Kubernetes (K8s)

Introduction ➡️ Deep Dive

Ravindu Nirmal Fernando | SLIIT | February 2025

How can we run containers at scale?

Container orchestration

- Container orchestration automates the deployment, management, scaling, and networking of containers.

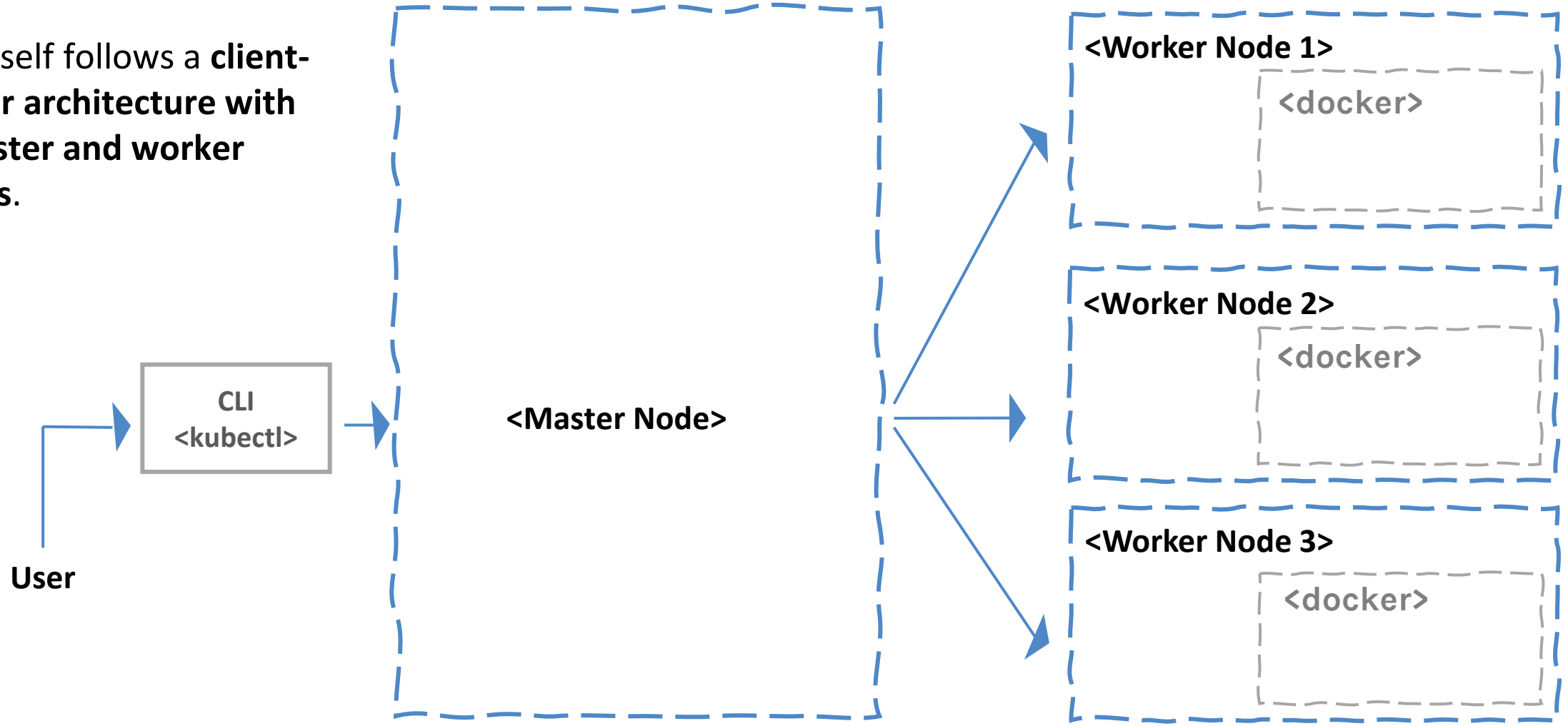
- Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it.

What is Kubernetes?

