

# Computation Support in DROP

**v5.00** 10 August 2020

**Kubernetes**

**Overview**

1. Focus of the Kubernetes System: *Kubernetes* is an open-source container orchestration system for automating computer application deployment, scaling, and management (Garrison (2016), Wikipedia (2020)).
2. Objectives of the System Design: It aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. It works with a range of container tools, including Docker.
3. Vendor Service Deployments using Kubernetes: Many closed services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform providing service. Many vendors also provide their own branded Kubernetes distributions.

**Kubernetes Objects**

1. Purpose behind the Kubernetes Objects: Kubernetes maintains a set of building blocks or primitives which collectively provide mechanisms that deploy, maintain, and scale applications based on CPU, memory, or custom metrics (Sharma (2017)).
2. Structure and Extensibility of Kubernetes: Kubernetes is loosely coupled and extensible to meet different workloads. The extensibility is provided in large part by the Kubernetes API, which is used by internal components as well as extensions and containers that run on Kubernetes. The platform exerts its control over compute and storage resources by defining resources as Objects, which can then be managed as such.

**Pods**

1. What is a Kubernetes Pod? A *pod* is a higher level of abstraction grouping containerized components. A pod consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources. The basic scheduling unit in Kubernetes is a *pod*.
2. IP Addresses for Kubernetes Pods: Each pod in Kubernetes is assigned a unique *Pod IP address* within the cluster, which allows applications to use ports without the risk of conflict (Langemak (2015a)). Within the pod, all containers can reference each other on localhost, but a container within one pod has no way of directly addressing another container within another pod; for that, it has to use the pod IP address.
3. Name Service to Reference Pods: An application developer should never use the pod IP address though, to reference/invoke a capability in another pod, as pod IP addresses are ephemeral – the specific pod that they are addressing may be assigned to another pod IP address on re-start. Instead, they should use a reference to a Service, which holds a reference to the target pod at the specific pod IP address.
4. Local/Network Disk Volume Pods: A pod can define a Volume, such as a local disk directory or a network disk, and expose it to the containers in the pod (Strachan (2015a)). Pods can be managed manually through the Kubernetes API, or their management can be delegated to a controller.
5. Storage for ConfigMaps and Secrets: Such volumes are also the basis for Kubernetes features of ConfigMaps – which provide access to configuration through the file-system visible to the container – and Secrets – which provide access to credentials required to access remote resources securely by providing those credentials on the filesystem visible only to authorized containers.

**ReplicaSets**

1. Purpose of the ReplicaSet: A ReplicaSet’s purpose is to maintain a stable set of replica pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical pods.
2. Selector Based ReplicaSet Specification: The ReplicaSets can also be said to be a grouping mechanism that lets Kubernetes maintain the number of instances that have been declared for a given pod. The definition of ReplicaSet uses a selector whose evaluation will result in identifying all pods that are associated with it.

**Services**

1. Definition of a Kubernetes Service: A Kubernetes source is a set of pods that work together, such as one-tier of a multi-tier application. The set of pods that constitute a service is defined by a label selector.
2. Modes of Service Discovery in Kubernetes: Kubernetes provides two modes of service discovery, using environmental variables of Kubernetes DNS. Service discovery assigns a stable IP address and DNS name to the service, and load balances traffic in a round-robin manner to network connections of that IP address among the pods matching the selector – even as failures cause the pods to move from machine to machine.
3. Exposing Front/Back End Services: By default, a service is exposed inside a cluster, e.g., back end pods might be grouped into a service, with requests from the front-end pods load balanced among them, but a service can also be exposed outside a cluster, e.g., for clients to reach front-end pods (Langemak (2015b)).

**Volumes**

1. Transient Nature of Kubernetes Filesystems: Filesystems in Kubernetes containers provide ephemeral storage, by default. This means that re-start of the pod will wipe out any data on such containers, and therefore, this form of storage is quite limiting in anything but trivial applications.
2. Persistent Storage Mechanism - Kubernetes Volume: A Kubernetes Volume provides persistent storage that exists for the life of the pod itself. This storage can also be used as shared disk space for containers within the pod.
3. Restrictions on the Mount Points: Volumes are mounted at specific mount points within the container, which are defined by the pod configuration, and cannot mount onto other Volumes or link to other Volumes. The same Volume can be mounted at different points in the filesystem tree by different containers.

**Namespaces**

Kubernetes provides a partitioning of the resources it manages into non-overlapping sets called namespaces. They are intended for use in environments with many users spread across multiple teams or projects, even separating environments like development, test, and production.

