Contents

1. Introduction

1.1. Purpose of the Thesis

1.2. Scope of the Thesis

1.3. Structure of the Thesis

In chapter Background and Related Work

2. Background and Related Work

Placeholder placeholder

2.1. Containerization

Contenerization is packaging an app along with all necessary runtime stuff like libraries, executables or asets into an object called "container". The main benefits of container are[1]:

- Protable and Flexible – container can be run on bare metal or virtual machine in cloud regardless of operating system. Only a container runtime software like Docker Engine or containerd is required, which allows to interact with host system.
- Lightweight – container is sharing operating system kernel with hostmachine, there is no need to install separate operating system inside
- Isolated – does not depends on host's environment or infrastructure
- Standarized – Open Container Initiative standarize runtime, image and distribution specifications

A container image is set of files and configuration needed to run a container. It is immutable, only new image can be crated with new changes. Consists of layers. The layer contain one

modification made a image. All layers are cachable and can be reused when building an image. The mechanism is really usefull when compiling large application components inside one container[2].

2.2. Contianer Orchestration

Container orchestration is coordinated deploying, managing, networking, scaling and monitoring containers process. It automates and manages whole container's lifecycle, there is no need to worrying about of deployed app, orchestration software like Kubernetes will take care of its availability [1].

The Kubernetes Authors says: "The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s"" [3]. K8s is open-source orchestration platform capable of managing containers [3]. Key functionalities are [3]:

- Automated rollouts and rollbacks – updates or downgrades version of deplotyed containers at controller rate, replacing containers incrementally
- Automatic bin packing – allows to specify exact resources needed by container (CPU, Memory) to fit on appropriate node
- Batch execution – possible to create sets of tasks which can be run without manual intervention
- Designed for extensibility – permits to add feautres using custom resource definitions without changing source code
- Horizontal scaling – scales (replicate) app based of its need for resources
- IPv4/IPv6 dual-stack – allocates IPv4 or IPv6 to pods and services
- Secret and configuration management – allows store, manage and update secrets. Containers do not have to be rebuilt to access updated credentials
- Self-healing – restarts crashed containers or by failure specified by user
- Service discovery and load balancing – advertises a container using DNS name or ip. Loadbalances traffic across all pods in deployment

– Storage orchestration – mounts desired storage like local or shipped by cloud provider and make it available for containers

Understanding Kubernetes workflow becomes significantly easier by familiarizing with its architecture, which will be discussed in the following seciton.

2.3. Kubernetes Architecture

A Kubernetes cluster is a group of machines that run containers and provide all the necessary services to enable communication between containers within the cluster, as well as access to the cluster from the outside. There are two types of components, a control plane and worker node. Minumum one of each is needed to run a container, but to provide more robus and reliable production cluter is better to use two to three control plane nodes [4].