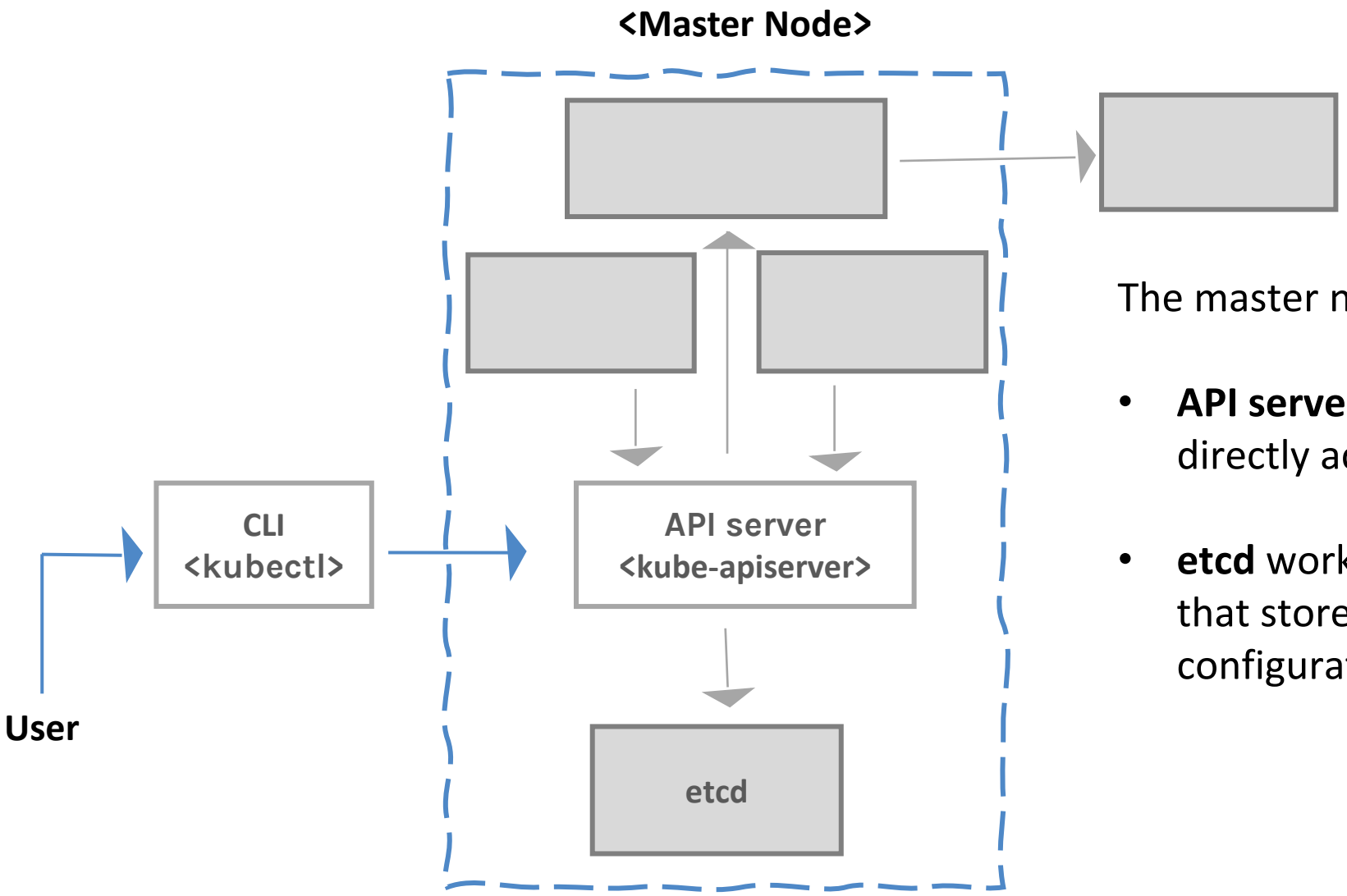
- “Kubernetes (k8s) is an open source platform for automating deployment, scaling and management of containers at scale”
- Project that was created by Google as an open source container orchestration platform. Born from the lessons learned and experiences in running projects like Borg and Omega @ Google
- It was donated to CNCF (Cloud Native Computing Foundation) who now manages the Kubernetes project
- Current Kubernetes stable version – 1.32

K8s Components & Architecture

K8s itself follows a **client-server architecture with a master and worker nodes**.



