Centaurus – An Open Cloud Infrastructure Project at Scale

Whitepaper

1. INTRODUCTION AND CHALLENGES

In today's transformational digital journey, the business, social, economic and technology trends play a major part in shaping the future of an enterprise. Cloud infrastructure and services have been established as the core part of Enterprise's IT and their digital transformation. More and more enterprises are leveraging cloud computing technologies to accelerate their business innovation by either migrating their applications and data to public cloud or building their own private cloud or using hybrid cloud model. The rise of emerging 5G, AI, Edge Computing, and IoT application landscape is offering Cloud Computing further more exciting opportunities as well as challenges to meet today's and tomorrow's enterprise digitization needs.

An open source cloud, the cloud built by open source technologies such as Openstack and Kubernetes, has been leading the way in the innovation of cloud computing technology itself, and we have seen more and more companies leveraging cloud technologies to accelerate their business innovations. At the same time, new types of applications and/or workloads pose new challenges to the cloud infrastructure platforms. The following is a list of challenges and/or trends we've observed that face cloud technology platforms:

- As more and more applications move to the cloud, there is an increasing demand for cloud infrastructure to manage ever increasing pool of compute nodes with scale, and to provision and deploy ever increasing workloads with consistent speed.

This challenge has been driving the new development and/or optimization of distributed cluster management platforms and lightweight virtualization technologies such as Container and Serverless. Current and future compute cluster management platforms will be continuously challenged to manage 100K+ compute nodes in a cluster and be able to provision and startup hundreds and even thousands of application instances within a minute.

- Both Cloud providers and Enterprises have been asking for the capability in their cloud technology infrastructure to provide support for managing and orchestrating heterogeneous resource types (bare-metal, VMs, containers, Serverless, Uni-Kernels, etc.) via an “unified ” resource management and orchestration platform as a single pane of glass. Therefore, the future cloud infrastructure needs to be the “unified” platform to meet the challenge and to simply reduce the management cost for both cloud providers and enterprise customers.

Modern cloud native applications are mostly designed for scale-out architectures that are more suited for containerized environments. A typical enterprise cloud environment isn’t just about containers only as containers may not be appropriate for all enterprise workloads and use cases. Most enterprises still run large number of legacy apps that run on bare metal and traditional VM environments

- With the convergence of traditional cloud computing and edge computing, and the emergence of new types of workloads such as 5G, AI and IoT applications, the cloud infrastructure platforms are being challenged to manage not only data center resources but also the edge compute nodes to support the new types of distributed applications cross data center and edge.

Accordingly, current and new cloud will need a distributed cloud architecture that streamlines workloads and data management; and optimizes performance and resource usage for cloud, edge, and devices in the new intelligent and connected world.

The current open source cloud platforms mostly treat Edge and AI as afterthought. The new open source cloud platform needs to be architected with Edge as part of the overall architecture from day one. For example, AI modeling can be done on the Cloud, while AI inferencing can be done on the Edge connecting to billions of IoT devices and sensors running 5G speed networks. Cloud-Edge computing combined with the optimized latency performance of 5G Core processing can reduce round-trip-time by up to two orders of magnitude in situations where there is tight control over all parts of the communication chain. This has enabled a brand-new class of intelligent cloud applications in the areas of industrial robotic/drone automation, V2X, and AR/VR infotainment, associated innovative business models, etc.

- Digitization is upending many core tenets of competition among industries by lowering the cost of entering markets and providing high-speed passing lanes to scale up enterprises. Hyperscale businesses are pushing the envelope of digitization as these businesses have users, customers, devices, interactions, and connectivity numbered in the hundreds of millions, billions, and more.

Based on the numerous recent surveys, enterprises aspire to emulate the network infrastructure designs of the hyperscale cloud service providers in their own network architecture. This means the network throughput and scalability offered by the current open source cloud platforms are no longer sufficient. The new requirements are 100K+ compute nodes, running at the 10K+ concurrent workloads per minute, and are capable of routing and managing communication among 10M+ endpoints/ports in a regional. Current cloud platforms in the open source communities provide very limited support for extremely high scalable networking in the virtualized cloud environment. This is primarily because contemporary cloud networking virtualization solutions are still cobbled together on top of age-old static networking designs. Such solutions are incapable of provisioning & management scale of the modern dynamic cloud workloads.

- Hybrid cloud and multi-cloud become a norm of Enterprise cloud strategy, and application portability cross cloud becomes a requirement to many companies. Open API and compatibility with the industry cloud ecosystem is a challenge to the new generation of cloud infrastructure technology development.

With these challenges and trends in the cloud computing technologies, we believe that as an industry and an open source community, there is a need for building the next generation open source, hyper-scale and unified cloud infrastructure that works with existing cloud technologies and can help enterprises meet the continuously growing digitization requirements. Today’s hyper-scale is tomorrow’s enterprise IT.

Centaurus is an open source project that we’ve been working on to build the next generation cloud infrastructure that meets the 5G, AI, Edge, and IoT needs. In this white paper, we will discuss its vision, project ideas, and use cases of Centaurus. We hope we can pique your interest and invite you to join us in making Centaurus a viable open cloud infrastructure platform for the future of Enterprise IT digitization journey.

2. VISION OF CENTAURUS

The vision of Centaurus open source project is to build a unified, high performance, and large-scale cloud infrastructure platform meeting the challenges discussed in the previous section.

Centaurus is designed to meet the infrastructure requirements for the new types of cloud workloads such as 5G, AI, Edge, and IoT applications. It is an open cloud native infrastructure that manages unified resources and provides the same API experience for all workload types (VMs, Containers, and Serverless etc.) through a single converged cloud technology stack, as shown in the diagram in **Figure 2.1**.

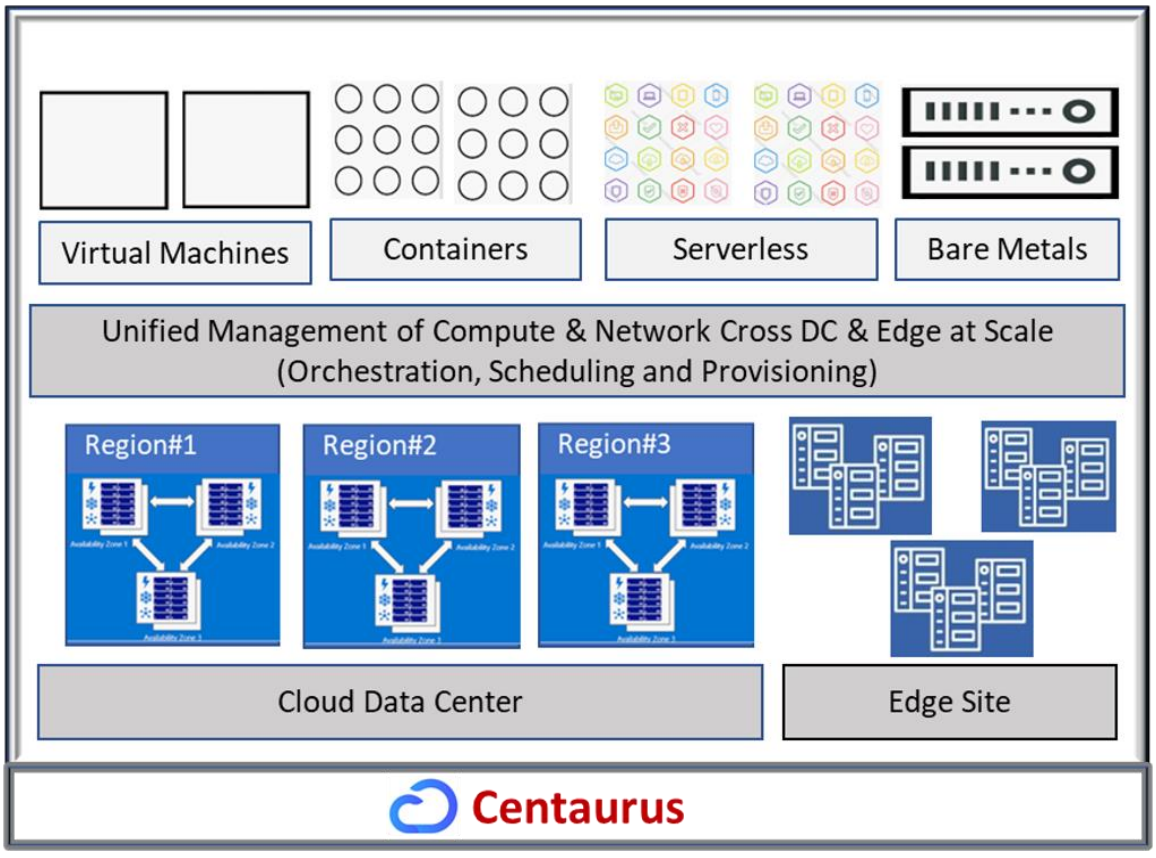

Figure 2.1: Vision of Centaurus

Centaurus is aimed to be deployed at regional level to support the management of 100K-300K compute nodes per region, and the provisioning of 10M+ network endpoints for applications and/or workloads. These hyper-scale capabilities will dramatically change the economics of enterprise IT. With multi-tenancy built from ground up, Centaurus project provides a fully

featured multi-tenant environment for all types of workloads running in the platform, supporting virtualization technologies such as KVM, Kata container, Firecracker and more in a single runtime environment. Centaurus networking solution is designed to support large scale of virtual networks and network endpoints with extremely low latency for forwarding and routing network traffics among VMs and containers. It aims to support near line rate of traffic routing between VMs and/or containers. Overarching goal is to eliminate the burden of orchestrating physical/virtual compute, network, and storage infrastructure, and enable application operators and developers to focus entirely on application-centric primitives for self-service operations environments. The goals are as shown in **Figure 2.2**.

Figure 2.2: Centaurus Goals

Centaurus platform addresses all the gaps illustrated earlier and enables a holistic hyper-scale cloud environment as open source software.

