Запись: 1

Название: Kubernetes in IT administration and serverless computing: An empirical study and research challenges

Авторы: Mondal, Subrota Kumar[Aff1]
Pan, Rui
Kabir, H M Dipu
Tian, Tan
Dai, Hong-Ning

Источник: *The Journal of Supercomputing: An International Journal of High-Performance Computer Design, Analysis, and Use*. 78(2):2937-2987

Состояние публикации: Published

Информация об издательстве: Springer US

Год издания: 2022

Термины по предметам: Container
Container orchestration
Kubernetes
Serverless computing
FaaS
Security
Attack tree

Описание: Today’s industry has gradually realized the importance of lifting efficiency and saving costs during the life-cycle of an application. In particular, we see that most of the cloud-based applications and services often consist of hundreds of micro-services; however, the traditional monolithic pattern is no longer suitable for today’s development life-cycle. This is due to the difficulties of maintenance, scale, load balance, and many other factors associated with it. Consequently, people switch their focus on containerization—a lightweight virtualization technology. The saving grace is that it can use machine resources more efficiently than the virtual machine (VM). In VM, a guest OS is required to simulate on the host machine, whereas containerization enables applications to share a common OS. Furthermore, containerization facilitates users to create, delete, or deploy containers effortlessly. In order to manipulate and manage the multiple containers, the leading Cloud providers introduced the container orchestration platforms, such as Kubernetes, Docker Swarm, Nomad, and many others. In this paper, a rigorous study on Kubernetes from an administrator’s perspective is conducted. In a later stage, serverless computing paradigm was redefined and integrated with Kubernetes to accelerate the development of software applications. Theoretical knowledge and experimental evaluation show that this novel approach can be accommodated by the developers to design software architecture and development more efficiently and effectively by minimizing the cost charged by public cloud providers (such as AWS, GCP, Azure). However,

serverless functions are attached with several issues, such as security threats, cold start problem, inadequacy of function debugging, and many other. Consequently, the challenge is to find ways to address these issues. However, there are difficulties and hardships in addressing all the issues altogether. Respectively, in this paper, we simply narrow down our analysis toward the security aspects of serverless. In particular, we quantitatively measure the success probability of attack in serverless (using Attack Tree and Attack–Defense Tree) with the possible attack scenarios and the related countermeasures. Thereafter, we show how the quantification can reflect toward the end-to-end security enhancement. In fine, this study concludes with research challenges such as the burdensome and error-prone steps of setting the platform, and investigating the existing security vulnerabilities of serverless computing, and possible future directions.

Тип документа: Original Paper

Язык: English

Членство автора в организациях: [Aff1]Faculty of Information Technology, Macau University of Science and Technology / 0000 0000 8945 4455
[Aff2]Faculty of Science, Vrije Universiteit Amsterdam / 0000 0004 1754 9227
[Aff3]Institute for Intelligent Systems Research and Innovation (IISRI), Deakin University / 0000 0001 0526 7079
[Aff4]Faculty of Information Technology, Macau University of Science and Technology / 0000 0000 8945 4455
[Aff5]The Department of Computing Decision Sciences, Lingnan University / 0000 0004 1770 0716

ISSN: 0920-8542
1573-0484

DOI: 10.1007/s11227-021-03982-3

Author Contribution: Subrota Kumar Mondal and Rui Pan have contributed equally.

Примечания: Publisher's Note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Номер доступа: edssjs.6C80EDF

База данных: Springer Nature Journals

Kubernetes in IT administration and serverless computing: An empirical study and research challenges

Today's industry has gradually realized the importance of lifting efficiency and saving costs during the life-cycle of an application. In particular, we see that most of the cloud-based applications and services often consist of hundreds of micro-services; however, the traditional monolithic pattern is no longer suitable for today's development life-cycle. This is due to the difficulties of maintenance, scale, load balance, and many other factors associated with it. Consequently, people switch their focus on containerization—a lightweight

virtualization technology. The saving grace is that it can use machine resources more efficiently than the virtual machine (VM). In VM, a guest OS is required to simulate on the host machine, whereas containerization enables applications to share a common OS. Furthermore, containerization facilitates users to create, delete, or deploy containers effortlessly. In order to manipulate and manage the multiple containers, the leading Cloud providers introduced the container orchestration platforms, such as Kubernetes, Docker Swarm, Nomad, and many others. In this paper, a rigorous study on Kubernetes from an administrator's perspective is conducted. In a later stage, serverless computing paradigm was redefined and integrated with Kubernetes to accelerate the development of software applications. Theoretical knowledge and experimental evaluation show that this novel approach can be accommodated by the developers to design software architecture and development more efficiently and effectively by minimizing the cost charged by public cloud providers (such as AWS, GCP, Azure). However, serverless functions are attached with several issues, such as security threats, cold start problem, inadequacy of function debugging, and many other. Consequently, the challenge is to find ways to address these issues. However, there are difficulties and hardships in addressing all the issues altogether. Respectively, in this paper, we simply narrow down our analysis toward the security aspects of serverless. In particular, we quantitatively measure the success probability of attack in serverless (using Attack Tree and Attack–Defense Tree) with the possible attack scenarios and the related countermeasures. Thereafter, we show how the quantification can reflect toward the end-to-end security enhancement. In fine, this study concludes with research challenges such as the burdensome and error-prone steps of setting the platform, and investigating the existing security vulnerabilities of serverless computing, and possible future directions.

Keywords: Container; Container orchestration; Kubernetes; Serverless computing; FaaS; Security; Attack tree

Subrota Kumar Mondal and Rui Pan have contributed equally.

Introduction

Virtualization has been evolving for decades from the initial running multiple applications on one host machine to multiple VMs on the host machine. Until today, community start to use containers to replace VM for better resource utilization and security reason. Since the container is operating-system-level virtualization that sheds hypervisor, multiple containers could share a Linux kernel, which enables the container to interact with host OS directly. Those advantages make container own lightweight, high-efficient, and low-cost features. We find that it is difficult and tedious to manage containers manually, so it is need of container orchestration platforms to run and collectively manage multiple container instances. In particular, the industry adopts container orchestration technology and develops various orchestration platforms, such as Docker Swarm [[1]–[4]], Nomad [[2]], Kubernetes [[2], [5]], and many others. Besides, a great number of organizations deploy their applications/services on cloud platforms where cloud providers offer them container orchestration platforms. However, with increasing demand, the community think that they need a smarter framework that could help them to run containers at scale and take care of the underlying infrastructure. To meet the requirement, leading cloud providers like AWS [[7]], GCP [[8]], Azure [[10]] adopt a serverless computing paradigm [[11]]. Even though serverless computing is neither a new concept, community derives FaaS (Function as a Service) from it, which is literally a form of serverless computing [[13]–[15]]. In fact, FaaS combines container orchestration and serverless computing together. In FaaS, a developer simply needs to write some stateless functions and deploy on the cloud, and then it works respectfully. Compared to traditional serverful cloud, where cloud vendors charge for VM's running time, serverless cloud simply needs to pay for function invoking times. Therefore, FaaS paradigm not only saves developer's capital but also transfers the responsibility of handling the underlying infrastructure and network configurations to FaaS providers. Aspect such as nowadays serverless computing has been widely used in

back-end APIs, data processing, IoT scenarios, and many other areas. We see that it has great commercial potential.

Nevertheless, serverless computing encounters a number of security threats and risks that are typically exploited by attackers. Besides, the serverless functions suffer from Cold start problem (incurs a latency as the function is re-deployed in every call), inadequacy of function debugging, and many other. Consequently, the challenge is to find ways of securing serverless platform and functions, optimize the cold start time, devise better function debugging approach. We find that a significant research works has been done in optimizing the cold start time (discussed in the Related Works section). On the other hand, further notable research works are needed to be carried out in the context of security, monitoring, and debugging. However, there are difficulties and hardships in addressing all the issues altogether. Respectively, in this paper, we simply narrow down our analysis toward the security enhancement of serverless platform and functions. In excellent, the contributions of this paper are delineated as follows:

- We analyze and provide a side-by-side comparison of the pros and cons of today's major orchestration platforms, such as Kubernetes.
- We throughout describe the composition and internal architecture of Kubernetes.
- We fully implement a Kubernetes cluster from scratch, concluding the difficulty of configuring a cluster without auxiliary tools.
- We present how to upgrade a Kubernetes cluster and how to update applications deployed in the cluster. Most importantly, we show the steps of taking backup of a deployed Kubernetes cluster so that we can restore the cluster any time if there is any disaster.
- We deeply analyze the principles and features of serverless computing, showing the advantages of serverless computing, concluding its potential business value.
- We analyze possible security issues of the serverless and describe related countermeasures and existing unresolved vulnerabilities.
- We comprehend and analyze the prominent and standard security quantification mechanisms such as, Attack Tree [[16]] and Attack–Defense Tree [[18]–[20]]. Besides, we carefully compare a set tools for modeling the Attack Tree and Defense Tree, and conclude with ADTool[1] [[21]–[23]].
- We build up Attack Tree and Attack–Defense Tree for the possible serverless attack scenarios and the related countermeasures. Thereafter, we quantitatively measure the success probability of attack with and without countermeasures. In fine, we show how the quantification can reflect toward the end-to-end security enhancement.

The rest of this paper is organized as follows: Section 2 introduces the evolution of virtualization to highlight the advantages of Kubernetes. In Sect. 3, we present the structure of an image and a container and their relationship. Section 4 presents a summary of a set of mainstream orchestration tools and their comparison to find the suitable orchestration tool(s). Then, we introduce the Kubernetes core components in Sect. 5. Note that the principle inside the Kubernetes cluster is presented in Sect. 6 alongside the commonly used objects and their internal relations. Besides, in Sect. 7, we briefly introduce the networking in Kubernetes.

We demonstrate a detailed process of deploying a cluster from scratch in Sect. 8. Section 9 introduces a set of general methods and strategies used by administrators to maintain and administer a cluster. Section 10 discusses several standard error types of Kubernetes and their countermeasures. Later, we shortly discuss a general process to containerize an application in Sect. 11. Importantly, we discuss about serverless computing and shows a comparative analysis of its pros and cons with traditional server-based computing in Sect. 12. Besides, we demonstrate the process of deploying an app to the serverless platform and analyze the prominent features of serverless toward the distributed design and development. We present a number of security issues of serverless frameworks and the related countermeasures in Sect. 13. In Sect. 14, we introduce a set of research challenges, we encounter during our study and analysis. Moreover, a summary of related works about the time-critical application and serverless computing is presented in Sect. 15. Conclusion and future works are given in the final section.

Graph: Fig. 1 Evolution of virtualization [[24]]

Why do we need Kubernetes?

- A traditional deployment where developers need to deploy different applications at various physical machines due to multiple development environments, so it could cause low resource utilization because developers cannot set an upper limit of resources the application can use. Furthermore, since the hardware resource is fixed, the application is hard to scale when the traffic increases unless increase infrastructure and reallocate resources. Considering the high cost of equipment, community think it is not an ideal solution.

- Then the virtual machine was invented to solve the above resource and cost issues as we can see from Fig. 1, which clearly shows three stages of virtualization. The advantage of VM is that the developer can deploy multiple applications that need to be compiled and deployed on different operating systems on one physical machine with the help of VM. Since VMs are based on a hypervisor, which is in charge of monitoring and controlling guest OS running on the host machine. So users can install any platforms on a hypervisor that could simulate hardware and drivers needed by the guest OS. Nevertheless, this method also has some resources utilization problems. For example, developers have to run the whole virtual OS on the host so that they can install applications on VM, but the guest OS already takes up plenty of resources to maintain the environment, let alone the scale of applications.

- Finally, the idea of the container was born, it enables different applications can share an OS, and each can be allocated the independent resource, administrators could precisely manage each app's resources. To better organize and automatically monitor multiple containers' status, the concept of orchestration appeared, and with it came the mainstream orchestration platform—Kubernetes [[24]]. Kubernetes enables developers to quickly and efficiently unify management, scale apps, rollback, update apps, and load-balance. Table 1 lists three stages of evolution of virtualization.