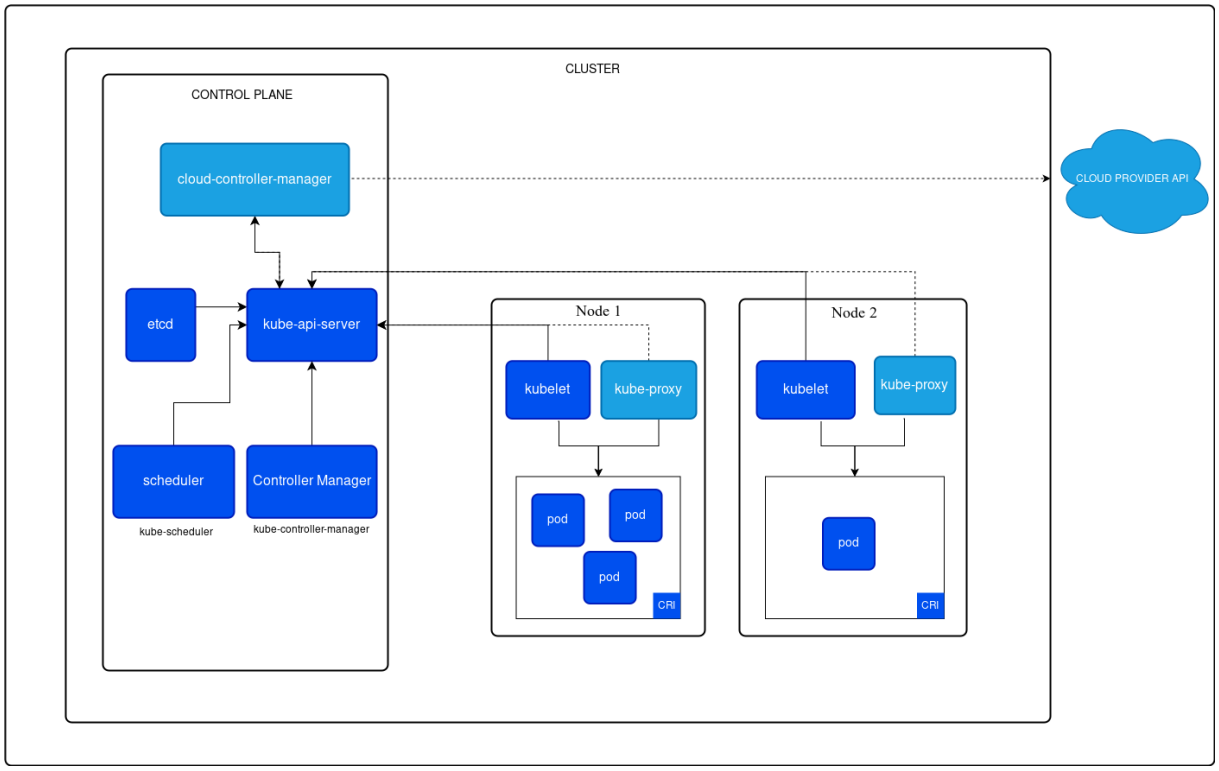

Figure 2.1: Kubernetes Cluster Architecture [4]

On figure figure 2.1 there is graphical representation of kubernetes cluster. Not all of components shown on figure are mandatory for kubernetes to work correctly. Take a look on control plane part, *cloud-controller-manager* might not be mandatory, in on-premise configurations where interacting with cloud provider is not needed. On the right side of figure in node representation is *kube-proxy* component, which is not mandatory as some networking plugins can

provide own implementation of proxy [4]. This is example of "Designed for extensibility", where Kubernetes can acquire 3rd-party features without changing its source code [3].

2.3.1. Control Plane

Control plane is like a brain in Kubernetes cluster. To interact with use kubcetl, asks *kube-apiserver*It is responsible for communication with worker nodes running pods, a smallest unit managed by K8s that has containers inside.

cloud-controller-manager

This component allows connect Kubernetes cluster and interact with cloud provider's API. It is combined with kube-controller-manager as single binary and can be replicated. This is the only component that talks to the cloud provider, separating other components from direct communication with the cloud. When running without cloud envirnoment this component is absent [4].

etcd

Etcd is an open source distributed key-value store service often used in distributed systems. It is responsible for maintaining both the current state and its previous version in its persistent memory [4][5].

kube-apiserver

Exposes Kubernetes API to interact with a cluster. Takes responsibility for handling all requests from components and users. This is the component which answers for cluster administrator requests sent by kubectl [4].

kube-controller-manager

Component which run controller processes. Its compiled binary consists of multiple controllers. Example controllers are [4]:

– Node controller – observes worker nodes if are up and running

– Job controller – responsible for batch execution jobs

– EndpointSlice controller – connects services with pods

More controller names can be found in kubernetes source code.

kube-scheduler

Takes care of pods which are not assigned to a worker node yet. kube-scheduler is looking for node that meets pod's scheduling requirements and fit a pod on that node. Such a node is called feasible node [6].

2.3.2. Nodes

All of below mentioned components run on every node in a cluster.