Currently, there are very limited existing open source solutions that address very large-scale cloud environments and at the same time manage infrastructure resources across all workload types in a unified manner. We see a need to build such a cloud infrastructure platform in an open source environment and to provide a choice for enterprises and/or customers to build their public or private cloud.

3. CENTAURUS ARCHITECTURE

This section talks about the architecture of Centaurus, its key features and technology spec. We will also present a reference deployment of Centaurus in this section to show how to use Centaurus to build a multi-region, high-available and large-scale cloud infrastructure.

3.1 Architecture

Centaurus currently consists of the following two sub-projects:

- Arktos: for large-scale cloud compute
- Mizar: for large-scale cloud virtual networking

Figure 3.1 below is an abstract illustration about how Arktos and Mizar work together:

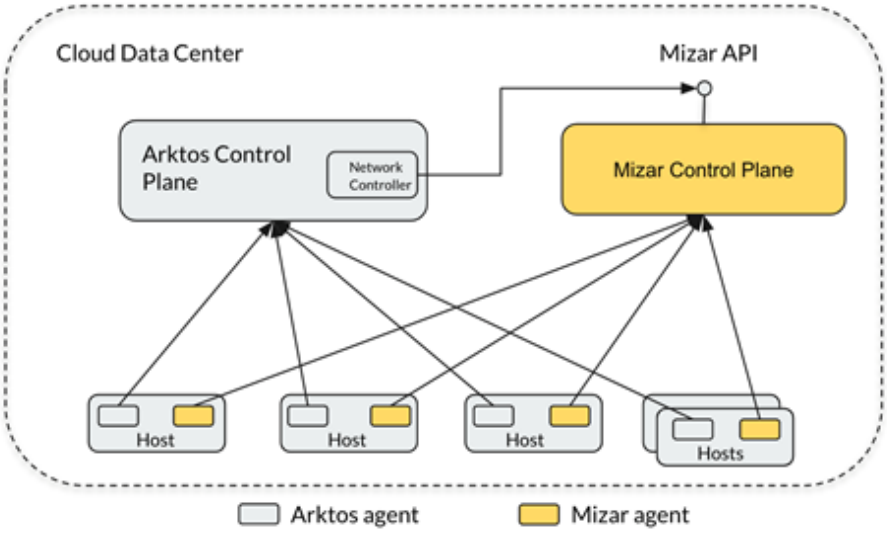

Figure 3.1: Interaction between Arktos and Mizar

Centaurus project also has a portal to manage Arktos and Mizar resources in a visualized way. The Centaurus portal can manage multiple Centaurus clusters located in different regions or availability zones (AZs).

The diagram below in **Figure 3.2** shows a more detailed version of how these components work together, and how they interact with external cloud services:

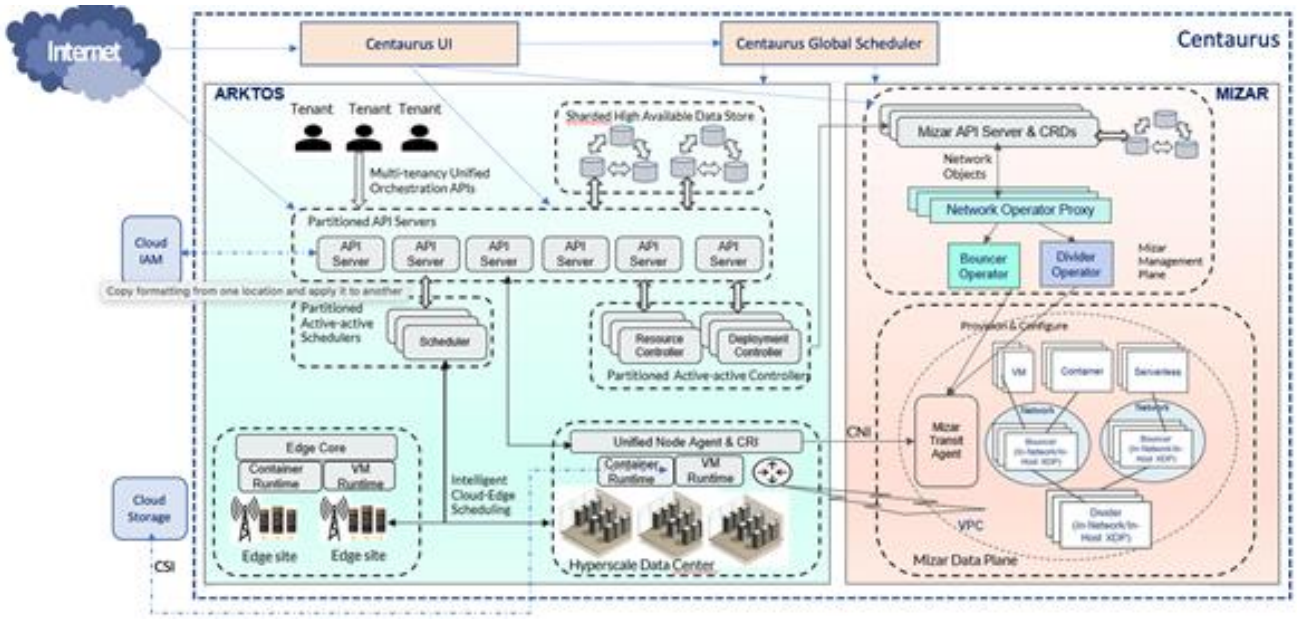

Figure 3.2: the overall architecture of Centaurus

Arktos and Mizar are two services designed in a decoupled way. Each of them has its own control plane and agents. Their agents are deployed on the host machine and talk to their corresponding control plane to manage compute or network resources on the host.

Please note that Arktos and Mizar are both designed as independent services. While they are usually deployed together to build a large-scale cloud platform, they can also be deployed in a standalone way and work with other compute and network solutions. Inside Arktos, there is a plugin module called “network controller”. By configuring different network providers Arktos can work with different network solutions.

3.2 Highlights

Compared to existing open-source technologies, Centaurus provides the following technology advantages:

- Single control plane for both containers and VMs.
- Single resource pool for both containers and VMs.
- Large scale regional control plane.
- Built-in strong multi-tenancy isolation
- Cross-AZ high available control plane.
- Fast VPC network endpoints provisioning and scaling.
- XDP-based efficient packet encapsulation and forwarding.

3.3 Centaurus Technical Goals

Scalability

Centaurus is designed to support up to 300,000 hosts per region, and up to 100,000 hosts per physical cluster, and support 10M+ network endpoints per region. On one physical cluster there can be up to 10,000,000 pods with the following performance constraint:

- 99 percentile API call latency is less than 1s.
- 99 percentile pod startup time is within 5s, including network endpoint provisioning time (not including image downloading time, which can vary depending on image size and image repo throughput capability).

Availability

All Centaurus APIs are provided as RESTful style for the best client compatibility. It's recommended to deploy Centaurus control-plane across multiple availability zones, and thus to be resilient to the failure of any single availability zone.

If a Centaurus cluster has hosts from different availability zones, Centaurus is able to deploy customer workloads in a cross-AZ way, and restart workloads in a different AZ if one of them AZs goes down.

Compatibility

All Centaurus APIs are provided as RESTful style for the best client compatibility. Centaurus is compatible with Container Storage Interface (CSI), and any storage service that provides a CSI driver can be deployed to the Centaurus cluster, and available to Centaurus workloads.

3.4 Reference Deployment

This section presents a reference deployment of a multi-region, high-available, large-scale Centaurus deployment. Let's first start with how Centaurus is deployed inside one region.

Inside a Region

A region consists of several AZs. Both compute and network in Centaurus provide a regional control plane. It's recommended to have a 3 AZ deployment for these two control planes. Because both Arktos and Mizar use quorum-based cluster metadata storage, it will be resilient from any single AZ failure. The below diagram (**Figure 3.3**) shows the logical view for a 3 AZ high-available deployment of Centaurus in one region:

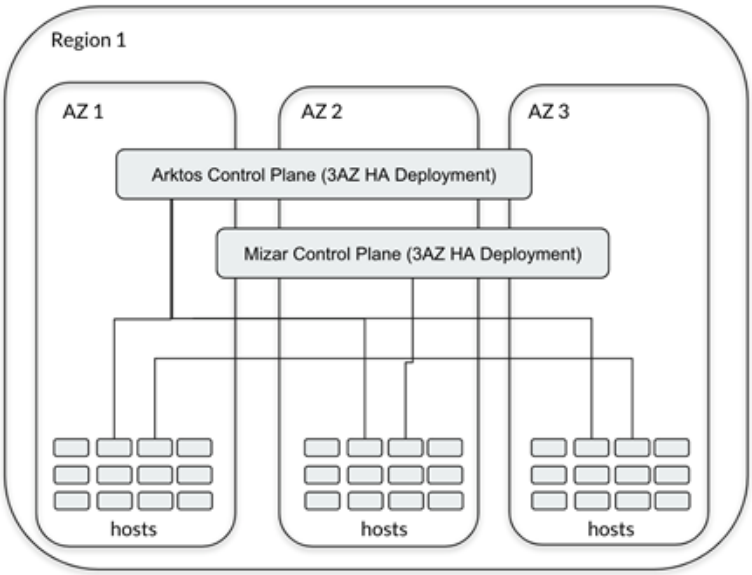

Figure 3.3: logical HA deployment of Centaurus inside one region

The diagram in **Figure 3.4** uses Arktos as an example to show the physical implementation of the above HA deployment. Because Arktos control plane is based on the same architecture, the same physical configuration applies to Mizar control plane.