**ConfigMaps and Secrets**

1. Types of Kubernetes Configuration Data: A common application challenge is deciding where to store and manage configuration information, some of which may contain sensitive data. Configuration data can be anything as fine-grained as individual properties or coarse-grained information like entire configuration files or JSON/XML documents.
2. Access to Configuration Data: Kubernetes provides two closely related mechanisms to deal with this need – *configmaps* and *secrets*, both of which allow for configuration changes to be made without requiring an application rebuild. The data from configmaps and secrets will be made available to every single application to which these objects have been bound via deployment.

**StatefulSets**

1. Kubernetes Stateless versus Stateful Scaling: It is very easy to address the scaling of stateless applications: one simply adds more running pods – which is something that Kubernetes does very well. Stateful workloads are much harder, because the state needs to be preserved if a pod is restarted, and if the application is scaled up or down the state may need to be redistributed.
2. Ordering of the Stateful Instances: Databases are an example of stateful workloads. When run in high-availability mode, many databases come with the notion of a primary instance and secondary instance(s). In this case, the notion of ordering the instances is important.
3. Uniqueness of the Stateful Instances: Other applications like Kafka distribute the data among their brokers – so that one broker is not the same as another. In this case, the notion of instance uniqueness is important.
4. Uniqueness and Ordering among Pod Instances: StatefulSets are controllers that are provided by Kubernetes that enforce the properties of uniqueness and ordering among the instances of a pod and can be used to run stateful applications.

**DaemonSets**

Normally, the location where the pods are run are determined by the algorithm implemented in the Kubernetes scheduler. The ability to do this kind of pod scheduling is implemented by the feature called DaemonSets.

**Secrets**

Secrets contain ssh keys, passwords, and OAuth tokens for the pod.

**Managing Kubernetes Objects: Labels and Selectors**

1. Label Selector Based Object Matches: Kubernetes enables clients – users or internal components – to attach keys called *labels* to any API Object in the system, such as pods or nodes. Correspondingly, *label selectors* are queries against labels that resolve to matching objects.
2. Service Routing using Label Selectors: When a service is defined, one can define the label selector that will be used by the service router/load balancer to select the pod instances that the traffic will be routed to.
3. Controlling Service Routing Using Labels: Thus, simply changing the labels of the pods or changing the label selectors on the service can be used to control which pod gets the traffic and which doesn’t, and this can be used to support various deployment patterns like blue-green deployments or A-B testing. This capability to dynamically control how services utilize implementing resources provides a loose coupling within the infrastructure.
4. Multi-Criterion Label Selection Example: For example, if the application’s pods have labels for a system – with values such as , , for example – and a – with values such as , , for example – then an operation on all of and nodes can use a label selector, such as:

**Field Selectors**

1. Object-Attribute-Value Based Selection: Just like fields, field selectors also let one select Kubernetes resources. Unlike labels, the selection is based on the attribute values inherent to the resource being selected, rather than user-defined categorization.
2. Built-in Metadata Field Selectors: and are field selectors that will be present on all Kubernetes objects. Other selectors can be used depending on the object/resource type.

**Replication Controllers and Deployments**

1. Focus of the Replication Controller: A ReplicaSet declares the number of instances of a pod that is needed, and the Replication Controller manages the system so that the number of healthy pods that are running matches the number of pods declared in the ReplicaSet, determined by evaluating its selector.
2. Kubernetes Deployments versus ReplicaSets: Deployments are a higher-level management mechanism for ReplicaSets. While the Replication Controller manages the scale of the ReplicaSet, Deployments will manage what happens to the ReplicaSet – whether an update has to be rolled out, rolled back, etc.
3. Deployment Induced Changes in ReplicaSets: When Deployments are scaled up or down, this results in the declaration of the ReplicaSet changing – and this change in the declared state is managed by the Replication Controller.

**Cluster API**

1. Programmatic Control of Kubernetes Clusters: The design principles underlying Kubernetes allow one to programmatically create, configure, and manage Kubernetes clusters. This function is exposed via an API called the Cluster API.
2. Cluster as a Controllable Resource: A key concept embodied in the API is the notion that the Kubernetes cluster itself is a resource object that can be managed just like any other Kubernetes resources. Similarly, machines that make up the cluster are also treated as a Kubernetes resource.
3. Cloud Provider Specific API Functions: The API has two pieces – the core API, and a provider implementation. The provider implementation consists of cloud-provider specific functions that let Kubernetes provide the cluster API in a fashion that is well-integrated with the cloud provider’s services and resources.

**Architecture**

Kubernetes follows a primary/replica architecture. The components of Kubernetes can be divided into those that manage an individual node, and those that are part of the control plane.

