Hyper-Converged Kubernetes

Contents

Executive Summary

Kubernetes is the de-facto standard for container orchestration for microservices and applications. However, enterprise adoption of big data and databases using containers and Kubernetes is hindered by multiple challenges such as complexity of persistent storage, network, and application lifecycle management. Kubernetes provides the agility and scale modern enterprises need. Although, it provides the building blocks for infrastructure, not a turnkey solution.

On the other hand, Hyper-converged Infrastructure (HCI) provides a turnkey solution by combining virtualized compute(hypervisor), storage, and network in a single system. It eliminates the complexity of integrating infrastructure components by providing an out of the box solution that runs enterprise applications.

We believe combining Kubernetes and the principles of HCI brings simplicity to Kubernetes and creates a turnkey solution for data-heavy workloads. Hyper-converged Kubernetes technology with built-in enterprise-grade container storage and flexible overlay networking extends Kubernetes' multi-cloud portability to big data, databases, and AI/ML.

Introducing: Hyper-Converged Kubernetes

What is hyper-convergence?

Hyper-converged Infrastructure is a software-defined IT framework that combines compute, storage, and networking in a single system. HCI virtualizes all components of the traditional hardware-defined IT infrastructure. Typically, HCI systems consist of a hypervisor for virtualized computing, a software-defined storage (SDS) component, and a software-defined networking (SDN) component.

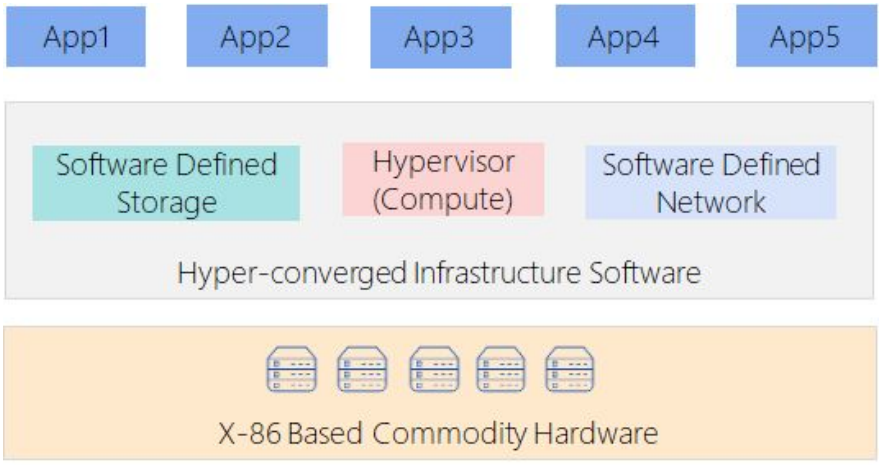

Figure 1: Hyper-converged Infrastructure

Hyper-converged Infrastructure software runs on X-86 based commodity hardware. It provides a complete environment for running enterprise applications, which means IT teams do not have to stitch together various pieces needed to to run the applications. All the required components are provided out of the box.

What is Kubernetes?

Kubernetes (also commonly referred to as K8s) is a container orchestration system that automates lifecycle operations such as deployment, scaling, and management for containerized applications. It was initially developed by Google, and later open-sourced. It is now managed by Cloud Native Computing Foundation (CNCF).

Kubernetes groups containers into logical units called “Pod”s. A pod is a collection of containers that belong together and should run on the same node. Kubernetes provides a Pod-centric management environment. It orchestrates compute, storage, and networking resources for workloads defined as Pods. Kubernetes can be used as a platform for containers, microservices, and private clouds.