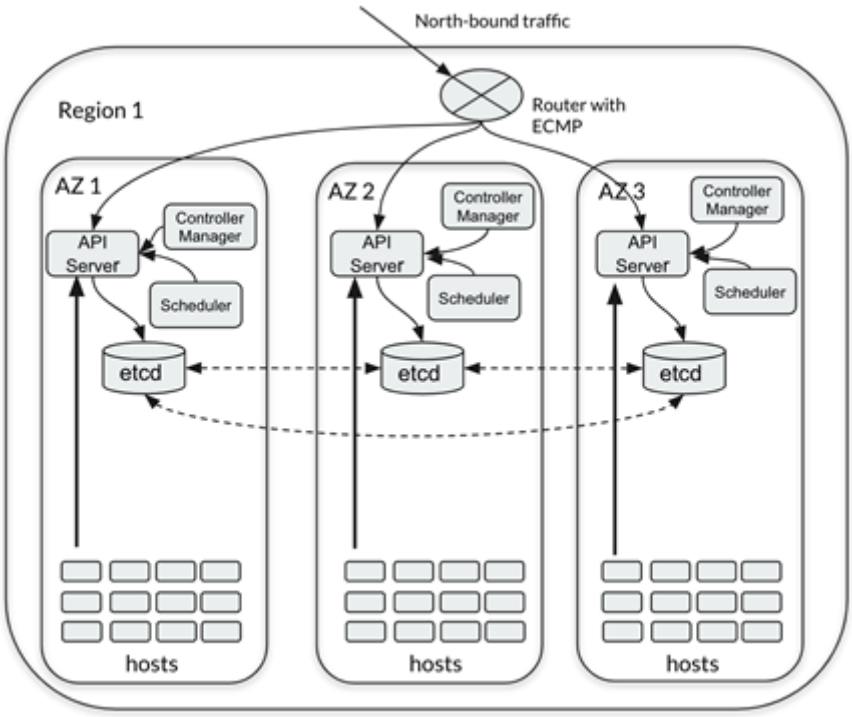

Figure 3.4: physical HA deployment of Centaurus inside one region

High-available deployment is not mandatory. Due to various limitations, 3AZ HA deployment may not always be available. In that case, control plane can be deployed in one AZ or two AZs. The combination can be very flexible. For example, Arktos control plane is deployed in 2 AZs and Mizar control plane is deployed in a single AZ. Of course, availability will be reduced with this deployment.

Please note that 2 AZ deployment doesn't necessarily increase Centaurus availability. Among the two AZs, the one which hosts more control plane nodes will be the major AZ, and the other AZ is the minor AZ. If the major AZ goes down, the entire control plane goes down. But if the minor AZ goes down, the control plane continues to work.

Multi-Region Cloud

After we have regional deployment, the diagram in **Figure 3.5** shows how to build a multi-region, high-available cloud infrastructure based on Centaurus.

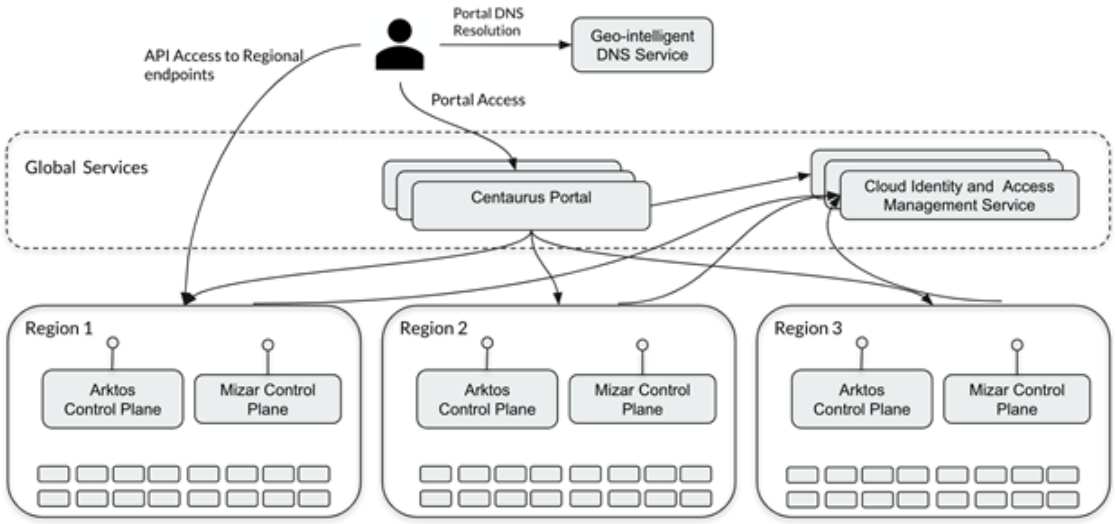

Figure 3.5: a multi-region deployment of Centaurus

Centaurus portal is a stateless service and multiple instances are deployed in a global layer. Physically they are just distributed in all regions. Each portal instance can talk to any regional Arktos/Mizar API endpoint. IAM services are synchronized among regions, so that different regions can share the same account system.

4. ARKTOS – AN OPEN COMPUTE INFRASTRUCTURE

4.1 Introduction

Arktos is an open source project designed for large scale cloud compute infrastructure. It evolved from the open source project Kubernetes codebase with core design changes. Arktos aims to be an open source solution to address key challenges of large-scale clouds, including system scalability, resource efficiency, edge computing, and the native support for the fast-growing modern workloads such as containers and serverless functions.

Specifically, the main goals for Arktos project include:

- Application-centric API: Compared to traditional cloud infrastructure that provides a resource-centric API (like VM), Centaurus provides a one-level-up abstraction API including deployment/job/replica Set/rolling update, health & readiness, etc.
- Container/VM unified infrastructure.
- Geographic-distributed scalability.

Below is a feature comparison to other similar systems:

Features	Arktos	Kubernetes	OpenStack Nova	Cloud IaaS
Container Management	X	X		
Virtual Machine	X		X	X
Multi Tenancy	X		X	X
Regional Scalability	X			X
Edge Integration	X			

Table 1: Arktos Feature Comparison

4.2 Arktos Architecture

The below diagram in **Figure 4.1** shows the architecture of Arktos:

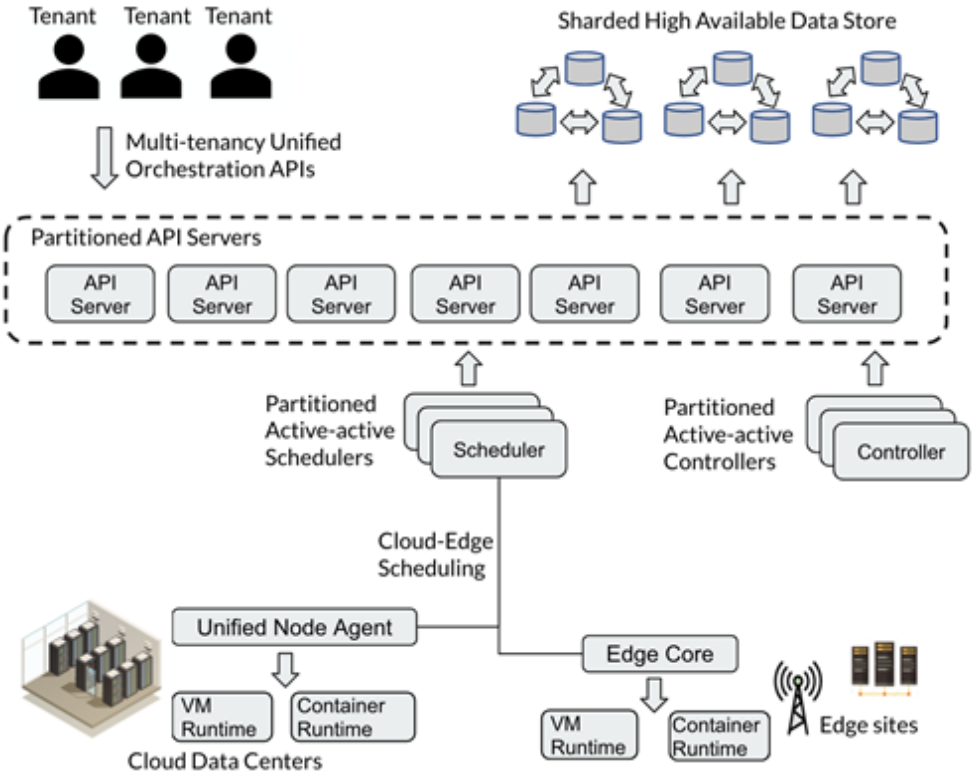

Figure 4.1: Arktos Architecture

Arktos inherits the Kubernetes architecture leveraging cloud native design principles, However, Arktos made extensive design changes and/or enhancements, which include:

- Sharded data store - a high-available and strong-consistent data store for cluster configuration and state. Data is sharded into multiple clusters based on variable criteria.

- Partitioned API Server - provides API access to both external and internal components. Since API Server caches data, API servers are also partitioned so that they can scale out when the cluster grows to an enormous size.

- Active-active controllers - unlike Kubernetes controllers that can have only one active instance at any given time, Arktos controllers work in an active-active way to scale and distribute workloads.

- Unified node agent - supports both container runtimes and VM runtimes in a pluggable style.

- Global scheduler: supports scheduling workloads among cloud data centers and edge sites.

4.3 Arktos APIs

In addition to inherited Kubernetes APIs such as nodes, pods or deployments (see https://kubernetes.io/docs/concepts/overview/kubernetes-api/ for full list of Kubernetes APIs), Arktos also supports the following new APIs:

- Tenants: /api/v1/tenants
- VM pod: /api/v1/tenants/{tenant}/namespaces/{namespace}/pod
- Actions for VMs: /api/v1/ tenants/{tenant}/namespaces/{namespace}/actions
- Network: /api/v1/ tenants/{tenant}/networks
- Cluster partition
- Data partition

4.4 Key Features

4.4.1 Unified Orchestration

With more and more containers running on bare-metals, today's cloud data centers have separate orchestration stacks for containers and VMs. For example, they might use VMWare or OpenStack to manage VM hosts and use Kubernetes to manage container hosts. Having two systems introduces lots of problems like separated resource pools, duplicated components, increased maintenance and operation cost, etc.