Container runtime

Node's key component, has ability to run, execute commands, manage and delete containers in efficient way [4].

kube-proxy

Create networking rules which allows to communicate with Pods from outside cluster. If available kube-proxy uses operating system packet filtering to create set of rules. It is also able to forward traffic by itself. This component is optional, can be replaced with different one if desired one implements key features. [4].

kubelet

It is responsible for managing containers inside pod on its node. Uses Container Runtime Interface to communicate with containers [4] [7].

2.3.3. Objects

Namespace

The purpose of namespace object is to isolate group of resources like pods, deployments, services etc. in a cluster. It helps to organise cluster into virtual sub areas of working space. If *Service* is created in some custom namespace <service-name>.<namespace-name>.svc.cluster.local DNS entry within cluster is created [8].

Pods

Pods are the smallest deployable objects in Kubernetes. It contains in one or more containers, which can communicate with each other using localhost interface. Since they share IP address, the can not use the same ports. It is really useful, when our service consist of two apps

which are coupled together. For example there is a pod which has two containers, one responsible for compiling a code, second one is creating cache entry from compiled object and uploads to some data storage. It has more sense, as sharing data among containers in a pod is rather easier than on node between pods. Scaling is simpler as replicating one pod instead of two. Moreover communication between apps happends using localhost, in scenario where there are two pods with one container, ClusterIP *Service* is needed. However the most common approach is to run one container per pod, where pod is just managing wrapper for contenerized app. Also rather then creating pod directly it is more common to use workload resource like *Deployment* [9].

ReplicaSet

Basically *ReplicaSet* consists of pod template, and runs desired number of pods [10].

Deployment

Deployment is a higher level abstraction over *ReplicaSet*, that manages its lifecycle. It provides more features like rollingback an app, as it keeps history of configurations [11].

DaemonSet

Running pods using DaemonSet guarantee that every node will have copy of desirded pod (if resource requirements are met etc.). It has ability to automatically add or remove pods, if number of nodes changes. The typical usege is creating monitoring pod on every node [12].

StatefulSet

StatefulSet unlike *Deployment* is stateful. It saves an identity of each pod and if e.g. some persistent storage is assigned to specific e.g. database pod and it dies, kubernetes will recreate pod on the same node as it was previously [13].

Job

Runs pod that does one task and exists. Kubernetes will retry execution if pod fails specific number of tries set in its configuration [14].

CronJob

Behaviours like *Job*, but is able to run regularly every given time for tasks like database backups or log rotation [15].

Service

Service exposes an application running inside a cluster by using an enpoint. As a pod is ephemeral resource and its address changes from time to time (e.g. when pod is recreated) it better to create dns name that resolves IP address. Moreover service will not advertise unhealthy pods. Usually service exposes one port per service, but for example web app might expose http and https ports. There are four types of services [16].

1. ClusterIP – makes one pod available to other inside cluster by exposing application using inter-cluter IP address. Although its oriented to be accessible within the cluster, objets like *Ingress* or *Gateway API* can expose service to the outside.

2. NodePort – by default allocates port (from range 30000-32767) to publish service on every node's ip. In this scenario every node on specified port acts like a proxy to deployed app.

3. LoadBalancer – kubernetes does not provide loadbalancer by default and when creating such a service it interacts with cloud provider to create external service for traffic balancing. Loadbalancer can be installed inside cluster.

4. ExternalName – allows pods inside Kubernetes to access external service using defined name rather than using IP address

Ingress

Ingress is an object that manages outside cluster access to services inside a cluster. It is a single point of entry to route traffic to specified pod based on configuration. This is only high abstract object that specifies routing rules in cluster. Real functionalities are provided by an *Ingress Controller*. Nowadays the development of Ingress is frozen, Kubernetes authors pay attention to its successor a *Gateway API* [17].

Ingress Controller

Ingress Controller fulfills an *Ingress* and starts serving an application which performs configured rules. Any implementation has its own features, but common functionalities are L4/L7 loadbalancing, host and path based routing, SSL termination. This is the real application that runs in a pod. Ingress Controller have to be installed manually and is not part of Kubernetes, however the container orchestration tool developers maintain AWS, GCE, and nginx ingress controllers [17][18].

