the Dockershim, e.g., cgroups v2 and user namespaces [5]. User namespaces are an important security enhancement. Sometimes it is necessary to run a container in privileged mode. By default, Docker uses Linux namespaces for isolation. This means that running in privileged mode in a contained also leads to gaining root privileges on the host. Thus, the entire host can be compromised through one container. With user spaces on the other hand, a process running as root in a container can run as a user with lower privileges in the host. Apart from mitigating the security risk, removal of the Dockershim also reduces the operational overhead and resource usage of communicating with the Docker engine. Kubernetes to directly communicate with the low-level runtime, e.g., Containerd. Migrating away from Docker also gives access to the latest features of Containerd that Docker doesn't support. One of them is a time optimization feature - lazy pulling. There is also a new security feature – Ocicrypt. It allows creating encrypted container images and running containers from them. To ease the migration from Docker, new container management tools and high-level container runtimes support similar CLI to that of Docker. Examples are Podman and Nerdctl. Of particular interest to us is containerd because it supports the latest features Containerd by design.

We intend to create a state-of-the art container orchestration solution with Kubernetes. To make the solution optimized for the latest K8s (version 1.26), we decided we must also run Dockerless and use Nerdctl instead, to make use of the latest feature of the Containerd runtime.

The container orchestration use-case of our interest is a containerized cloud NIDS. As shown in section 2, we didn't find any previous research that combined the popular standards for container orchestrator (Kubernetes) and network intrusion detection system (Snort). We found some research that included containerized Snort service, but no orchestration. Our suggested improvements are 1) to use Kubernetes as the base platform for the service and 2) to test the functionality of the container cluster with the latest Snort 3, which was not examined in the previous studies.

So, we decided to explore this new research area by creating an on-premises test environment where we could build our container cluster and NIDS service from scratch. We chose to test the current latest releases of Kubernetes and the other software for a state-of-the-art solution.

We recorded the problems encountered during installation and configuration, because we believe this could help other researchers and IT professionals to adopt these technologies and give a basis for future research.

4. FUTURE IMPROVEMENT

As a future improvement, to make the most of the cloud service model, we could create a secured honeypot environment in our Kubernetes cluster. There we could learn about new (zero-day) attacks, newly compromised networks (botnets), etc. The gathered data could be used to generate new rules for our NIDS cloud service, which we can both provide for third parties or use it to secure out own IT infrastructure. The next step is to analyze gathered attack traffic for the creation of a behavioral IDS. Machine learning algorithms could be applied on the gathered data. This new functionality could be plugged into the existing NIDS service as a Snort preprocessor or used as a separate service.

5. RESEARCH QUESTIONS

In accordance with our goals, we formulated the following research questions:
RQ1. How to manage an on-premises Kubernetes (latest release 1.26) cluster, to provide a private cloud container orchestration service?
RQ2. Can researchers (especially who are used to working with the rich functionality of the Docker engine) seamlessly migrate to using and the Nerdctl to fully utilize the capabilities of Containerd?
RQ3. Can we evaluate the benefits of running Docker less – performance, security?
RQ4. How to apply container orchestration to provide a state-of-the art NIDS cloud service?
We've partially answered **RQ3** in section 3.

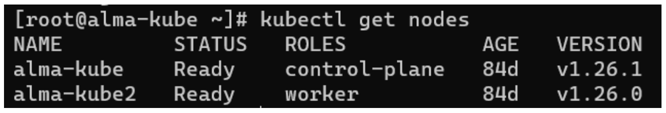
6. TEST ENVIRONMENT AND IMPLEMENTATION

An on-premises cluster of 2 virtual machines was installed, one of the VMs designated as a Kubernetes master node, the other one – as a worker node. Follows an account of the most important software used in our implementation. The main components in our test environment are presented in Table 1. The left column shows the role of each component, on the right side is the corresponding software used. During our work we updated some of the components. That is why some of the items in the table have two versions. Most of the tests were done with both versions.