Arktos implements a native support of VM in addition to the mature container support inherited from Kubernetes. Now in Arktos a pod can be a container pod or a VM pod. The advantages of unified VM/container orchestration include:

- Users run containers and their VM workloads with the same APIs.
- Container orchestration paradigm, such as replicaSet or deployment, can also be used by VM applications.
- Cloud providers have a shared resource pool for both containers & VMs.
- Cloud providers only need to manage & operate one platform

The below screenshot of **Figure 4.2** shows a YAML sample for a VM pod and container pod.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: vm1
spec:
  virtualMachine:
    name: vm
    image: download.cirros-cloud.net/0.3.5/cirros-0.3.5-x86_64-disk.img
    resources:
        requests:
          cpu: "1"
          memory: "1Gi"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: container1
spec:
  containers:
    - name: container1
      image: ubuntu
      command: ["/bin/bash", "-ec", "while :; do echo '.'; sleep 5 ; done"]
      resources:
        requests:
          cpu: "1"
          memory: "1Gi"
```

Figure 4.2: Arktos Native VM support

4.4.2 Multi Tenancy

Arktos has built-in multi-tenancy support based on the idea of virtual cluster, which implements a hard-multi-tenancy model. All tenants are sharing a single physical cluster, but they all feel they are using the cluster exclusively. They are not aware of the existence of other tenants at all, as shown in **Figure 4.3**.

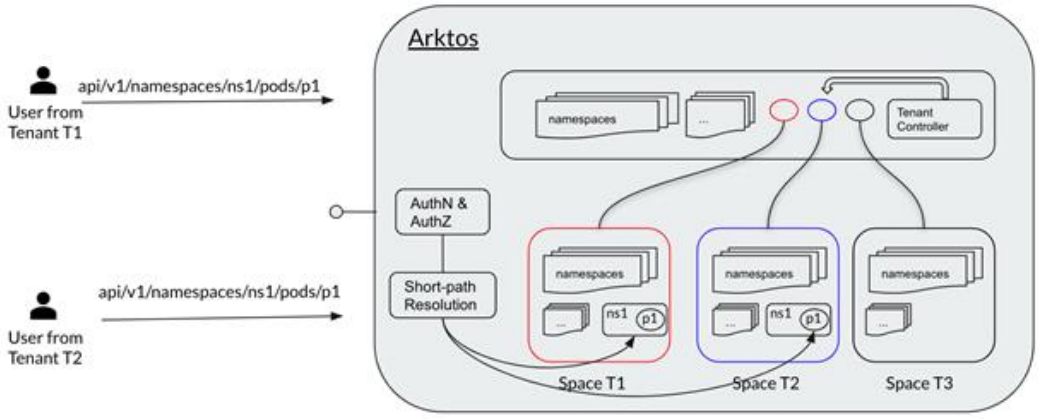

Figure 4.3 Arktos Multi-Tenant

Arktos achieves these based on a new concept "space". By default, a tenant is allocated a space with the same tenant name, and only that tenant and system tenant has access to the space. All API object visibility and interactions are restricted to the space.

By retrieving tenant credentials from API requests and automatically filling in the space information, we make spaces transparent to users. A tenant user accesses resources in its own space just like how he or she accesses Arktos without multi-tenancy. The APIs and tools are still compatible. Arktos multi-tenancy model has the following advantages:

- Strong isolation: tenants are strongly isolated from each other. They are not aware of the existence of other tenants at all. One tenant cannot access another tenant's resources or impact other tenants' performance. However, cross-tenant access is also supported if the authorization policy is explicitly set.

- Autonomy: tenant admins can perform management work within their own tenant without turning to cluster admins. Such work includes managing quota, roles, service accounts, installing CRDs, etc.

- Manageability: cluster operators can manage tenants in an easy way and are able to enforce some common policies across all tenants.

- Compatibility: customers can still use the existing tools and object manifests.

As a part of multi-tenancy, Arktos supports a fully isolated network among tenants. Usually this is implemented by a VPC network.

Arktos defines a new API object “network” to represent a fully isolated network. A network controller is a pluggable module that watches the network object and provides network connectivity. By deploying different network controllers Arktos can support different network providers, such as Mizar, Neutron, AWS VPC, etc.

The diagram in **Figure 4.4** below shows the component interaction when network controller for Mizar is deployed:

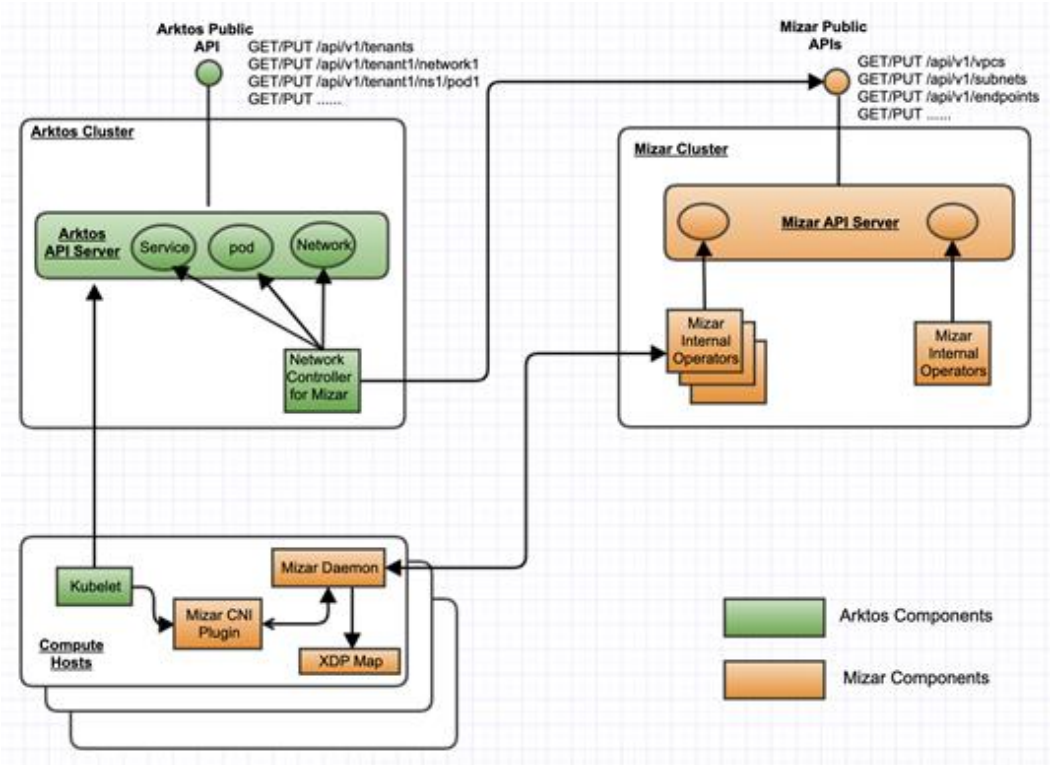

Figure 4.4: component diagram when network controller for Mizar is deployed

4.4.3 Cloud Scalability

Arktos provides a public cloud level of scalability. Our design aims to support 300K hosts per region and 100K hosts per cluster. From backend storage to frontend API server, all control plane components can scale out and are highly available:

- Data store support multiple clusters. Data is usually sharded by tenant ID, but it can be sharded by other criteria as well.
- API Servers are partitioned to different groups to overcome the cache limit of a single machine.
- Workloads are partitioned based on hash ID and handled by different controller instances.

The **Figure 4.5** below shows the implementation of sharding and partitioning mentioned above:

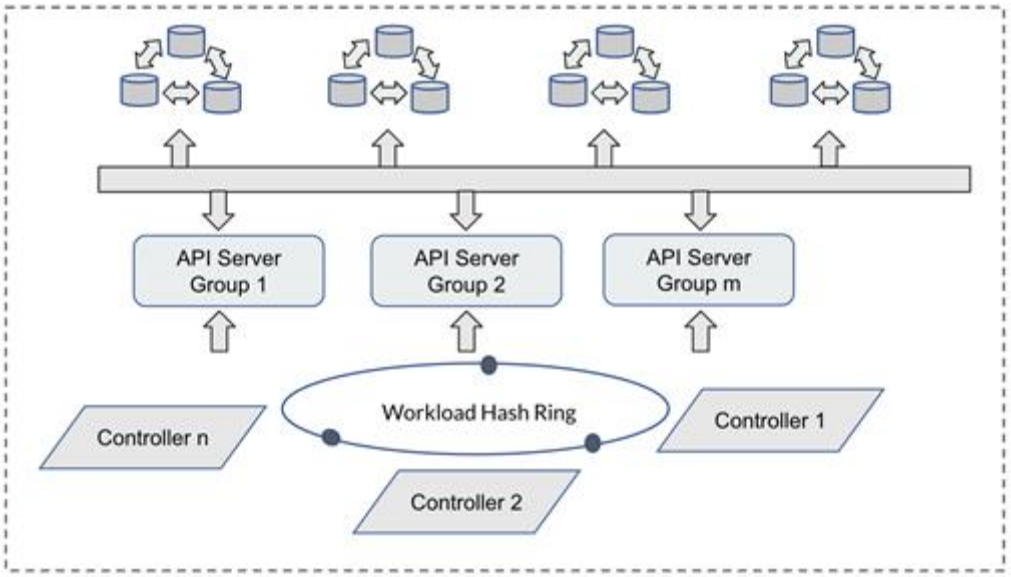

Figure 4.5: Data Shard and Workload Partition

Arktos Regional Control Plane

Arktos also supports partitioning on an even higher level. The entire regional control plane can be deployed as multiple “tenant partitions”. A tenant partition has its own API server, etcd, controllers and schedulers. It holds all workload resources owned by tenants, such as deployments, pods, jobs and configmaps.

All non-tenant resource objects (such as nodes) are distributed in various resource managers. A resource manager is also a cluster partition and has its own API Server, etcd, and some controllers. Usually we deploy one resource manager for each AZ. This resource manager has all the node objects for the AZ and serves heartbeat and node health check.

Please note that for each tenant partition, the scheduler still talks to all resource managers to make scheduling decisions. So scheduling is still across all hosts. This is also the key difference between one partitioned cluster and multiple clusters.

The diagram in **Figure 4.6** below shows the architecture of a partitioned regional control plane that consists of three tenant partitions:

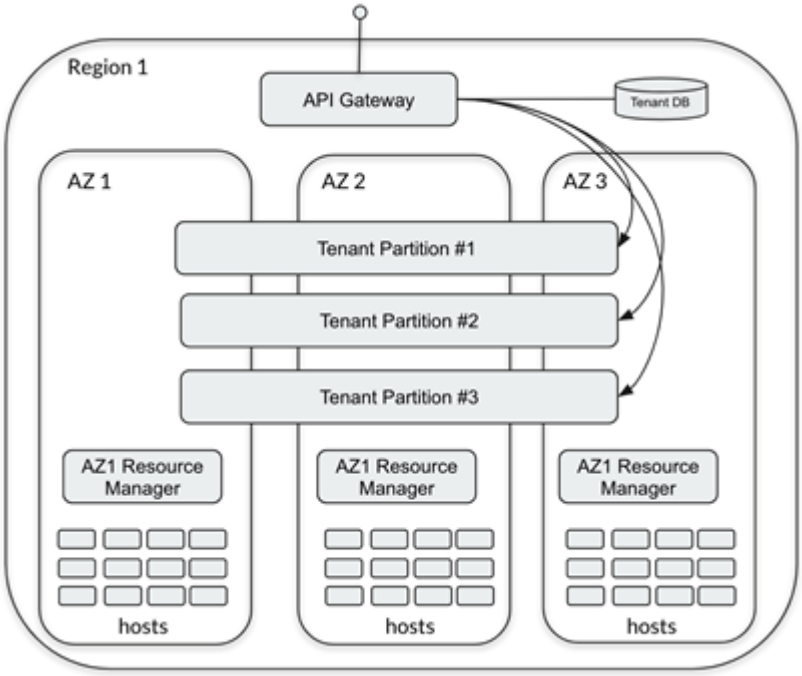

Figure 4.6: Arktos Partitioned Regional Control Plane

The advantages of Arktos partitioned regional control plane are:

- The whole control plane can scale out independently on two dimensions: tenant workloads and cluster resources. There is no single system partition in the entire system.
- Each tenant partition is a fault domain, and it's fully isolated from other tenant partitions.
- Resource scheduling is still global and across all hosts in the region. This ensures optimized workload scheduling instead of statically partitioning nodes.