Gateway API

The functionalities of Gateway API are so wide, that the Kubernetes authors use term "project". The project mainly focus on L4 and L7 routing in a cluster. It succeeds *Ingress*, Load Balancing and service mesh APIs. The Gatewa API resource model is role-oriented [19].

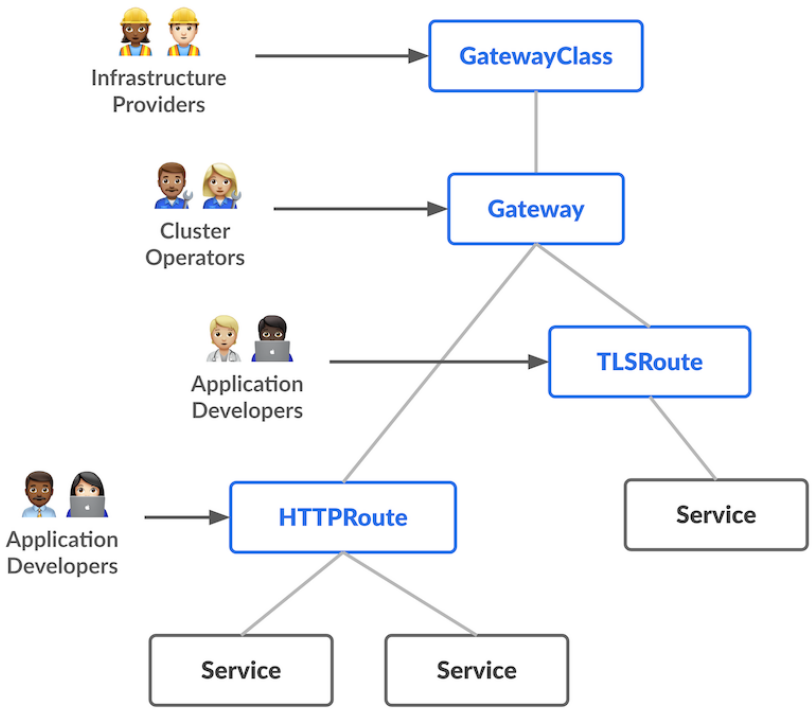

Figure 2.2: Gateway API roles-oriented resource model [19]

The model focus on 3 separate group of peoples who interact with a cluster on different levels.

On a top of figure 2.2 there are infrastructure providers, who provide GatewayClass resource. They are responsible for the overall multiple clusters infrastructure, rather than ensuring developers can access pods correctly [19]. The Gateway API craetors provides clear overview of what GatewayClass: "This resource represents a class of Gateways that can be instantiated.". It defines specific types of loadbalancing implementations and provide clear explenation of capabilities availabe in Kubernetes resource model. The functionality is similiar to *Ingress* [19] [20]. There can be more than one GatewayClasss created. [19].

Cluter operators are in the middle of figure 2.2, they make sure that cluster is meets requirements for several users. As maintainers define Gateway resource, some loadbalancing system is provisioned by GatewayClasss. Gateway resource defines specific instance which will handle

incomming traffic. Allows to define specific protocol, port or allowed resources route incomming traffic [19] [21].

End users specified on Gateway API resource model on 2.2 figure are application developers. They focus on serving application to the clients by creating resource named HTTPRoute. The resource defines HTTP routing from defined gateway to end API object like service. GRPCRoute is simmiliar, but operates on different protocol. [19] [22].

Gateway API is not an API Gateway. An API Gateway in general is resposible for routing, loadbalancing, information exchange manipulation and much more depending on specific implementation. Gateway API is set of three resources mentioned earlier, which creates a role-oriented Kubernetes service networking model. Creators of Gateway API provide a clear explenation: "Most Gateway API implementations are API Gateways to some extent, but not all API Gateways are Gateway API implementations" [19].