Table 1 The evolution of virtualization

	Pros	Cons
Traditional deployment	No.	1. Have to deploy different apps at different physical machines. 2. Hardly set resource limitation for each application. 3. Hardly scale apps on demands. 4. Very expensive.

	Pros	Cons
VM	1. Reduce hardware costs. 2. Can deploy different applications at the same physical machine. 3. Support to set resource limitation for each VM.	1. VM has resource utilization problems. 2. Still hardly and expensive to scale apps on demands.
Container	1. Different apps can share the same Linux kernel. 2. Container can make full use of resources. 3. Container can be collectedly managed and monitored. 4. Container can be scaled on demands. etc.	Learning curve is steep for some newbies.

Prerequisites

Before talking about Kubernetes and containers, we need to know a few basic concepts about the container and image because they are the fundamental of Kubernetes. The inside view of a container and image is shown in Fig. 2.

The inside of image

Image is a file used to create a container. An application is containerized or packed into an image, where the image has a runtime, dependent files required by the application. The image is composed of a File system shot, which contains the application file and some dependent libraries, and a startup command that will be executed to run a service or enter app's shell while the program is running.

Graph: Fig. 2 The structure of an image and container [[25]]

The inside of container

Container's nature is a running process, which is created by its image. The container is composed of a binary application, a Linux Kernel and a set of resources, like hard disk, CPU, memory, network, etc. Note that the resources are allocated by Namespace[2]. Since containers run in a Linux virtual machine environment, so applications and pre-installed commands can make use of the Linux Kernel, which invokes system-calls to get resources [[26]].

Discussion

Kubernetes Application Developers often have to deal with image and container. Specifically, they need to divide an app into several micro-services because those are easy to packet into images. Then those images are uploaded to Docker Hub[3] or private repositories from where Kubernetes can pull images. Finally, those images can be run as containers waiting to be assigned by schedulers.

The comparison of orchestration tools

Nowadays, to fill up various market needs, different orchestration tools are invented. There are several mainstream tools in the market, such as Kubernetes, Mesos [[27]], ECS [[29]], Docker Swarm [[1]–[4]], Nomad [[2]], and OpenShift [[31]–[33]]. Although the above tools have unique features that others could do not have, their limitations dissuade some potential users. However, through multi-dimensional comparison, we found that Kubernetes is the best choice. Table 2 lists five mainstream orchestration platforms and their features and compares the pros and cons.

Table 2 Comparison of orchestration tools

	Features	Pros	Cons
K8S	1. Automatic deployment, scaling, and management.	1. Applications can be deployed via micro-services, deployment, and pods	1. Only support containerized workload

	Features	Pros	Cons
	2. Rolling update	2. Multiple network solutions	2. Container deployment and scaling is slow
	3. Container isolation	3. Native dashboards	3. YAML file needs to be rewritten according to different platforms
	4. Monitoring service states	4. Persistence Volume can be set on external storage	
	5. Service discovery		
	6. Load balancing		
	7. Support maximum 5,000 nodes per cluster		
Mesos	1. Run both containerized and non-containerized applications in the same cluster	1. Suitable for big systems to realize maximum redundancy	1. Learning curve is steep, and hard to configure environment
	2. Mesos-DNS provides all access services, including load balancing	2. Support maximum 10,000 nodes per cluster	2. Persistent volumes couldn't set at external storage
	3. Support multiple frameworks to share the same cluster		3. Container isn't assigned IP by default, and multiple containers cannot share the same network namespace
Docker swarm	1. Native Docker cluster management and scheduling	1. Configuration is convenient	1. Can't automatically scale
	2. Use Docker Engine API	2. Container deployment is very efficient	2. Limited to the capabilities of the Docker API
	3. Use Docker compose file to configure multiple containers applications	3. Suitable for development and test phases	3. Not suitable for at scale production environment
		4. New adding node creates a new overlay network, which is easy to encrypt communication	4. No logging and monitor tools
			5. No native GUI
Nomad	1. Nomad is a single binary	1. Support different types of apps	Cluster management and scheduling only, no other services
	2. Focus on scheduling	2. Simple architecture	
	3. Automatically manage failure	3. Support maximum 10,000 nodes	
Open shift	1. K8S distribution, facing to enterprise	1. Package the network, Ingress, load balancing, storage, without user manual configuration	1. Charge and expensive
	2. Productivity and security	2. GUI is easy to use	2. Only can install on special Linux distribution
	3. Use their own container registry	3. Integrated CI/CD with Jenkins	3. Templates are less flexible

• **Kubernetes**

- Kubernetes(K8S) is the leader of the container orchestration tool because it makes a series of standards and realizes many fundamental and useful features. For instance, it supports automatically deploy, scaling, and rolling update containers, and container isolation. It also has logging and monitoring utilities that can use various metrics to exam object healthy and states. In the network configuration, K8S has Ingress and services to realize all types of communications. Moreover, administrators are allowed to

configure objects through 2 ways - imperative and declarative, the former means using K8S API commands, the latter means using YAML file to configure objects. Lastly, the K8S cluster can support a maximum of 5,000 nodes in the production environment [[28], [34]]. Those features are so useful that other orchestration tools often copy them. Another advantage of K8S is that applications can be deployed through multiple patterns, such as micro-service, Deployment, and pods, and its flexibility is suitable for most enterprises. Furthermore, K8S persistent volume can be placed at external storage, like AWS and GCP, which gives users more choices. It also supports native dashboard. However, it still has some drawbacks [[28], [34]], such as it only supports containerized workload, deploying and scaling is very slow, and YAML configuration needs to be rewritten according to different platforms.

- **Mesos/Marathon**

- Mesos is a fine-grained resource sharing platform for data center, developed by the UC Berkeley team. Its main objective is to enable multiple frameworks to share one cluster. Therefore, Mesos allows users to run containerized and non-containerized workload at the same time. To achieve this feature, it has a unique architecture - master nodes, slave nodes, and schedulers. The slave nodes are used to run different frameworks; each framework has its scheduler to assign resource; the master nodes are used to receive slave nodes information and send requests to their schedulers. Nevertheless, since Mesos is a target to large systems to achieve maximum redundant, which may need to support 10,000 nodes, this makes Mesos very hard to configure. Furthermore, those are familiar with K8S could hardly adapt to Mesos because it does not support some K8S features. For example, Mesos's persistent volume cannot set on external storage [[27], [35]], and containers are not assigned IP by default, which cannot share the same network namespace. Therefore, considering the above limitations, even if Mesos is mighty, it has narrow audiences. To use a simpler Mesos, some users choose Marathon, which is a container orchestration framework running on Mesos.

- **Docker Swarm**

- Docker Swarm is Docker's native cluster management and container scheduler. Since Swarm integrates with Docker Engine API, which is very convenient and easy to learn, users can use the command-line interface(CLI) to control clusters. Swarm architecture is straightforward either, it only has 3 components - Service, Node manager, and Task. The Service is a file used to pre-define the container's state, image, and some meta-data, and then the node manager parses the Service and maintains the states; it is similar to K8S Deployment. The task is an executor, like a K8S job. When the Service defines an application, then the manager will run multiple tasks on demand; each task is a container and has different states [[27], [35], [37]]. However, although Docker Swarm is pretty easy to use, it lacks some features, like automatic scale, native logging and monitoring components, and native GUI. Most fatally, it is limited to the Docker API. Therefore, the Docker Swarm is only suitable for some small scale clusters and lightweight workloads.

- **Nomad**

- Nomad is a single binary, which focuses on schedule. It is invented for multi-datacenter and multi-region configurations like big data analysis. It has several unique features, such as automatically manage failure, hide node manages to users, and support different types of applications ranging from container-based to legacy to batch applications. Moreover, it is very lightweight due to the simple architecture. However, Nomad only supports cluster management and scheduler, without any services [[2], [27], [35], [38]].

• **OpenShift**

- OpenShift is a K8S distribution introduced by RedHat and aimed at enterprises. Its main features are productivity and security. Since OpenShift needs to face to commercial users, to highlight productivity, it puts network, Ingress, load balance, storage, monitor, logging, and RBAC modules all in one package, which means that the users do not need to install each control components from scratch, which is very convenient [[33], [39]]. However, each coin has two sides, the OpenShift forbids users to install images from Docker Hub for safety, and they must pull the image from a private repository. Moreover, OpenShift is limited installation, which means users are only allowed to use designated Linux distributions. The most important is the price, it costs tens of thousands of dollars a year, but K8S is free and maintained by the community.

The component of Kubernetes

After deploying K8S, we get a cluster, which is an abstract concept. The cluster is composed of multiple machines, which can be a physical machine or VM. Those machines are collectively called Nodes. Nodes can be divided into two groups - master nodes and worker nodes. The former is in charge of controlling and monitoring the cluster. Each worker node takes charge of loading and executing some jobs and applications. Table 3 lists the components of master nodes and worker nodes. Today's web applications are mostly composed of micro-services to take advantage of the container, each one takes charge of a single function, but when they communicate and interact with other services, they could reach a full-order Eco-system. For example, an E-commerce platform consists of payment system, store system, shopping cart system, and user management system; each Service can be deployed on four worker nodes and communicate with other services to support the whole platform. These services can be called micro-services. The advantages of micro-services are that it is easy to maintain, update and scale.

Table 3 K8S components

	Master node	**Worker node**
Components	– Kube-API-Server	– Kubelet
	– ETCD	– Kube-Proxy
	– Kubectl	– Runtime
	– Kube-Scheduler	
	– Kube-Controller-Manager	
	– Pod-NetWork	

The master node's components

- **Kube-API-server:** Kube-API-server is the core part of K8S because it is in charge of offering K8S API, which is used to control and manage K8S' components. Since Kube-API-server needs to talk with each component, each component must connect to Kube-API-server.

- **ETCD server** ETCD is a distributed key-value database, which makes retrieval and management very flexible. It takes charge of storing the cluster's configuration data and resources' status. When Kube-API-server receives new commands, it will first update ETCD data and then inform other control components.

- **Kubectl** Kubectl is a CLI tool. It reads commands from the administrator and then transfer the commands to Kube-API-server to manipulate the cluster.

- **Kube-scheduler:** The scheduler takes charge of picking up the most suitable nodes for pending pods. It will assess each node's resource and workload to rank and filter the nodes which not fulfil the pod's

need, and then choose the best one.

- **Kube-controller-manager** Kube-controller-manager supervises cluster's resource. It is a common control component, which has dozens of sub-controllers to monitor the status of a coincident object in the cluster. The controller will ensure objects to reach to the desired status. The most common managers are the replication controller, endpoint controller, etc.

- **Pod Network** Since pods[4] can communicate with each other, the cluster needs to deploy a Pod Network. The most common solution is Weave.

The worker node's components

- **Kubelet** Kubelet is an agent between worker nodes and master nodes; it takes charge of managing and maintaining worker nodes' pods. Kubelet connects to the Kube-API-server, when the Kube-Scheduler chooses a suitable node, it will send pod's configuration to Kubelet that will create a container according to the config file. Kubelet also monitors the status of containers running in pods and transmits containers' log and status to Kube-API-server.

- **Kube-proxy** Kube-proxy is a network proxy that implements some pod-network policies. It manages services in the cluster and route traffic to designated services.

- **Runtime** Runtime is a container engine, which offers containers an environment to run. The most popular runtime is Docker [[6], [40]], but other engines like Rkt [[41]] can replace it.

The principle inside the cluster

Common objects in Kubernetes

- **Pod** The pod is the minimum work unit in K8S. It runs a set of highly suitable containers, such as a Web server and a database. Containers inside the pod share the same network namespace, and volume[5] resource.

- **ReplicaSet** ReplicaSet contains a set of pods; its main duty is keeping the pods in the desired number; if one broke down, it would create a new one. Users can easily scale ReplicaSet size according to demands.