4.4.4 Edge Computing

With rapid deployments of IoT devices and the arrival of 5G wireless, edge computing has become more and more important in the next cloud architecture. Arktos builds some core features to extend cloud computing to edge sites and provides a seamless integration between cloud computing and edge computing.

This section talks about two core features in Arktos for integration with Edge computing: cloud edge scheduling and cloud-edge communication.

Cloud-Edge Scheduling

In Arktos a component called “global scheduler” intelligently schedules resources across cloud data centers and edge sites.

The global scheduler serves the following purposes:

- Improve application QoS: schedule workloads on the edge sites that are closest to clients. Re-schedule workloads when traffic pattern changes or some edge sites are down.
- Improve resource utilization: coordinate resource utilization among cloud data centers and edge sites.

Global scheduler aims to provide a service that breaks resource boundaries, extends resource limitation, and optimizes resource utilization among cloud data centers and edge sites. The scheduler has a global view of all data centers’ available resource capacity and edge sites' available resource capacity, and intelligently places VMs/Containers on a DC/edge site that meets the VM/Container specification and achieves optimal global resource utilization.

In addition to that, the global scheduler monitors client traffic patterns for an application during runtime. It dynamically moves application instances to provide low latency and high QoS computing service to applications. For example, imagine an application that has few instances initially running in a cloud data center. After some time, the global scheduler finds out that a significant partition of client traffic is from a particular region. As long as there is available resource capacity in the edge sites in that region and the application deployment spec allows migration, the global scheduler will reduce the running instances in the cloud data center and deploy several instances on the edge sites in that region.

The below image in **Figure 4.7** shows the architecture of the global scheduler.

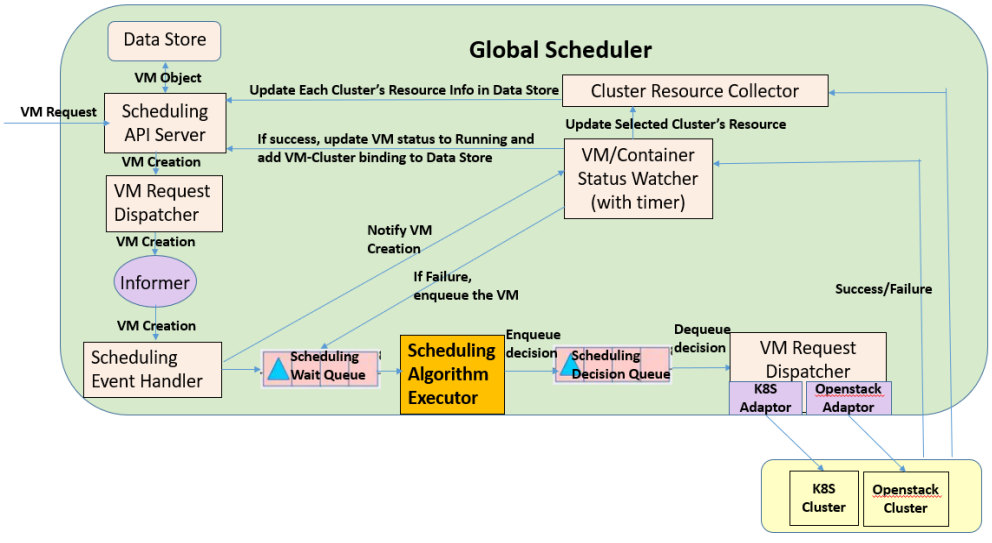

Figure 4.7: Architecture of Global Scheduler

Cloud-Edge Communication

Compared to cloud data centers, edge data centers usually have a less trusted physical environment. An edge data center might be a CDN data center, a small MEC site, or a private data center owned by some tenants. Access control and manage in these edge data centers are usually not as strict as it is in public cloud data centers. Therefore, these edge data centers are in a different trust zone.

Some control plane components need to run on the edge sites when we extend from cloud to edge. For example, we need to run network agents on edge nodes when we establish a VPC across cloud and edge. These edge-located agents need to communicate with their master components in the central cloud, just like other agents deployed in cloud data centers do. Currently they communicate with cloud-located master components through VPN network.

To solve the different trust level problem aforementioned, administrators need to carefully configure firewall and ACL rules to enforce the cross-boundary security protection. However, there are many problems when VPN is used to directly connect the components/services of different security levels:

- VPN exposes an attack surface larger than necessary.
- It is too risky to directly expose the central cloud management network/IP to users.
- IP needs to be planned globally in advance to ensure that the management components/service IP of the central cloud and each edge site do not conflict
- Network change configuration and management is difficult.

To solve these problems, Arktos introduces two new components: cloud gateway and edge gateway. See the diagram in **Figure 4.8** for the high-level overview:

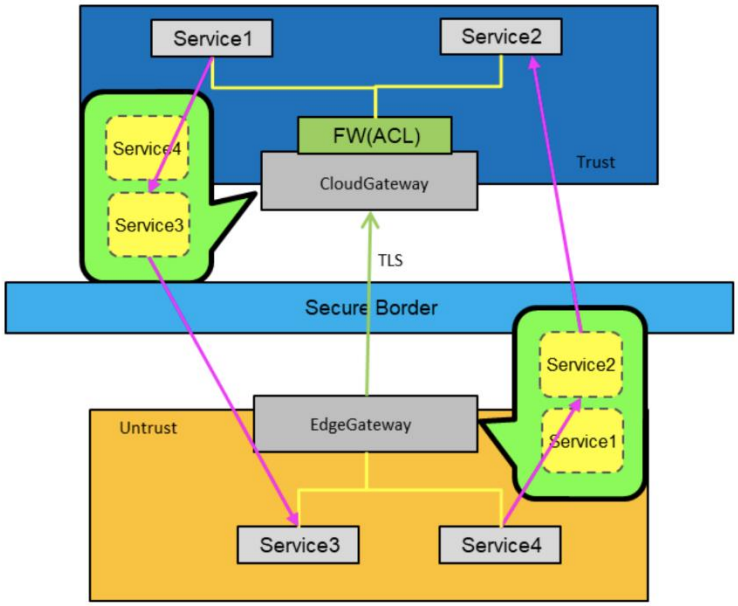

Figure 4.8: Cloud gateway and edge gateway in Arktos

Cloud gateway and edge gateway are deployed in the cloud data centers and edge sites respectively. They are designed from the beginning with different trust levels in mind. So only necessary services and communication ports are exposed.

When an edge-located control plane component or cloud service wants to access its cloud located counterpart, it doesn't need to talk to actual service endpoints directly. Instead, they talk to a restricted virtual presence of the service.

With this approach, Arktos provides a fine-grain controlled and easy-to-manage communication channel between cloud and edge when we offload services or control plane components to edge sites.

5. MIZAR – AN OPEN NETWORK INFRASTRUCTURE

5.1 Introduction

Mizar is an open-source high-performance cloud-network powered by eXpress Data Path (XDP) and Geneve protocol for highly scale cloud. It is a simple and efficient solution that lets you create a multi-tenant overlay network of a massive number of endpoints with extensible network functions that:

- Support provisioning and management of enormous number network endpoints
- Accelerate network resource provisioning for dynamic cloud environments
- Achieve high network throughput and low latency
- Create an extensible cloud-network of pluggable network functions
- Unify the network data-plane for containers and virtual machines, etc
- Isolate multi-tenant's traffic and address space

Mizar can be thought as a server-less platform for networking functions, in which developers extend it with capabilities without compromising performance or scale. The following diagram in

Figure 5.1 illustrates Mizar's high-level architecture.

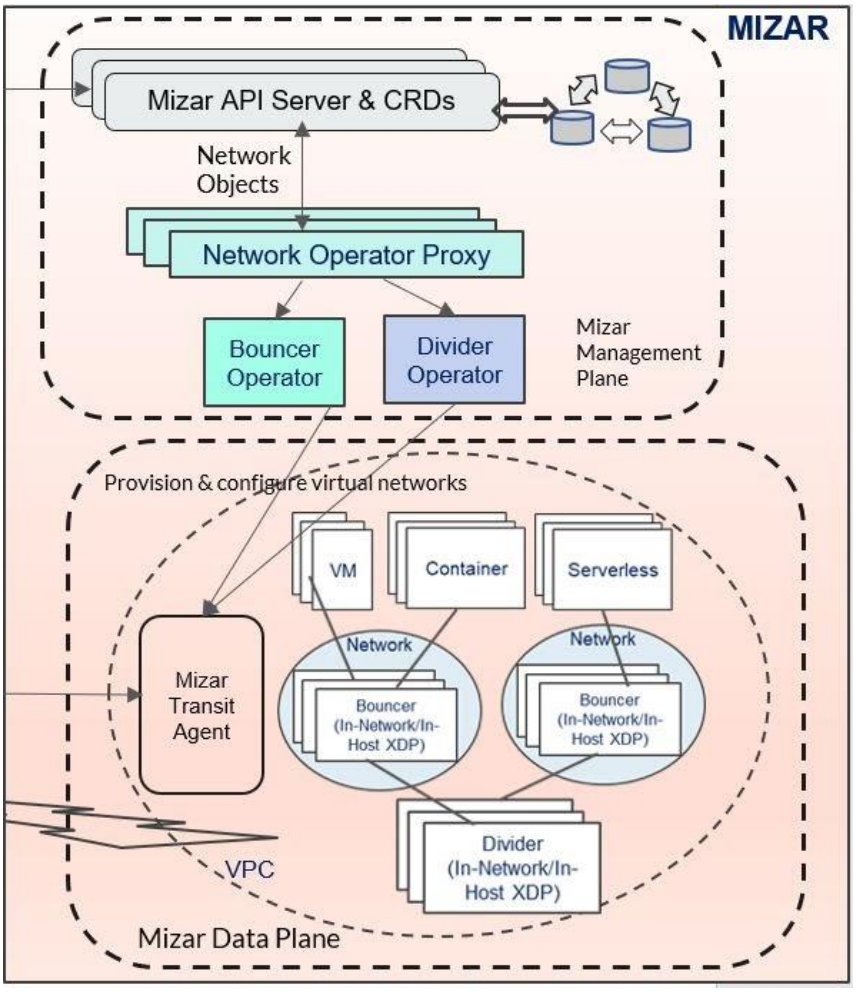

Figure 5.1: Mizar High Level Architecture

Mizar's data-plane is built from ground-up on top of XDP. Mizar's main building block is an XDP program that runs on each host. The program implements virtual functions including overlay switching, routing, virtual endpoints, load-balancing, NAT, etc. By running multiple instances of XDP programs per tenant network (VPC or Subnet) Mizar data plane provides high performance and extensible packet processing pipeline and functions that achieve Mizar's functional, scale, and performance goals.