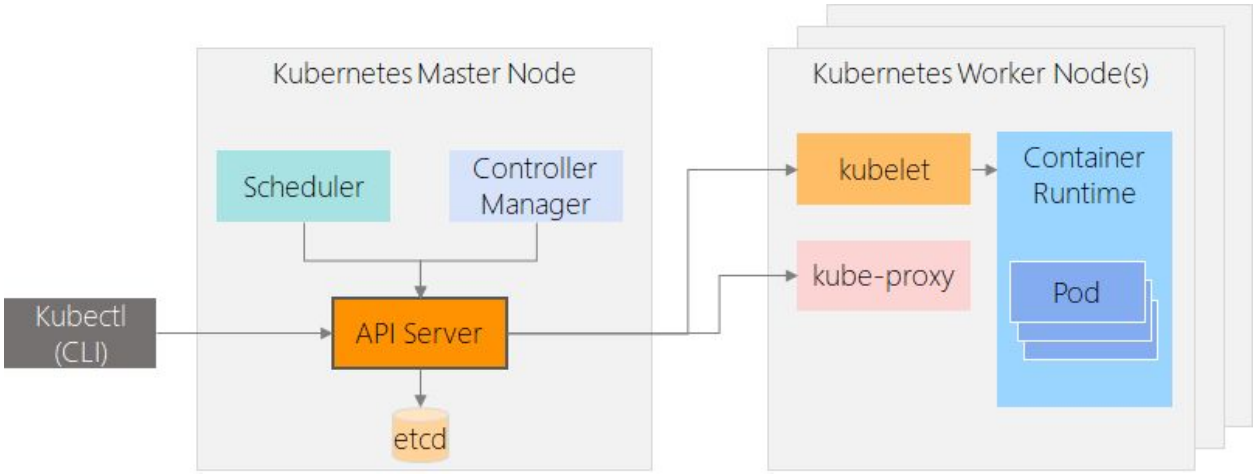

Figure 2: Kubernetes Architecture

Kubernetes is built using master-slave architecture common to most distributed computing systems. A K8s cluster consists of at least one master node and multiple worker nodes. Master nodes are responsible for running the master components that provide the cluster's control plane. Master components can run on multiple master nodes, but for simplicity, they are started on the same node by set up scripts. The designated master node does not run any user containers. On the slave side, Node components run on each worker node, and are responsible for maintaining running pods and providing the K8s runtime environment.

What is hyper-converged Kubernetes?

Hyper-converged Kubernetes is a software-defined application orchestration framework that combines containerized storage, networking, compute (Kubernetes), and the application management layer into a single system.

As the name suggests, we are bringing together the simplicity of the hyper-converged infrastructure and the agility of Kubernetes. HCI significantly simplifies IT operations by providing a turnkey infrastructure solution, but the technology is yet to fully adopt containers. On the other hand, Kubernetes is the de-facto standard for container orchestration, but it simply provides the building blocks for infrastructure. It is far from a turnkey solution.

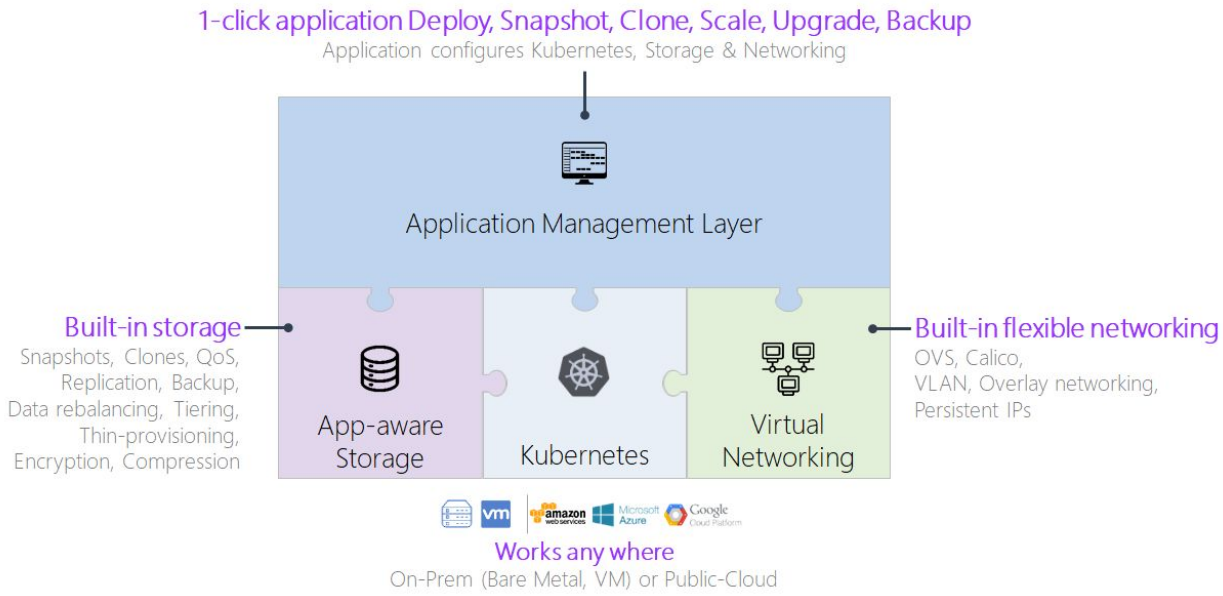

Figure 3: Hyper-converged Kubernetes Concept

The idea is to use the building blocks provided by Kubernetes to achieve a turnkey solution similar to hyper-converged infrastructure. All infrastructure components will be natively integrated in a cohesive system with Kubernetes at its core. The end result is a hyper-converged solution that is simple to use, while also providing the agility and scale of the container technology.

The need for hyper-converged Kubernetes

Kubernetes was initially developed with stateless applications in mind. A stateless application is a program that does not store data generated in one session for use in next session. No data is persisted. A common example is a web server. Every session counts as new. A stateful application on the other hand needs to persist data after interacting with a client. A common example is a database server. The data created during a session with a client needs to persisted by the database server for later use. The persisted data is available even if the database server fails.

The microservices based application paradigm suggests each application should be broken down into independent microservices and these microservices should interact with each other to deliver the end-to-end application. The paradigm dictates microservices should be stateless, i.e. independent of any data persistence. The stateful applications such as databases and big data were meant to run outside of Kubernetes. Only the application logic would run in microservices on Kubernetes.

