Copyright Notice

Prakash Badhe

prakash.badhe@vishwasoft.in

Cluster of Machines

- Increase processing power
- Handle large amount of data and users
- Make the applications Highly Available to users.
- Load Balancing
- Fail Over and data replication
- Transaction management
- Utilize the idle resources

Container Management

- The Docker platform and other tools support to manage the lifecycle of a container.
- The Docker Command Line Interface (CLI) supports the following container activities:
  - Pulling a repository from the registry.
  - Running the container and optionally attaching a terminal to it.
  - Committing the container to a new image.
  - Uploading the image to the registry.
  - Terminating a running container.

Manage Large No of Containers

- CLI meets the needs of managing one container on one host,.
- It is not useful for managing multiple containers deployed on multiple hosts.
- To go beyond the management of individual containers, we turn to container orchestration tools.
- The container orchestration tools extend lifecycle management capabilities to complex, multi-container workloads deployed on a cluster of machines.
- The orchestration tools allow users to treat the entire cluster as a single deployment target.

Orchestration process

- The process of orchestration involves tools that can automate all aspects of application management from initial placement, scheduling and deployment to steady-state activities such as update, deployment, update and health monitoring functions that support scaling and failover.
- These capabilities characterize some of the core features users expectations offer modern container orchestration tools.
- These tools use container configuration in a standard schema, using languages such as YAML or JSON.

Configuration options

- Declarative definition in YML
- **Rules and Constraints** for placements of containers, performance and high availability.
- Provisioning, or scheduling involves negotiating the placement of containers within the cluster and launching them. This involves selecting an appropriate host based on the configuration and load balancing algorithm.

Container Discovery

- In a distributed deployment consisting of containers running on multiple hosts, container discovery becomes critical. Web servers need to dynamically discover the database servers, and load balancers need to discover and register web servers.
- The Orchestration tools provide, or expect, a distributed key-value store, a lightweight DNS or some other mechanism to enable the discovery of containers.

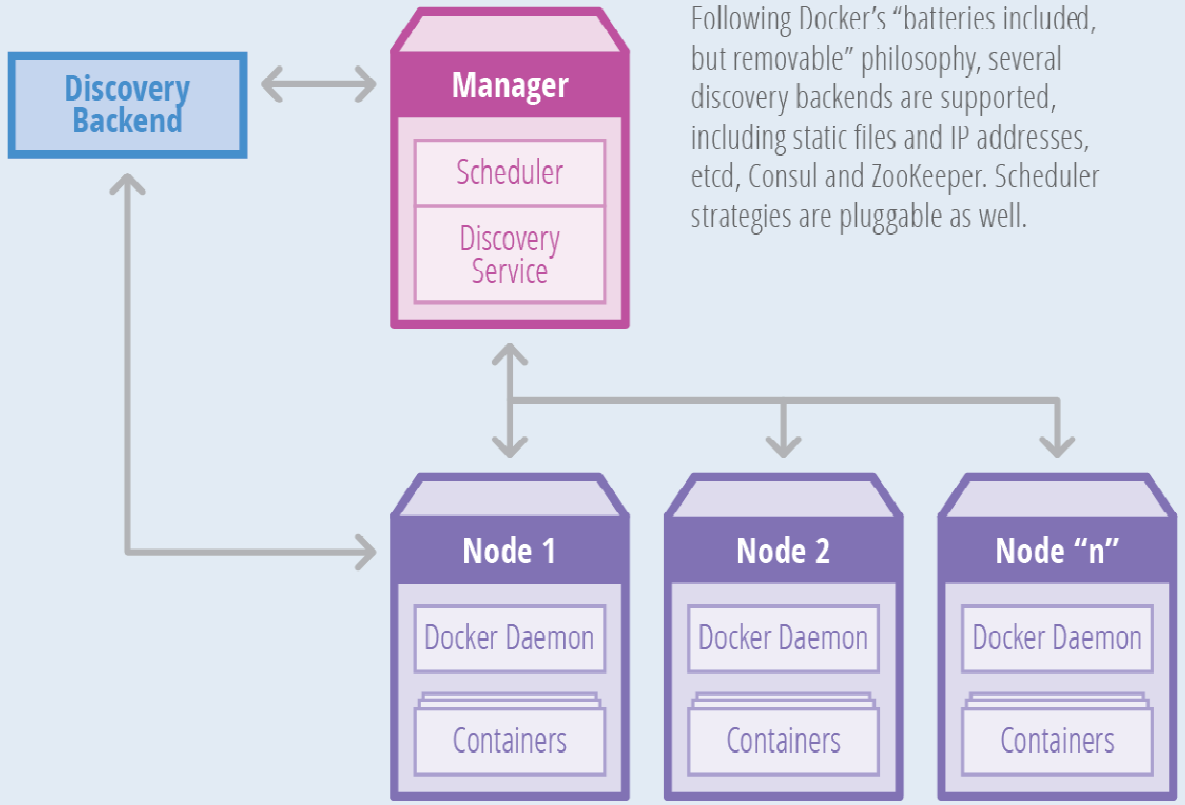
Health Monitoring

- **Health Monitoring and corrective actions**
- Since orchestration tools are aware of the desired configuration of the cluster, they should be able to track and monitor the health of the clusters' containers and hosts.
- In the event of host failure, the tools can relocate the container.
- Similarly, when a container crashes, orchestration tools can launch a replacement.
- Orchestration tools ensure that the deployment always matches the desired state declared by the developer or operator.

