الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and scientific research
المدرسة الوطنية العليا للتكنولوجيا
National High School of Technology
Department of Electrical Engineering and Industrial Informatics

---

Final Year Project to Obtain the Diploma of
Master

Field:
Telecommunications

Speciality:
Telecommunication Systems and Networks

Subject:
State of the Art of Kubernetes Architecture and Components

Realized by:
LECHEHEB Manel

Members of The Jury:

Mrs BOUTARFA Souhila	Chair	M.C.B	ENST
Mrs ABBAD Leila	Supervisor	M.A.A	ENST
Mr SEFTA Mahfoud	Co-supervisor	Head of Cloud Infrastructure	Icosnet
Mr BELAHCENE Abdelkader	Examinator	M.A.A	ENST
Mrs CHIALI Imane	Examinator	M.C.B	ENST

Algiers, 06/07/2023

Academic year 2022 –2023

State of the Art of Kubernetes Architecture and Components

Manel LECHEHEB
Department of Electrical Engineering and Industrial Computing
National Higher School of Technology
Algiers, Algeria
m_lecheheb@enst.dz

***Abstract*—Kubernetes is a key component in data centers, offering advanced container orchestration capabilities. It simplifies the management and scaling of containerized applications, enabling data centers to efficiently manage large-scale deployments and optimize resource utilization. This article presents an introduction to the Kubernetes container orchestrator, its standards, and its benefits. In addition, it aims to clarify the Kubernetes architecture, its main components, and how they communicate with each other.**

***Index Terms*—kubernetes, k8s, container, cluster.**

I. Introduction

Kubernetes is a revolutionary technology that has completely transformed the operation of today's data centers. Developed as an open-source container orchestration platform, Kubernetes plays a crucial role in managing and scaling applications in data center environments.

Kubernetes' key role is its ability to abstract the underlying infrastructure, enabling developers and operators to focus on the logic of the application rather than the specifics of the environment. It offers a unified API and a declarative configuration approach, allowing users to define the desired state of the application and letting Kubernetes manage the orchestration and management tasks needed to maintain that desired state.

In addition, Kubernetes facilitates service discovery and load balancing, enabling smooth communication between containers and efficient distribution of network traffic. It supports updates and rollbacks, facilitating the deployment of new application versions.

In order to deploy and manage containerized workloads, it is necessary to master the architecture and components of Kubernetes. The main objective of this article is to present the Kubernetes architecture and its essential components.

II. Overview about Kubernetes

A. Definition

The word "Kubernetes" is Greek in origin and means "helmsman" or "pilot." K8s is an acronym created by counting the eight letters between the "K" and the "s". [1]

Kubernetes is a lightweight, flexible, open source container orchestration platform developed by google for managing containerized workloads and services. It enables declarative setup and automation in various deployment environments which facilite approache of exploiting the virtualisation by Containerization.

B. Containerization

Containerization is the process of packaging software code with all its necessary components, including libraries, frameworks, and other dependencies, into separate containers to ensure their isolation and independent execution.

Containerization virtualization involves partitioning directly at the operating system level. Each container runs in its own environment but on the same host operating system. That's why containers are generally used to virtualize a program rather than an entire server.

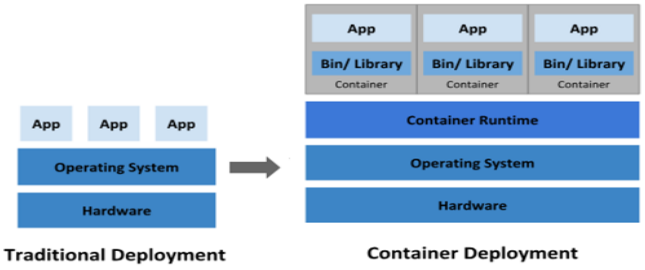


Fig. 1. Comparaison between Traditional and Container deployment

Traditional application deployment involves running applications on physical servers, where an operating system and other necessary dependencies are installed. This strategy is often beset by difficulties such as resource isolation, scalability issues, and high maintenance costs. Container deployment, on the other hand, is based on technologies such as Docker, and virtualizes the operating system at the OS layer, combining code and dependencies. Containers offer advantages such as size reduction, faster start-up times, efficient use of resources, portability and ease of use, and scalability via orchestration systems.