As Kubernetes became mainstream, the demand for including stateful applications on Kubernetes led to development of StatefulSets and Persistent Volumes. These mechanisms enabled running stateful applications on Kubernetes.

However, there are the following challenges:

Big Data and databases are not built as microservices applications

Scaling is simple for stateless microservices - simply add more pods. For big data and databases, scaling isn't as simple as adding more pods, registering with a load balancer to spread load around. Databases are operated using built-in assumptions around storage and network state management, which fall apart during HA on Kubernetes. There is also a significant investment in custom scripting that assume SSH access to hosts running apps. It is important for the maintenance of databases. On Kubernetes, there no SSH access to pods, which creates complexity for database administrators who rely on SSH access to run maintenance scripts for backing up, patching and tuning the database.

Storage

Kubernetes supports storage for pods using a construct called Persistent Volumes.However, Kubernetes by itself doesn't ship with any storage or data management stack. An external storage stack must be used to provide the actual storage and data management capabilities for the Persistent Volumes. Because Kubernetes doesn't have its own storage stack there is no provision to ensure performance SLAs or IOPs guarantees when running multiple databases on the same Kubernetes cluster. Users cannot limit or guarantee IOPs for an application, giving rise to the noisy neighbour problem. Persistent Volumes also do not support data locality, affinity, anti-affinity, and isolation constraints. Handling applications that modify Root filesystem is also a challenge.

Networking

Pods are considered mortal in Kubernetes.When they die, they are not recreated. A new pod with the same configuration, but with a different identity is created instead. You can use the StatefulSet controller to create pods that will retain the hostname if they die, but their IP address changes. Most Big Data and Database applications predate both docker and kubernetes and are built with the assumption that the IP address assigned to them is preserved in the event of relocation or restarts. Several of them embed this IP address deep inside their configuration data. Therefore, not persistent IP address breaks these Big Data and Database applications. Exposing Big Data and Databases to other applications running on different L3 subnets in Kubernetes requires several steps including defining service endpoints and creating IP routing rules. This is both cumbersome and complex.

Benefits of hyper-converged Kubernetes

Hyper-converged Kubernetes (HCK) combines the benefits of HCI, containers, and Kubernetes.

Agility to adopt to business needs
Hyper-converged Kubernetes brings public cloud like flexibility to data centers. DevOps and IT teams can start with small deployments of, and as applications grow, they can add more resources. HCK runs on commodity hardware, making it easy to scale-out by adding commodity servers to existing deployments. This is in contrast to traditional IT infrastructure where scaling can be a challenge due to requirement of specialized hardware, licensing issues with multiple vendors etc.

Provision stateful applications in minutes
By providing all the pieces required to deploy any application (including databases and big data) out of the box, hyper-converged Kubernetes allows DevOps teams to provision applications quickly. Without HCK, users would have to integrate storage, networks, and Kubernetes, and carefully use Kubernetes primitives such as Persistent Volumes to reach a desired stage before deploying stateful applications. With HCK, all the plumbing is already taken care of.

Simplify application lifecycle operations
Native integration between Kubernetes, storage, network, and application management layer enables 1-click operations to scale, snapshot, clone, backup, migrate applications. Simplified operations allow DevOps teams to be more productive and improve time-to-market for new products and features.

Extend Kubernetes for data-intensive applications
Hyper-converged Kubernetes enables uses cases such as big data, databases, and distributed AI/ML on Kubernetes. These use cases need built-in storage and network to successfully run on Kubernetes. HCK enable a broad range data-intensive applications such as Hortonworks, Cloudera, Elastic stack, RDBMS, NoSQL etc.

Consolidate workloads without compromising SLAs
Consolidating workloads on shared infrastructure can lead to significant cost savings. However, consolidating stateful workloads on Kubernetes may give rise to noisy neighbour issue, because there is no mechanism to limit maximum IOPs or guarantee minimum IOPs. Hyper-converged Kubernetes provides mechanism to set minimum and maximum IOPs for applications running on shared infrastructure. As a result, organizations can now enjoy the cost savings through workload consolidation, without compromising SLAs.

Lower TCO
HCK systems run on commodity hardware, leading to cost saving compared to systems that need specialized hardware such as SAN. Also, simplified operations lead to cost saving on the IT administration side. HCK also eliminates the need for specialized software systems, resulting in cost savings on the software licensing front.

Faster time-to-market
Deploying applications on HCK is much easier due to simplified infrastructure. It saves valuable time at each stage of application lifecycle. As a result, DevOps teams are successful taking new applications and/or features to market faster.

Key Differentiating Features

Built-in application-aware storage

Hyper-converged Kubernetes comes with a built-in software-defined storage solution that is custom-built for providing enterprise class storage features on top of commodity direct attached disks. This storage stack is application-aware at the volume and disk management layers, which means that the storage stack exposes programmable primitives that allow controlling disk allocation logic and QoS from an application-context instead of worrying about per-volume level configuration. In simple terms, users can control disk allocation and IOPs for applications deployed through pods on Kubernetes.