Container Orchestration Tools

- **Docker Swarm**
- Matches with Docker's own YML configurations or command line approaches.
- Swarm transparently deals with an endpoint associated with a pool of Docker Engines.
- The existing tools and APIs continue to work with a cluster in the same way they work with a single instance.
- Docker's tooling/CLI and Compose are how developers create their applications, and therefore, they don't have to be recoded to accommodate an orchestrator.

Swarm Architecture



Source: The New Stack.

THENEWSTACK

Swarm Usage

- Docker Swarm supports constraints and affinities to determine the placement of containers on specific hosts.
- Constraints define requirements to select a subset of nodes that should be considered for scheduling.
- They can be based on attributes like storage type, geographic location, environment and kernel version. Affinity defines requirements to collocate containers on hosts.

Discovery in Swarm

- For discovering containers on each host, Swarm uses a pluggable backend architecture that works with a simple hosted discovery service, static files, lists of IPs, etcd, Consul and ZooKeeper.
- Swarm supports basic health monitoring, which prevents provisioning containers on faulty hosts.

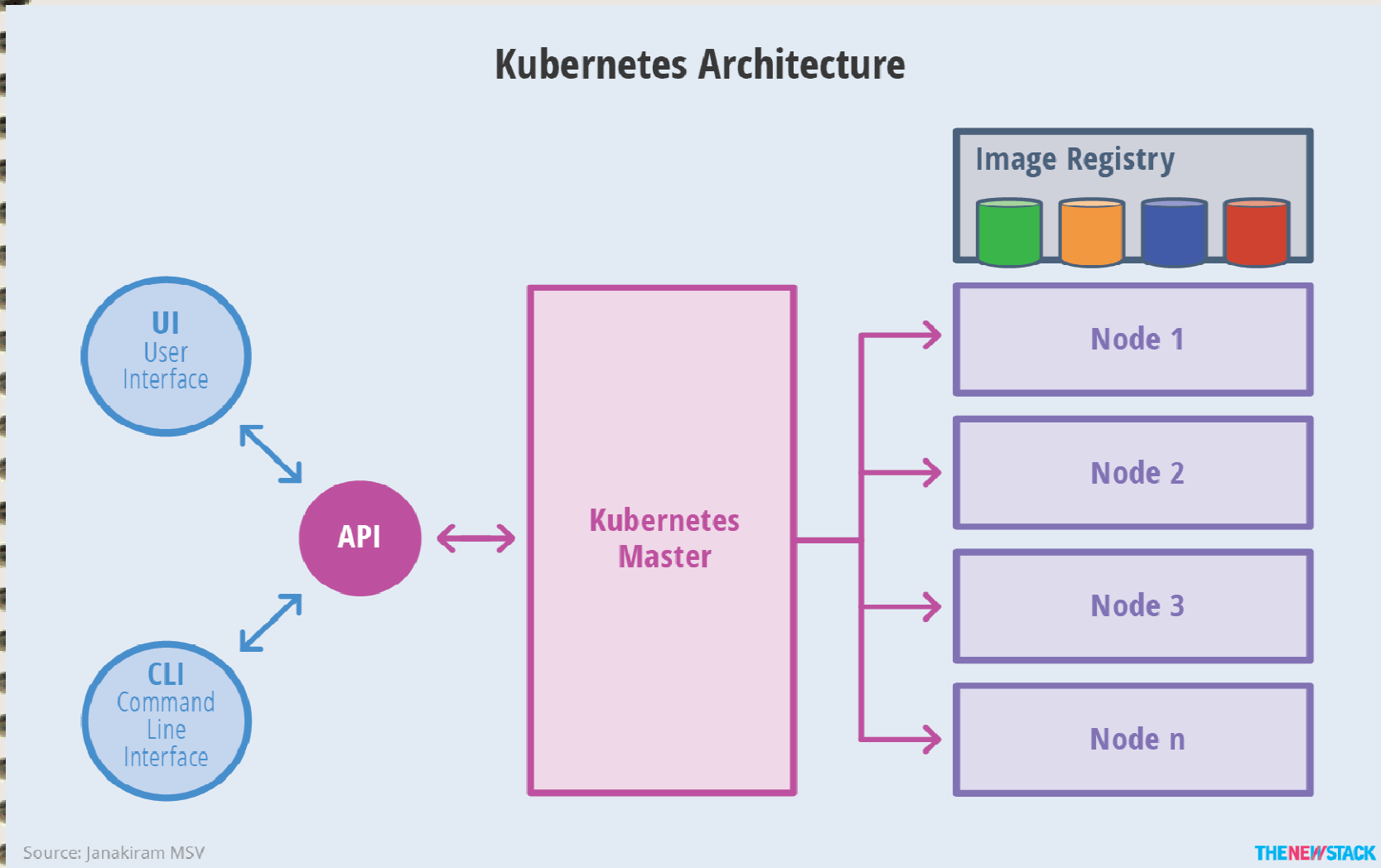
Apache Mesos

- Apache Mesos is an open source cluster manager that simplifies the complexity of running tasks on a shared pool of servers.
- Originally designed to support high-performance computing workloads, Mesos added support for Docker in the 0.20.0 release.
- A typical Mesos cluster consists of one or more servers running the mesos-master and a cluster of servers running the mesos-slave component.

Kubernetes

- Container orchestration tool from Google that claims to deal with two billion containers every day.
- Kubernetes enjoys unique credibility.
- Kubernetes works with different container such as Docker and RKT containers.
- Kubernetes controls the containers through Virtual Machine Drivers.
- VM is needed to work with Kubernetes cluster.
-

Kubernetes architecture



Kubernetes Tools

- Virtual Machine Manager : Oracle Virtual Box
- Minikube : Local Cluster Manager
- Kubectl : Cluster Client
- Tools available for platforms like windows, Linux with flavors ,Mac and multiple other platforms.
- Works with Container Runtimes
  - Docker
  - RKT
  - CRI-O

Kubernetes in nutshell

- Kubernetes' architecture is based on a master server with multiple nodes which are pod managers.
- The command line tool, called kubecfg, connects to the API endpoint of the master to manage and orchestrate the pods through nodes.
- Below is the definition of each component that runs within the Kubernetes environment:

Kubernetes design

- Kubernetes is designed on the principles of scalability, availability, security and portability.
- It optimizes the cost of infrastructure by efficiently distributing the workload across available resources.