C. Why Kubernetes?

Because of the rising use of containers and microservices, Kubernetes is the required tool to manage these services that

can't be handled by scripts. K8s provides an extensive set of container orchestration features. They are as follows: [2]

- **Automatic bin packing:** It arranges containers to maximise utilisation depending on resource needs and constraints.
- **Self-healing:** It replaces and reschedules containers from failed nodes automatically.
- **Horizontal scaling:** It entails manually or automatically scaling apps based on CPU or utilisation.
- **Secret management:** It maintains sensitive data and application configuration parameters separately from the container image.

In addition, this open source orchestrator is mainly oriented to the devellopers who don't need to deal with VM management. Instead, they have a runtime environment in the form of a container to deploy their code. Kubernetes takes care of the underlying infrastructure layers, allowing the developers to focus solely on their application. With Kubernetes, the developer doesn't need to know where the apps are located or worry about the underlying infrastructure, as it is hidden from them.

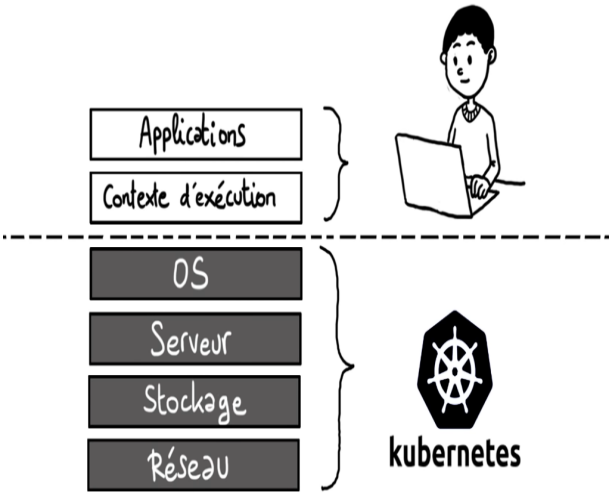


Fig. 2. Kubernetes and Developing

III. Kubernetes' Architecture

Kubernetes is a distributed system that has multiple components spread across different servers over a network, forming a cluster. A Kubernetes cluster consists basically of control plane, worker nodes, and a representational state transfer application programming interface (REST API) which ensure the communications between the users and the control plane.

A. Control Plane

The control plane, master node, is in charge of container orchestration, controlling and keeping the cluster in the appropriate condition, in other words, it makes global decisions about the cluster, detecting and responding to cluster events.

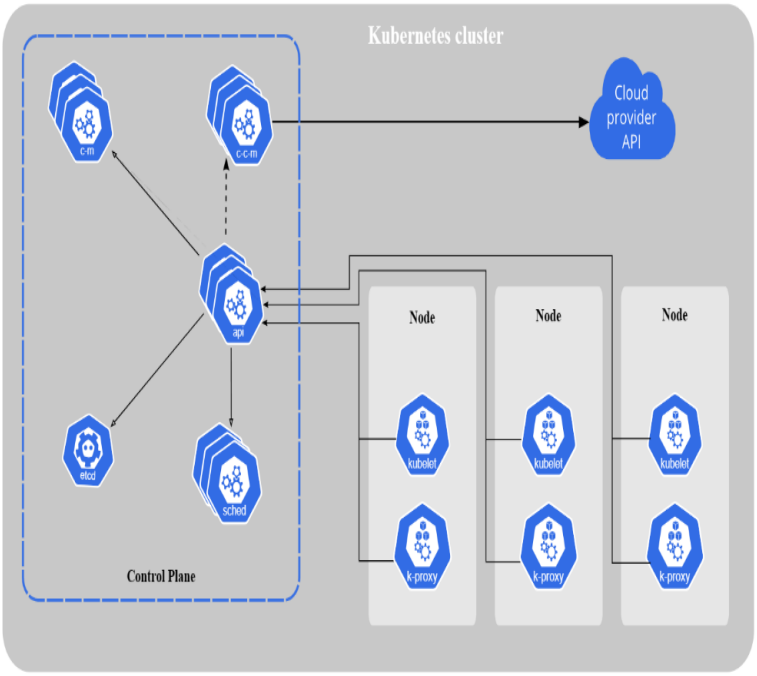


Fig. 3. Kubernetes' Architecture [3]

B. Worker nodes

Worker nodes are in charge of maintaining running containerized apps and providing runtime environment.