Built-in networking with persistent IPs

The built-in networking solution in hyper-converged Kubernetes provides a mechanism to assign each pod a unique IP address. It also provides network routing to help pods communicate with each other or with external services, with minimum overhead. Data-heavy applications, such as Big Data, NoSQL, and RDBMS databases require high-performance network communication with minimal to no network translation overhead. Also, in the event of node failures when the pods running these applications are relocated to other healthy nodes, it is important to preserve the IP address of each relocated pod. Hyper-converged Kubernetes provides overlay networks to help preserve the IP addresses of the pods in the event of node failure.

Data locality constraints

Data-heavy applications such as Hortonworks, Cloudera etc. rely on data locality to provide high performance. They require the compute (pod) and its corresponding data to be located on the same node. Hyper-converged Kubernetes provides data locality to ensure distributed data platforms operate with high performance. The built-in storage exposes primitives that help users control disk allocation for pods (and in turn for services running inside the pods).

Advanced affinity and anti-affinity policies

Kubernetes provides mechanism for node affinity and inter-pod affinity using labels. However, implementing the affinity constraints using labels can be very complex. Moreover, some applications may need data level affinity and anti-affinity rules, defining which data can reside on the same disk, or on a node containing that disk. For example, Hadoop administrators may want to set up and anti-affinity rule that dictates the data for Zookeeper and Data Nodes should

not reside on the same disk. Hyper-converged Kubernetes provides advanced affinity and anti-affinity policies for nodes, pods, and disks with simple mechanism to enforce the policies.

QoS guarantee

The built-in storage in hyper-converged Kubernetes provides programmable primitives to control maximum and minimum IOPs for pods. The applications deployed as pods can be limited with maximum IOPs constraint. On the other hand, applications with high priority can be assigned minimum IOPs guarantee, making sure they perform as expected. This mechanism allows hyper-converged Kubernetes to eliminate the noisy neighbour problem.

Application and cluster level snapshots

Hyper-converged Kubernetes provides the ability to snapshot entire application including data, configuration, and cluster topology. Snapshotting an entire application enables restoring or cloning an entire application as-is. There is no need to restore individual storage volumes and remap them to their respective partitions/shards, etc. An entire application snapshot captures (1) application topology and its configuration, (2) snapshot of all data volumes, and (3) all placement and QoS policies.

Application cloning

With hyper-converged Kubernetes, one can clone an entire application cluster from a previously taken snapshot. Cloning creates a fully functional application cluster, say MongoDB, that can be used right away. Cloning applications goes beyond just cloning the storage volumes. An entire application (in this case the entire MongoDB cluster) is cloned resulting in a new MongoDB cluster to be spun up that matches the configuration and topology of the cluster when the snapshot was taken. The cloned MongoDB cluster gets a new network and cluster identify, but the data is the same from when the snapshot was taken. Cloning an entire application cluster is beneficial for running test/dev reports, UAT testing, and for testing patching and upgrades.

Implementing Hyper-converged Kubernetes

ROBIN reference architecture

ROBIN is the first implementation of the hyper-converged Kubernetes concept. ROBIN hyper-converged Kubernetes platform combines software-defined block storage, software-defined network, Kubernetes, and application management in a single system.