Kubernetes components

- **Master**: The server that runs the Kubernetes management processes, including the API service, replication controller and scheduler.
- **Node**: The host that runs the kubelet service and the Docker Engine. Nodes receive commands from the master.
- **Kubelet**: The node-level manager in Kubernetes; it runs on a minion.
- **Pod**: The collection of containers deployed on the same Node.

Kubernetes components

- **Replication controller**: Defines the number of pods or containers that need to be running.
- **Service**: A definition that allows the discovery of services/ports published by each container, along with the external proxy used for communications.
- **Kubecfg**: The command line interface that talks to the master to manage a Kubernetes deployment.

Kubernetes Node

- A node is a worker machine in Kubernetes, previously known as a minion.
- A node may be a VM or physical machine, depending on the cluster.
- Each node has the services necessary to run **pods** and is managed by the master components.
- The services on a node include Docker, kubelet and kube-proxy.

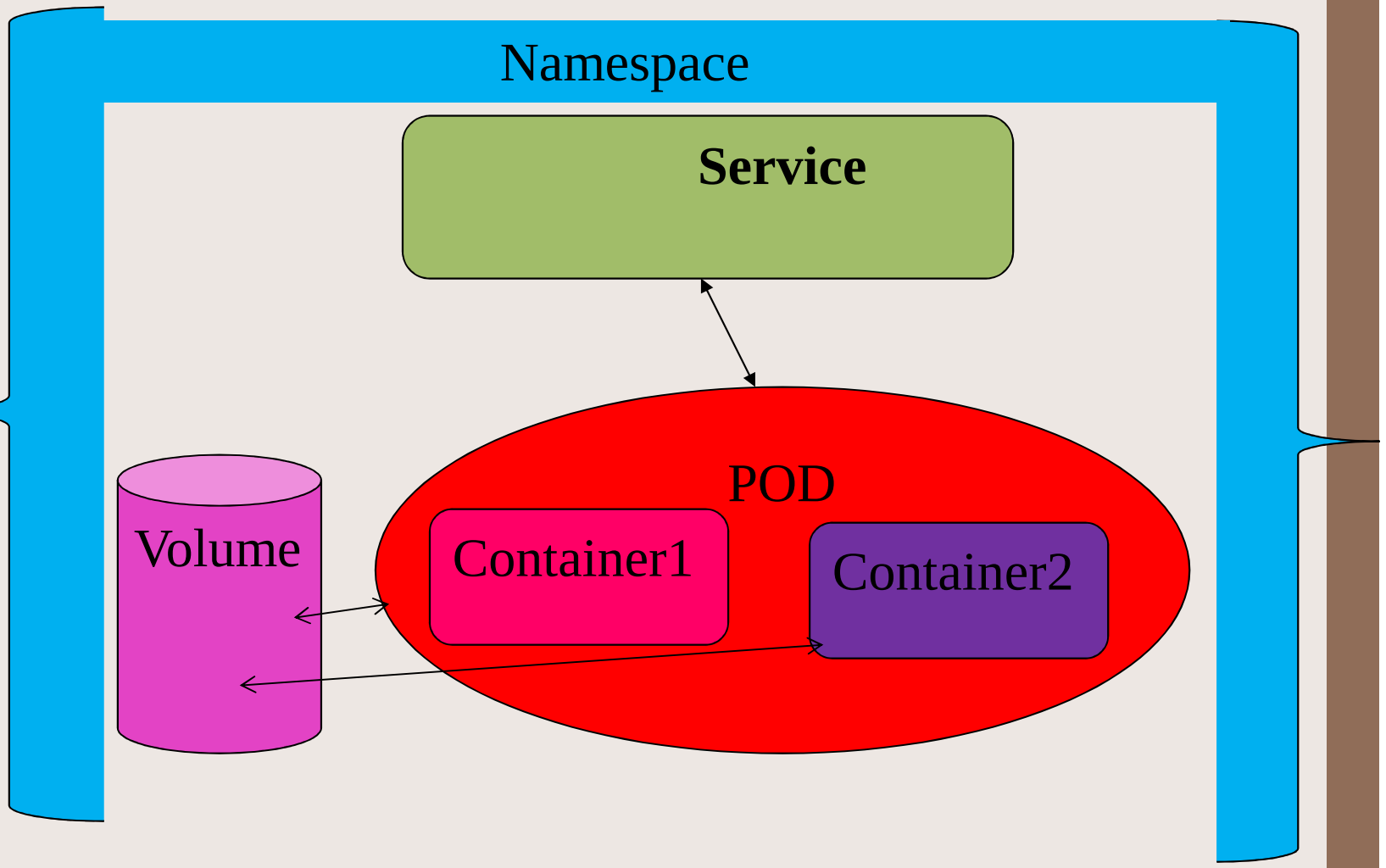
Kubernetes Cluster

- Minikube : Single Node Cluster
- With Kubeadm tools in LAN across the physical machines
- Across the Virtual Machine Nodes in Network
- RedHat Openshift Cluster
- Cluster as service from cloud Amazon EC2, GCE,Azure etc.
- Docker in Docker

Pods in Kubernetes

- A pod is a collection of one or more containers. The pod serves as Kubernetes' core unit of management.
- Pods act as the logical boundary for containers sharing the same context and resources.
- The grouping mechanism of pods make up for the differences between containerization and virtualization by making it possible to run multiple dependent processes together.
- At runtime, pods can be scaled by creating replica sets, which ensure that the deployment always runs the desired number of pods.