C. Kubernetes API

The Kubernetes API is a the basic component that allows users, different elements of a cluster, and external components to communicate with the Kubernetes system by accessing and altering the state of objects within it. The API provides a standardised interface for administering and controlling many components of Kubernetes.

The API server, which offers an HTTP API that supports communication between different components and entities inside the cluster, is at the heart of Kubernetes' control plane. Most activities can be performed by users and administrators via command-line tools such as kubectl, which connect with the API server. However, REST calls can be used to gain direct access to the API server.

Most operations can be performed through the kubectl command-line interface or other command-line tools, such as kubeadm, which in turn use the API. However, you can also access the API directly using REST calls.

Client libraries for many programming languages are available to make it easier to use the Kubernetes API. These libraries provide higher-level abstractions that facilitate programmatic interaction with the API. Go, Python, Java, JavaScript, and C# are among the client libraries that are officially supported.

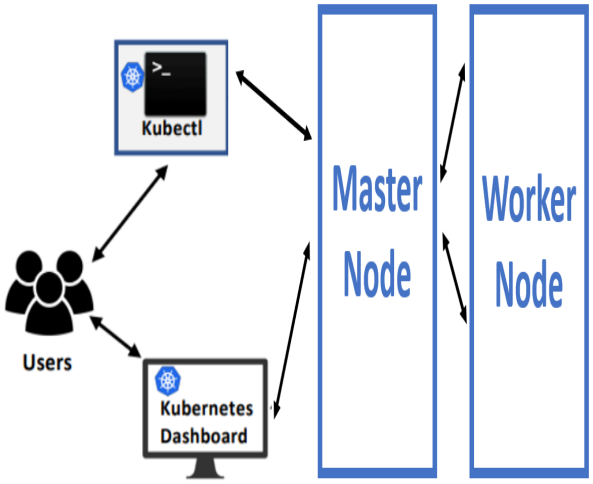


Fig. 4. Kubernetes API

IV. Components of the Kubernetes Cluster [2]

A. Control Plane Components

According to the Kubernetes requirements, the components of the control plane can be run on any machine in the cluster, but it's recommended to run all the devices on the same machine which doesn't contain user's containers. The control plane consists of several elements including:

1) Kube-apiserver: The Kubernetes API server is a layer of control element that offers the Kubernetes API. The API server serves as the control plane's front end.

The main Kubernetes API server implementation is kube-apiserver. kube-apiserver is intended to scale horizontally, i.e. by deploying extra instances. Many instances of kube-apiserver might be runed and balanced traffic among them.

2) Etcd : Etcd is a freely available, dispersed, persistent key-value store utilised for shared configuration of distributed systems or machine clusters, service discovery, and scheduler coordination. etcd facilitates safer automatic updates, coordinates tasks scheduled to hosts, and aids in the configuration of overlay networking for containers.

As Kubernetes' principal datastore, etcd stores and duplicates all Kubernetes cluster states. Because etcd is a vital component of a Kubernetes cluster, it is critical that it has a dependable approach to setup and management.

3) Kube-scheduler: The scheduler in Kubernetes is a control level operation that assigns Pods to Nodes. According to limitations and available resources, the scheduler evaluates which Nodes are valid placements for each Pod in the scheduling queue. The scheduler then selects each valid Node and assigns the Pod to one of them. Within a cluster, several schedulers can be deployed; the reference implementation is kube-scheduler.

4) Kube-controller-manager : The Kubernetes controller manager is a daemon that integrates the Kubernetes basic control loops. A control loop is a non-terminating loop that regulates the status of a system in robotics and automation applications. A controller in Kubernetes is a control mechanism that monitors the cluster's shared state via the apiserver while performing adjustments to try to shift the current state closer to the intended state. The replication controller, endpoints controller, namespace controller, and serviceaccounts controller are examples of controllers that ship with Kubernetes today.

5) cloud-controller-manager: A control plane component in Kubernetes that includes cloud-specific control logic. The cloud controller manager allows users to connect the cluster to the cloud provider's API and distinguishes between components that interface with the cloud platform and those that only interact with the cluster. Only controllers related to the cloud provider are run by the cloud-controller-manager. The cluster does not have a cloud controller manager if the user is running Kubernetes on its own premises or in a learning environment inside its own PC.