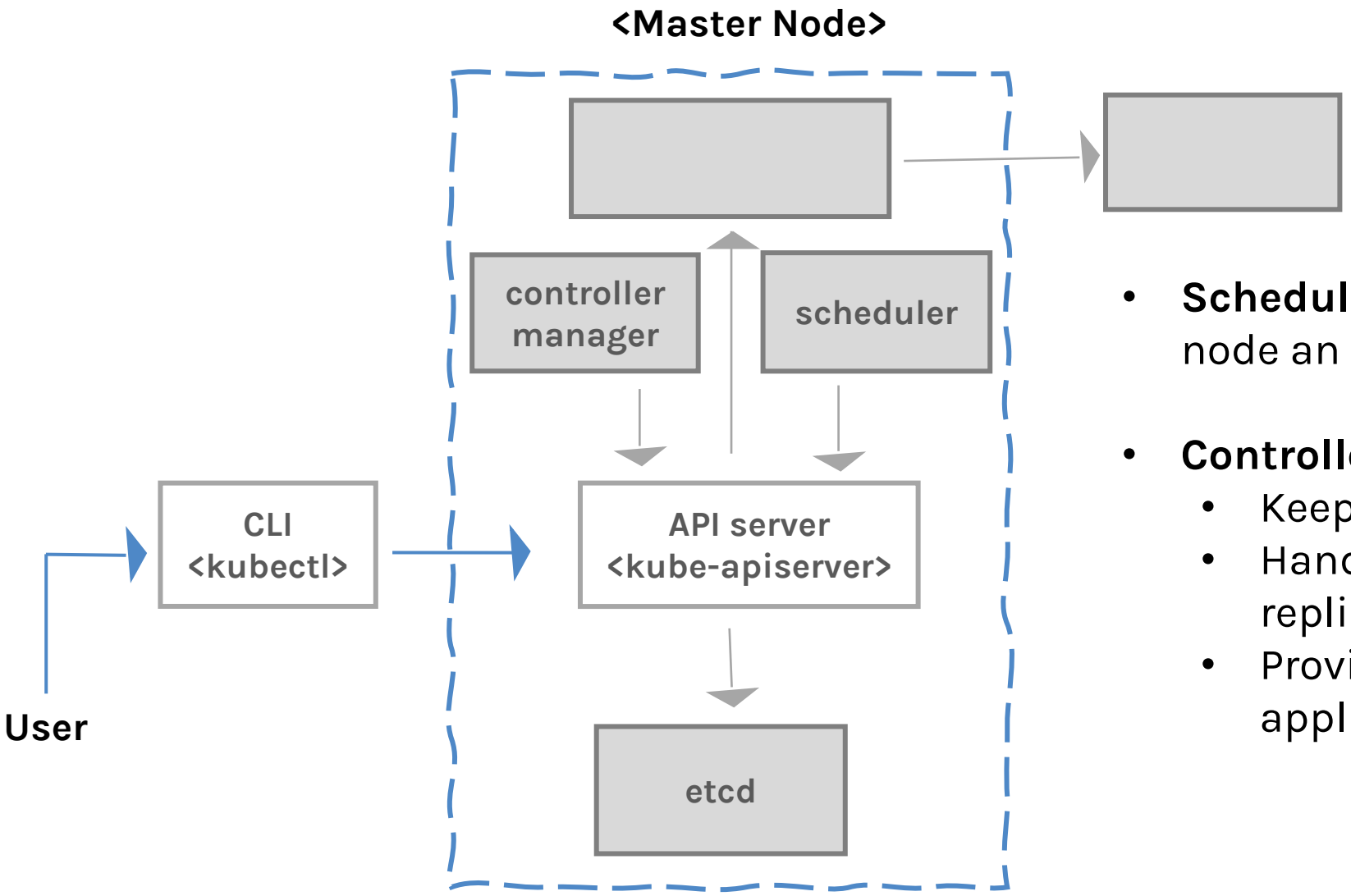
K8s Components & Architecture <cont>



The master node has:

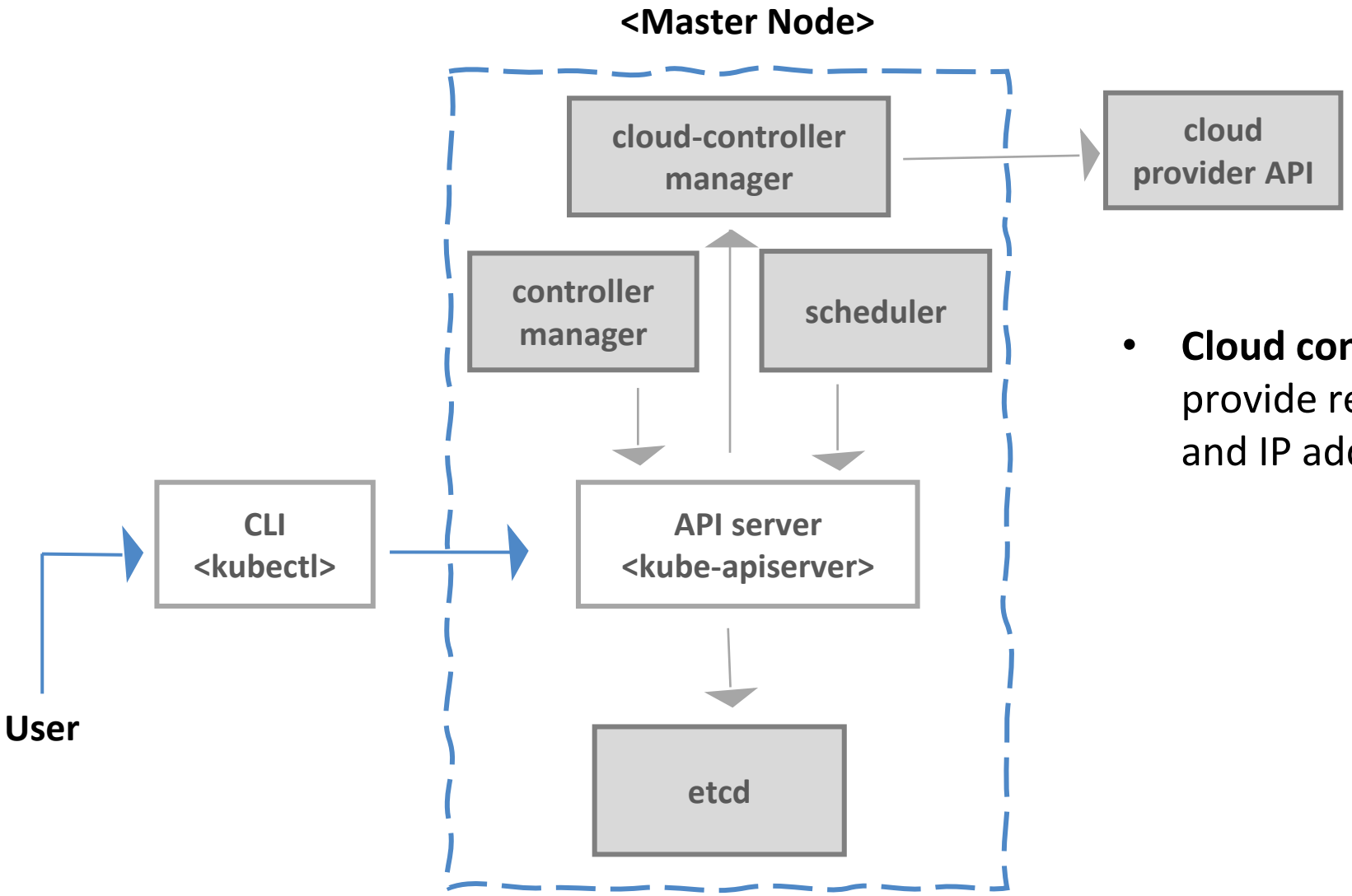
- **API server** contains various methods to directly access the Kubernetes
- **etcd** works as backend for service discovery that stores the cluster's state and its configuration