Table 1: Test Environment

Component	Name & Version
Cluster nodes OS	AlmaLinux 9.1
Container orchestrator	Kubernetes v1.26.0, (updated to) v1.26.1
Container runtime	Containerd v1.6.12, (updated to) v1.6.16
Container management	Nerdctl v1.1.0, (updated to) v1.2.0
Container image building	Buildctl v0.10.6, (updated to) v0.10.6
Container Network Interface plug-in implementation	Calico v3.24.5

The following Bash CLI output shows the nodes in our cluster (Figure 1).



```
[root@alma-kube ~]# kubectl get nodes
NAME         STATUS   ROLES           AGE   VERSION
alma-kube    Ready    control-plane   84d   v1.26.1
alma-kube2   Ready    worker          84d   v1.26.0
```

Figure 1: Cluster Nodes

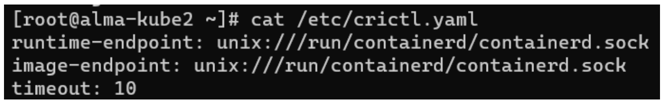
We've built the Snort container image from an Ubuntu:22.04 container image. Unlike the other study that containerized Snort [7], we used Snort version 3, which has some important differences with the previous versions. The previous work used Barnyard2 "to parse binary output log files that contains alerts created by Snort and save the alerts in a centralized MySQL database" [7]. We've assumed that Snort v3 does not require the Barnyard2 spooler because it is multi-threaded. Writes should be handled in parallel.

7. PROBLEMS AND SOLUTIONS

Follows a discussion of the problems we encountered while configuring the cluster and possible solutions.

7.1. Securely Distribute Local Container Images in the Cluster.

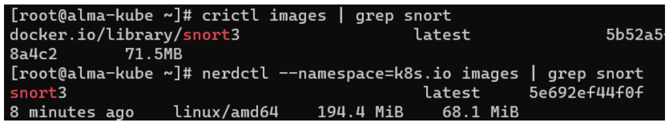
The issue is how to distribute locally built or cached container images to all Kubernetes cluster nodes.
In our setting, we used the Nerdctl container manager in combination with the Buildctl software to build our own customized Snort v3 container images. One of the reasons is because we didn't find an official, trusted contained image for the latest version of Snort in the public repository. However, Kubernetes doesn't seem to provide built-in automated image cache sharing between all cluster nodes – this means that one might not be able to run a pod if it is scheduled on a worker node that doesn't have the image in its local cache.
A hint for troubleshooting such scenarios is to find out which images K8s sees via Container Runtime Interface (CRI) using the tool Crictl, so our advice is to make sure to configure it in your environment. To do that, insert the following lines in the Crictl configuration file on each cluster node (Fig. 2).



```
[root@alma-kube2 ~]# cat /etc/crictl.yaml
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
timeout: 10
```

Figure 2: Crictl Configuration

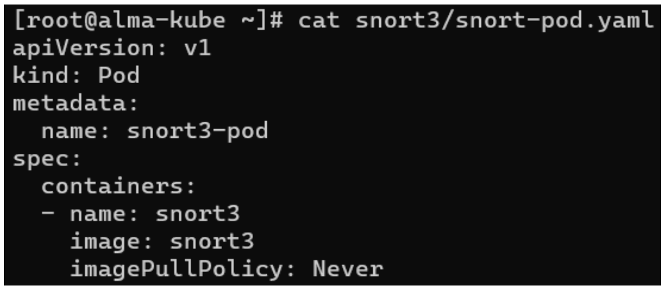
When the configuration above is done, the command `crictl images` can be used to see locally cached container images. Pods can also be listed with the command `crictl pods`.
Back to our Problem 1. For example, let's say we want to start a K8s Pod from the locally built image "snort3". The image is visible in the "k8s.io" namespace – we cloud test this both with Nerdctl and Crictl, as shown in Fig. 3.



```
[root@alma-kube ~]# crictl images | grep snort
docker.io/library/snort3                        latest               5b52a5
8a4c2       71.5MB
[root@alma-kube ~]# nerdctl --namespace=k8s.io images | grep snort
snort3                                           latest      5e692ef44f0f
8 minutes ago    linux/amd64    194.4 MiB    68.1 MiB
```

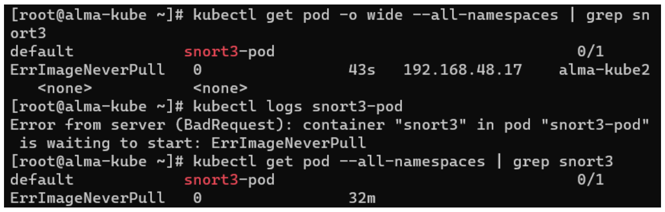
Figure 3: Locally Built Container Image

For the test, we are using a simple Pod configuration (Fig. 4).



```
[root@alma-kube ~]# cat snort3/snort-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: snort3-pod
spec:
  containers:
  - name: snort3
    image: snort3
    imagePullPolicy: Never
```