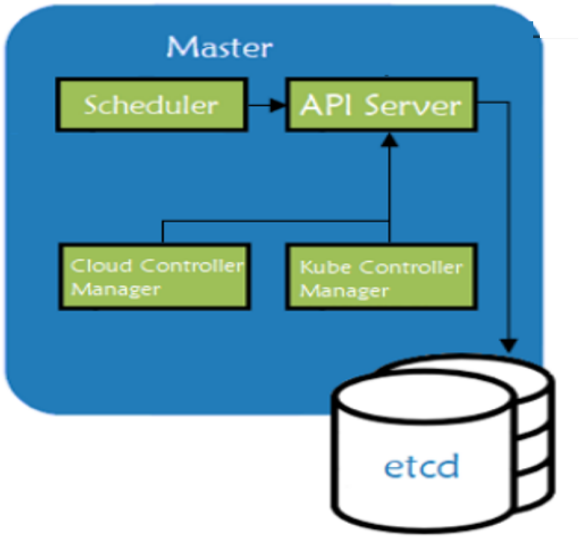


Fig. 5. Control Plane Components

The cloud-controller-manager, like the kube-controller-manager, integrates a number of conceptually distinct control loops into a single binary that runs as a single process. To boost performance or to assist accept faults, the user can scale horizontally (run more than one copy).

The controllers listed below may be cloud provider dependent:

- Node controller: Checks with the cloud provider to see if a node in the cloud has been deleted once it stops responding.
- Route controller: This component is responsible for configuring routes in the underlying cloud infrastructure.

- Service controller: For building, modifying, and deleting cloud provider load balancers.

B. Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1) Kubelet : An agent that operates on every cluster's node. It ensures that containers in a Pod are operating.

The kubelet accepts a set of PodSpecs delivered by different methods and guarantees that the containers defined in those PodSpecs are up and operating. Containers that were not created by Kubernetes are not managed by the kubelet.

2) kube-proxy : kube-proxy is a network proxy that operates on every node in the cluster and provides a component of the Kubernetes Service paradigm.

kube-proxy keeps network rules up to date on nodes. These network rules enable network communication to the Pods from network sessions both inside and outside the cluster.

If an operating system packet filtering layer is available, kube-proxy will use it. Otherwise, kube-proxy will forward the traffic.

3) Container runtime : The container runtime is the software that is in charge of container execution.

Container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface) are supported by Kubernetes.

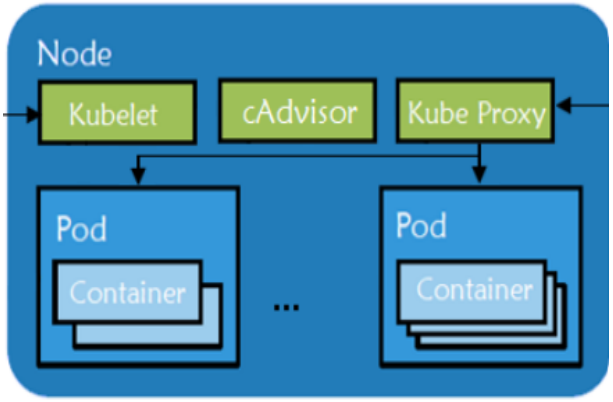


Fig. 6. Node Components

C. Addons

Addons in Kubernetes are additional components or extensions that enhance the functionality and capabilities of a Kubernetes cluster. These addons provide features such as networking, observability, security, and policy enforcement beyond the core functionalities of Kubernetes. They are designed to extend and customize the behavior of Kubernetes to meet specific requirements and use cases. It includes:

1) DNS : While the other addons are optional, all Kubernetes clusters ought to have cluster DNS because many examples rely on it.

Cluster DNS is a DNS server that serves DNS records for Kubernetes services in addition to the other DNS server(s) in the environment.

Containers launched by Kubernetes include this DNS server in their DNS searches by default.

2) Web UI (Dashboard): Dashboard is a web-based user interface designed to Kubernetes clusters. It enables users to handle and debug clustered software as well as the cluster itself.