K8s Components & Architecture <cont>



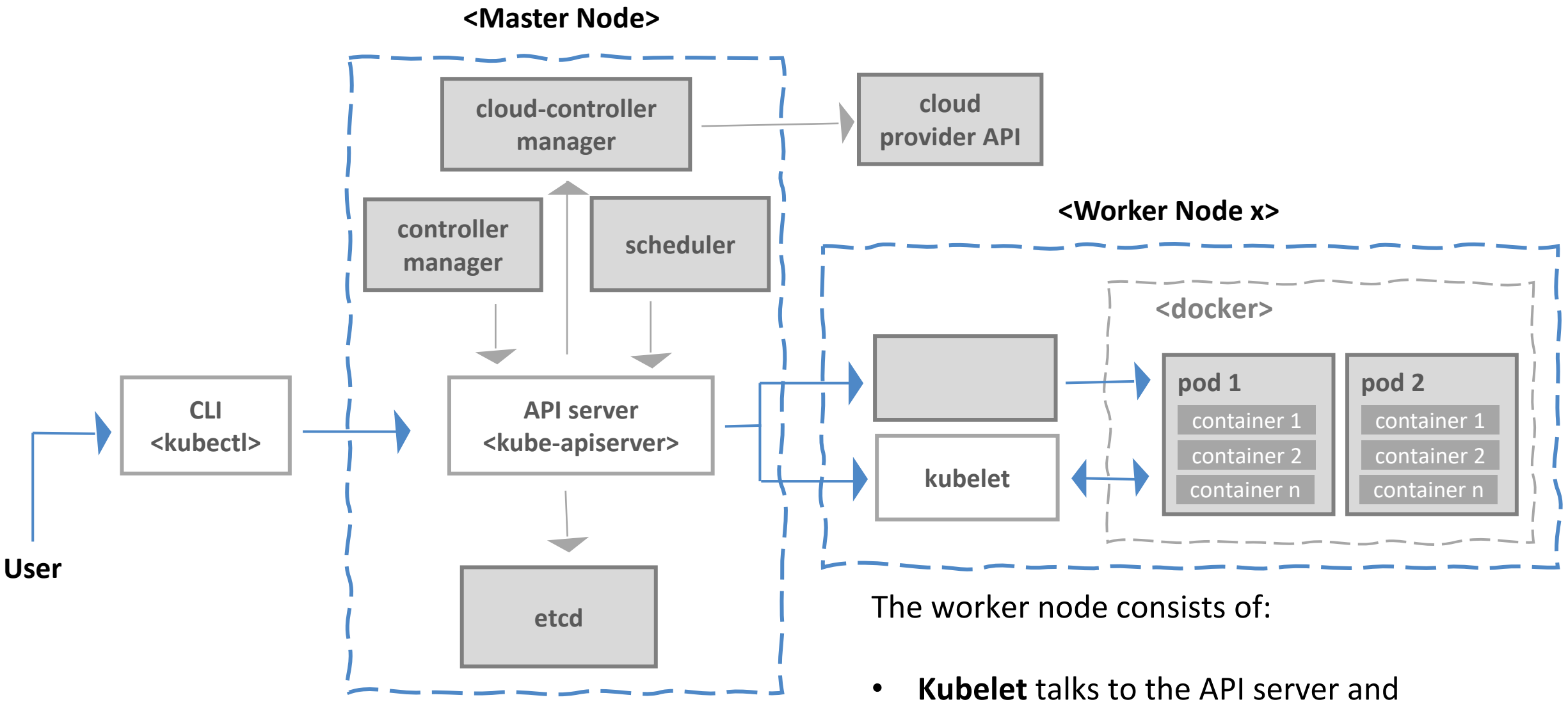
- **Scheduler** assigns to each worker node an application
- **Controller manager:**
  - Keeps track of worker nodes
  - Handles node failures and replicates if needed
  - Provide endpoints to access the application from the outside world