- **Deployment** It is a commonly used object, which manages multiple pods to make sure they are running as expected. It invokes a replica set, but compared to a replica set; it supports rolling update and downgrade pod images.

- **DaemonSet** DaemonSet is a set of pods that needs to be deployed on every node, such as monitoring and logging tools.

- **Job** The Job is used to execute some one-time tasks and then exits.

- **Service** Service is a group of strategies for accessing pods. It has three micro-services:

- **NodePort** Establishing communication between the outside network and inside pods.
- **ClusterIP** Enabling pods can be visited inside the node.
- **LoadBalancer** Using Cloud Server's Load Balancer, which routes outside traffic to back-end services.
- **Ingress** Ingress includes a set of functions: load balancer, SSL, URL routing. It offers an external URL; the developer can config the URL with multiple services.
- **Secret** Secret stores data in cipher text, avoiding directly saving sensitive information in config files. It will be mounted to pod as a volume; then the pod can access the sensitive data like database password from the secret.
- **Namespace** namespace logically divides a physical cluster into multiple virtual clusters so that multi-user could develop, deploy all kinds of objects in the same physical cluster without interfering with other's jobs because namespace isolates resource. There are two namespaces in a cluster by default, default and kube-system. Suppose, developers do not specify the namespace while creating new objects that will be assigned to the default namespace. Cluster control-components objects use the latter.
- **ConfigMap** ConfigMap is used to save some pod's configuration, such as environment variables and port numbers. It can facilitate pods to retrieve.
- **PV** Persistent volume is used to save the data produced by pods in nodes rather than pods where the data will be deleted as long as the pod gets deleted. By contrast, PV can ensure the data will not be deleted when the pods go down or get deleted.
- **PVC** Persistent volume claims [[43]] is used by pods, which hopes to store data permanently. PVC will bind with a PV as long as it matches requirements.

Graph: Fig. 3 The control components of K8S [[45]]

Under the hood of master node

We find that to efficiently handle heavy tasks and simultaneously maintain the whole framework, the cluster needs some tools to monitor and manage nodes and objects status, and that is core control components. Those components are like human organs being responsible for different functions and duties. Figure 3 shows an overview relationship of them.

- **Kube-API-server** It is a hub that connects every control components; it exchanges information and requests among them. The workflow of Kube-API-server is to authenticate incoming commands and identity to check if the request command is valid and then transmit the request to other components to execute.
- **ETCD** ETCD connects to API-server, it will update and store the information when Kube-API-server executes a command. To prevent server crashing, administrators usually deploy more than 2 ETCD servers for backup.
- **Kubectl** It invokes K8S API to manipulate objects.

- **Kube-scheduler** Its job is to monitor Kube-API-server, when it needs to create a pod object, the scheduler chooses the best node for the pod.

- **Kube-controller-manager** It contains different managers to supervise objects' status. E.g., Take an example of Node-controller, which listens to the reply from nodes every 5s to check whether they run correctly. If a node does not reply and lasts for the 40s, the controller will mark the node as unreachable and wait for it to restart for 5 mins. However, if the node does not restart, then the controller will remove it and allocate a new one. Take another example of Replication-controller, which monitors the replica set's pod number; if a pod goes down, the controller will automatically create a new pod.

Scheduling

When Kube-scheduler works, it evaluates the most suitable nodes for the new pods utilizing several techniques:

- **Label** Labels are pod's attribute; it is a key-value pair, which can be used to label pod's function so that admin can easily filter different pods according to functions. For example, a pod running front-end server can be labelled with "tier: front-end", and a pod running back-end server can be labelled with "tier: back-end".

- **Selector** Selector [[43]] is a tool used to find some pods with individual labels. It can specify coincident labels and filters out non-conforming pods. For example, the Deployment uses pod selector to discover coincident pods, which can be created by itself or users. The mechanism enables the Deployment to manage the same labels pods collectively. Moreover, if users hope to bind a service to a pod, the Service uses the selector to match that pod.

- **Taint** Taint is used to define rules for the node, and the rules only allow pods that conform to the rules to enter. For example, the taint of master node states that a typical pod cannot be deployed on the master.

- **Toleration** Toleration is used to define rules for the pod that match the taint condition of the node.

Table 4 summarizes above components and objects in the K8S.

Table 4 The commonly used components inside K8S

Common objects	Control components	Scheduling tech
– Pod	– Kube-API-Server	– Label
– ReplicaSet	– ETCD	– Selector
– Deployment	– Kubectl	– Taint
– DaemonSet	– Kube-Scheduler	
– Job	– Kube-Controller-Manager	
– Service		
– Ingress		
– Secret		
– Namespace		
– ConfigMap		
– PV, PVC		

Networking in Kubernetes

K8S cluster can have a very simple or very complex network, which depends on need. This section will address the analysis of three basic network situation in K8S. Moreover, Fig. 4 presents a bridge network inside the cluster, and it combines the following three condition. Table 5 lists all the network solutions in K8S.

Table 5 The network solutions inside K8S

	Solutions
Inside Pod	Containers share the same network namespace
Among Pods	ClusterIP
Outside Access	NodePort, Load Balancer

The communication between containers inside the pod

Each pod in the same cluster has its own network namespace, that means they have an individual IP address since containers can communicate through localhost, which belongs to their network namespace and prevents conflict.

The communication between pods

A Pod's IP/endpoint is visible to other pods in the same cluster, and they could directly communicate with others without NAT, tunnel, or proxy.

Graph: Fig. 4 A bridge network inside the cluster [[46]]

The communication between pods and services

Although pods can talk to each other through IP, their IP may frequently be changed due to Kube-API-server operations. Therefore, to solve the problem, service offers pods an abstract layer for unified access. The services use pod selectors to bind designated pods and route traffics to the pods.

Outside access

We find that the IP/endpoint of a Pod or Service is invisible to other nodes because they are private addresses. To access K8S services from outside, we have to use NodePort and LoadBalancer, which maps pod inside ports to the host ports, and then clients can visit the pod through the ports.

CNI (Container network interface)

To make sure the network is standard and extensible and flexible, K8S came up with CNI standard [[26]]. CNI defines some basic work rules, and developers can use their technology to create their plugins.

How to deploy a cluster from scratch

We create and deploy a K8S cluster using Ubuntu[6] virtual machine. Note that we can have different types of Kubernetes clusters for different work requirements. For our analysis, we build the cluster following the Kubernetes The Hard Way[7] approach. This approach does not bring a fully automated command to bring up a Kubernetes cluster. On the other hand, it is optimized for learning to ensure that we understand each task required to bootstrap a Kubernetes cluster. This installation helps us know how Kubernetes can generate security certificates for the components and how the security is defined for the components and applications. To analyze the security protocol of Kubernetes cluster for IT administration and application development, we follow this way of installation. Now, VM configuration is listed in Table 6. Note that we can freely add as many nodes as we need.

Table 6 The VM configuration

VM	**Purpose**	**IP**	**Forwarded port**

VM	Purpose	IP	Forwarded port
kubemaster01	Master	192.168.5.11	2711
kubemaster02	Master	192.168.5.12	2712
kubenode01	Worker	192.168.5.21	2721
kubenode02	Worker	192.168.5.22	2722
loadbalancer	LoadBalancer	192.168.5.30	2730

Configure VMs

- We use Vagrant[8] and VirtualBox to create 5 VMs - 2 Master nodes, 2 Worker nodes, and 1 LoadBalancer node. Then, we set 2 master nodes to High Availability (HA) and prevent a node from going down or not responding. The LoadBalancer acts as master nodes' gateway.
- We install container runtime Docker on nodes.
- We allocate IP addresses to each nodes and make sure they are in the same network.

Configure authentication

- We create a pair of secret keys on the master node.
- We install Kubectl on master nodes.
- We choose master nodes to act as certificate authority(CA), and then create a root certificate(ca.crt) and secret key(ca.key) for CA, which are used to authenticate other certificates.
- We use ca.crt and ca.key to generate K8S control components' certificates and secret keys:
- We use ca.crt and ca.key to generate admin.crt and admin.key for admin users.
- We generate Kube-config files for each control components because clients need to authenticate the server.

Graph

Configure master's components bootstrapping installation

ETCD server, Kubernetes control plane, Kube-scheduler, Kube-API-server.

Configure worker's components bootstrapping installation

- We use ca.crt and ca.key to create worker.crt and worker.key so that NodeAuthorizer can authorize Kubelet.
- We creates worker.kubeconfig
- We install Kubelet, Kube-proxy, Kubectl

Configure admin user's permission to access kubectl remotely

Next step, we need to create a kubeconfig file to enable users to access Kubectl remotely. Kube-API-Server can read the file to authenticate authorized users. To create the configuration file, we must set our cluster, credentials with coincident certificates and keys.

Establish CNI-weave network For pods

If we check nodes status, the worker nodes still show 'NotReady' because we have not configured the pod's network. Therefore, we deploy a popular network solution - Weave on worker nodes.

Configure API-server's permission to access kubelet through RBAC

Since Kube-API-Server needs frequently communicate worker nodes through Kubelet to retrieve nodes logs, and status. We create a cluster role for Kube-API-Server to assign it permission to access Kubelet.

Installing coreDNS on the cluster

Although we have set most of the cluster, we also need a DNS service for applications. Therefore, we install coreDNS[9] in the cluster.

Summary

Deploying a cluster from scratch is a troublesome process if users install and configure each control component individually. Administrators need to configure VMs for nodes and cluster authentication for security. Moreover, they need to choose a suitable network solution and install coincident plugins. Finally, they may need to set access permission to different users for safety. Therefore, a more convenient way is to use Kubadm; the tool can help rapidly deploy a cluster. Note that we have a Kubernetes cluster following the Kubeadm Way[10] as well.

Cluster maintenance

Common maintenance methods

Upgrade version

Each part of the cluster can be downloaded and installed manually, so there are many versions of each part. Since Kube-API-server controls each component, its version must always be higher than or equal to the other parts version; otherwise, incompatibilities may arise. Moreover, when updating each part of the cluster, we cannot directly update to the latest version, like from 1.1.0 to 1.2, but we have to update gradually step by step, like from 1.1.0 to 1.1.1, then 1.2.

Application upgrade

The software update strategy of the cluster is rollingUpdate because it will not affect users to access the server during the updating because when a node is upgrading, it transfers objects to other working nodes, and then update the app, then the new version node can be loaded with new pods.

Backup ETCD

We can backup a cluster so that we can restore it from the last saved state if there is any disaster. To back up, we can directly use the get command, as shown:

Graph

It converts the configuration information for all services to a YAML file. However, this only records the object configuration information of the cluster and some status information in the cluster is stored in the ETCD. Therefore, we have to back up the ETCD separately. Since accessing to ETCD requires authentication of

files such as server.crt server.key ca.crt, the path to these files must be specified when backup. This is the common backup ETCD command:

Graph

This command uses the subcommand - snapshot in the etcdctl tool to create a database file with a snapshot of ETCD.

Restore ETCD

Graph

We must specify the data-dir path to store the files for the ETCD, fill in the token file, the address and port of the cluster node, and the address port of the cluster peer. Finally, we also need to enter the path /etc/kubernetes manifests/etcd.yaml, and modify ETCD configuration file. We need add data-dir path, change volume, and volumeMount path.

Summary

K8S is neat and sturdy, but it needs constant maintenance. To keep the consistency of user experience, the administrator needs to watch out application and cluster upgrades. Otherwise, some users could not access the server during system maintenance. Moreover, even if K8S supports high availability(HA)[11], regularly backup cluster data are also essential. The maintenance helps the administrator to recover the whole cluster in seconds.

Troubleshooting

Error types

- **Application Can Not Be Accessed** Since most applications are composed of front-end and back-end, both of them have coincident services to be accessed. Therefore, the first step is to check whether services can be visited through curl http://service, if the terminal display timeout, then we need to look at pod's configuration and Service's configuration to ensure that their labels and selector are same. We also have to check that the Service's endpoint is the correct host address.