Mizar's management-plane programs the data-plane by translating typical networking related APIs and resources to Mizar specific configuration. The programmability of the data-plane involves loading and unloading network functions at various stages of the packet processing pipeline.

Both Mizar data-plane and management-plane will be discussed in detail in the following sections of this whitepaper.

5.2 Why Mizar is Different

Unlike traditional networking solutions, Mizar relies on the natural partitioning of a cloud network to scale. Mizar simplifies the programming of data-plane to scale by flexible in-network processing, compared to flow-based programming models. As it primarily targets use cases of cloud-networking among virtual machines and containers, Mizar reduces the control-plane overhead of several routing and switching protocols within a cloud environment (e.g., L2 learning, ARP, BGP, etc.).

In Mizar architecture shown in Figure 5.1, Virtual Private Cloud (VPC) is a flat-network of endpoints specific to a single tenant, and a network within a VPC is a group of endpoints within a VPC. An operator may identify Networks as subnets of the VPC address space or any other partitioning scheme. Endpoints, a group of which forms a network, have IP addresses from the VPC address space.

Traditionally routing between VPCs and subnets is managed by virtual switches and routers. These mandates, for example, that endpoints belong to the same subnets, and a network of endpoints must have a subnet address. Mizar does not have this restriction. Mizar introduces new abstract components called Bouncers and Dividers. Bouncers and Dividers are in-network and horizontally scalable hash tables. The management-plane populates the Bouncers and Dividers tables according to network domain partitioning.

Bouncers' decision domain is constrained to a network. A Bouncer holds the configuration of endpoints within a network. When a packet arrives at a Bouncer, it is expected to find the destination endpoint's host and bounce the packet back to the host. Unlike a switch - where packet forwarding is performed by L2 learning - Bouncer's configuration maintains a mapping of an endpoint within a VPC to its host. The endpoint is identified by its IP address within a VPC (VNI). Bouncers rewrite the destination IP address of the outer packet to the endpoint's host.

Dividers' decision domain is constrained to VPCs. A Divider holds the configuration of all networks within a VPC; hence it divides (shards) the traffic inside the VPC across multiple bouncers. Dividers do not maintain endpoint-to-host mapping information. When a divider receives a packet, it determines which bouncer has the host information of the destination endpoint according to the network partitioning logic and rewrites the destination IP of the outer packet to the bouncer.

This overall architecture allows - among many advantages - to accelerate endpoints provisioning, as the management plane programs a finite number of hosts designated as Bouncers instead of propagating the endpoint configuration to each host.

5.3 Mizar Data Plane

Mizar's data-plane is the core engine behind its scalability, performance, and programmability. The data-plane is primarily a group of XDP programs.

Essential Concepts

Before diving deep into how the data-plane works, we need to introduce the essential architectural elements of Mizar.

- Droplet - A physical interface on a host with an IP to the substrate network. In the typical case where a host has a single interface, a droplet represents the host.

- Transit Agent - An XDP/ebpf program that processes an endpoint egress traffic. The transit agent is attached to the veth-pair of an endpoint in the root namespace.

- Transit XDP - The main XDP program that processes all ingress traffic to a droplet. The transit XDP program is extensible to support several data-plane functions, including interacting with user-space networking programs.

- Transit Daemon - A user-space program that interfaces with the management plane. The daemon primarily passes configuration to the Transit XDP and Transit Agent by updating ebpf maps. It provides RPC services to the management plane to pass these configurations. Additionally, the daemon provides services, including controlling kernel networking through the Netlink interface, programming other components on the host as needed (e.g., OpenVSwitch). Please refer to the management plane design for details on the Transit Daemon functionalities. Users can also interact directly with the transit daemon through a CLI program, which may run on the same host of the daemon or from a remote machine. The primary usage of the CLI program is for testing and troubleshooting.

The following diagram in **Figure 5.2** illustrates the architecture of a Mizar host.

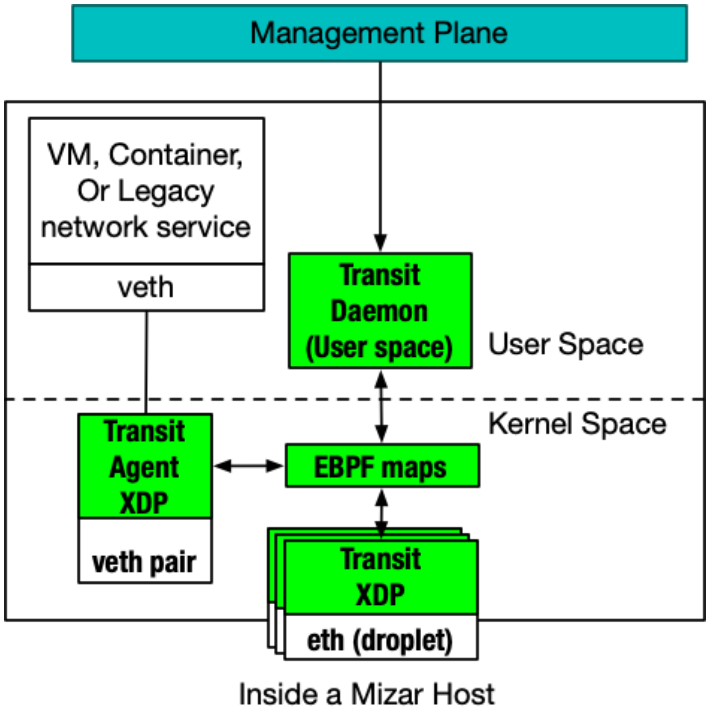

Figure 5.2: Architecture on a Mizar Host

Packet Forwarding (Ingress/Egress)

On the ingress path, the Transit XDP program determines if the host shall process the packet. The packet may be redirected to an endpoint (container/VM) or a user-space network function. Packets are redirected to the TXQ of the veth pair of the endpoint after decapsulation bypassing the host network stack in the root-namespace. Packets destined to a network function may be processed by another XDP program through a tail-call or by redirecting the packet to an AF_XDP socket. This approach has the advantage of making use of hardware offloading to processing packets entirely or in part. The Transit XDP programs and the transit daemon provide mechanisms to program the decisions of packet processing for network functions through RPC interfaces.