Object Structure



Basic Objects

- **Pod** : Group of containers
- **Service** : Contains multiple pods
- **Volume** For shared data to containers
- **Namespace** : Package for all above objects
-

Kubernetes configuration

- The service definition, along with the rules and constraints, is described in a JSON file.
- For service discovery, Kubernetes provides a stable IP address and DNS name that corresponds to a dynamic set of pods.
- When a container running in a Kubernetes pod connects to this address, the connection is forwarded by a local agent (called the kube-proxy) running on the source machine to one of the corresponding backend containers.

Minikube Features

- Minikube supports Kubernetes features
  - DNS
  - NodePorts
  - ConfigMaps and Secrets
  - Dashboards
  - Container Runtime: Docker, rkt and CRI-O
  - Enabling CNI (Container Network Interface)
  - Ingress for loadbalancer

Minikube VM Drivers

- virtualbox
- vmwarefusion
- kvm2
- kvm
- hyperkit
- xhyve (deprecated)

Get Started with Cluster

- On Linux : minikube start --vm-driver=virtualbox
- On Windows10 with Hypervisor :
- minikube start --vm-driver hyperv --hyperv-virtual-switch “MySwitch”
- **Minikube ip :** get the ip address of cluster
- **Minikube dashboard:** opens the dashboard in browser.
- minikube get-k8s-versions : get a list of all available kuebernets versions
- minikube get-k8s-versions
- Start for the specific version of Kubernetes
- minikube start --Kubernetes-version v1.7.3

Minkube Commands

- **Proxy usage**
- https_proxy=<my proxy> minikube start --docker-env http_proxy=<my proxy> --docker-env https_proxy=<my proxy> --docker-env no_proxy=192.168.99.0/24
- **To stop the cluster**
- minikube stop
- **Delete the Cluster**
- **minikube delete:** delete the minikube cluster. This command shuts down and deletes the minikube virtual machine. No data or state is preserved.

Behind Proxy

- To access internet through proxy server
- minikube start --docker-env http_proxy=http://$YOURPROXY:PORT \ --docker-env https_proxy=https://$YOURPROXY:PORT

Cluster Interactions

The minikube start command creates a “kubectl context” called “minikube”.

This context contains the configuration to communicate with the minikube cluster.

- Minikube sets this context to default automatically, but if you need to switch back to it in the future, run:
- kubectl config use-context minikube
- Or pass the context on each command like this: kubectl get pods --context=minikube.

Minikube cluster address

- The minikube VM is exposed to the host system via a host-only IP address, is obtained with the minikube ip command.
- Any services of type NodePort can be accessed over that IP address, on the NodePort.
- To determine the NodePort for the service use a kubectl command
- kubectl get service $SERVICE --output='jsonpath="{.spec.ports[0].nodePort}“’

Dashboard and Service

- To access the Kubernetes Dashboard, run the command in a shell after starting minikube to get the address:
- minikube dashboard
- **Services**
- To access a service exposed via a node port, run this command in a shell after starting minikube to get the address
- minikube service [-n NAMESPACE] [--url] NAME

Kubectl

- kubectl is a command line interface for running commands against Kubernetes clusters.
- Kubectl command structure
- Kubectl command, TYPE, NAME, and flags are:
- Commands are
  - Create
  - Get
  - Describe
  - Delete

Kubectl Type

- Specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms.
- For example, the following commands produce the same output:
- kubectl get pod pod1
- kubectl get pods pod1
- kubectl get po pod1

Kubectl resource Name

- NAME: Specifies the name of the resource.
- Names are **case-sensitiv**e.
- If the name is omitted, details for all resources are displayed,
- kubectl get pods.
- **kubectl get pod pod1 pod2**
- **Multiple resource types**
- **kubectl get pod/-pod1 replicationcontroller/rc1**
- To specify resources with one or more files: -f file1 -f file2 -f file<#>
- **kubectl get pod -f ./pod.yaml**

Kubectl Command Flags

- flags: Specifies optional flags.
- Use the -s or --server flags to specify the address and port of the Kubernetes API server.
- Flags that you specify from the command line override default values and any corresponding environment variables.
- To get the help, just run ’kubectl help’ from the terminal window.
- To print information about the status of a pod
- kubectl get pods <pod-name> --server-print=false

Kubectl commands

- To start the deployment
  - kubectl run app --image=pbadhe34/my-apps:app1 --port=8090
- To view the pods
  - kubectl get pods
- To view the deployments
  - kubectl get deployments
- To view the services
  - kubectl get services

Kubectl commands

- Expose the service for the deployment
- kubectl expose deployment app --type=NodePort
- Delete the service and deployments
- kubectl delete service, deployment app

Kubectl Configuration

- The definition of Kubernetes objects, such as pods, replica sets and services, are submitted to the master.
- Based on the defined requirements and availability of resources, the master schedules the pod on a specific node.
- The node pulls the images from the container image registry and coordinates with the local container runtime to launch the container.

POD in Host Network Mode

- The containers support the network communication in two modes
- Host Mode: The container address space utilizes the host address space.
- Multiple container can use the same host mode where their ip and port are assigned to host address and port.
- The same pod name cannot be reassigned.
- The pod name can be generated by the tag generateName.