- **The Control Plane Cannot Be Accessed** Sometimes failures can arise from control components because they interact and dependent. If one component does not respond, it will affect the whole cluster. Therefore, the first thing we should do is to check the node status to ensure them working accordingly. We can use the command, as shown: To check whether all control components are running. Besides, if we install control plane components through binary, we should check system component services' status, using the command: It prints the details of services. If a service fails, we can simply restart the Service. Lastly, if the service is not in working state after the restart and the above information do not help solve the issue, we have to look at their logs, scrolling down to the bottom of the log, where the latest updated logs are displayed.

- **Worker Node Cannot Be Accessed** If there is a problem with any worker node, we need to check the status of it. If the node status is 'NotReady,' then we can use describe command and see the node conditions. It tells us why the node breaks down. Next step, we have to login to worker nodes and list the processes running in the nodes so that we could know whether any processes are abnormal and kill them. We also should check disks/volumes status on nodes because we usually ignore disk capacities when they are full. Moreover, Kubelet also needs to be checked. If there is nothing wrong with the above

stuff, the problem could be worker nodes' certificates because certificates may be out of date and have not been reassigned. Therefore, the solution is sending a CSR to CA and approve the request.

Graph

Graph

Discussion

If a service does not work, usually it takes time to figure out what is wrong with it. To quickly find and solve the problem, understanding K8S architecture and principles is vital. Administrators can analyze by object relationships, such as service is the container-based object's gateway, Kube-API-Server is the core of control components, Kubelet is an agent between Worker node and the Master node. Moreover, proficiency in using commands is helpful, which can save significant time to check the documentation.

How to containerize an application

We see that traditional apps using monolithic architecture cannot be directly moved to a container platform. Therefore, we need to reconstruct all stateful functions and transform them into stateless functions to adapt to the new environment. After that, we can use any continuous integration and continuous deployment (CI/CD) platform to test and deploy our applications. Containerizing and deploying, an application can be done in the following way:

Steps

- First, we have to divide an application into several micro-services according to their original functions.
- Second, we need to build docker images for each micro-service according to their runtime environment and dependencies.
- Third, we can upload the docker images to a CI/CD platform. In our analysis, we use the TravisCI platform [[48]], which is a popular CI/CD platform. It executes test suits and pushes the image to the Docker hub as long as they pass the test. Specifically, we need to write a script to tell the TravisCI how to install needed dependencies and the environment and finally deploy the application on the platform. After a successful build, the platform can read the test suite from codes and executes them. If the app pass the test, it means the new feature is ready for Deployment. Then CI/CD platform then pushes the code to the production environment and wait for the next update.

Discussion

Containerizing an application is a topic that belongs to software engineering; in this paper, we only briefly discuss it because there are dozens of paradigms to develop an application. Furthermore, K8S requires stateless functions, which means developers must reconstruct a legacy function to adapt to the new environment. However, the principle is the same; they simply need to design different micro-services and build images, and then deploy the containers.

Serverless computing

What is serverless computing?

Serverless computing is a new development computing paradigm, which does not mean the developer does not need servers. Instead, the development team does not need to take care of the underlying infrastructure, like how to allocate CPU resource, RAM resource, or network to the application. Developers

can focus on coding and developing. Furthermore, serverless computing could help application owners to save cost and efficiently scale applications and take full advantage of computing resources.

How serverless computing works?

Serverless computing is based on Function as a Service (FaaS) [[50]] concept. By conventional, community follow IaaS (Infrastructure as a Service) or SaaS (Software as a service) to build applications. The former asks developers to plan and manage the infrastructure of hardware and software; the latter could ease their workload because they do not have to deal with the underlying resource. However, it is an inconvenience for developers to develop and decouple apps on the SaaS platform flexibly. Therefore, FaaS emerged and solved these two problems; for those who can have no background of hardware infrastructure, they also can decouple their services into many independent functions, which is equal to containers and can be deployed on FaaS platform. Only requests can invoke functions to process tasks, and functions will not run all the time. Users just pay for the duration of code execution. Moreover, FaaS platform can manage functions and auto-scale the function on demand according to incoming requests. It also supports to monitor function metrics so that administrators can easier maintain the application.

Advantages of serverless computing

- Developers do not have to know and manage the underlying infrastructure, which is collectively managed by serverless platform vendors. It is helpful for some startup companies to save manpower cost.

- Serverless computing charges per function invoke [[51]]; developers can save more compared to IaaS or SaaS where applications have to stay active in 24/7. However, serverless functions only execute on-demand when events are triggered, and stay idle in spare time.

- Serverless computing applications are much easier to scale and update due to decoupled functions [[52]]. Moreover, developers can get rid of language restrictions and use the most suitable language to create new features. It allows developers to choose the right tools and dependencies in different scenarios without influencing other parts of the system and organization.

- Since administrators do not need to care about the underlying infrastructure, they can put more attention to code.

- Serverless computing is more convenient to monitor and safety because functions are stateless [[53]]; status information is collectively stored in the cloud platform. Moreover, communication between functions is end-to-end, which can be tracked.

- Since public cloud platforms set vendor lock-in that restricts programming languages and container orchestration frameworks. Therefore, some open-source serverless computing frameworks merge in the market, such as Kubeless [[12], [54]], OpenWhisk [[12], [55]], OpenFaaS [[12], [57]], and many other. Those frameworks can be flexibly adapted to all kinds of private environments without artificial restrictions. For instance, OpenFaaS can support most container orchestrations and nearly all languages. Consequently, we focus on OpenFaaS in the literature. Table 7 summarizes the pros and cons of serverless computing and the traditional server.

Table 7 The comparison of serverless computing and traditional server

Pros	Cons

	Pros	Cons
Serverless computing	1. Reduced cost, no longer need to spend money on a lot of physical machine and staff. 2. Elastic scalability is very convenient. 3. Charge for each function call. 4. Comparatively more secure. 5. Reliable.	1. Cold start functions. 2. No mature monitoring applications. 3. Lacking debugging applications. 4. Potential man in the middle attack.
Traditional server	Developers can fully control servers and design their own architecture.	1. Servers can become increasingly expensive, like house, operation and maintenance staff, etc. 2. Server's life cycle is short.

Why do we need openFaaS?

Today, there are several mainstream public cloud platforms, such as AWS, Azure, GCP, support FaaS service to the developer to build and deploy their services and charge for the duration of code execution. However, public cloud platforms usually set resource thresholds to restrict users, they only support a few container orchestration platforms and programming languages, which require the developer to adopt rules, or they could not use it. Therefore, several open-source FaaS frameworks were invented, which allow running serverless computing on private infrastructure, thereby avoiding vendor lock-in. Among those open-source frameworks, OpenFaaS is one of the most extensible, flexible, and popular platforms. Figure 5 shows OpenFaaS architecture, users can communicate with OpenFaaS gateway through CLI. The gateway connects with Prometheus [[59]–[62]] that is a function monitor tool, and it also brings K8S' features to manage container.

Graph: Fig. 5 OpenFaaS architecture [[57]]

What is openFaaS?

OpenFaaS covers most container orchestration platforms, like K8S, Docker Swarm, and Nomad. Developers can extend other platforms. It is composed of 2 parts - API gateway and function watchdog. The former is in charge of providing an external route for functions; it relies on native functionalities provided by the chosen Docker orchestrator. It uses Prometheus running as a service to records function's metrics. It scales function when receiving alerts sent from Prometheus. Function watchdog is packed with function, and it is the entry of functions.

How to deploy a simple app on a serverless platform?

We pick OpenFaaS and K8S as FaaS framework. VMs are configured with Ubuntu. First, we need to install faas-cli, which is used to create, build, deploy and invoke functions. Then we have to install docker-engine and K8S. It will create 2 namespaces - openfaas and openfaas-fn, the former belongs to OpenFaaS basic components, and the latter is used for functions. After creating the cluster, we install OpenFaaS and OpenFaaS gateway, which run as deployments in the cluster. Then we can generate a password and config "OpenFaaS URL' as a global variable, and log in OpenFaaS UI where we could easily deploy new functions and monitor function status. Note that we can also use OpenFaaS command line interface, faas-cli for deploying and monitoring functions. After creating the OpenFaaS environment, we can create new functions. There are two ways—(1) create a function using a code template, or (2) take an existing binary. We adopt the first method. First, we need to pull the template from GitHub.

Graph

Then create a new function.

Graph

This command will create three files and a directory under the current directory.

Graph

The YML file is used to configure CLI for building, pushing and deploying the function. handler.py is a core file, where function logic will be implemented. The file will react to invoke. Finally, we can use the command to build, push, and deploy the function, and then the function image will be pushed to the docker hub. Note that we can use the single command, faas-cli up to build, push, and deploy commands of faas-cli in one shot.

Graph

Serverless toward distributed design and development

We see that the following prominent features of serverless helps enhance distributed design and development. Note that in this section, the empirical analysis is shown with OpenFaaS. As stated earlier, it is open source and the most popular among the community. Now, the features are.

Build templates

The serverless frameworks provide their own set of templates. We can use that build templates to build and generate our functions. For example, the OpenFaaS has a large set of build templates. If we run the following command, then we will get the list of templates.

Graph

The templates are, as shown in Table 8. Note that we can build our own templates and store them to OpenFaaS store for our personal use.

Table 8 OpenFaaS build templates

Openfaas build templates

Name	Source	Description
Csharp	Openfaas	Classic C# template
Dockerfile	Openfaas	ClassicDDockerfile template
Go	Openfaas	Classic golang template
Java8	Openfaas	Java 8 template
Java11	Openfaas	Java 11 template
Java11-vert-x	Openfaas	Java 11 Vert.x template
Node14	Openfaas	HTTP-based node 14 template
Node12	Openfaas	HTTP-based node 12 template
Node	Openfaas	Classic nodeJS 8 template
Php7	Openfaas	Classic PHP 7 template
Python	Openfaas	Classic python 2.7 template
Python3	Openfaas	Classic python 3.6 template
Python3-dlrs	intel	Deep learning reference stack v0.4 for ML workloads
Ruby	Openfaas	Classic ruby 2.5 template
Ruby-http	Openfaas	Ruby 2.4 HTTP template
Python27-flask	Openfaas	Python 2.7 flask template
Python3-flask	Openfaas	Python 3.7 flask template
Python3-flask-debian	Openfaas	Python 3.7 flask template based on Debian
Python3-http	Openfaas	Python 3.7 with flask and HTTP
Python3-http-debian	Openfaas	Python 3.7 with flask and HTTP based on Debian
Golang-http	Openfaas	Golang HTTP template

Openfaas build templates

Name	Source	Description
Golang-middleware	Openfaas	Golang middleware template
Python3-debian	Openfaas	Python 3 debian template
Powershell-template	Openfaas-incubator	Powershell core Ubuntu:16.04 template
Powershell-http-template	Openfaas-incubator	Powershell core HTTP Ubuntu:16.04 template
Rust	Booyaa	Rust template
Crystal	Tpei	Crystal template
Csharp-httprequest	Distantcam	C# HTTP template
Csharp-kestrel	Burtonr	C# Kestrel HTTP template
Vertx-native	Pmlopes	Eclipse vert.x native image template
Swift	Affix	Swift 4.2 template
Lua53	Affix	Lua 5.3 template
vala	Affix	Vala template
Vala-http	Affix	Non-forking vala template
Quarkus-native	Pmlopes	Quarkus.io native image template
Perl-alpine	Tmiklas	Perl language template based on Alpine image
Crystal-http	Koffeinfrei	Crystal HTTP template
Rust-http	Openfaas-incubator	Rust HTTP template
Bash-streaming	Openfaas-incubator	Bash streaming template
Cobol	Devries	COBOL template

Scaling from zero

Serverless supports scaling to and from zero. In this way, we can save cost, manage optimally, increase agility, and many more. For example, we can achieve this by passing the command, kubectl scale deploy –replicas=0 helloworld -n openfaas-fn. Note that when a create a deployment, by default the replica count is set to 1. However, if we want it scaling from 'Zero,' we can do that during the deployment of the function by setting the label to com.openfaas.scale.zero=true. Now, we have the helloworld function as deployed earlier. In this case, we scale down its replica count to 'Zero.' We can check that in OpenFaaS UI, in Prometheus Graph, in Grafana[12] dashboard [[63]], or we can pass the command kubectl get deploy helloworld -n openfaas-fn in the terminal and see that the helloword has 0 replica. Thereafter, we invoke the function, by faas-cli invoke helloworld and see that its replica count is changed to 1. Note that we can invoke the function from OpenFaaS UI and see the replica changes accordingly. The steps are shown in the Listing below.