The transit agent processes all egress packets of an endpoint. TX packets from an endpoint are typically encapsulated and sent to another host. In that case, the transit agent redirects the packets to the main interface of the host configured in the transit agent's metadata. If the packet's destination is another endpoint on the same host, the transit agent simply invokes the Transit XDP program through a tail call, which redirects the packet to the veth interface of the destination endpoint (on the same host).

The **Figure 5.3** below illustrates typical packet forwarding in a Mizar droplet.

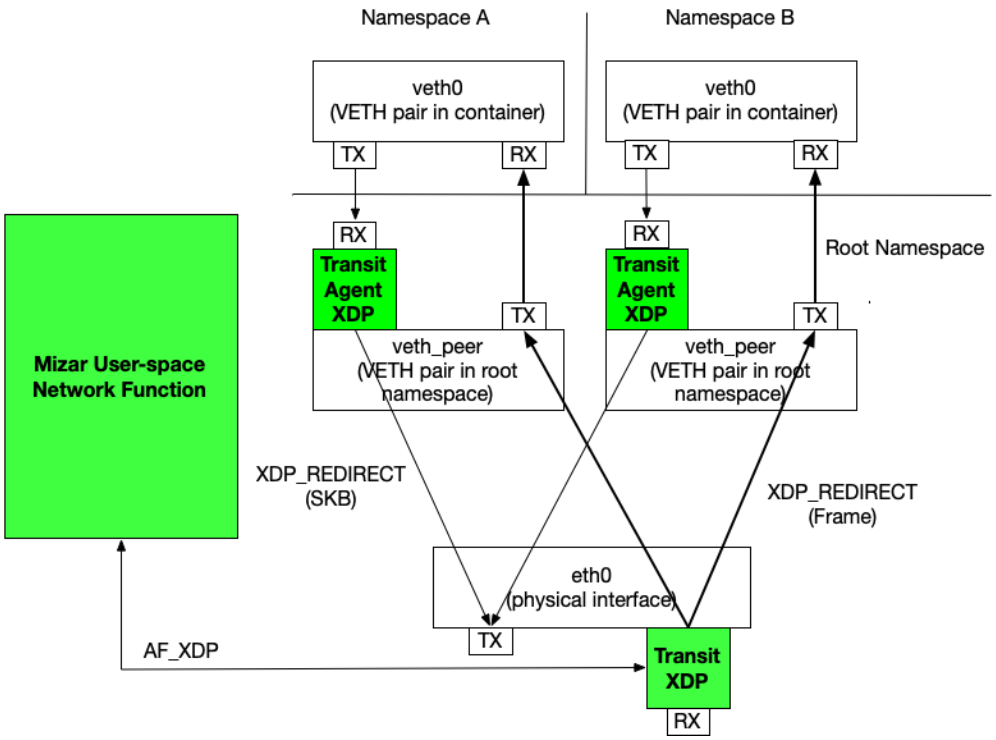

Figure 5.3: Packet Forwarding in a Mizar Droplet

5.4 Mizar Management Plane

Mizar's management-plane is based on Kubernetes operator framework. The management-plane is a fully distributed, elastically scaling, and kubernetes-native design. Several existing components allows this design to happen, which we reuse:

- Kubernetes Custom Resource Definition (CRD): Allows us to extend the K8s API server with networking objects. Some of these objects are generic to any networking solutions and some of them are specific to Mizar.

- Kubernetes Operator Framework: Operators Pattern helps extend Kubernetes with domain-specific operators that client APIs that act as a controller for CRDs. Operator frameworks allow us to write custom lightweight operators that derive the networking objects life cycle. We particularly use Kopf extensively as it allows us to easily refactor the test controller.

- LMDB: Provides a lightweight in-memory transactional database that is local to each Operator.

- Luigi: Facilitates developing, scheduling and executing workflows.

Object-Pipeline Architecture

The following diagram in **Figure 5.4** illustrates the overall architecture of the management plane. The main components of this architecture are objects and operators.

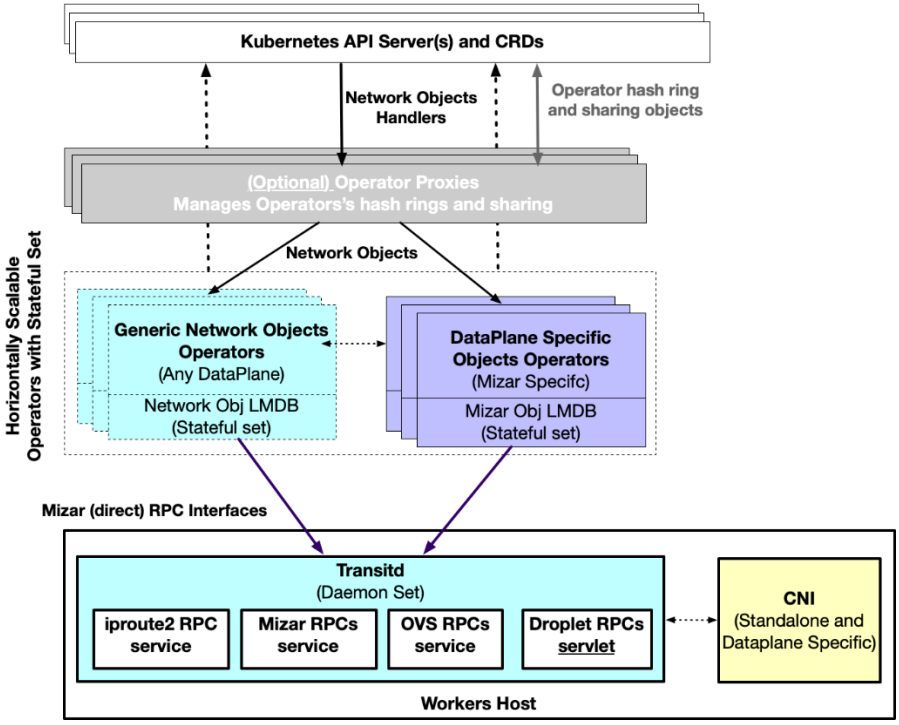

Figure 5.4: Architecture of Mizar Management-Plane

Objects are any networking object, of which the management plane derives their life cycle. When it's first created an object becomes in Init state. Multiple operators work together - in a distributed way - to evolve an object state to a Provisioned state. Between the Init and the Provisioned state, an Operator may transition the object's state to other intermediate states that are specific to the data-plane requirements. Once the object is Provisioned it implies that the data-plane is ready to function given that object semantics.

We refer to this process as the Object-pipeline, which is illustrated in the following diagram of a Generic Object Pipeline. We will explain several examples of the application of this approach to develop Mizar's workflows.

Operators program the data-plane through direct RPC interfaces exposed by Transitd (a daemon-set running in each worker host). These RPC interfaces allow the management plane to follow the well-known design pattern of having less number of management-plane clients and call a large number of servers to prevent failures to cascade between components. Also having

"One" daemon-set exposes multiple extensible RPC interfaces minimizes the number of agents on the hosts, improves deployment story, and reduces the chances of cycle-stealing from actual workload on the host.

We split Objects and Operators into two main categories: Generic, and Data-Plane specific. The Generic objects are necessary for any cloud-networking solutions such as: VPC (or Network in Neutron terminology), Network (or Subnet in Neutron), Endpoint (or Port in Neutron), And Droplet which specifies a host (Or a Worker node in Kubernetes). The other category of objects is those that are data-plane specific; For Mizar these are Bouncers and Dividers Objects. Each Object's must specify a "status" field that indicates the status of the object in the pipeline. Two statuses are necessary: Init, and Provisioned. In the PoC, the Management plane has four generic objects: Droplet, VPC, Network, and Endpoint. Each Object is specified using Custom Resource Definitions (CRDs).

Each Operator is responsible for maintaining the data of one and only one object (Stateful Object). We call this a Stateful Object from the operator perspective, since the Operator maintains its data. It is mandatory that an Operator updates the data for its Object when it is in the Provisioned state and optional in any other state. Operators may also mutate the state of objects other than its Stateful Object, but never store these objects data. We call these Objects Mutable objects from an Operator perspective. The Object types with respect to Operators will become clear as we detail Mizar's workflow.

The specification of an Operator's Stateful Object, Mutable Object, and Mutating Actions are what defines the management plane workflow.

Inter-Operators Communications

Operators communicate with each other to evolve the object's state. When an operator mutates the state of the object it invokes the next operator(s) in the object pipeline. The invocation is done through a callback mechanism. According to the deployment and specific implementation the callback is translated into one of multiple modes. In all these modes, the operator developer only focuses on the workflow through calling the callback function without worrying about the details of the communication implementation.

Through the API Server(s): In this mode, an operator only changes the object state and updates the object through the API Server. As one operator makes the callback to invoke the next step of the object pipeline, the only changes the object state and updates the object through the API server. As the next Operator in the pipeline handles the object in the new state, communication

happens implicitly. The drawback of this approach is the load on the API server, and the latency involved to evolve the object's pipeline. The advantage is that the object state is always tracked through the system.