Pod in Host Port mode

- The pod’s port and address is explicitly mapped to the host address and port no.
- Not more than one pod is able to get assigned the same port mapping.
- The pod name can be get generated.
- But still the port mapping once done from host side, cannot be reassigned again the same.

Exec Command

- Execute a command against a container in a pod.
- kubectl exec <pod-name> date
- Get output from running 'date' in container <container-name> of pod <pod-name>.
- kubectl exec <pod-name> -c <container-name> date
- Get an interactive TTY and run /bin/bash from pod <pod-name>.
- kubectl exec -ti <pod-name> /bin/bash : For singkle container pod. Which is default.

Logs command

- Print the logs for a container in a pod.
- kubectl logs <pod-name>: Single pod logs
- kubectl logs <pod-name> <container name>
- Start streaming the logs from pod <pod-name>
- kubectl logs -f <pod-name>

Linking the Containers in POD

- All the containers deployed with the same pod definition, get the same host name as localhost.
- So they are accessible to each other as localhost:<port No>
- The parent container has to change the way earlier it was linked to the child container.
- It has to look for linked container bas localhost insetead of any other name.

Sharing the Volume in POD

- As in docker, the containers here can have their own home directory path from the POD and multiple containers in the same pod can share the same home path as volume mount.
- The containers can exchange data through this shared local volume.
- But when the POD is restarted, every data initialized by the containers is lost.
- We can have Persistent volume from host file system as shared volume to manage the state across the restarts.

Persistent Volume

- The PersistentVolume subsystem abstracts details of how storage is provided from how it is consumed.
- Two new objects manage the persistent volume.
- PersistentVolume and PersistentVolumeClaim.
- The PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator.
- The PersistentVolume is a resource in the cluster just like a node is a cluster resource.

Persistent Volume Support

- Kubernetes supports **PersistentVolumes** of type hostPath. These PersistentVolumes are mapped to a directory inside the minikube VM.
- The Minikube VM boots into a tmpfs, so most directories will not be persisted across reboots (minikube stop).
- However, Minikube is configured to persist files stored under the following host directories.
  - /data
  - /var/lib/localkube
  - /var/lib/docker

PersistentVolume Config

```
apiVersion: v1
kind: PersistentVolume
metadata: name: pv0001
 spec:
    accessModes: - ReadWrite Once
    capacity:
      storage: 5Gi
   hostPath: path: /data/pv0001/
```

PV to POD

- The PVs are volume plug-ins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.
- So they maintain the path from **host file system** and independently manage the storage.le.
- It is initialized by a definition in YML file and loaded with kubectl command.
- It is monitored just as any other resource in the cluster.

PersistentVolumeClaim

- A PersistentVolumeClaim (PVC) is a request for storage by a user.
- It is similar to a pod.
- Pods consume node resources and PVCs consume PV resources.
- Pods can request specific levels of resources (CPU and Memory).
- Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

PVC to Container in POD

- While PersistentVolumeClaims allow a user to consume abstract storage resources, it is common that users need PersistentVolumes with varying properties, such as performance, for different problems.
- Cluster administrators need to be able to offer a variety of PersistentVolumes that differ in more ways than just size and access modes, without exposing users to the details of how those volumes are implemented.

PV and PVC Lifecycle

- PVs are resources in the cluster.
- PVCs are requests for those resources and also act as claim checks to the resource.
- The interaction between PVs and PVCs follows this lifecycle:
- **Provisioning:** statically or dynamically.
- **Binding:** linkiing
- **Using:** The Pods use claims as volumes.
- Once a user has a claim and that claim is bound, the bound PV belongs to the user for as long as they need it

Controller Objects

- Controllers build upon the basic objects, and provide additional functionality and convenience features.
- ReplicaSet : New version of Replication Controller
- Deployment: *Deployment* controller provides declarative updates for Pods and ReplicaSets.
- StatefulSet: StatefulSet is the workload API object used to manage stateful applications.

Controller Objects

- **DaemonSet:A** *DaemonSet* ensures that all or some Nodes run a copy of a Pod.
- Deleting a DaemonSet will clean up the Pods it created
- As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.
- **Job**: A *job* creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the *job* tracks the successful completions.

Replica set with pods

- Replica sets deliver the required scale and availability by maintaining a pre-defined set of pods at all times.
- A single pod or a replica set can be exposed to the internal or external consumers via services.

Abstraction

- The etcd is an open source, distributed key-value database from CoreOS, which acts as the single source of information for all components of the Kubernetes cluster.
- The master queries etcd to retrieve various parameters of the state of the nodes, pods and containers.
- This architecture of Kubernetes makes it modular and scalable by creating an abstraction between the applications and the underlying infrastructure.

Workload Scalability

- Each container is designed to perform only one task(image).
- Pods can be composed of stateless containers or stateful containers.
- Stateless pods can easily be scaled on-demand or through dynamic auto-scaling.
- Kubernetes supports horizontal pod auto-scaling, which automatically scales the number of pods in a replication controller based on CPU utilization.

App Deployments

- Applications deployed in Kubernetes are packaged as MicroServices.
- These MicroServices are composed of multiple containers grouped as pods.
- Hosted Kubernetes running on Google Cloud also supports cluster auto-scaling. When pods are scaled across all available nodes, Kubernetes coordinates with the underlying infrastructure to add additional nodes to the cluster.

List sorting

- To output objects to a sorted list in the terminal window, add the --sort-by flag to a supported kubectl command.
- Sort your objects by specifying any numeric or string field with the --sort-by flag.
- To specify a field, use a jsonpath expression.
- kubectl [command] [TYPE] [NAME] --sort-by=<jsonpath_exp>
- To print a list of pods sorted by name, run:
- kubectl get pods --sort-by=.metadata.name

High Availability

- The application workloads demand availability at both the infrastructure and application levels.
- In clusters at scale, everything is prone to failure, which makes high availability for production workloads strictly necessary.
- While most container orchestration engines and PaaS offerings deliver application availability, Kubernetes is designed to tackle the availability of both infrastructure and applications.