2.4. Cluster Networking

Networking is a most important thing in Kubernetes, the whole point is to obtain reliable and robust communication among containers, pods, services, nodes and external systems in a cluter [23]. There are four types of network communication: [23]:

1. container-to-container – communicates by sharing network resources inside a pod
2. Pod-to-Pod – every pod can communicate with any other pod without need to use NAT as every of them has its own IP address [24].
3. Pod-to-Service – covered by service type ClusterIP, which provides inter-cluster IP address
4. External-to-Service – held by services type NodePort and Loadbalancer, which expose pod to the outside

Kubernetes allocates IP addresses to nodes, services and pods [23]:

– *kubelet* or *cloud-controller-manager*, depending on local or cloud infrastructure allocates IP address for nodes

– *kube-apiserver* allocates IP address for services

– for allocation of IP address to pod is responsible networking plugin which is an implementation of *Container Network Interface (CNI)*

2.5. Container Network Interface (CNI)

CNI is standarized by Cloud Native Computing Foundation set of API rules which defines container networking. Generally speaking CNI is responsible for pod-to-pod communication, which include assigning IP addresses, configuring network interface inside container and routing [24].

2.6. Overview of Selected CNI Plugins

Table 2.1: Comparison of Antrea and Cilium [25][26][27][28][29].

Feature/Plugin	**Antrea**	**Cilium**
Dataplane	Open vSwitch	eBPF
Encapsulation	VXLAN or Geneve	VXLAN or Geneve
Encryption	IPsec or WireGuard tunnels	IPsec or WireGuard tunnels
Security	Extends Kubernetes Network Policies	Advanced security policies
Observability	Theia and Grafana for visualization	Hubble
Purpose	Simplified Kubernetes networking management	For large-scale cluters
Additional features	Network policies for non-Kubernetes nodes	BGP to advertise network outside cluster
Gateway API	No support	Fully supports Gateway API
Egress Gateway	Basic egress gateway capabilities	Advanced egress gateway support

Antrea is an open-source CNI plugin which is built on Open vSwitch [25]. OvS is a virtual swtich with capability of handling traffic flow between vitrual machines and containers [30]. Antrea's main focus is L3/L4 networking ad security services, such as network policies. The resource is responsible for managing traffic flow between pods. By default every pod can communicate with any other pod, but network policies can specify if pod A is able to talk to pod B [31]. Consider scenario with three pods, client, frontend and backend. There is no need to allow client communication directly with backend, so network policies allows traffic flow from client to frontend and direct communication with backend is not allowed [32].

Cilium, open-source CNI which uses eBPF (extended Barkeley Packer Filter) for packet processing, security and deep observability using Hubble [33]. eBPF is a technology that allows running defined programs, with custom logic inside operating system kernel in privileged context without need of any kernel source code changes or loading modules. Lack of switching between kernel and user space, which reduces latency [34].