K8s Components & Architecture <cont>



- **Cloud controller** communicates with cloud provide regarding resources such as nodes and IP addresses

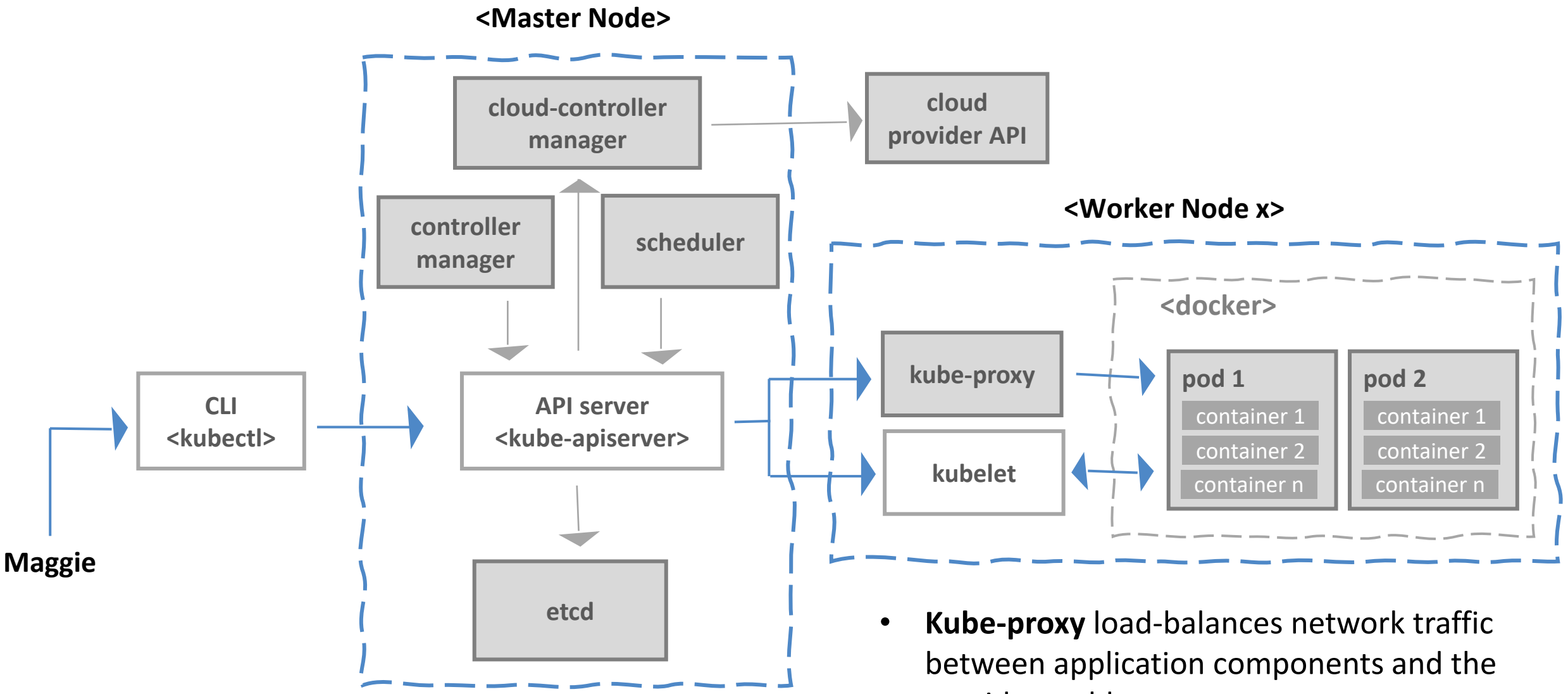
K8s Components & Architecture <cont>



The worker node consists of:

- **Kubelet** talks to the API server and manages containers on its node

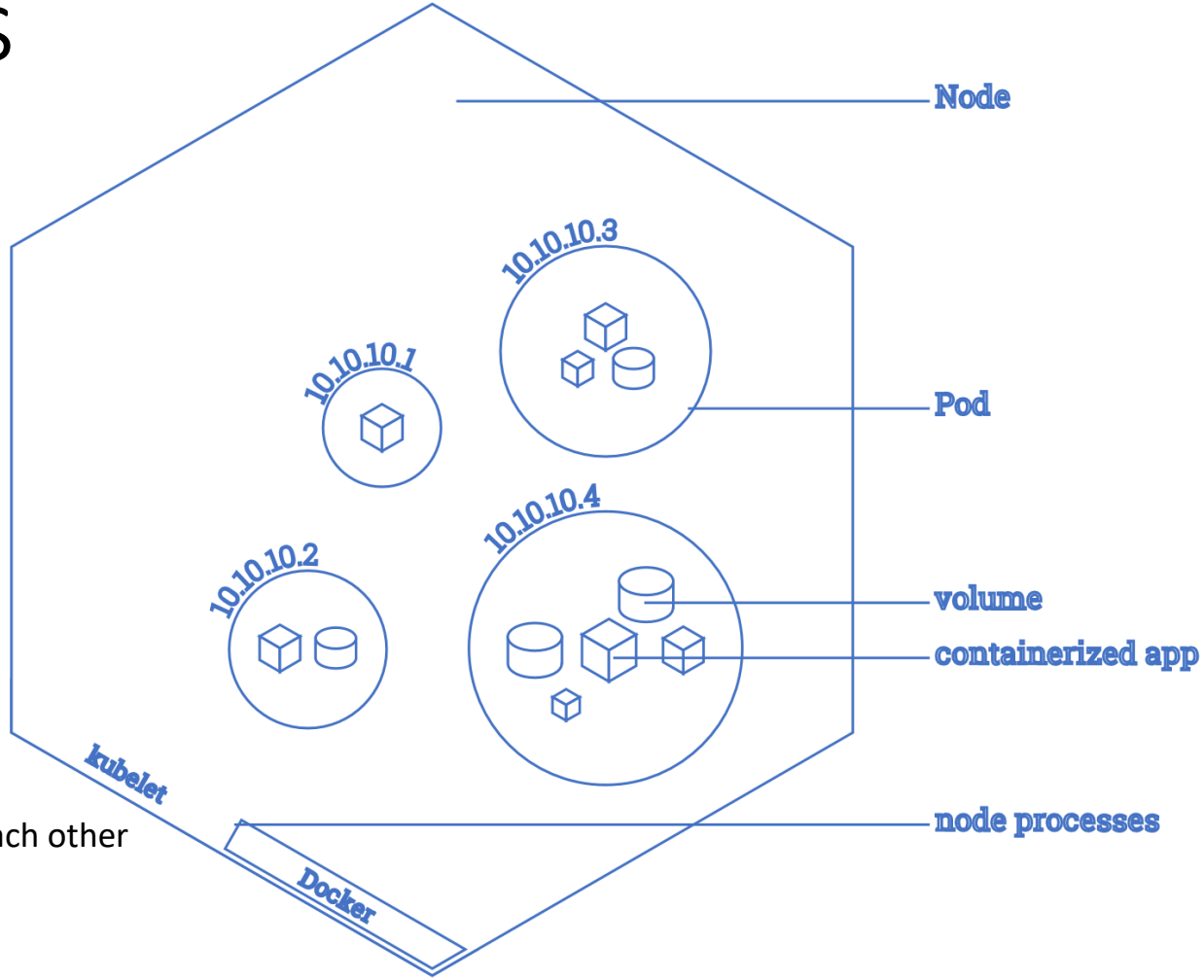
K8s Components & Architecture <cont>



- **Kube-proxy** load-balances network traffic between application components and the outside world

Basic building blocks

- Containers
  - Define single running process*
  - E.g. docker container
- Pods
  - the way of running containers in Kubernetes
  - basic deployable and scaling unit
  - defines one or more containers
  - containers are co-located on a node
  - flat network structure
- Nodes:
  - physical worker machines
  - can run multiple pods
  - pods running within single node don’t know about each other



Running things locally

- Minikube:
  - single node cluster
  - running in a VM
  - supports linux, windows and macOS
  - mature project
- kind:
  - Requires you to have Docker or Podman installed in your local computer
- Docker Desktop with built-in kubernetes:
  - single node cluster
  - running in a VM
  - windows and macOS
  - drag & drop installation
  - bound to specific kubernetes version

https://kubernetes.io/docs/tasks/tools/

Managing cluster resources

- **Create resource from file** - *kubectl create -f resource_file.yml*
- **Change existing (or create) resource based on file** - *kubectl apply -f resource_file.yml*
- **Delete existing resource** - *kubectl delete resource_type resource_name*
- **List resources of type** - *kubectl get resource_type*
- **Edit resource on the server** - *kubectl edit resource_type resource_name*