Make it available

- Kubernetes ensures high availability by means of replica sets, replication controllers and pod sets.
- Operators can declare the minimum number of pods that need to run at any given point of time.
- If a container or pod crashes due to an error, the declarative policy can bring back the deployment to the desired configuration.
- Stateful workloads can be configured for high availability through pet sets.

Security

- Security in Kubernetes is configured at multiple levels.
- The API endpoints are secured through transport layer security (TLS), which ensures the user is authenticated using the most secure mechanism available.
- Kubernetes clusters have two categories of users — service accounts managed directly by Kubernetes, and normal users assumed to be managed by an independent service.

Portability

- Kubernetes is designed to offer freedom of choice when choosing operating systems, container runtimes, processor architectures, cloud platforms and PaaS.
- A Kubernetes cluster can be configured on mainstream Linux distributions, including CentOS, CoreOS, Debian, Fedora, Red Hat Linux and Ubuntu.

Containers

- It can be deployed to run on local development machines; cloud platforms such as AWS, Azure and Google Cloud; virtualization environments based on KVM, vSphere and libvirt; and bare metal.
- Users can launch containers that run on Docker or rkt runtimes, and new container runtimes can be accommodated in the future.

Health Checks

- Kubernetes supports user-implemented application health checks.
- These checks are performed by the kubelet running on each minion to ensure that the application is operating correctly. Currently,

Supported Health Checks

- Kubernetes supports three types of health checks:
- **HTTP health check**: The kubelet will call a web endpoint. If the response code is between 200 and 399, it is considered a success.
- **Container exec**: The kubelet will execute a command within the container. If it returns “OK,” it is considered a success.
- **TCP socket**: The kubelet will attempt to open a socket to the container and establish a connection. If the connection is made, it is considered healthy.

Dashboard

- The dashboard is the web application, part of cluster to monitor and control the minikube cluster.
- To launch it : minikube dashboard
- URL : http://192.168.99.100:30000
- View the cluster objects like pods,services,volumes,replicas etc.
- Control the life cycle of cluster components
- Monitor the events for cluster object
- Deploy new components

POD Life cycle

- Pods are the deployment units in Kubernetes clsuer which represent the groups of containers.
- Kubernetes *Pods* are itself mortal. They are born and when they die, they are not resurrected.
- Each Pod gets its own IP address when started , those IP addresses cannot be relied upon to be stable over time.

Services to track PODs

- When some set of Pods as back-ends provides functionality to other Pods as frontends inside the Kubernetes cluster, how do those frontends find out and keep track of which back-ends are in that set?
- What if some pods in the backend gets down/crashed/started new instances ?
- How to manage the pods life automatically.
- How to decide which pod to send request from front end pods.?
- The service objects support these pod management.

Services

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them.
- The set of Pods targeted by a Service is usually determined by a Label Selector .
- In an application cluster, the frontend pods do not care which backend they use.
- While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that or keep track of the list of back-ends themselves.
- The Service abstraction enables this decoupling.

Service port mapping

- Kubernetes Services support TCP and UDP for protocols. The default is TCP.
- The service can map an incoming port to any targetPort of the POD containers.
- By default the targetPort will be set to the same value as the port field in container port.
  - The targetPort of service can be mapped to the POD container port as port mapping .
  - The service targetPort value can be dynamically assigned which will act as front end port for pOD containers.

Service Port Name

- The targetPort can be a string, referring to the name of a port in the backend Pods. The actual port number assigned to that name can be different in each backend Pod.
- This offers a lot of flexibility for deploying and evolving the Services.
- For example, the port number that pods expose in the next version of the backend software can be changed , without breaking clients.

Discovery of POD

- Services enable the discovery of pods by associating a set of pods to a specific criterion.
- Pods are associated to services through key-value pairs called labels and selectors.
- Any new pod with labels that match the selector will automatically be discovered by the service. This architecture provides a flexible, loosely-coupled mechanism for service discovery.

Service Types

- The applications in pods such as frontends expose through a Service onto an external world outside of the cluster boundary.
- **ClusterIP**: Exposes the service on a cluster-internal IP. This makes the service only reachable from within the cluster.
- This is the default Service Type.

NodePort Service

- **NodePort**: Exposes the service on each Node's IP at a static port (the NodePort).
- A ClusterIP service, to which the NodePort service will route, is automatically created.
- To contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>.
  - Kubernetes master allocates a port from a range specified by --service-node-port-range flag (default: 30000-32767), and each Node will proxy that port (the same port number on every Node) into the Service.
  - That port get reported in the Service's .spec.ports[*].nodePort field.

LoadBalancer service

- **LoadBalancer**: Exposes the service externally using a cloud/cluster provider's load balancer.
- Internally the NodePort and ClusterIP services, to which the external load balancer will route, are automatically created.
- The service is accessible by an external ip address assigned by the cluster provider.
- Traffic from the external load balancer is directed at the backend Pods.