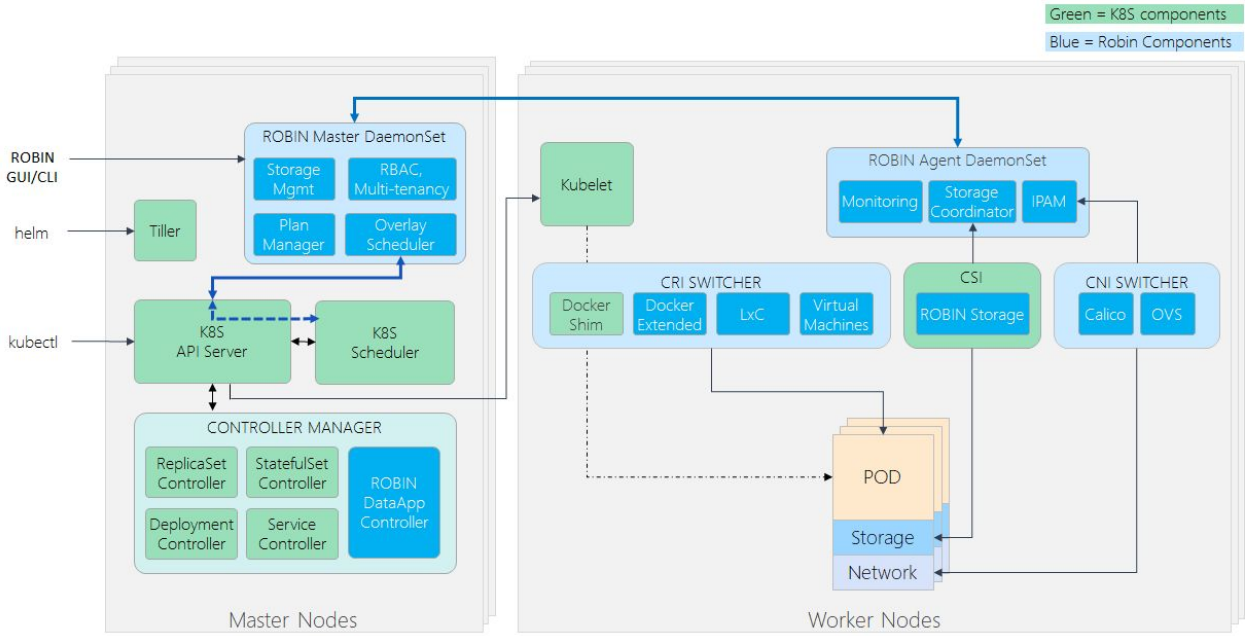

Figure 4: ROBIN Hyper-converged Kubernetes Platform Architecture

To implement Hyper-converged Kubernetes, ROBIN extends Kubernetes by adding components to master and worker nodes. These components are implemented as Daemonsets, which means they run on every node in the K8s cluster.

Key Kubernetes components

Master Components

- **API Server:** The API server exposes the Kubernetes API to the end user. It is the front end of all the REST commands used to control the K8s cluster.
- **etcd storage:** It is a distributed key-value store used for storing all cluster data.
- **Scheduler:** Scheduler watches for newly created pods that do not have node allocation. It is responsible for allocating nodes for pods. Scheduler takes into consideration all necessary requirements such as hardware resources, affinity policies, labels etc.
- **Controller-manager:** This component run all controllers for the K8s cluster. Controllers communicate with the API Server to learn what the desire state of the cluster is, and take corrective action if the current state of the cluster does not match the desired state.

Node Components

- **Kubelet:** This is an agent that runs on each node responsible for making sure pds are running and healthy. It gets the configuration for pods from the API Server master component and makes sure the pods are running with correct configuration.
- **Kube-proxy:** Acts as a network proxy and load balancer on every node. Takes care of network routing.
- **Container runtime:** Responsible for running containers (for example, Docker).

Beyond the basic architectural components, let's also take a look at components and subsystems in Kubernetes that are relevant to the HCK architecture:

Persistent Volumes
Persistent Volumes are storage resources that maintain data independent of any individual pods that use the storage resource. Persistent Volumes are created by administrators (can be created programmatically) and made available for the pods to attached to. Default storage in Kubernetes is ephemeral. If a pod dies, you lose related data. Persistent Volumes are meant to solve the problem. They persist data irrespective of pod health. As new pods are created to take place of the failed pods, they can attach themselves with the same Persistent Volumes the failed pods were attached to. This means the new pods maintain the application state. Pods attach to Persistent Volumes using "Persistent Volume Claims".

Container Storage Interface (CSI)
CSI provides standardized mechanism to expose arbitrary storage systems to the pods. Before CSI, volume plugins had to be included in the core Kubernetes binaries. CSI enables storage plugins to be developed independently, containerized, and exposed through standard Kubernetes primitives such as PersistentVolumes, PersistentVolumeClaims, StorageClasses etc.

Container Network Interface (CNI)
CNI contains specifications and libraries for writing plugins to configure network interfaces in Linux containers. It also contains a number of supported plugins such as Calico, Weave etc. CNI makes network layer pluggable for Kubernetes, allowing users to select a networking solution that fits their needs.

Container Runtime Interface (CRI)
CRI is a plugin interface that enable Kubernetes to use wide variety of container runtimes. Kubernetes uses Docker as the default container runtime, but users may need support for other runtimes. CRI solves the problem by providing an interface to plug in additional runtimes. It provides an abstraction layer between Kubelet (the component responsible for running pods) and the container runtimes, allowing Kubelet to choose relevant runtime for pods, without having to recompile the pods.

DaemonSets
DaemonSets are pods that run on every node. The provide an easy way to implement a runtime environment across the Kubernetes cluster.

Key ROBIN components

In addition to the core Kubernetes components, following components are essential to ROBIN:

ROBIN software-defined block storage
The ROBIN platform ships with a built-in software-defined storage stack that provides enterprise class storage on top of commodity direct attached disks.. The ROBIN storage stack exposes programmable primitives that allow controlling disk allocation logic and QoS from an application-context. ROBIN uses the Container Storage Interface (CSI) provided by Kubernetes to set its block storage as the default storage for the cluster.

ROBIN storage stack is designed for high-performance and high-scalability. Data can be spread across different disks on different nodes or can be stored on a single disk on a specific node. The exact disk allocation and data spreading logic is determined by the ROBIN Master services based on the application needs.

ROBIN software-defined network
ROBIN uses the industry standard Open vSwitch and Calico software-switching technology. Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols. ROBIN uses bridged-networking mode to ensure that pods communicate with the other pods and nodes. This ensures that network packets don't incur performance degradation due to network address translation (NAT) or IP-in-IP encapsulation overhead that one normally sees with other container platforms.

ROBIN uses overlay network mechanism to provide persistent IPs to all pods, so that pods can retain their IP addresses in the event of node failure. Persistent IP for pods is a critical requirement for hyper-converged Kubernetes.