**Kubernetes Control Plane**

1. Kubernetes Master and Control Plane: The Kubernetes master is the main controlling unit of the cluster, managing the workload and directing communication across the system. The Kubernetes control plane consists of various components, each its own process, that can run both on a single master node or on multiple masters supporting high-availability clusters. The various components of the Kubernetes control plane are as follows.
2. etcd - Distributed Key-Value Store: etcd is a persistent, lightweight, distributed key-value store developed by CoreOS that reliably stores the configuration data of the cluster, representing the overall state of the cluster at any given point in time.
3. Choosing Cluster Consistency over Availability: Just like Apache ZooKeeper, etcd is a system that favors consistency over availability in the event of network partition. This consistency is crucial for correctly scheduling and operating services.
4. Usage of etcd’s Watch API: The Kubernetes API Server uses etcd’s Watch API to monitor the cluster or roll out critical configuration changes or simply restore any divergences of the state of the cluster back to what was declared by the deployer.
5. Example - Creating Additional Pod Instance: As an example, if the deployer specified that three instances of a particular pod need to be running, this fact is stored in etcd. If it is found that only two instances are running, this delta will be detected by comparison with the etcd data, and Kubernetes will use this to schedule the creation of an additional instance of that pod.
6. The Kubernetes API Server Component: The API Server is a key component and serves the Kubernetes API using JSON over HTTP, which provides both the internal and the external interfaces to Kubernetes (Marhubi (2015a)). The API server process and validates REST requests and updates the state of the API objects in etcd, thereby allowing clients to configure workloads and containers across worker nodes (Ellingwood (2018)).
7. Role of the Kubernetes Scheduler: The scheduler is the pluggable component that selects which node an unscheduled pod – the basic entity managed by the scheduler – runs on, based on resource availability. The scheduler tracks resource use on each node to ensure that the workload is not scheduled in excess of available resources.
8. Matching Resource Supply and Workload Demand: For this purpose, the scheduler must know the resource requirements, resource availability, and other user-provided constraints and policy directives such as quality-of-service, affinity/anti-affinity requirements, data locality, and so on. In essence, the scheduler’s role is to match the resource *supply* to workload *demand*.
9. The Kubernetes Controller Manager Component: The controller is a reconciliation loop that drives the actual cluster state toward the desired cluster state, communicating with the API Server to create, update, and delete the resources it manages, i.e., the pods, the services, and the end-points (Marhubi (2015a)). The Controller Manager is a component that manages a set of core Kubernetes controllers.
10. The Kubernetes Replication Controller Component: One kind of controller is the Replication Controller, which handles replication and scaling by running a specified number of pods across the cluster. It also handles creating replacement pods if the underlying node fails.
11. Kubernetes DaemonSet and Job Controllers: Other controllers that are a part of the core Kubernetes system include a DaemonSet Controller for running exactly one pod on every machine, and a Job Controller for running pods that run to completion, e.g., as part of a batch job (Sanders (2015)).
12. Control Exercised using Label Selection: The set of pods that a controller manages is determined by the label selection that are part of the controller’s definition.

**Kubernetes Node**

1. Nodes where Containers are Deployed: A Node, also known as a Worker or a Minion, is a machine where containers/workloads are deployed. Every node in the cluster must run a container runtime such as Docker, as well as the below-mentioned components, for communication with the primary for network configuration of these containers.
2. Kubelet - Kubernetes Node Health Monitor: Kubelet is responsible for running the state of each node, ensuring that all containers in the node are healthy. It takes care of starting, stopping, and maintaining application containers organized into pods as directed by the control plane (Marhubi (2015b)).
3. Mechanism Employed in Monitoring the Node: Kubelet monitors the state of a pod, and if not in the desired state, the pod re-deploys to the same node. Node status is relayed every few seconds via heartbeat messages to the primary. Once the primary detects a node failure, the Replication Controller observes this state change and launches pods on other healthy nodes.
4. Role of the Kube-Proxy: The Kube-Proxy is an implementation of the network proxy and a load balancer, and it supports service abstraction along with other networking operations. It is responsible for routing the traffic to the appropriate container based on the IP and the port number of the incoming request.
5. Kubernetes Workload/Container Runtime: A container resides inside a pod. The container is the lowest level of a micro-service, which holds the running applications, libraries, and their dependencies. Containers can be exposed to the world through an external IP address. Kubernetes supports Docker containers since its first version, and in July 2016 rkt container engine was added.