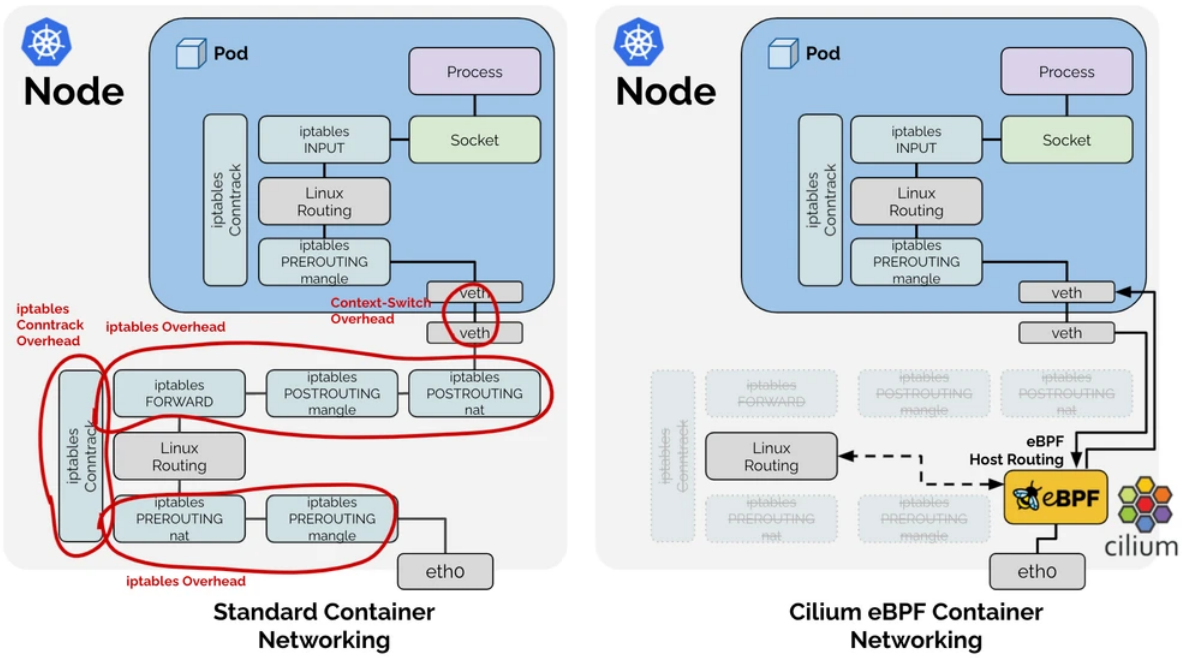

Figure 2.3: Cilium eBPF host-routing [35]

As seen on figure 2.3, the whole point of eBPF networking is skipping overhead that comes from iptables. Moreover eBPF implements hash tables for storing routing policies, which time complexity is O(log n), comparing to iptables array O(n). It makes clear that large-scale clusters will benefit from using eBPF [36].

2.7. Related Work

As discussed in [37] the performance of different CNIs can vary widely, some CNIs performing two to three times better than others, making it essential to choose right plugin for particular workload. Authors says, that developing automated methodology of CNI plugin evaluation is a key aspact, specifically in large High-Performance Computing (HPC) environments. This allows for reproducible and consistent tests across different configurations, reducing the overhead of manual testing. To achieve that, tools like Ansible can be helpful. They state that Linux Kernel or NIC can be a bottleneck in networking performance, so they extend maximum buffer size, scale TCP window, disable TCP Selective Acknowledgement, increase SYN Queue Size or enabled Generic Receive offload. The paper shows results of comparison four CNI plugins, such as Antrea, Cilium, Calico, Flannel using TCP/UDP in base and with optimized system settings [37].

In [38] the authors measure CNI plugins for inter-host and intra-host communication using

UDP and TCP protocols. They introduce the concept of CPU cycles per packet (CPP) to evalueate CNI efficiency. By measuring throughput, RTT, and latency, they compare how different CNI plugins (flannel, weave, cilium, kuber-router, calico).

– Tekst

Bibliography

[1] Redhat Inc. What is containerization? https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization. Accessed, 07-Dec-2024.

[2] Docker Inc. What is an image? https://docs.docker.com/get-started/docker-concepts/building-images/. Accessed, 07-Dec-2024.

[3] The Kubernetes Authors. Overview. https://kubernetes.io/docs/concepts/overview/. Accessed, 07-Dec-2024.

[4] The Kubernetes Authors. Cluster Architecture. https://kubernetes.io/docs/concepts/architecture/. Accessed, 07-Dec-2024.

[5] etcd Authors. Data model. https://etcd.io/docs/v3.5/learning/data_model/. Accessed, 08-Dec-2024.

[6] The Kubernetes Authors. Kubernetes Scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/. Accessed, 08-Dec-2024.

[7] The Kubernetes Authors. Container Runtimes. https://kubernetes.io/docs/setup/production-environment/container-runtimes/. Accessed, 08-Dec-2024.

[8] The Kubernetes Authors. Namespaces. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. Accessed, 10-Dec-2024.

[9] The Kubernetes Authors. Pods. https://kubernetes.io/docs/concepts/workloads/pods/. Accessed, 08-Dec-2024.

[10] The Kubernetes Authors. ReplicaSet. https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/. Accessed, 10-Dec-2024.

[11] The Kubernetes Authors. Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. Accessed, 10-Dec-2024.

[12] The Kubernetes Authors. DaemonSet. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. Accessed, 10-Dec-2024.