3) Container Resource Monitoring: Container Resource Monitoring stores generic time-series metrics about containers in a centralised database and provides a user interface for accessing the information in the database.

4) Cluster-level Logging : Container logs are recorded to a central log storage with a search/browsing interface via a cluster-level logging mechanism.

It refers to the practice of collecting, storing, analyzing, and querying logs generated by containers and pods running within a Kubernetes cluster. It involves having a separate backend or logging solution that handles the storage and retrieval of logs independently of the nodes, pods, or containers in the cluster.

5) Network Plugins : Network plugins are software components that fulfil the CNI specification for container network interfaces. They are in charge of assigning IP addresses to pods and allowing them to communicate with one another inside the cluster.

V. How Nodes communicate to their control plane

In this part, we will explain the communication paths between the API server and the Kubernetes cluster:

A. Node to Control Plane

Kubernetes employs a "hub-and-spoke" API design. All API usage from nodes (or the pods they run) terminates at the API server. No other control plane components are intended to expose remote services. The API server is set up to accept remote connections on a secure HTTPS port (usually 443), with one or more forms of client authentication enabled. One or more types of permission should be enabled, especially if anonymous requests or service account tokens are permitted.

Nodes should be provided with the cluster's public root certificate so that they can connect securely to the API server with proper client credentials. A smart way is to submit the client credentials to the kubelet in the form of a client ID.

Pods that want to connect to the API server can do so securely by using a service account, which allows Kubernetes to immediately inject the public root certificate and a valid bearer token into the pod when it is constructed. The kubernetes service (in the default namespace) is setup with a virtual IP address that is forwarded (through kube-proxy) to the HTTPS endpoint on the API server. The control plane components connect with the API server through the secure port as well.

As a result, the default operating mode for connections from nodes and pods running on nodes to the control plane is secure by default and can run over untrusted and/or public networks.

B. Control plane to node

There are two basic communication pathways between the control plane (the API server) and the nodes. The first is from the API server to the kubelet process, which operates on each node in the cluster. The second is from the API server to any node, pod, or service via the API server's proxy capabilities:

1) API server to kubelet : The connections from the API server to the kubelet aims to:

- Getting logs for pods.
- Attaching to running pods (often via kubectl).
- Providing port-forwarding capability for the kubelet.

These connections terminate at the HTTPS endpoint of the kubelet. The API server does not check the kubelet's serving certificate by default, which exposes the connection to man-in-the-middle attacks and makes it insecure to run over untrusted and/or public networks.

We could use the –kubelet-certificate-authority parameter to supply the API server with a root certificate bundle to use to verify the kubelet's serving certificate in order to verify this connection.

If that is not an option, we employ SSH tunnelling between the API server and the kubelet to avoid connecting over an untrusted or public network.

Finally, to secure the kubelet API, Kubelet authentication and/or authorization should be enabled.

2) API server to nodes, pods, and services : Connections from the API server to a node, pod, or service are always plain HTTP connections, which are neither authenticated nor encrypted. They can be launched via a secure HTTPS connection by prefixing https: to the node, pod, or service name in the API URL, but they will not validate the certificate provided by the HTTPS endpoint or offer client credentials. As a result, while the connection will be encrypted, there will be no assurances of integrity. These connections are not presently safe to use across untrusted or public networks.

SSH tunnels are supported by Kubernetes to safeguard communication pathways between the control plane and nodes. The API server initiates an SSH tunnel to each node in the cluster (connecting to the SSH server listening on port 22) and routes any communication meant for a kubelet, node, pod, or service through the tunnel. This tunnel ensures that traffic does not leave the network in which the nodes are executing.

VI. Conclusion

The Kubernetes architecture is composed of many components operating together to enable container orchestration and maintain the cluster in an appropriate state. To achieve this, Kubernetes uses secure communications between nodes and the control plane. Nodes can communicate securely with the control plane by configuring the API server with certificate-based authentication, authorization, and security procedures. This ensures the reliability, scalability, and manageability of containerized applications in the Kubernetes ecosystem.