Graph

Asynchronous call

We can call the functions asynchronously. To achieve this, we simply need to pass the – async option to the function invoke callback, as shown.

Graph

Autoscaling and loadbalancing

Serverless functions are autoscaled, i.e., the functions are scaled up or down based upon user requests. Autoscaling in OpenFaaS implemented with the help of Prometheus and AlertManager. A threshold value called request per seconds is defined in Prometheus, and AlertManager reads the value from it. If it is exceeded the limit, the AlertManager fires. Note that we need to set the minimum and maximum replica count. We can do that during the deployment of the function by adding labels com.openfaas.scale.min and com.openfaas.scale.max, and assigning the min and max replica counts, respectively. Note that we can use Kubernetes native Horizontal Pod Autoscaler (HPA) instead of AlertManager. Now, we show the status of

autoscaling the function call, helloworld. First, we deploy the helloworld by setting the min and max replicas to 2 and 15, respectively. Thereafter, we run a script to invoke the helloworld function over and over until we see the replica count goes from 2 to 15. The steps are shown in the Listing below

Graph

Now, we monitor the autoscaling in Prometheus (left pane in Fig. 6). Note that besides Prometheus, we use Grafana dashboard (right pane in Fig. 6) to monitor the autoscaling. We find that it depicts the status in a better way and helps us monitor other important parameters. As we can see that initially the replica count is 0. After a while, we see that replica count is 2, since the min value is set to 2. Besides, the max value is set to 15. Until, we invoke the function, the replica counts does not change—remains to 2. Afterward, we call the function over and over using the script as stated earlier. Then the replica count gradually goes from 2 to 15 as we see in the Fig. 6 (bottom right pane). Note that replica count becomes 2 when we finish invoking the function. In addition, from the Grafana dashboard (top right pane), we can monitor the requests rate per second. As we see that the request rate lies in around 10. This information helps us monitor load testing. We discuss about this in the next section.

Graph: Fig. 6 Autoscale monitoring in serverless

Further, we see that the replicas are distributed across the cluster in a well-balanced manner. Besides, the user requests are served in a balanced way. The Kubernetes components LoadBalancer and Ingress come into light to handle the loadbalancing task. Now, we see the load distribution in the cluster,

Graph

We have 15 replicas deployed and we see that replicas are distributed evenly among the 2 worker nodes (as stated earlier, we have only 2 worker nodes in our cluster).

Multi-user support (load testing)

Serverless offers the support of multiple requests and multi-users at the same time. To validate this, we perform load testing using a tool called hey[13] [[64]]. For example, we want to invoke the function helloworld for 4 users with 10 requests per second and over the duration of 300 seconds. Then we can pass the command as shown and monitor the status. Note that min and max replica counts are 2 and 15 as in Autoscaling scenario.

Graph

Figure 7 shows the output of load testing. The left pane is the output summary from the terminal and the right pane (Grafana dashboard) shows the Function rate and replica scaling. We see that multi-requests with multi-users are served duly.

Graph: Fig. 7 Load testing in serverless (4 users and 10 requests/second)

We see that not only the features, we mentioned herein, but also many other striking features are supported by serverless computing such as, event-triggering, easy integration and deployment, rich metrics, and many more.

Summary

As we stated earlier, serverless computing makes full use of container orchestration frameworks and reduces the developer's hurdle. Since developers/clients do not need to take care of underlying

infrastructure anymore, they can decrease team overhead and focus on developing new applications. Moreover, serverless computing framework is simple and powerful as there are many open-source and useful tools to ensure better services, such as logging and monitoring tools. Hence, serverless computing is an excellent opportunity for most developer teams. Besides, the striking features of serverless helps develop and enhance many fields of computing, such as distributed computing, edge computing, machine learning, and many other as discussed in the Related Works section.

Security quantification of serverless using attack tree

We know that if there is no enough security measures, there could have a huge revenue loss. Besides, quality of service, service level agreement, privacy, integrity, safety, confidentiality, reliability, availability, downtime, vulnerability, threats, attacks, and many other will be degraded [[65]–[69]]. Therefore, security (risk) quantification is important. To the end, security quantification is the process of identifying security risks, threats, or vulnerabilities and evaluating them and then validating, measuring and analyzing the available security data using mathematical modeling techniques to accurately represent the security environment in a manner that can be used to make informed security infrastructure investment and risk transfer decisions [[65]–[70]].

There are numerous mechanisms for quantifying security attributes, measures, or metrics. We find that one of the potential mechanisms for quantifying security is Attack Tree. Note that it can help us model all the possible attacks, threats, or vulnerabilities. Finally, we can predict risks and other security related metrics corresponding to vulnerabilities/attacks and their impact [[17]–[22], [67]–[71]].

First, we analyze the possible security risks of serverless. Afterward, we present the security quantification using Attack Tree.

Security risks of serverless

Serverless cloud platform vendors protect customers' information through isolating multi-tenant functions. Since serverless function is stateless, they cannot store data permanent and data will be deleted when the function exits. This mechanism seems very safe, but the potential issue is that data have to go across the network frequently, and there is a higher risk of leaking data. To solve this problem, some teams adopt IFC(information flow control) [[72]] to keep track of data flow and keep information safe. Since IFC can offer a system-level or function level monitoring, when data try to interact with other data, it gets tainted by labels that the other has. For example, if a process visits a "top secret" data, it will be tainted with a label, then OS will keep track of this process and prevent the process of accessing the Internet.

Nevertheless, the primary IFC method is not efficient enough because a typical application could have hundreds of functions; monitoring data flow could consume substantial system resources and influence the performance of the whole platform [[11]]. Moreover, basic IFC is not safe enough. Attackers can take advantage of observing the termination of processes, even if termination only leak 1 bit of information, the attacker could create multiple concurrent requests to crack the secret.

Graph: Fig. 8 Trapeze architecture [[74]]

Therefore, a research team invent an innovative solution–Dynamic IFC, and they implement an architecture called Trapeze [[72]]. Figure 8 shows how it works. This approach creates a sandbox and security shim in serverless systems, every incoming and outgoing function requests have to go through the shim. It also provides static security labels for each serverless function invocation and dynamic faceted labeling of data in the persistent store.

Another security issue is broken authentication [[11], [75]–[77]]. Since some platform APIs are in the public domain, most of them do not have authentication, which means anyone can send a request to functions. It leaves weakness to attackers to exploit those functions. To solve that, administrators must add authentication to every endpoint.

Another well-known security issue is SQL injection attack [[78]]. Attacks on functions can be caused by SQL injection, system command execution, and many others— developers do not filter or encode external data inside the function. To countermeasure an injection attack, we can separate data from command and queries. Besides, we also can perform input validation at server side.

Nevertheless, more security risks are waiting to be solved properly [[11], [76]–[79]]. For instance, some legacy or unused functions may be ignored by development teams. Those obsoleted functions could still take up roles, cloud resource and dependencies, which could be hacked by attackers. Some hackers take advantage of the fact that the platform will automatically scale the functions and use Denial-of-Service to attack serverless applications. As a result, developers have to pay huge amounts for function invocation. Table 9 summarizes serverless computing existing security problems.

With the advent of serverless computing, more and more security problems will emerge, serverless has disrupted the previous division of security responsibilities, shifting many responsibilities from cloud users to cloud providers [[80]] without fundamentally changing them. However, serverless must also address the inherent risks of multi-tenant resource sharing between applications.

Table 9 Serverless computing security issues

Security issues	Solved	Solution
Data leakage	Yes	Dynamic IFC
Broken authentication	Yes	To add authentication to every endpoint
SQL injection attack	Yes	Data separation from queries
Overprivileged function	No	
Denial-of-service	Yes	Isolating resources
Legacy function	No	

Attack tree

Attack tree (ATree) was proposed by Schneier [[16]] as a widely used way to represent and evaluate potential security threats on systems. It uses graphical, mathematical, structured decision tree notations to model attacks and systematically categorizes the ways in which a system can be attacked. An attack tree is constructed from the perspective of the adversary. Creating a nice attack tree requires that we play the role of an attacker—it is a multi-level tree consisting of a root and leaves as shown in Fig. 9. The top-level root node of an attack tree is the overall goal of the attack. On the other hand, leaf nodes are sub-goals which the attacker likely to implement for performing the attack.

Graph: Fig. 9 Attack tree structure [[81]]

Attack–Defense Tree

The fundamental formalism of Attack Tree does not think over defense mechanisms. As such, Attack–Defense Tree (ADTree) extends attack tree with defensive measures, also called countermeasures, yielding a graphical mathematical model of multi-stage attacks along with safeguards [[18]]. It has been developed to investigate the effect of defense mechanisms using measures such as attack/defend cost, attack probability, and many other [[19]]. A defense node can be placed in the Attack–Defense Tree. Therefore, each node belongs to either the attacker or the defender in our models. Countermeasures prevent an

adversary from reaching the goal, thus the ADTree represents an interplay between an attacker, whose goal is to attack the system, and a defender who tries to protect it [[18]].

Attack–Defense Tree, constructing by a diagram with a single root node, is similar with most mathematical tree models which create a visual record of a system. Beneath the root, it expands through forks to get more branches. This structure seems like the decision trees applied to pivotal business decisions or the fault trees helped to show the logical relationships between events and causes that lead to failure. ADTree-based analysis helps to better determine the vulnerability of a system against any specific type of attack. The root node represents the global goal of an attacker. The intermediate nodes are refinements of the global goal and define different stages of the attack leading to the root. The leaf nodes model attacks that can no longer be refined.

Graph: Fig. 10 Attack–Defense Tree structure

If we choose the goal carefully, it is likely to comprehensively analyze a system simply with a single ADTree. However, a particular hostile attacker may have some different goals, and in this case, sometimes we need multiple Attack–Defense trees to help us perform the whole attack analysis plainly. Thanks to the succinct structure of ADTree, a very complex attack scenario can be expressed clearly and simply. Besides, it is also possible that different hostile attackers have various unique goals. Benefit from its scalability and flexibility, ADTree is quite qualified to perceive attacks and countermeasures easily in these situations. When we plan to construct an Attack–Defense Tree, there are some steps we need to follow.

- Identify goals. Each goal can be a separate attack tree.
- Identify attack against goals. The attack against goals can be repeated as necessary.
- Existing attack trees can be plugged-in if needed.
- Considering possible countermeasures.
- Adding the defense node into the attack tree.

These constructs illustrate the attacks and defensive measures in a tree structure as shown in Fig. 10.

Boolean gates are used to explain whether a refinement can be conjunctive (AND-gate) or disjunctive (OR-gate).

- **AND** nodes represent different steps in achieving a goal.
- **OR** nodes represent different way to achieving the same goal.

The quantitative analysis helps us estimate the success probability of attack, attack/defend cost, and many other with or without countermeasures.

Applications of Attack/Attack–Defense Tree

Attack–Defense Tree helps us easily understand how an attacker can intrude into the target system by using vulnerabilities. Moreover, it is an effective way to display where to best spend a security budget.