Direct Service Communication: In this mode, operators expose a service interface through RPC or any other mechanism, and the callback effectively calls the RPC on the other operator. The calling operators must handle failure scenarios in such cases. The drawback of this approach is the complexity of handling failure scenarios. Also, as the intermediate states of the objects are not updated to the API server, it becomes harder to troubleshoot problems in the object-pipeline. This approach may have latency advantage, especially if the API server communication is a bottleneck. However, if the API server and kubernetes storage scales horizontally latency of this approach and the second approach may be comparable. In that case, this approach only adds implementation complexities that are hard to diagnose.

Simple Function Calls: In this mode, all operators are deployed into a single (horizontally scalable) operator. The functional implementation of the operators is still separated through call-back interfaces. But the callbacks are implemented as simply function calls. This approach is the simplest to develop, test, and diagnose. It also minimizes the latency since there is no network communication between the operators. Only the initial and final state of the objects must be updated through the API server. All other intermediate states are hidden. The drawback of this approach though is that local object store of the operator may have a huge size. The other drawback is similar to the second mode where the intermediate states of the objects are not available to the API server; hence, it is harder to troubleshoot and track the object state for problem diagnosis.

The drawbacks of the third approach are of a less impact on the development, performance, and operation. Not to mention simplicity. First, we can solve the local object store size problems by means of sharding through an operator proxy layer (which is a typical design pattern). The proxy layer shards the objects across replicas of the operator hence both reduce overload and local object store size. Second, the other drawback of tracking the object state for troubleshooting becomes less of a concern in the simple function calls mode because state transitions are within the same process with unified logging, and metering.

Local object store (LMDB) and Resumable Workflow

Each Operator will have a local K/V store implemented with LMDB. Each Operator stores its Stateful Object in this store when it reaches the state "Provisioned" and deletes it when the object is recycled (deleted). The objects are stored in their Provisioned state, there is no need

to develop a complicated cache synchronization mechanism as the object is cached into an final (immutable) state.

Given the operator being a Stateful Set, the store persists in the volume. When the operator is replaced it resumes the workflow by serving stalled objects in the API server (in Init or recorded intermediate state) utilizing the data of the provisioned objects in the local store. In this regard, the local store serves two purposes:

It caches the final object's data into its operator, hence reduces the need to frequently get the object through the API server.

The store accelerates the starting time of the operators after a failure.

Extension to an external store

The current implementation uses a simple hash map to cache the local state. However, the implementation is extensible if we find a need to interface with an external store other than that provided by Kubernetes.

Horizontal Scaling

Operators shall be deployed as stateful replica sets, where Kubernetes auto-scales the number of operators according to their resource metrics or custom metrics. To ensure that the horizontal scaling is effective, the operator proxy layer partitions the requests to be handled by each individual operator. The partitioning is by means of a typical hash ring, where the key is the VPC. Partitioning by the VPC ensures data locality of operators that relies on the function calls as inter-operator communications.

5.5 Some of Performance Test Results

The diagram in **Figure 5.5** below shows the packet rate per second. Mizar is able to achieve a near line rate of packet rate per second, with one bouncer or direct path from host to host.

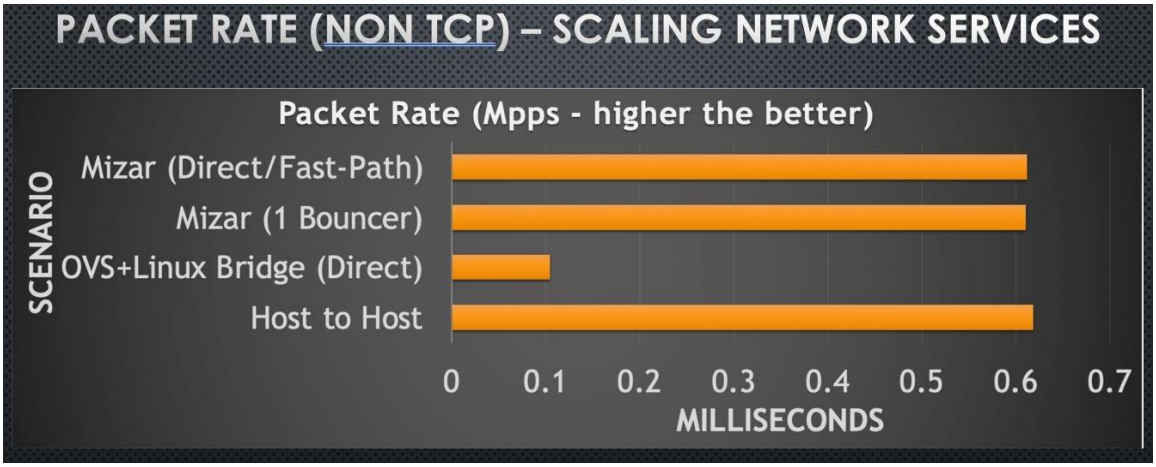

Figure 5.5: Packet Process Rate

Figure 5.6 shows the CPU usage on a host during the TCP performance test. Mizar has significantly less overhead during the provisioning of both endpoints and bouncers on the host.

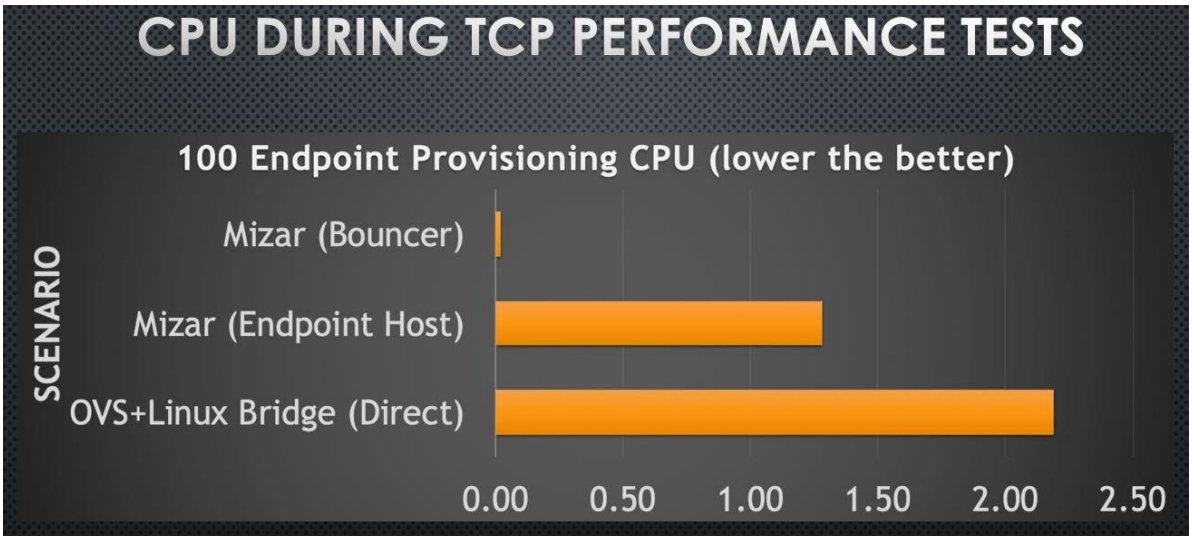

Figure 5.6: CPU Usage on a Mizar Host

The diagram in **Figure 5.7** shows the memory usage on a host during the TCP performance test. Again, Mizar solution causes negligible memory overhead. In fact, its memory usage is very close to an idle host without networking constructs even with Traffic processing.

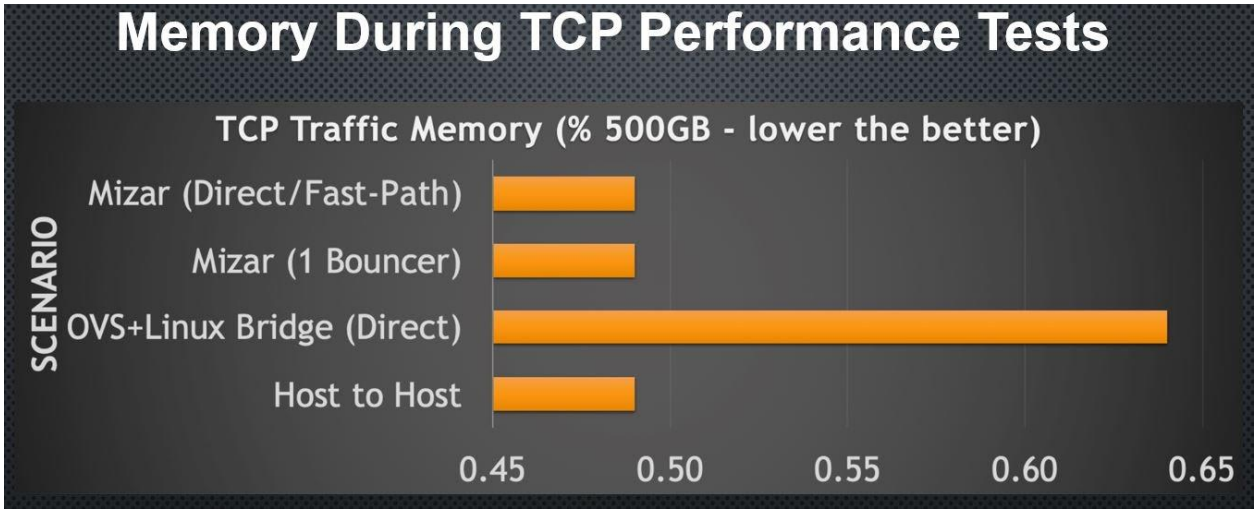

Figure 5.7: Memory Usage on a Mizar Host

6. CLOSING THOUGHTS

Digitization is upending many core tenets of competition among industries by lowering the cost of entering markets and providing high-speed passing lanes to scale up enterprises. As part of the ongoing digitization strategies, we believe there is a need for a hyper-scale and unified cloud infrastructure platform providing an option to build public or private cloud in open source communities for next generation applications.

Enterprises can leverage such a platform as the underlying cloud substrate for building their respective products & services as part of their ongoing digital modernization journey brought by the fast growing of 5G, AI, Edge, IoT technologies – today's hyper-scale is tomorrow's enterprise.

With open source community's participation and support, Centaurus platform has the potential to offer enterprises the hyper-scale and unified management capabilities which will dramatically change the economics of enterprise IT. It is still early days for Centaurus and we hope you can join us and make it a reality.

As a quick recap, Centaurus is an open source Distributed Cloud Native Infrastructure+ umbrella project for the 5G, AI, and Edge era. Centaurus currently includes the two core projects, a Compute project (Arktos) and a Networking project (Mizar). We would like to invite the open source community to join us to complete the Centaurus ecosystem.