Debugging cluster resources

- **Execute command on the container -** *kubectl exec [-it] pod_name process_to_run*
- **Get container logs -** *kubectl logs pod_name [-c container_name]*
- **Forward port from a pod -** *kubectl port-forward pod_name local_port:remote_port*
- **Print detailed description of a resource -** *kubectl describe resource_type resource_name*

Few resource objects in K8s

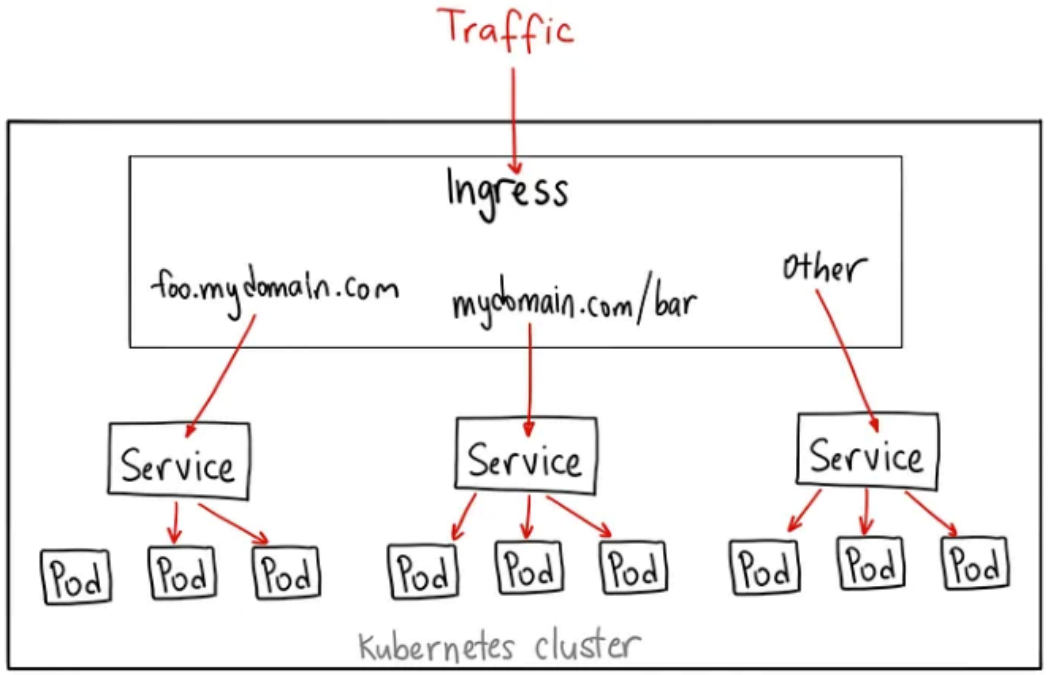
- **Replica Sets -** Ensures desired number of pods exist by: scaling up or down and running new pods when nodes fail
- ***Deployment***
  - A *Deployment* provides declarative updates for Pods and ReplicaSets. You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate.
- ***Service***
  - A method for exposing a network application that is running as one or more Pods in your cluster.
    - Cluster IP/ NodePort/ Load Balancer

`Feature`	`ClusterIP`	`NodePort`	`LoadBalancer`
Exposition	Exposes the Service on an internal IP in the cluster.	Exposing services to external clients	Exposing services to external clients
Cluster	This type makes the Service only reachable from within the cluster	A NodePort service, each cluster node opens a port on the node itself (hence the name) and redirects traffic received on that port to the underlying service.	A LoadBalancer service accessible through a dedicated load balancer, provisioned from the cloud infrastructure Kubernetes is running on
Accessibility	It is **default** service and Internal clients send requests to a stable internal IP address.	The service is accessible at the internal cluster IP-port, and also through a dedicated port on all nodes.	Clients connect to the service through the load balancer's IP.
Yaml Config	`type: ClusterIP`	`type: NodePort`	`type: LoadBalancer`
Port Range	Any public ip form Cluster	30000 - 32767	Any public ip form Cluster

Image Credit: StackOverflow

Few resource objects in K8s

- **Ingress -** An Ingress is a Kubernetes object that sits in front of multiple services and acts as an intelligent router. It defines how external traffic can reach the cluster services, and it configures a set of rules to allow inbound connections to reach the services on the cluster.



Thanks to Ahmet Alp Balkan for the diagrams

Demo

- ***Defining a Pod***
- ***Creating a ReplicaSet***
- ***Creating a Deployment***
- ***Creating a Service and exposing it***

I want to be able to deploy and share my app everywhere consistently, and manage it as a single entity regardless of the different parts.