ExternalName Service

- **ExternalName**: Maps the service to the contents of the externalName field (e.g. user.in.server.com), by returning a CNAME record with its value.
- No proxying of any kind is set up. This requires a kube-dns deployment in the cluster.
- Type NodePort

Scaling the Service

- You can specify the initial instances of containers in the pod as replicas in the deployment definition.
- The ReplicationController or ReplicaSet controls the replication management.
- The Service itself is another controller in Kubernetes API.
- The instances of containers in the pods can be scaled up and down by using kubectl scale command.
-

Service commands

- Read the services
- sudo kubectl get services
- sudo kubectl get service py-app
- sudo kubectl describe service py-app
- Scale the deployments
- kubectl scale --current -replicas=2 --replicas=3 deployment/mysql
- Scale the replication controller 'userData' to 3.
- kubectl scale --replicas=3 rc/userData

Life of POD in service

When any of the pods goes down or gets crashed, the service controller automatically creates new instances of it and tracks their life.

Update the service

- Update the services with
  - New version of image for pod containers
  - New configuration values
  - New POD definitions
  - Newer deployments
- No need to stop the existing service and restart with new configuration/new images etc.
- The rolling update supports in place updates.

Rolling Update

- **Rolling updates** allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones.
- The new Pods get scheduled on Nodes with available resources.
- Scaling the application to run multiple instances is also the requirement for performing updates without affecting application availability.
- By default, the maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one.

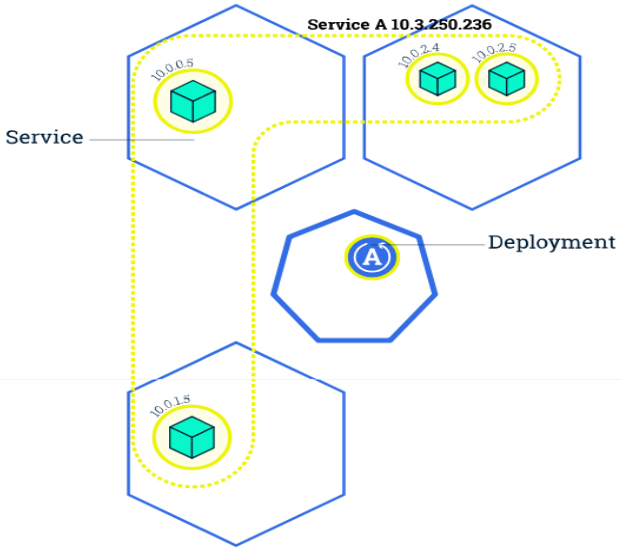
Versioned Updates

- In Kubernetes, updates are versioned and any Deployment update can be reverted to previous (stable) version.
- Similar to application Scaling, if a Deployment is exposed publicly, the Service will load-balance the traffic only to available Pods during the update.
- An available Pod is an instance that is available to the users of the application.

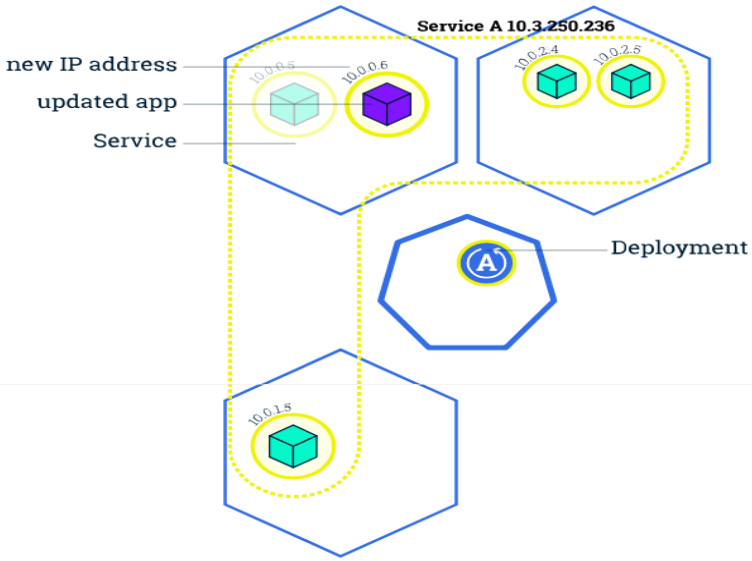
Rolling Update Actions

- Rolling updates allow the following actions:
- Promote an application from one environment to another (via container image updates)
- Rollback to previous versions
- Continuous Integration and Continuous Delivery of applications with zero downtime.
- Perform a rollback