ROBIN Master Daemonset
ROBIN Master manages the configuration, cluster topology, and runs the logic to place application pods on the relevant nodes. It is also the main administrative service of the platform. It exposes a single core REST API service that is used by the rest of the platform to programmatically configure the cluster and perform operations on applications. The REST API is served over secure SSL-encrypted HTTPS sessions. At any given time, there is a single ROBIN Master service that runs in the entire cluster. To prevent single point of failure the service can be configured to failover to a total of 3 nodes.

ROBIN Master exposes a REST API and a GUI to end users to help them deploy and manage applications. When a deployment request is made, the **Plan Manager** service intercepts the

request, creates a detailed deployment plan taking into consideration resource requirements and affinity/anti-affinity policies for each pod, and submits it to the **Overlay Scheduler** service. Overlay Scheduler then creates a simple configuration file for the default scheduler of Kubernetes. ROBIN Master also runs a **Storage Management** service that holds information about all storage disks in the cluster. It also runs the allocation logic to provide storage volumes for the applications orchestrated by ROBIN.

ROBIN Agent Daemonset
ROBIN Agent runs on every node in the ROBIN cluster. It is a task executor for the ROBIN Master service. When ROBIN Master wants to execute a particular task on a node, it communicates with the ROBIN Agent on that node and instructs it to perform that operation. Communication happens over an SSL-encrypted HTTPs channel.

The ROBIN agent includes services for monitoring, storage coordination, and IP address management.

ROBIN DataApp Controller
ROBIN DataApp Controller is runs inside the Controller Manager on the master node. This controller listens for configuration requests for the custom resource called “DataApp”. It is responsible for deploying and maintaining pods that belong to “DataApp”. DataApp is Custom Resource Definition (CRD) created by ROBIN.

CRI Switcher
CRI switcher is responsible for selecting the appropriate container runtime based on the type of application. ROBIN supports multiple container runtimes such as Docker, LxC, and plans to support virtual machines.

CNI Switcher
CNI Switcher is responsible for selecting the appropriate network plugin. ROBIN supports Open vSwitch and Calico.

How it works

To understand how ROBIN hyper-converged Kubernetes works, for the sake of simplicity, let us consider application deployment as a 3-step process.

1. Creating pods and assigning them to nodes

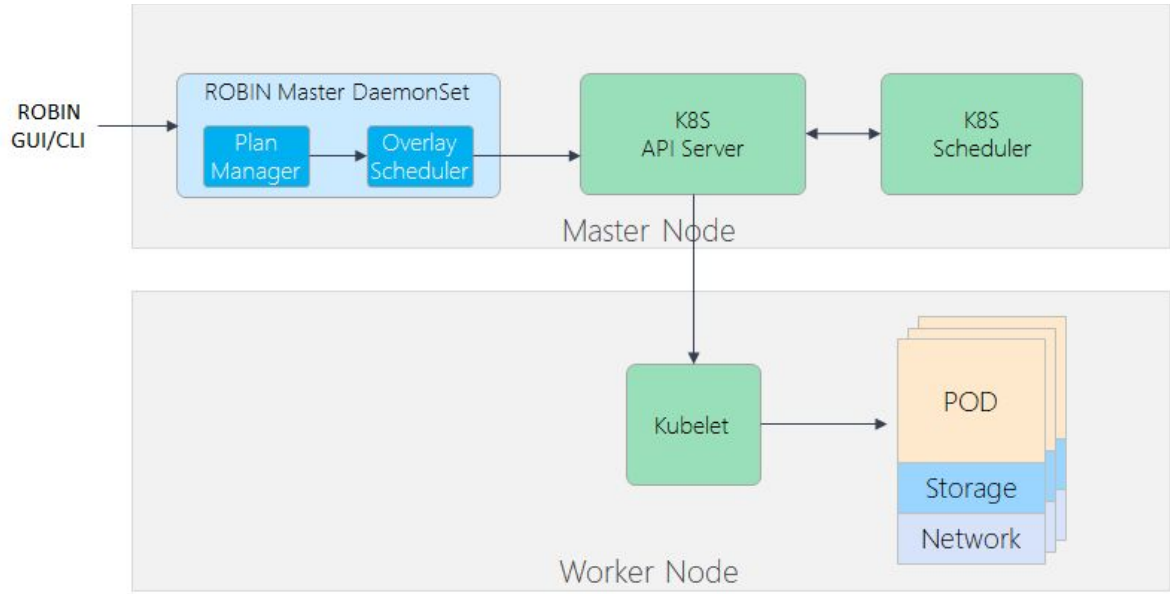

Figure 5: Pod creation workflow

When an application deployment request is made through ROBIN CLI or GUI, the Plan Manager service accepts the request. The Plan Manager considers application requirements, decides how many pods are needed, takes into account affinity and anti-affinity policies, and submits a detailed plan to the Overlay Scheduler service.