Reference - https://github.com/IBM/helm101/tree/master

Deploying an App – *kubectl* Way

- Let’s see what it takes to deploy an app on a running Kubernetes cluster
  - There will be lot’s of YAML Kubernetes manifest files
    - Ex:-
      - Application deployment and service configuration
      - Redis master deployment and service configuration
      - Redis slaves deployment and service configuration
  - Using the Kubernetes client, *kubectl*
    - Create Deployment
    - Manage Deployment

Refer - https://github.com/IBM/guestbook/tree/master/v1

Reference - https://github.com/IBM/helm101/tree/master

Deploying an App – *kubectl* Way – Pain Points

- CI/CD pipeline
  - *kubectl* deployments are not easy to configure, update and rollback
    - Deploying app to dev/test/production may require different configuration
      - Update deployment e.g. update with a new image
      - Change the configuration based on certain conditions
      - A different serviceType is needed in different environments (e.g. NodePort/LoadBalancer)
      - Need for rollback
      - Need of having multiple deployments (e.g. multiple Redis deployments)
  - Requires to track your deployment and modify YAML files (can be error prone)
  - Does not allow multiple deployments without updating metadata in manifest files
- Share your deployment configurations with your friend, team or customer?
  - You need to share many files and related dependencies
  - Your users are required to have knowledge of deployment configuration

Reference - https://github.com/IBM/helm101/tree/master

Here Comes Helm

- Deploying an app – Helm Way
  - No expertise of Kubernetes deployment needed as Helm hides Kubernetes domain complexities
  - Helm packages all dependencies
  - Helm tracks deployment making it easy to update and rollback
  - Same workload can be deployed multiple times
    - Helm allows assigning workload release names at runtime
  - Easy to share

Reference - https://github.com/IBM/helm101/tree/master

What is Helm?

- Helm is a tool that streamlines installation and management of Kubernetes applications
  - A tool or package manager for the Kubernetes, for deployment and management of applications into a Kubernetes cluster
  - Helm became a CNCF project in mid 2018
- It uses a packaging format called **charts**
  - A chart is a collection of files that describe Kubernetes resources
  - Think of Helm like apt/yum/homebrew for Kubernetes
- Helm is available for various operating systems like OSX, Linux and Windows
- Run Helm anywhere e.g. laptop, CI/CD etc.

Reference - https://github.com/IBM/helm101/tree/master

What Helm is NOT

- A fully fledged **system** package manager
- A configuration management tool like Chef, puppet etc.
- A Kubernetes resource lifecycle controller

Reference - https://github.com/IBM/helm101/tree/master

A chart is organized as a collection of files inside of a directory. The directory name is the name of the chart (without versioning information). Thus, a chart describing WordPress would be stored in a `wordpress/` directory.

Inside of this directory, Helm will expect a structure that matches this:

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  values.yaml         # The default configuration values for this chart
  values.schema.json  # OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file
  charts/             # A directory containing any charts upon which this chart depends.
  crds/               # Custom Resource Definitions
  templates/          # A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

Helm reserves use of the `charts/`, `crds/`, and `templates/` directories, and of the listed file names. Other files will be left as they are.

Reference - https://helm.sh/docs/topics/charts/

Demo – Guestbook Chart Deployment

- Check existing installation of Helm chart
  - ***helm ls***
- Check what repo do you have
  - ***helm repo list***
- Add repo
  - ***helm repo add helm101 https://ibm.github.io/helm101/***
- Verify that helm101/guestbook is now in your repo
  - ***helm repo list***
  - ***helm search helm101***
- Install
  - ***helm install helm101/guestbook --name myguestbook --set service.type=NodePort*** – follow the output instructions to see your guestbook application
- Verify that your guestbook chart is installed
  - ***helm ls***
- Check chart release history
  - ***helm history myguestbook***

Reference - https://github.com/IBM/helm101/tree/master

Demo – Guestbook Upgrades and Rollback

- First let’s see what we have
  - ***helm history myguestbook***
- Upgrade
  - ***helm upgrade myguestbook helm101/guestbook***
  - ***helm history myguestbook***
- Rollback
  - ***helm rollback myguestbook 1***
  - ***helm history myguestbook***

Reference - https://github.com/IBM/helm101/tree/master

Demo – Clean Up

- Remove repo
    - **helm repo remove helm101**
- Remove chart completely
    - **helm delete --purge myguestbook**
        - Delete all Kubernetes resources generated when the chart was instantiated

Reference - https://github.com/IBM/helm101/tree/master

Another Demo!!!

https://github.com/rav94/devops-in-practice

References

- https://kubernetes.io/docs/concepts/overview/components/
- https://helm.sh/docs/intro/quickstart/