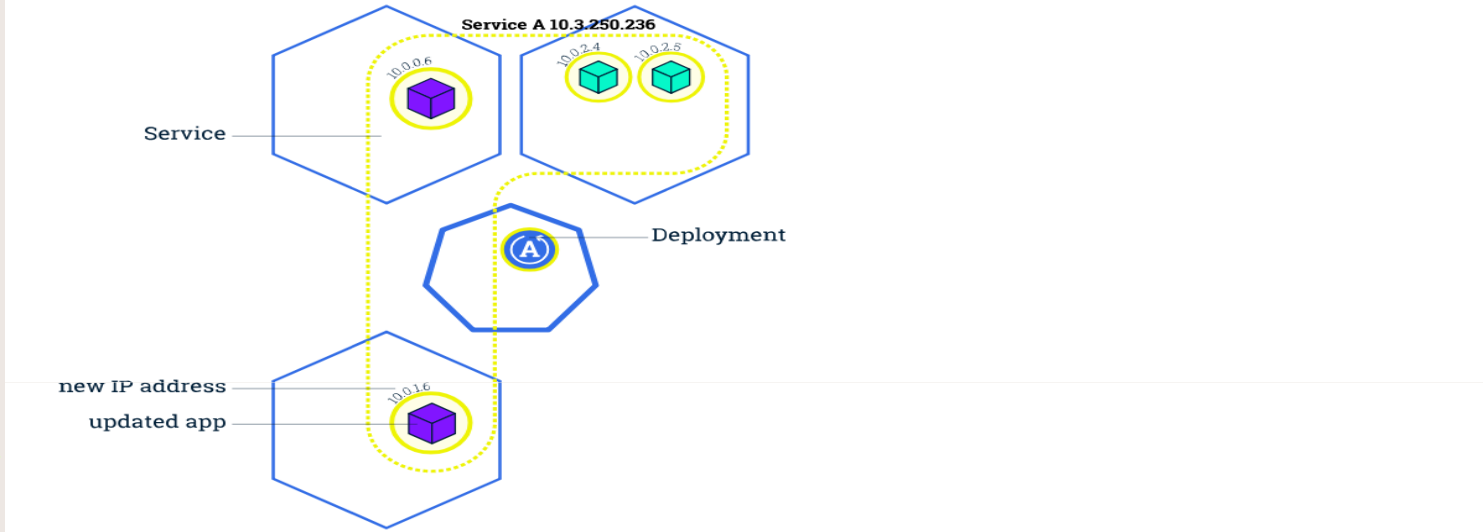
Rolling update service



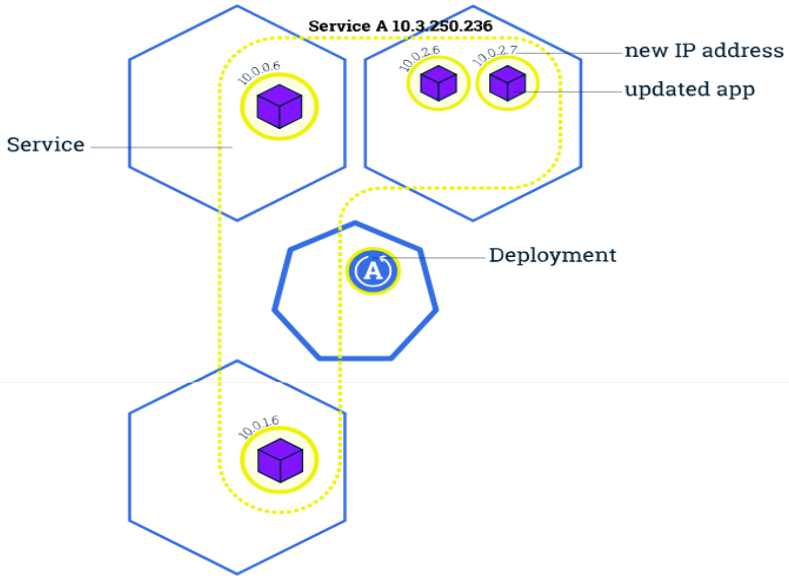
Update Stage1



Update Stage2



Update Stage3



RollOut commands

- The kubectl rollout command manages a deployment.
- Update the deployment by changing the image
- kubectl set image deployment/default-http-backend
- kubectl rollout status deployment/nginx-deployment
- kubectl rollout history deployment/nginx-deployment
- kubectl rollout pause RESOURCE
- kubectl rollout resume RESOURCE
- To roll back
- kubectl rollout undo deployment/nginx-deployment
-

Cluster Monitoring

- Monitoring an application's current state is one of the most effective ways to anticipate problems and discover bottlenecks in a production environment.
- Network Administrators handling availability, performance, and deployments are making teams to create, or use, orchestrators to handle all the services and servers.
- A container orchestration tool such as Kubernetes handles containers in several computers, and removes the complexity of handling distributed processing.

Application Health

- Monitors logs from
  - PODs
  - Containers
  - Services
- Monitor resources
  - Volume
  - Deployments
  - ReplicasSets

Resource Usage

- kubectl top command usage to to see the resource consumption for nodes or pods.
- Display Resource Usage for CPU,Memory,Storage for nodes and pods.
- Kubectl top <pod name/id>
- Kubectl top <Node name/id>
- This command requires **Heapster** to be correctly configured and working on the server.

Kubectl top options

- **--cluster**=The name of the kubeconfig cluster to use
- **--context**=The name of the kubeconfig context to use

REST API Access

- Run kubectl in proxy mode (recommended). it uses the stored api-server location and verifies the identity of the API server using a self-signed cert.
- Provide the location and credentials directly to the http client. This works with client code that is confused by proxies.
- http://localhost:8080/api/

Cluster Access

- To access a cluster, you need to know the location of the cluster and have credentials to access it
- Check the location and credentials
- $ kubectl config view
- Using kubectl proxy
- Runs kubectl in a mode where it acts as a reverse proxy.
- It handles locating the API server and authenticating.
- kubectl proxy --port=8080 &

Proxy config

- **Configuring the Proxy**
- The easiest way to access Kubernetes API is to configure a proxy.
- The kubectl comes with an option to configure it.
- Open a new terminal window and run the following command to create the proxy.
- kubectl proxy --port=8000
- Through this proxy session, any request sent to **localhost:8000** will be forwarded to the Kubernetes API server.

Kubectl REST API

- http://localhost:8000/api
- http://localhost:8000/api/v1/nodes | jq '.items[] .metadata.labels‘
- http://localhost:8000/api/v1/namespaces/default/services
- http://localhost:8000/v1/proxy/namespaces/default/services

Create New POD via POST

- Create nginx-pod.json file.
- Create POD by post
- curl -s http://localhost:8000/api/v1/namespaces/default/pods \ -XPOST -H 'Content-Type: application/json' \ -d@nginx-pod.json \ | jq '.status‘
- Create service by POST
- curl -s http://localhost:8000/api/v1/namespaces/default/services \ -XPOST -H 'Content-Type: application/json' \ -d@nginx-service.json \ | jq '.spec.clusterIP'

Verify the pods

- curl http://localhost:8000/v1/proxy/namespaces/default/services/nginx-service/