**Add-Ons**

1. Additional Special Purpose Cluster Applications: Add-ons operate just like any other application running within the cluster: they are implemented via pods and services, and are only different in that they implement features of the Kubernetes cluster. The pods may be managed by Deployments, Replication Controllers, and so on. There are many add-ons, and the list is growing. Some of the more important are as follows.
2. DNS: All Kubernetes clusters should have a cluster DNS; it is a mandatory feature. Cluster DNS is a DNS server, in addition to the other DNS server(s) in the environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
3. Web UI: This is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
4. Container Resource Monitoring: Providing a reliable application runtime, and being able to scale it up or down in response to workloads, means being able to continuously and effectively monitor workload performance. Container Resource Monitoring provides this capability by recording metrics about the containers in a central database, and provides UI for browsing that data. The cAdvisor is a component of the slave node that provides a limited metric monitoring capability. There are full metrics pipelines as well, such as Prometheus, which can meet most monitoring needs.
5. Cluster-Level Logging: Logs should have a separate storage and lifecycle independent of nods, pods, or containers. Otherwise, node or pod failures can cause loss of event data. The ability to di this is called luster-level logging, and such mechanisms are responsible for saving container logs to a central log-store with searching/browsing interface. Kubernetes provides no native storage for log data, but one can integrate many existing logging solutions into the Kubernetes cluster.

**Microservices**

Kubernetes is commonly used as a way to host microservice-based implementation, because it and its associated ecosystem of tools provides all the capabilities needed to address key concerns of any microservice architecture.

**Kubernetes Persistence Architecture**

1. Containers – Enabling Portability of Software: Containers emerged as a way to make software portable. The container contains all the packages you need to run a service. The provided filesystem makes the containers extremely portable and easy to use in development. A container can also be moved from development to test or production with no or relatively few changes.
2. Reasons for Requiring Container Persistence: Historically, Kubernetes was suitable only for stateless services. However, many applications have a database, which require persistence, which leads to creation of persistence storage for Kubernetes. Implementing persistent storages is one of the top challenges for Kubernetes administrators, DevOps, and cloud engineers. Containers may be more ephemeral, but more and more of their data is not, so one needs to ensure data’s survival in case of container termination or hardware failure.
3. Persistent Storage for Containerized Applications: When deploying containers with Kubernetes or containerized applications, companies of ten realize that they need persistent storage. They need to provide fast and reliable storage for databases, root images, and other data used by the containers.
4. CNCF Kubernetes Persistent Storage Guidelines: In addition to the landscape, the Cloud Native Computing Foundation (CNCF) has published other information about Kubernetes Persistent Storage including a blog helping to define the container attached storage pattern. This pattern can be thought of as one that uses Kubernetes itself as a component of the storage system or service.
5. Public Domain Storage Orchestration Projects: More information about the relative popularity of these and the other approaches can be found on the CNCF’s landscape survey as well, which showed that OpenEBS from MayaData and Rook – a storage orchestration project – were the two projects most likely to be in evaluation.

**References**

* Ellingwood, J. (2018): [An Introduction to Kubernetes](https://web.archive.org/web/20180705232851/https:/www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes)
* Garrison, J. (2016): [Why Kubernetes is Abbreviated k8s](https://medium.com/@rothgar/why-kubernetes-is-abbreviated-k8s-905289405a3c)
* Langemak, J. (2015a): [Kubernetes 101 – Networking](http://www.dasblinkenlichten.com/kubernetes-101-networking/)
* Langemak, J. (2015b): [Kubernetes 101 – External Access into the Cluster](http://www.dasblinkenlichten.com/kubernetes-101-external-access-into-the-cluster/)
* Marhubi, K. (2015a): [Kubernetes from the Ground Up: the API Server](http://kamalmarhubi.com/blog/2015/09/06/kubernetes-from-the-ground-up-the-api-server/)
* Marhubi, K. (2015b): [What even is a kubelet?](http://kamalmarhubi.com/blog/2015/08/27/what-even-is-a-kubelet/)
* Sanders, J. (2015): [Kubernetes: Exciting Experimental Features](https://livewyer.io/blog/2015/10/02/kubernetes-exciting-experimental-features/)
* Sharma, P. (2017): [Autoscaling based on CPU/Memory in Kubernetes — Part II](https://blog.powerupcloud.com/autoscaling-based-on-cpu-memory-in-kubernetes-part-ii-fe2e495bddd4)
* Strachan, J. (2015): [Kubernetes for Developers](https://blog.fabric8.io/kubernetes-for-developers-2a9c7202fcd3#.b6u76jxar)
* Wikipedia (2020): [Kubernetes](https://en.wikipedia.org/wiki/Kubernetes)