[13] The Kubernetes Authors. StatefulSets. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. Accessed, 10-Dec-2024.

[14] The Kubernetes Authors. Jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job/. Accessed, 10-Dec-2024.

[15] The Kubernetes Authors. CronJob. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/. Accessed, 10-Dec-2024.

[16] The Kubernetes Authors. Service. https://kubernetes.io/docs/concepts/services-networking/service/. Accessed, 10-Dec-2024.

[17] The Kubernetes Authors. Ingress. https://kubernetes.io/docs/concepts/services-networking/ingress/. Accessed, 10-Dec-2024.

[18] The Kubernetes Authors. Ingress Controllers. https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/. Accessed, 10-Dec-2024.

[19] The Linux Foundation. Introduction. https://gateway-api.sigs.k8s.io/. Accessed, 11-Dec-2024.

[20] The Linux Foundation. GatewayClass. https://gateway-api.sigs.k8s.io/api-types/gatewayclass/. Accessed, 11-Dec-2024.

[21] The Linux Foundation. Gateway. https://gateway-api.sigs.k8s.io/api-types/gateway/. Accessed, 11-Dec-2024.

[22] The Linux Foundation. HTTPRoute. https://gateway-api.sigs.k8s.io/api-types/httproute/. Accessed, 11-Dec-2024.

[23] The Kubernetes Authors. Cluster Networking. https://kubernetes.io/docs/concepts/cluster-administration/networking/. Accessed, 10-Dec-2024.

[24] Stephanie Susnjara and Ian Smalley. What is Kubernetes networking? . https://www.ibm.com/topics/kubernetes-networking. Accessed, 11-Dec-2024.

[25] The Antrea Contributors. Documentation. https://antrea.io/docs/v2.2.0/. Accessed, 12-Dec-2024.

[26] Cilium Authors. Cilium BGP Control Plane. https://docs.cilium.io/en/stable/network/bgp-control-plane/bgp-control-plane/. Accessed, 12-Dec-2024.

[27] Cilium Authors. Egress Gateway. https://docs.cilium.io/en/stable/network/egress-gateway/egress-gateway/. Accessed, 12-Dec-2024.

[28] Cilium Authors. Gateway API Support. https://docs.cilium.io/en/stable/network/servicemesh/gateway-api/gateway-api/. Accessed, 12-Dec-2024.

[29] Cilium Authors. Transparent Encryption. https://docs.cilium.io/en/stable/security/network/encryption/. Accessed, 12-Dec-2024.

[30] A Linux Foundation Collaborative Project. Production Quality, Multilayer Open Virtual Switch. https://www.openvswitch.org/. Accessed, 12-Dec-2024.

[31] The Kubernetes Authors. Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/. Accessed, 12-Dec-2024.

[32] Inc. Tigera. Kubernetes policy, demo. https://docs.tigera.io/calico/latest/network-policy/get-started/kubernetes-policy/kubernetes-demo. Accessed, 12-Dec-2024.

[33] Jeremy Colvin. Introduction to Cilium & Hubble. https://docs.cilium.io/en/latest/overview/intro/. Accessed, 12-Dec-2024.

[34] eBPF.io authors. eBPF Documentation. https://ebpf.io/what-is-ebpf/. Accessed, 12-Dec-2024.

[35] Cilium Authors. CNI Benchmark: Understanding Cilium Network Performance. https://cilium.io/blog/2021/05/11/cni-benchmark/. Accessed, 12-Dec-2024.

[36] Jeremy Colvin. What is Kube-Proxy and why move from iptables to eBPF? https://isovalent.com/blog/post/why-replace-iptables-with-ebpf/. Accessed, 12-Dec-2024.

[37] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić. Performance and latency efficiency evaluation of kubernetes container network interfaces for built-in and custom tuned profiles. *Electronics*, 13(3972), 2024.

[38] Shixiong Qi, Sameer G Kulkarni, and K. K. Ramakrishnan. Understanding container network interface plugins: Design considerations and performance. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pages 1–6, 2020.