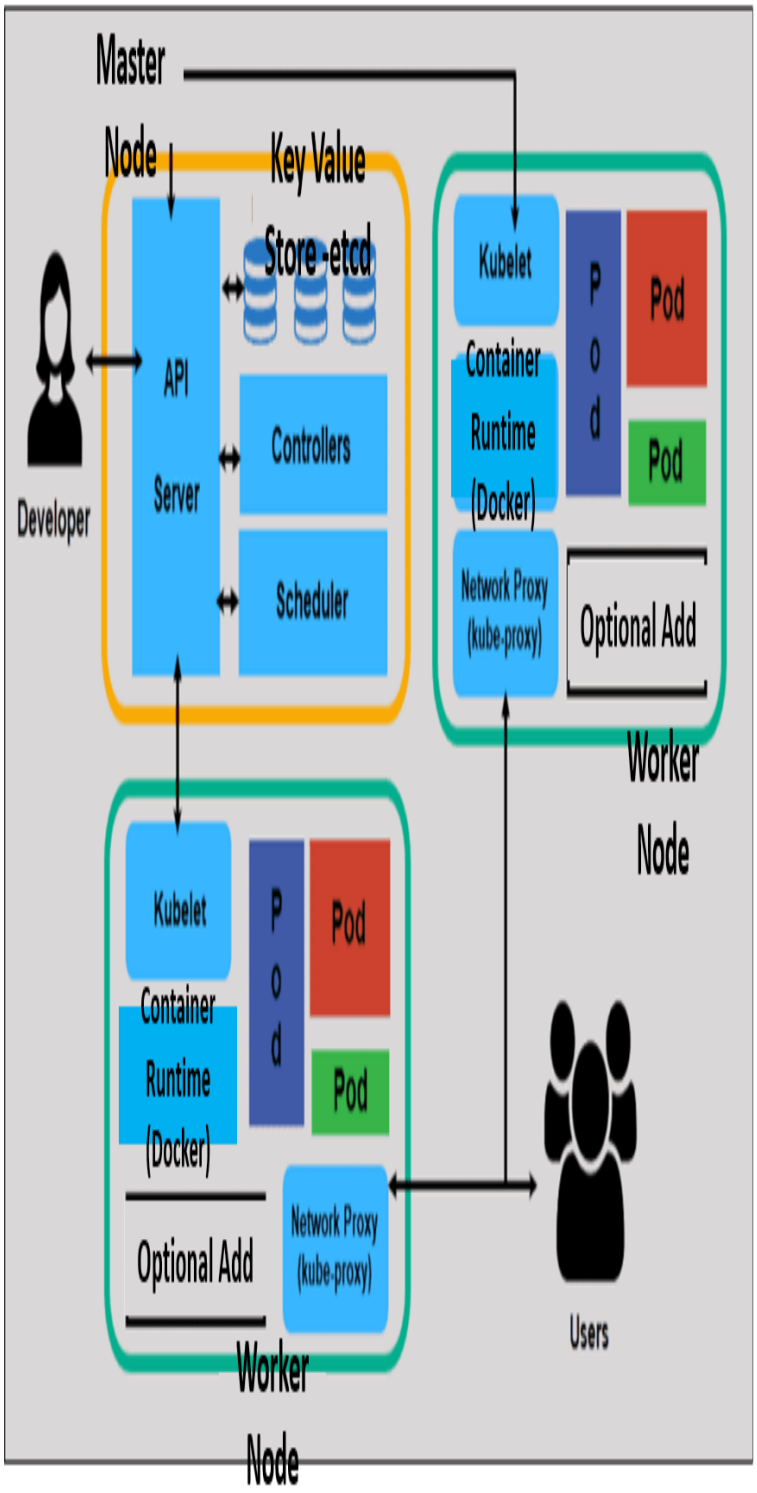


Fig. 7. Kubenetes' Cluster Components [4]

References

[1] Khalel, M. M., Pugazhendhi, M. A., & Raj, G. R. Sensors (2022). Enhanced Load Balancing in Kubernetes Cluster By Minikube.

[2] Production-Grade Container Orchestration. (n.d.). Kubernetes. https://kubernetes.io/

[3] Educative. (n.d.). The Kubernetes architecture simplified. Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/the-kubernetes-architecture-simplified
[4] Johnson, B. (2020). Carbon-Aware Kubernetes. Sustainable Software. https://devblogs.microsoft.com/sustainable-software/carbon-aware-kubernetes/