As we know, serverless computing primarily focuses on code rather than infrastructure. Besides, it allows decomposing the workload into smaller artifacts for better scaling, easy managing, optimum loadbalancing, and many more. However, as stated earlier, it has issues to address in terms of security. To this end, we take help of Attack–Defense Tree to analyze the security threats toward the enhancement of it.

Attack tree tool

We thoroughly analyze a set of existing commercial and open-source attack tree modeling tools. Commercial tools, such as Attack-Tree+ from Isograph, SecurlTree from Amenaza Technologies, and RiskTree from 2T Security. Besides, Open-source tools are ADTool from University of Luxembourg, Ent, and SeaMonster.

In our analysis, we do not use the commercial tools since we need to pay for it. On the other hand, we decide to pick the best open-source tool and we select the ADTool[14] [[21]–[23]]. Note that it has no system limitation, more up-to-date, and provides better quantitative analysis compared with other open-source tools.

Attack tree modeling and quantification

Graph: Fig. 11 Attack tree of serverless with possible attack scenarios

The possible attack cases/scenarios are demonstrated earlier. In this section, we use the ADTool to establish a simple attack tree, shown in Fig. 11 demonstrating the cases that the serverless may be attacked/threatened.

Graph: Fig. 12 Probability of attack with possible attack scenarios

Now we try to quantify the security risks from the standpoint of probability which indicates the likelihood of a threat occurring. We choose the Probability of success attribute domain. Literally, it is apparently evident that the probability of success is congruent with the probability of a risk occurring, because in this context, as for a risk, the success implies that the presence of the risk instead of being avoided.

We assign the probabilities to the attack scenarios as shown in Fig. 12 and compute the success probability of attack. Note that the probability values are simply the assumption based on the severity of attack. Especially, we need to carry out a statistical analysis to get probabilities of attacks. However, in this paper, we limit our objectives and simply assume the probability values. What we do here is that we assign the probabilities to the leaf nodes. Consequently, ADTree derives the attack probability of success which is 0.77.

Attack–Defense Tree modeling and quantification

After modeling the attack tree for attacking the serverless and doing the quantitative analysis, we construct the Attack–Defense Tree with the related countermeasures which is shown in Fig. 13.

Graph: Fig. 13 Attack–Defense Tree of serverless with related countermeasures

Now, we do a quantitative analysis of the Attack–Defense Tree with the Probability of success attribute domain. Same as before, we assume the value of each defensive measure and show the analysis as shown in Fig. 14.

Graph: Fig. 14 Probability of attack with related countermeasures

When we assign the probability to a countermeasure then the probability of the corresponding attack is reduced which has an overall reflection to the cumulative (root) attack. As we see, there is a great reduction of risk after adding the defensive measures. The automatically generated value in the root node has been changed to 0.563 which is 0.77 in the attack tree (refer to Fig. 12). Thus, it reflects that though we cannot totally eliminate the risks, but we can mitigate the impact of them through some effective countermeasures.

Toward a secure serverless computing

We make a discussion about the above security quantification results and give some recommendations on getting a securer serverless platform. The success probability of attack in Attack tree and Attack–Defense Tree is shown in Table 10. The attack scenarios we have established are consistent with the serverless platform and functions. From the table, we find a satisfactory reduction of the risk probability when we add the defensive measures. The results also indicate that the quantification method we used does make sense.

Table 10 Serverless security quantification

	Attack tree	Attack–Defense Tree
Success probability of attack	0.77	0.563

According to this result, we can confidently state that the system is insecure. There is an urgent demand for us to enhance the security aspect of serverless—we need find more effective ways to defend against risks or develop a safer system. According to our research, we have reasons to strongly suggest that the community should pay more attention to the security issues of serverless.

The fact is that our analysis is limited with probability assumption and empirical evaluation and validation. In the future, we will work on devising a quantitative measure to logically compute the probability of an attack and countermeasure. Besides, we will perform empirical evaluation and validation of every attack and the related countermeasure.

Research challenges

Kubernetes

- For beginners, deploying a K8S cluster is a painstaking process because they need to configure everything by themselves, such as checking control component's version, and choosing an appropriate network solution, and many other. It is time-consuming and easy to make mistakes.

- For larger organizations, since they have to consider more deeply about cluster security, reliability. In other areas, they need to consider networking stuff because they face more clients and traffics, which require high-standard bandwidth and sites. Those may consume more cost.

- For individual developers, they need to reconstruct their monolithic architecture to micro-services architecture, which requires stateless programming. Hence, it will spend developers much time on adapting the new program architecture.

Table 11 summarizes the above challenges.

Table 11 K8S research challenges

User	Problems
Individual user	– Difficult to deploy a K8S cluster. – Learning curve is steep, time consuming.

User	Problems
Enterprises	– Need to consider more about cluster to achieve security and reliability.
Developers	– Need to adapt micro-service architecture

Serverless computing

- **DDoS billing issue** Some hackers use DDoS attack to maliciously invoke functions, which would bring a great financial loss to the cloud client.

- **Broken authentication** When developers deploy and release services, they could expose some function's API to the public domain. However, some hackers may use unauthenticated events to call the function and bypass the system logic to steal data.

- **Vulnerable third-party dependencies** Since today's developers are used to using some third-party libraries and dependencies for convenience. Nevertheless, it may leave some unexpected risk if those dependencies do not have a good security strategy, then some hackers still can make use of their vulnerability to attack the system.

Related works

We find that researchers and developers have devoted to serverless computing from different perspectives. In particular, some of them work on improving serverless computing performance so that users can get better user experience. Therefore, most of the works focus on time-critical applications [[82]] and cold start problem. There are a large group of works in these areas. We try to summarize a few of them.

In the area of time-critical applications, Stefanic et al. [[83]] derive a unique approach called SWITCH workbench. It aims at offering a complete development environment for time-critical microservice-based cloud-native applications. Note that an application of this kind needs to deal with huge sensitive data and its components often be distributed to different places. As such, the traditional service modelling tools using Docker and K8s as virtualization and orchestration technologies, cannot fulfil the demand. On the other hand, SWITCH provides a package, which contains three components, such as SIDE, DRIP, ASAP to meet the demand, as discussed. SIDE takes advantage of Docker compose to create application components; DRIP can allocate resource for each component, ASAP is used to monitor applications. The whole system can perfectly adopt time-critical applications.

A similar work comes in the area of handling problem that serverless functions accessing a database are much slower than traditional VM. Bishakh et al. [[84]] demonstrate that the root cause of it is that the function and database instances are not in the same containers. As a result, if a function wants to read from (or write to) the database frequently, there will be a high latency. Therefore, to kill the latency, they try to create a cache space in the runtime of the container. Every time, the function wants to fetch data, and it checks the cache first. If it is a cache hit, then it can retrieve directly from there. Otherwise, it needs to access the database. This design dramatically optimizes the performance of serverless frameworks.

Moreover, several researchers explore the way to adapt current serverless frameworks into low-latency and low hardware application scenarios, such as IoT. Adam and Ramachandran [[85]] explore an alternative solution called WebAssembly which can fulfil the above requirements. Therefore, it is a perfect substitute. There is a similarly plenty of works in the area of optimizing serverless performance. Suppose, an application needs a stable performance or handle burst requests. In this case, Nguyen et al. [[86]] build an extension of the serverless interface called real-time, where administrators can pass a value called Service-Level Objective (SLO) to the system, and then the system can maintain the function invocation rate at SLO

to ensure optimum performance. This idea can also meet applications that need low-latency and high transmission quality. The aforementioned research, study, and analysis all come up with innovative methods to handle time-critical problems.

Similarly, in the area of cold start problem, researchers have proposed numerous solutions. For example, Dong Du et al. [[87]] derive a novel serverless design, Catalyzer which can sharply optimize the startup time of functions. In most scenarios, when users initiate requests, functions often need to go through cold start. In particular, the system pulls up images from the mentioned image repository, e.g., Docker Hub and creates sandbox containers for functions. The issue is that this end-to-end processing is quite sluggish, i.e., takes a while to process. In order to solve the issue, Catalyzer reuses images and the state of the sandbox, which eliminates application initialization time and sandbox creation time. Finally, it helps optimize the user experience.

Further, we see that Bermbach et al. [[88]] build a lightweight choreography middleware, which can reduce the cold start. The principle of middleware is that it uses process knowledge to predict the approximation of incoming function and prepare containers. Therefore, when users want to invoke multiple functions, the provisioned containers would be created before functions. In fine, it effectively helps reduce the cold start.

Another research team [[89]] address the cold start problem by analyzing redundant content when deploying containers. Since, scaling and versioning will create duplicate blocks, so to reuse code in optimizing latency, they build and evaluate a novel system composed of 4 parts - peer-to-peer network (P2P), a virtual file system with content-addressable storage (CAS), partial delivery execution and TCP splitting. Users can upload source codes to the file system. If the system needs to execute a container, it will check the file system first, and if there are no codes, it will fetch the codes from other worker nodes through the P2P network.

In addition, we find that Bardsley et al. [[90]] present several strategies explicitly to optimize serverless performance, especially the cold start and latency issue. They show that latency often comes from external systems and events. They believe a request should not wait for a function to start. Instead, it should take the next free resource and consequently it can help alleviate cold start. That is to say, the function call stack should be as short as possible. Otherwise, it would cause severe latency issues.

In addition, we see that several researchers have explored the methods to break through the vendor lock-in problem. As stated earlier, serverless providers often hide some critical system managements from users for convenience, but sometime those strategies could not make the best choice for functions. More specifically, the algorithm used to spread functions to different nodes is not robust. For example, the FaaS platform could place several different types of functions to the same node without considering their performance implications. Therefore, to fix the issue, Mahmoudi et al. [[91]] adopt machine learning based technologies to train multiple models and find the best spreading algorithm called the smart spread. It not only dramatically improves serverless computing performance but also has high compatibility and can be installed to the platform's scheduler. Moreover, we see that Aske et al. [[92]] build the MPSC framework that can automatically choose the "right" provider and schedule applications across the providers.

At the same time, there is a large body of work in the area of monitoring and tracking functions. We see that it is hard to effectively supervise complex applications using the traditional monitoring and tracking modules/tools provided by the platform. On the other hand, we find that Correia et al. [[93]] propose a queuing theory-based modelling approach to analyze the performance of the micro-service platform. This approach could facilitate the platform to predict the distribution of response times. Besides, it can decide to determine how many instances should be maintained under a specific circumstance.

In short, we can say that since serverless computing is compelling, significant research works have been done and are continuously going on to overall improve its quality of services (Qos), service level agreements, security, and many other.

Now, we review what are the perspectives of Kubernetes and serverless computing in distributed computing, hybrid cloud, data science and other areas.

Tsai et al. [[94]] adopt Kubernetes to their distributed system, which analyzes a large volume of data collected from multiple IoT devices. In this aspect, K8s monitors the state of system, and allocates reasonable resources to IoT applications. The results show that K8s ensures higher performance and lower overhead consumed by resource limited devices. The authors in [[95]] introduce Katib, an AutoML platform based on K8s, which not only can support multiple AutoML algorithms but also reduce the prohibitive computational costs. To adopt K8s in AutoML, the research team divides and abstracts machine learning algorithms into a single function and containerizes it into micro-service. Consequently, with this flexible and extensible design, Katib becomes a user-friendly and powerful platform for enterprise users.

Another research proposes a high-level architecture [[96]] designed for a hybrid cloud environment, which can combine multiple cloud environments to scale computing resources and reduce costs. The architecture adopts K8s as an infrastructure layer, where each cloud is composed of multiple worker nodes. Moreover, there is a K8s federation layer on the top of control nodes, which can collectively manage clusters ranging from the public cloud to the private cloud. Besides, the authors in [[97]] proposes a heterogeneous mobile cloud computing model for hybrid clouds, which is based on distributed computing. The model can utilize mobile devices to do edge computing operations to relieve the overload of cloud center. We see that this model adopts K8s to scale up and down the services.