Figure 4: K8s Pod Definition File
We create the pod using the command `kubectl apply -f <pod-filename>.yaml`. The pod fails to start with `ErrImageNeverPull` error (Fig. 5).



```
[root@alma-kube ~]# kubectl get pod -o wide --all-namespaces | grep sn
ort3
default             snort3-pod                                  0/1
ErrImageNeverPull   0                43s   192.168.48.17    alma-kube2
   <none>           <none>
[root@alma-kube ~]# kubectl logs snort3-pod
Error from server (BadRequest): container "snort3" in pod "snort3-pod"
 is waiting to start: ErrImageNeverPull
[root@alma-kube ~]# kubectl get pod --all-namespaces | grep snort3
default             snort3-pod                                  0/1
ErrImageNeverPull   0                32m
```

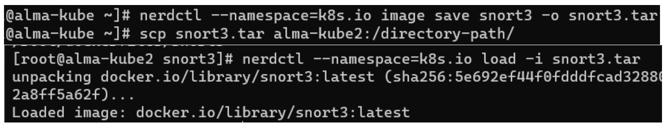
Figure 5: Container Image Pull Error

Obviously K8s can't see the locally built image. The problem is that the pod was scheduled on another worker node (alma-kube2), not the node where the image is present (alma-kube). This can be seen on Fig. 5.

Solutions to 7.1.

The pod is scheduled on a worker node and the image must be in the cache of every worker node where it might be scheduled, so we have several options:

1) *Preloaded Images.*
Preload the images to all the worker nodes of the cluster manually or with an automated script. The distribution could be done by exporting the container image to a tar archive, securely transferring it to the other nodes and loading it in their image caches as exemplified on Fig. 6.



```
@alma-kube ~]# nerdctl --namespace=k8s.io image save snort3 -o snort3.tar
@alma-kube ~]# scp snort3.tar alma-kube2:/directory-path/
 [root@alma-kube2 snort3]# nerdctl --namespace=k8s.io load -i snort3.tar
 unpacking docker.io/library/snort3:latest (sha256:5e692ef44f0fdddfcad32880
 2a8ff5a62f)...
 Loaded image: docker.io/library/snort3:latest
```

Figure 6: Container Image Distribution Between Nodes

2) *Local Image Repository.*
Run a local repository and protect it with authentication so that all cluster nodes can connect to it.

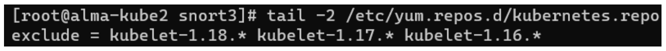
3) *Ocicrypt Encryption*
Upload your locally created images in the public repository and, to secure them, use the capability of Containerd and Nerdctl to create encrypted container images (ocicrypt) [8].

7.2. Bugs: RPM Repository Error

After installing Kubernetes from the RPM repository (which is the recommended way), we encountered an error `cannot install both kubelet-1.26.1-0.x86_64 and kubelet-1.18.4-0.x86_64` when trying to update the RPM packages.

Solution to 7.2.

This turned out to be a bug reproducible when installing Kubernetes on Centos 8 (there is an open issue in Github on this topic). We have the same issue with Alma Linux 9.1. It is solved by adding the following exclude to the configuration of the Kubernetes RPM repository, shown in Fig. 7.

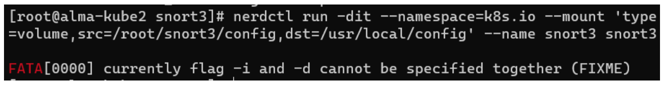


```
[root@alma-kube2 snort3]# tail -2 /etc/yum.repos.d/kubernetes.repo
exclude = kubelet-1.18.* kubelet-1.17.* kubelet-1.16.*
```

Figure 7: Kubernetes RPM Repository

7.3. Migration from Docker to Nerdctl CLI

As already explainer in section 2 and 3, many researchers are familiar with the Docker CLI and might have some difficulties when migrating to nerdctl. For example, running Docker commands with the arguments below would not produce errors (Fig. 8).



```
[root@alma-kube2 snort3]# nerdctl run -dit --namespace=k8s.io --mount 'type
=volume,src=/root/snort3/config,dst=/usr/local/config' --name snort3 snort3

FATA[0000] currently flag -i and -d cannot be specified together (FIXME)
```

Figure 8: Nerdctl Specific Error

With Docker we would often use the `docker run -dit` flag combination. With it we could first run the container in "detach" mode and at the same time be able to get an interactive bash terminal session into the container later. That way we could have the container running without interruption and attach the standard input to the pseudo-TTY in when needed. This is quite convenient, because it allows a system administrator to manage the container in a way that's very similar to a standard virtual machine (or a bare-metal server). This functionality is still not present in nerdctl, despite being requested by some IT professionals who are trying to migrate to nerdctl.