The Overlay Scheduler service is aware of all available resources in the cluster. Based on resource availability on nodes, resource requirements, and affinity policies for pods defined in the plan, it maps pods to nodes. Based on the mapping, the Overlay Scheduler creates a manifest that enforces strict node affinity for pods. The default Kubernetes Scheduler then uses the manifest to schedule pods on the nodes as defined in the manifest. ROBIN uses this process to control the pod creation and allocation process without disrupting the K8s workflow, by letting the default K8s scheduler schedule the pods.

The K8s Scheduler notifies the API Server that it has scheuled pods for creation. The API server notifies Kubelets on corresponding worker nodes, which create the pods.

2. Creating persistent storage volumes and assigning them to pods

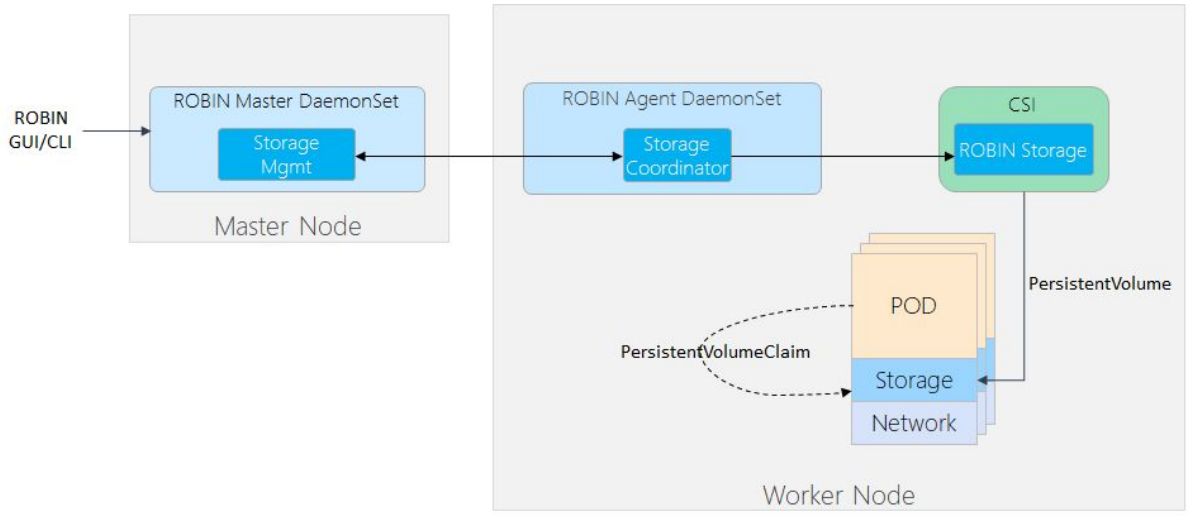

Figure 6: Persistent storage assignment workflow

The ROBIN Overlay Scheduler service, along with creating the schedule for pod allocation, also creates the schedule for assigning Persistent Volumes to pods based on the data locality and affinity policies defined for the application. The Storage Management service is responsible for taking the Persistent Volumes allocation information, and relaying it to the Storage Coordinator service running inside the ROBIN Agent on the corresponding worker nodes where the relevant disks are located.

On the worker node, the Storage Coordinator service uses K8s CSI to interact with the ROBIN built-in storage to create a Persistent Volume. It also creates a Persistent Volume Claim, which is used by the associated pod to attach the Persistent Volume to itself.