A research team analyzes several FaaS platforms [[98]] to investigate the ability of orchestration of serverless functions to handle parallel processing. They find none of the existing platforms have promising performance result. However, they provide valuable directions toward the performance enhancement of FaaS services for parallel processing. On the other hand, the authors in [[99]] proposes an innovative framework used to execute parallel computations. It integrates container platforms into distributed computing system. Consequently, it helps users easily implement parallel applications and offers an efficient image management utility, which enables developers to easy scale up and down applications.

Overall, a significant number of works aim to optimize serverless performance in various aspects - supporting the time-critical application, optimizing cold start time, mitigating latency, enhancing monitoring module, and many other aspects. Our mission is to collect those state-of-art methods, empirically analyze the usage prospect of serverless computing, and make it more firm and robust.

Conclusion and future works

Conclusion

Over the years, containerization has gradually shown its potential edges in the market. Developers utilize container technology and serverless computing to solve numerous real-world challenges, such as VM's performance loss problem, auto scaling, load balancing, optimizing cost, and many others.

To this end, in this paper, we mainly present the principles of K8S and its associated serverless computing framework, OpenFaaS. Our primary motivation is to build a complete K8S platform and evaluating its manageability for administrators. To achieve this, we build a simple K8S cluster and analyze how to create, deploy, and manage containers. Next, we discuss the K8S cluster structure from an administrator's

perspective, focusing on the relationship between Kube-API-Server and other control components and their functions. Besides, we figure out K8S objects and their usage and effectiveness. To establish communication among pods, we configure pod's proxy, node's agent, and third-party CNI network solutions, such as Weave and coreDNS. After that, we study the K8S security mechanism since all the data have to go through Kube-API-Server. The cluster needs to authenticate the role and authorize permission for each access.

Next, we deeply discuss serverless computing that transfers the responsibility of handling the underlying architectures and networks to the cloud providers. To build a simple serverless computing platform, we choose one of the most popular open-source frameworks called OpenFaaS. We see that it helps easy deploy and test functions. Besides, our study shows that the prominent features of serverless helps enhance the different fields of computing, such as distributed computing, edge computing, machine learning, and many other.

Nevertheless, serverless has issues in terms of security, cold start, application monitoring, function debugging, and many others. Consequently, the challenge is to find ways to address the issues. We see that significant research works have been done in optimizing the cold start time and optimizing the performance of time-critical applications. To this end, we summarize a number of related works in the area of cold start, time-critical application, and serverless performance optimization.

Respectively, in this paper, we simply narrow down our analysis toward the security aspects of serverless. Therefore, we analyze possible serverless security issues and their related countermeasures. We see that some vulnerabilities are still waiting to be addressed and need better countermeasures instead of the current/existing ones. Thereafter, we quantitatively measure the probability of serverless attack (using Attack Tree and Attack-Defense Tree) with the possible attack scenarios and the related countermeasures. In fine, we show how the quantification can reflect toward the end-to-end security enhancement. In particular, it shows where to pay more attention to defend the security attack.

However, our attack tree analysis is limited with probability assumption of possible attacks and the related countermeasures. Besides, empirical evaluation and validation is important to carry out for this attack tree analysis. In fine, this study concludes with research challenges such as the burdensome and error-prone steps of setting the platform, and investigating the existing vulnerabilities of serverless computing, and possible future directions.

Future works

In this paper, we have shown a thorough analysis of Kubernetes and serverless computing. Besides, we have analyzed the possible security issues of serverless and related countermeasures. Moreover, we have quantitatively analyzed the security of serverless and showed how the quantification reflects toward the security enhancement. However, there are some deficiencies and rooms for further enhancement of Kubernetes and serverless computing. Such as,

- One of our future works includes efficiently picking up legacy functions that own some cloud resources and roles, which could be attacked by hackers.

- Second, for the issue of the over-privileged functions, we hope to solve this problem by creating a permission agent to assign privileges to different functions according to their jobs automatically.

- Third, improving function monitoring and logging, since today's logging utilities provided by cloud platforms are too weak to monitor each function's status from the application layer. Therefore, it is necessary to come up with an easy-use and efficient function monitoring tool/framework in the future.

- We have discussed a set of research challenges in section 14. We focus on analyzing these challenges to address them in a better way.

- As stated earlier, our attack tree security analysis is limited with probability assumption and empirical evaluation and validation. In future, we will work on devising a quantitative measure to logically compute the probability of an attack and countermeasure. Besides, we will perform empirical evaluation and validation of every attack and the related countermeasure. In particular, we focus on performing penetration testing to detect each and every security flaw and finally ensuring better countermeasures.

Abbreviations

• AWS

- Amazon Web Service

• ASAP

- Autonomous System Adaptation Platform

• CAS

- Content-Addressable Storage

• CI

- Continuous Integration

• CD

- Continuous Deployment

• CLI

- Command Line Interface

• CNI

- Container Network Interface

• CA

- Certificate Authority

• DRIP

- Dynamic Real-time Infrastructure Planner

• FaaS

- Function as a Service

• GCP

- Google Cloud Platform

• HA

- High Availability

• IoT

- Internet Of Thing

• IFC

- Information Flow Control

• IaaS

- Infrastructure as a Service

• K8S

- Kubernetes

• HPA

- Horizontal Pod Autoscaler

• MPSC

- Multi-Provider Serverless Computing

• P2P

- Peer-to-Peer

• SaaS

- Software as a Service

• SWITCH

- Software Workbench for Interactive, Time Critical and Highly self-adaptive Cloud applications

- SLO
- Service-Level Objective
- SIDE
- SWITCH Interactive Development Environment
- VM
- Virtual Machine
- ATree
- Attack Tree
- ADTree
- Attack–Defense Tree

Publisher's Note

References

1 Naik N (2016) Building a virtual system of systems using docker swarm in multiple clouds. In: 2016 IEEE International Symposium on Systems Engineering (ISSE), IEEE, pp 1–3

2 Guerrero C, Lera I, Juiz C. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. The Journal of Supercomputing. 2018; 74; 7: 2956-2983. 10.1007/s11227-018-2345-2

3 Cérin C, Menouer T, Saad W, Abdallah WB (2017) A new docker swarm scheduling strategy. In: 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), IEEE, pp 112–117

4 Soppelsa F, Kaewkasi C (2016) Native docker clustering with swarm. Packt Publishing Ltd

5 Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, omega, and kubernetes: lessons learned from three container-management systems over a decade. Queue. 2016; 14; 1: 70-93. 10.1145/2898442.2898444

6 Bernstein D. Containers and cloud: from lxc to docker to kubernetes. IEEE Cloud Computing. 2014; 1; 3: 81-84. 10.1109/MCC.2014.51

7 Ifrah S (2019) Deploy a containerized application with amazon EKS. In: Deploy Containers on AWS, Springer, pp 135–173

8 Bisong E (2019) Containers and google kubernetes engine. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform, Springer, pp 655–670

9 Ifrah S (2021) Deploy containerized applications with google kubernetes engine (GKE). In: Getting Started with Containers in Google Cloud Platform, Springer, pp 105–135

Orchestration C, Buchanan S, Rangama J, Bellavance N. Introducing Azure Kubernetes Service. 2019: Berline; Springer

Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, Mitchell N, Muthusamy V, Rabbah R, Slominski A, et al. (2017) Serverless computing: current trends and open problems. In: Research Advances in Cloud Computing, Springer, pp 1–20

Mohanty SK, Premsankar G, Di Francesco M, et al. (2018) An evaluation of open source serverless computing frameworks. In: CloudCom, pp 115–120

Back T, Andrikopoulos V (2018) Using a microbenchmark to compare function as a service solutions. In: European Conference on Service-Oriented and Cloud Computing, Springer, pp 146–160

Fox GC, Ishakian V, Muthusamy V, Slominski A (2017) Status of serverless computing and function-as-a-service (faas) in industry and research. arXiv preprint arXiv:170808028

Shahrad M, Balkind J, Wentzlaff D (2019) Architectural implications of function-as-a-service computing. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pp 1063–1075

Schneier B. Attack trees. Dr Dobb's journal. 1999; 24; 12: 21-29

Schneier B. Secrets and lies: digital security in a networked world. 2015: NewYork; Wiley. 10.1002/9781119183631

Kordy B, Mauw S, Radomirović S, Schweitzer P. Attack-defense trees. J Logic Comput. 2014; 24; 1: 55-87. 3159372. 10.1093/logcom/exs029

Audinot M, Pinchinat S, Kordy B (2017) Is my attack tree correct? In: European Symposium on Research in Computer Security, Springer, pp 83–102

Roy A, Kim DS, Trivedi KS. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security Commun Netw. 2012; 5; 8: 929-943. 10.1002/sec.299

Kordy P, Schweitzer P (2012) The ADTool Manual. University of Luxembourg

Kordy B, Kordy P, Mauw S, Schweitzer P (2013) ADTool: security analysis with attack–defense trees. In: International conference on quantitative evaluation of systems, Springer, pp 173–176

Gadyatskaya O, Jhawar R, Kordy P, Lounis K, Mauw S, Trujillo-Rasua R (2016) Attack trees for practical security assessment: ranking of attack scenarios with ADTool 2.0. In: International Conference on Quantitative Evaluation of Systems, Springer, pp 159–162

KubernetesOfficialDocumentation (2021) What is Kubernetes? https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Grider S (2019) Docker and kubernetes: the complete guide https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide/learn/lecture/11437326#overview

Hightower K, Burns B, Beda J (2017) Kubernetes: up and running: dive into the future of infrastructure. " O'Reilly Media, Inc."

Truyen E, Van Landuyt D, Preuveneers D, Lagaisse B, Joosen W. A comprehensive feature comparison study of open-source container orchestration frameworks. Appl Sci. 2019; 9; 5: 931. 10.3390/app9050931

Al Jawarneh IM, Bellavista P, Bosi F, Foschini L, Martuscelli G, Montanari R, Palopoli A (2019) Container orchestration engines: a thorough functional and performance comparison. In: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, pp 1–6

Acuña P. Amazon EC2 container service. Deploying rails with docker. 2016: Kubernetes and ECS; Springer: 69-98. 10.1007/978-1-4842-2415-1_4

Ifrah S (2019) Deploying containerized applications with amazon ECS. In: Deploy Containers on AWS, Springer, pp 83–133

Pousty S, Miller K (2014) Getting Started with OpenShift: a Guide for Impatient Beginners. " O'Reilly Media, Inc."