One factor that greatly eases the migration from Docker is that every image form Docker Hub Container Image Library because of their compatibility with the OCI standard. These images could be used with Nerdctl, or any other high-level runtime that is compatible with the CRI.

7.4. Designing a NIDS Service in a Kubernetes Cluster

Our initial goal was to create a Kubernetes cluster running containerized Snort v3 as a cloud service. However, that is not a trivial task. The Kubernetes Service object is designed for exposing containerized applications that listen on a specific (TCP, UDP or SCTP) port. To our knowledge, this K8s feature is not suitable for Snort, which needs to do packet sniffing, i.e., it needs to receive all network traffic, including data from the lower levels of the TCP/IP stack. In this respect, Kubernetes Network Policies can be of help. DaemonSets can be employed to enforce running one or more Snort Pods on every cluster node. As it is known, one of the strong points of Kubernetes is that is highly extensible. By using the K8s CustomResourceDefinition (CRD), a custom resource can be added to the API. The networking plugin Calico has a deep packet inspection CRD that allows monitoring the traffic of a selected workload or application in the K8s cluster [8]. Further investigation of Calico capabilities will be part of our future research.

8. CONCLUSION

In our study we've discussed specific issues related to the latest release Kubernetes and container runtimes. We've given an overview of security and other problems of combining Kubernetes with Docker as a high-level runtime. We've shown research gaps in this area. To address these issues, we've tested the migration to another container management tool, that enables using the latest functions or the Containerd runtime (Nerdctl). During our tests that we've run on a Kubernetes v1.26 cluster we've identified some problems, e.g., how to securely manage and distribute locally built container images and other issues that might be encountered during the migration. Solutions are provided in section 7. We examine applying container orchestration for a cloud NIDS service, by using the latest Snort v3. We've also outlined directions for future research.

REFERENCES

1. "What is container orchestration?" Red Hat. https://www.redhat.com/en/topics/containers/what-is-container-orchestration (accessed Jan 30, 2023).
2. "What is container orchestration?" VMware. https://www.vmware.com/topics/glossary/content/container-orchestration.html (accessed Jan 30, 2023).
3. Naydenov, N., & Ruseva, S. (2022). **Cloud Container Orchestration Architectures, Models and Methods: a Systematic Mapping Study.** Paper presented at the 2023 22nd International Symposium INFOTEH-JAHORINA, INFOTEH 2023 - Proceedings, doi:
4. "Dockershim Deprecation", "Kubernetes 1.20: The Raddest Release" kubernetes.io. https://kubernetes.io/blog/2020/12/08/kubernetes-1-20-release-announcement/#dockershim-deprecation (accessed Jan 30, 2023).
5. "Updated: Dockershim Removal FAQ" kubernetes.io. https://kubernetes.io/blog/2022/02/17/dockershim-faq/ (accessed Mar 4, 2023).
6. Hårek Haugerud, Huy Nhut Tran, Nadjib Aitsaadi, Anis Yazidi, **A dynamic and scalable parallel Network Intrusion Detection System using intelligent rule ordering and Network Function Virtualization, Future Generation Computer Systems**, Volume 124, 2021, p. 254-267, ISSN 0167-739X, https://doi.org/10.1016/j.future.2021.05.037.
7. F. A. Saputra, M. Salman, J. A. N. Hasim, I. U. Nadhori, and K. Ramli, "**The Next-Generation NIDS Platform: Cloud-Based Snort NIDS Using Containers and Big Data**," Big Data and Cognitive Computing, vol. 6, no. 1, p. 19, Feb. 2022, doi: 10.3390/bdcc6010019.
8. "OCIcrypt" github.com. https://github.com/containerd/nerdctl/blob/main/docs/ocicrypt.md (accessed Mar 21, 2023).
9. "Deep packet inspection" docs.tigera.io. https://docs.tigera.io/calico-enterprise/latest/reference/resources/deeppacketinspection (accessed Mar 21, 2023).