3. Assigning persistent IP addresses to pods

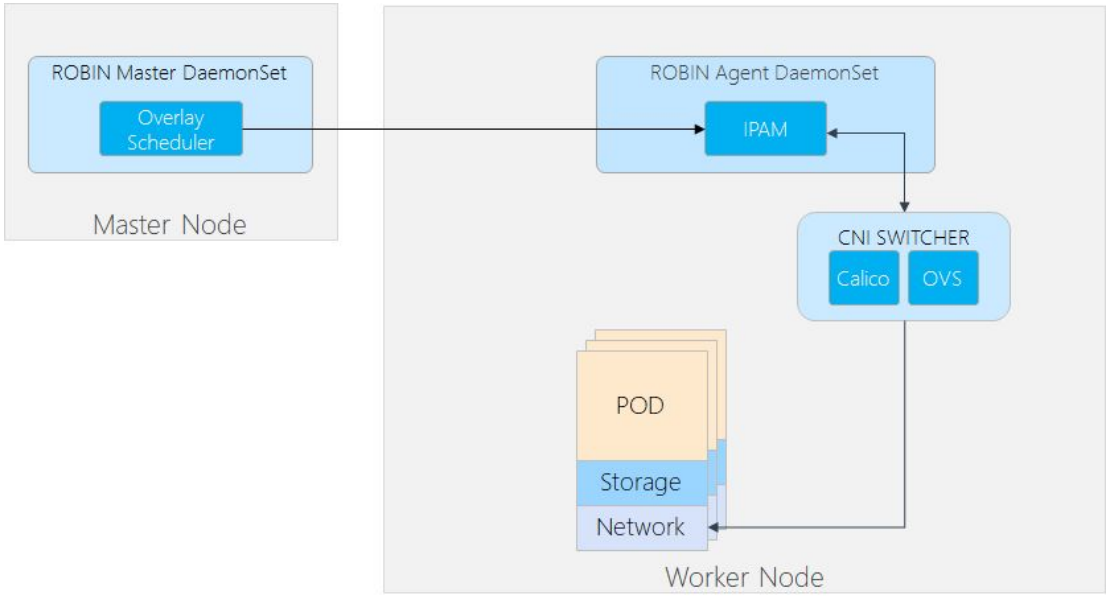

Figure 7: Assigning persistent IP addresses to pods

ROBIN extends Kubernetes networking via both Calico and Open vSwitch based CNI drivers. This offers flexibility in using either overlay networks to create flexible L3 subnets that span multiple datacenters or cloud environments or use bridged networking to get wire-speed network access for high performance applications. In both modes ROBIN also enhances the CNI driver to retain the IP address of the pod when it is restarted or moved from one host to another. This provides greater flexibility during scaling, migration and high availability.

Many Big Data and Database applications predate both Docker and Kubernetes and have made strong assumptions around how network and storage persistency is preserved in the event of pod restarts. ROBIN's handling of both network and storage ensures that these applications function correctly when running on a Kubernetes cluster.

Key Use Cases

Big Data

Big data platforms and ecosystem components such as Hortonworks, Cloudera, Kafka, Spark, HBase, Tensorflow, Druid and others are popular tools used in modern data analytics and ML/AI projects. However, deploying these apps typically starts with weeks of careful infrastructure planning to ensure good performance, ability to scale to meet anticipated growth and continued fault tolerance, as well as high availability of services. Post deployment, the rigidity of the infrastructure poses operational challenges in adjusting resources to meet changing needs, patching, upgrades, and performance tuning of the analytics apps.

ROBIN takes an innovative new approach with its implementation of hyper-converged Kubernetes, where developers and IT teams just describe the needs of their data intensive analytics apps without worrying about the assembly of the underlying infrastructure to host them. Hyper-converged Kubernetes combines the benefits of containers, a built-in application-aware storage stack, built-in software-defined network, and an application-aware workflow manager to provide an app-store like experience where complex applications can be deployed in minutes and their lifecycle managed through radically simplified 1-click operations.

Key benefits for using hyper-converged Kubernetes for big data include:

- Slashing deployment and management times from weeks to minutes
- Sharing resources among multiple big data clusters while guaranteeing isolation
- Dynamically scaling resources to meet changing needs
- Decoupling compute and storage and scaling them independently

Databases

Relational databases such as Oracle, DB2, SQLServer etc. and NoSQL databases such as MongoDB, Cassandra, Couchbase etc. are key components of the application stack. Deploying these databases requires careful planning and performance tuning. Also, once deployed, scaling, upgrades, and cloning are day-to-day challenges DBAs face.

The ROBIN hyper-converged Kubernetes provides an app-store experience where DevOps teams can provision databases within minutes using a 1-click operation. Once deployed, ROBIN also provides simple 1-click operations for scaling-up, scaling-out, patching, upgrading, cloning, and migrating.

Key benefits of using hyper-converged Kubernetes for databases include:

- Simplify lifecycle operations
- Improve collaboration between multiple teams using the cloning feature
- Share infrastructure resources without compromising SLAs

Distributed AI/ML

AI/ML is an upcoming field that offers advanced analytics techniques to uncover new insights. Many enterprises are in the midst of building their AI/ML infrastructure. However, with many libraries to choose from, and the need to experiment with multiple libraries while building a model, deploying and maintaining multiple AI/ML clusters consume a large amount of time, eating into the productivity of data scientists. Also, for optimum performance, these clusters need GPUs, which are fairly expensive. The IT department either needs to keep a close watch on the consumption of these expensive resources, which creates delays, or they need to create quotas.

ROBIN hyper-converged Kubernetes provides data scientists the ability to deploy distributed AI/ML clusters with their favorite libraries using 1-click. Data scientists can define how many GPUs, CPUs, and RAM their new deployment would need, and as long as they are within their quota, ROBIN takes care of the deployment for them. They can also scale-up or scale-out if they find the performance of their algorithm is too slow and they need more resources.

Key benefits of using hyper-converged Kubernetes for AI/ML include:

- Self-service provisioning of AI/ML clusters in minutes, making data scientists more productive
- Creating quotas for expensive GPU resources (as well as other resources)
- Dynamically scaling resources to meet changing needs

Conclusion

Hyper-converged Kubernetes brings together the simplicity of the hyper-converged infrastructure and the agility of Kubernetes. Kubernetes is the de-facto standard for container orchestration, however, there are many challenges for adopting big data and databases on Kubernetes. Hyper-converged Kubernetes technology with built-in enterprise-grade container storage and flexible overlay networking, eliminates these challenges and extends Kubernetes' multi-cloud portability to big data, databases, and AI/ML.

ROBIN SYSTEMS | 224 AIRPORT PKWY SAN JOSE CA 95110 | (408) 216-0769 | ROBINSYSTEMS.COM