Lossent A, Peon AR, Wagner A (2017) PaaS for web applications with OpenShift Origin. In: J Phys: Conf Series, IOP Publishing, vol 898, p 082037

Aly M, Khomh F, Yacout S (2018) Kubernetes or openShift? Which technology best suits eclipse hono IoT deployments. In: 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), IEEE, pp 113–120

MANGAT M (2021) Best container orchestration tools for 2020 https://phoenixnap.com/blog/container-orchestration-tools

Pan Y, Chen I, Brasileiro F, Jayaputera G, Sinnott R (2019) A performance comparison of cloud-based container orchestration tools. In: 2019 IEEE International Conference on Big Knowledge (ICBK), IEEE, pp 191–198

Naser H (2017) Kubernetes Vs. mesos: a comparison of containerization platforms part II https://vexxhost.com/blog/kubernetes-mesos-comparison-containerization/

Modak A, Chaudhary S, Paygude P, Ldate S (2018) Techniques to secure data on cloud: docker swarm or kubernetes? In: 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), IEEE, pp 7–12

Nomad (2020) Nomad vs. Kubernetes https://www.nomadproject.io/intro/vs/kubernetes/

Linzel B, Zhu E, Flores G, Liu J, Dikaleh S (2019) How can OpenShift accelerate your Kubernetes adoption: a workshop exploring openShift features. In: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, pp 380–381

Anderson C. Docker [software engineering]. IEEE Software. 2015; 32; 3: 102-c3. 10.1109/MS.2015.62

Martin JP, Kandasamy A, Chandrasekaran K. Exploring the support for high performance applications in the container runtime environment. Human-centric Comput Inf Sci. 2018; 8; 1: 1-15. 10.1186/s13673-017-0124-3

Xie XL, Wang P, Wang Q (2017) The performance analysis of Docker and rkt based on Kubernetes. 2017 13th International Conference on Natural Computation. Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), IEEE, pp 2137–2141

Arundel J, Domingus J (2019) Cloud native devops with kubernetes: building, deploying, and scaling modern applications in the cloud. O'Reilly Media

Saito H, Lee HCC, Wu CY (2019) DevOps with Kubernetes: accelerating software delivery with container orchestrators. Packt Publishing Ltd

KubernetesOfficialDocumentation (2021) Kubernetes Components https://kubernetes.io/docs/concepts/overview/components/

Mannambeth M (2020) Certified kubernetes administrator (CKA) with practice tests. https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/learn/lecture/14296142

Kumar R, Trivedi MC. Networking analysis and performance comparison of kubernetes CNI Plugins. Advances in computer. 2021: Berline; Springer: 99-109

Boettiger C. An introduction to Docker for reproducible research. ACM SIGOPS Oper Syst Rev. 2015; 49; 1: 71-79. 10.1145/2723872.2723882

Belmont JM (2018) Hands-On continuous integration and delivery: build and release quality software at scale with Jenkins, Travis CI, and CircleCI. Packt Publishing Ltd

Sewak M, Singh S (2018) Winning in the era of serverless computing and function as a service. In: 2018 3rd International Conference for Convergence in Technology (I2CT), pp 1–5

Eivy A. Be wary of the economics of" Serverless" cloud computing. IEEE Cloud Comput. 2017; 4; 2: 6-12. 10.1109/MCC.2017.32

Van Eyk E, Toader L, Talluri S, Versluis L, Uţă A, Iosup A. Serverless is more: from PaaS to present cloud computing. IEEE Internet Comput. 2018; 22; 5: 8-17. 10.1109/MIC.2018.053681358

Yan M, Castro P, Cheng P, Ishakian V (2016) Building a chatbot with serverless computing. In: Proceedings of the 1st International Workshop on Mashups of Things and APIs, pp 1–4

Spillner J (2017) Snafu: Function-as-a-service (faas) runtime design and implementation. arXiv preprint arXiv:170307562

Kuntsevich A, Nasirifard P, Jacobsen HA (2018) A distributed analysis and benchmarking framework for apache openwhisk serverless platform. In: Proceedings of the 19th International Middleware Conference (Posters), pp 3–4

Djemame K, Parker M, Datsev D (2020) Open-source serverless architectures: an Evaluation of Apache OpenWhisk. In: 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), IEEE, pp 329–335

Ellis A (2019) The power of interfaces in OpenFaaS https://blog.alexellis.io/the-power-of-interfaces-openfaas/

Kaewkasi C (2018) Docker for serverless applications: containerize and orchestrate functions using OpenFaas, OpenWhisk, and Fn. Packt Publishing Ltd

PrometheusOfficialDocumentation (2021) What is Prometheus? https://prometheus.io/docs/introduction/overview/

Sabharwal N, Pandey P (2020) Getting started with prometheus and alert manager. In: Monitoring Microservices and Containerized Applications, Springer, pp 43–83

Turnbull J (2018) Monitoring with Prometheus. Turnbull Press

Brazil B (2018) Prometheus: up & running: infrastructure and application performance monitoring. " O'Reilly Media, Inc."

Brattstrom M, Morreale P (2017) Scalable agentless cloud network monitoring. In: 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), IEEE, pp 171–176

Nguyen N, Kim T. Toward highly scalable load balancing in kubernetes clusters. IEEE Commun Mag. 2020; 58; 7: 78-83. 10.1109/MCOM.001.1900660

Trivedi KS, Sahner R. SHARPE at the age of twenty two. ACM SIGMETRICS Perform Eval Rev. 2009; 36; 4: 52-57. 10.1145/1530873.1530884

Ou X, Singhal A. Quantitative security risk assessment of enterprise networks. 2011: Berlin; Springer. 10.1007/978-1-4614-1860-3

Henley EJ, Kumamoto H (1996) Probabilistic risk assessment and management for engineers and scientists. IEEE Press (2nd Edition)

Stallings W, Brown L, Bauer MD, Bhattacharjee AK. Computer security: principles and practice. 2012: NJ, USA; Pearson Education Upper Saddle River

Hubbard DW, Seiersen R (2016) How to measure anything in cybersecurity risk. Wiley Online Library

Ingoldsby TR (2010) Attack tree-based threat risk analysis. Amenaza Technologies Limited pp 3–9

Coles-Kemp L, Bullée JW, Montoya L, Junger M, Heath C, Pieters W, Wolos L (2015) Technology-supported Risk Estimation by Predictive Assessment of Socio-technical Security

Alpernas K, Flanagan C, Fouladi S, Ryzhyk L, Sagiv M, Schmitz T, Winstein K (2018) Secure serverless computing using dynamic information flow control. arXiv preprint arXiv:180208984

Bacon J, Eyers D, Pasquier TFM, Singh J, Papagiannis I, Pietzuch P. Information flow control for secure cloud computing. IEEE Transac Netw Service Manag. 2014; 11; 1: 76-89. 10.1109/TNSM.2013.122313.130423

Alpernas K, Flanagan C, Fouladi S, Ryzhyk L, Sagiv M, Schmitz T, Winstein K (2018) Secure serverless computing using dynamic information flow control. Proc ACM Program Lang 2(OOPSLA), https://doi.org/10.1145/3276488

O'Meara W, Lennon RG (2020) Serverless computing security: protecting application logic. In: 2020 31st Irish Signals and Systems Conference (ISSC), IEEE, pp 1–5

Podjarny G (2019) Serverless Security. O'Reilly Media Inc

Li X, Leng X, Chen Y (2021) Securing serverless computing: challenges, solutions, and opportunities. arXiv preprint arXiv:210512581

Datta P, Kumar P, Morris T, Grace M, Rahmati A, Bates A. Valve: securing function workflows on serverless computing platforms. Proc The Web Conf. 2020; 2020: 939-950

Kelly D, Glavin FG, Barrett E. Denial of wallet-defining a looming threat to serverless computing. J Inform Security Appl. 2021; 60: 102843

Jonas E, Schleier-Smith J, Sreekanti V, Tsai CC, Khandelwal A, Pu Q, Shankar V, Carreira J, Krauth K, Yadwadkar N, et al. (2019) Cloud programming simplified: a berkeley view on serverless computing. arXiv preprint arXiv:190203383

Ruddle A, Ward D, Weyl B, Idrees S, Roudier Y, Friedewald M, Leimbach T, Fuchs A, Gürgens S, Henniger O, et al. (2009) Deliverable D2. 3: Security requirements for automotive on-board networks based on dark-side scenarios. EVITA project

Guo J, Rahimi M, Cleland-Huang J, Rasin A, Hayes JH, Vierhauser M (2016) Cold-start software analytics. In: Proceedings of the 13th International Conference on Mining Software Repositories, pp 142–153

Štefanič P, Cigale M, Jones AC, Knight L, Taylor I, Istrate C, Suciu G, Ulisses A, Stankovski V, Taherizadeh S. SWITCH workbench: a novel approach for the development and deployment of time-critical microservice-based cloud-native applications. Future Gener Comput Syst. 2019; 99: 197-212. 10.1016/j.future.2019.04.008

Ghosh BC, Addya SK, Somy NB, Nath SB, Chakraborty S, Ghosh SK (2020) Caching techniques to improve latency in serverless architectures. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, pp 666–669

Hall A, Ramachandran U (2019) An execution model for serverless functions at the edge. In: Proceedings of the International Conference on Internet of Things Design and Implementation, pp 225–236

Nguyen HD, Zhang C, Xiao Z, Chien AA (2019) Real-time serverless: enabling application performance guarantees. In: Proceedings of the 5th International Workshop on Serverless Computing, pp 1–6

Du D, Yu T, Xia Y, Zang B, Yan G, Qin C, Wu Q, Chen H (2020) Catalyzer: sub-millisecond startup for serverless computing with initialization-less booting. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp 467–481

Bermbach D, Karakaya AS, Buchholz S (2020) Using application knowledge to reduce cold starts in FaaS services. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp 134–143

Mahajan K, Mahajan S, Misra V, Rubenstein D (2019) Exploiting content similarity to address cold start in container deployments. In: Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies, pp 37–39

Bardsley D, Ryan L, Howard J (2018) Serverless performance and optimization strategies. In: 2018 IEEE International Conference on Smart Cloud (SmartCloud), IEEE, pp 19–26

Mahmoudi N, Lin C, Khazaei H, Litoiu M (2019) Optimizing serverless computing: introducing an adaptive function placement algorithm. In: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, pp 203–213

Aske A, Zhao X (2018) Supporting multi-provider serverless computing on the edge. In: Proceedings of the 47th International Conference on Parallel Processing Companion, pp 1–6

Correia J, Ribeiro F, Filipe R, Arauio F, Cardoso J (2018) Response time characterization of microservice-based systems. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), IEEE, pp 1–5

Tsai PH, Hong HJ, Cheng AC, Hsu CH (2017) Distributed analytics in fog computing platforms using tensorflow and kubernetes. In: 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp 145–150

Zhou J, Velichkevich A, Prosvirov K, Garg A, Oshima Y, Dutta D (2019) Katib: A distributed general automl platform on kubernetes. In: 2019 USENIX Conference on Operational Machine Learning (OpML 19), pp 55–57

Trakadas P, Nomikos N, Michailidis ET, Zahariadis T, Facca FM, Breitgand D, Rizou S, Masip X, Gkonis P. Hybrid clouds for data-intensive, 5G-enabled IoT applications: an overview, key issues and relevant architecture. Sensors. 2019; 19; 16: 3591. 10.3390/s19163591

Alonso-Monsalve S, García-Carballeira F, Calderón A. A heterogeneous mobile cloud computing model for hybrid clouds. Future Gener Comput Syst. 2018; 87: 651-666. 10.1016/j.future.2018.04.005

Barcelona-Pons D, García-López P, Ruiz Á, Gómez-Gómez A, París G, Sánchez-Artigas M (2019) Faas orchestration of parallel workloads. In: Proceedings of the 5th International Workshop on Serverless Computing, pp 25–30

Ramon-Cortes C, Serven A, Ejarque J, Lezzi D, Badia RM. Transparent orchestration of task-based parallel applications in containers platforms. J Grid Comput. 2018; 16; 1: 137-160. 10.1007/s10723-017-9425-z

Footnotes

ADTool https://satoss.uni.lu/members/piotr/adtool/

Namespace encapsulates the Kernel's global resources, so that each namespace can have an independent and isolated resource sets. It helps ensure different processes to use the same resource in their namespaces without interfering with each other.

Docker Hub is a registry service on the cloud that allows developers to download Docker images that are built by other communities.

Pods is a collection of containers whose functions are highly related.

Volume is a store technique, which maps local file to container's directory. It enables developers to modify local code and dynamic update the container.

Ubuntu https://ubuntu.com/

Kubernetes The Hard Way https://github.com/kelseyheightower/kubernetes-the-hard-way

Vagrant is automatic tool for creating and managing multiple VMs at the same time.

CoreDNS is a DNS server. It is written in Go. It can be used in a multitude of environments because of its flexibility.

Kubeadm Way https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

High availability: HA means that developers usually create more than one node for some services to prevent the emergency. If one node breaks down, then other nodes can back up and work.

Grafana Dashboard https://grafana.com/grafana/dashboards

hey https://github.com/rakyll/hey

ADTool https://satoss.uni.lu/members/piotr/adtool/

~~~~~~~~

By Subrota Kumar Mondal; Rui Pan; H M Dipu Kabir; Tan Tian and Hong-Ning Dai

Reported by Author; Author; Author; Author; Author

~~~~~~~~