**Basics**

Container Evolution

Container Runtime Engine

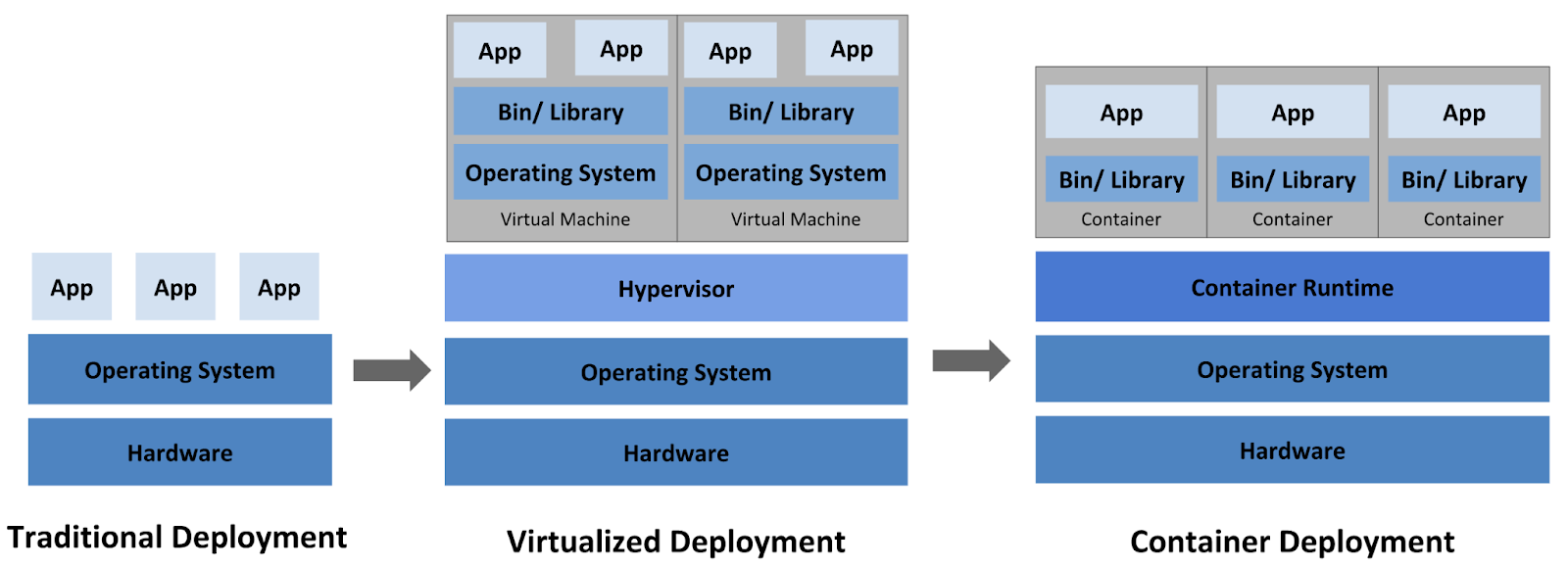
Docker Basics

Creating application images and tagging

Continuous Integration with Images

Basics

# Container Evolution



Containerized App -

What’s inside? - App + Library + Bin

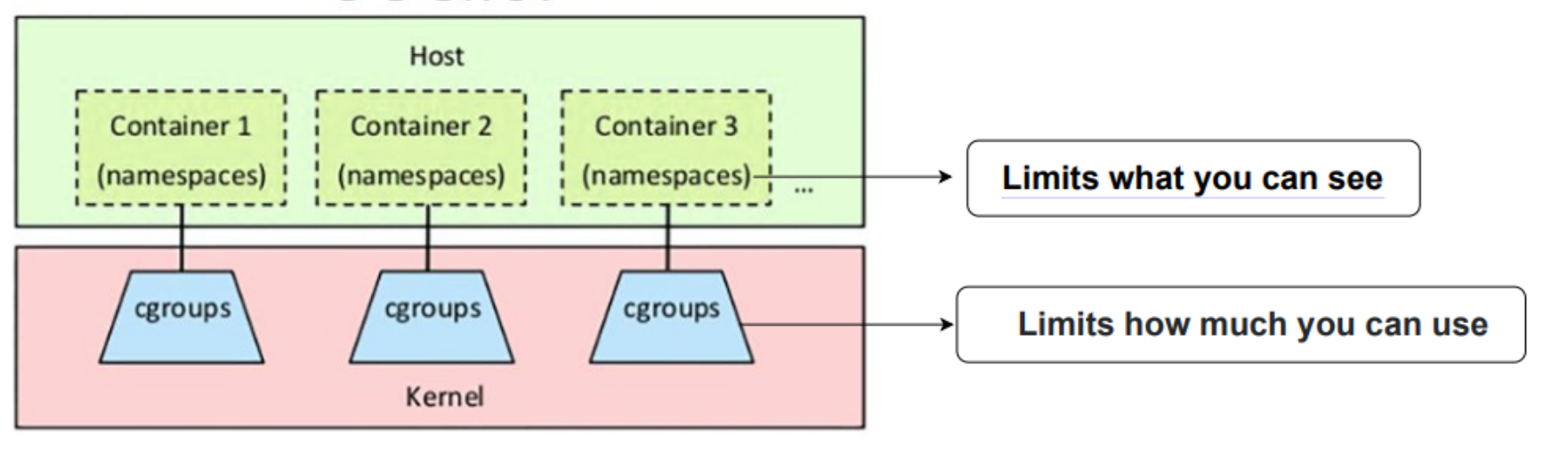
Image - App bundled with Dependencies

Container - Image deployed as Container

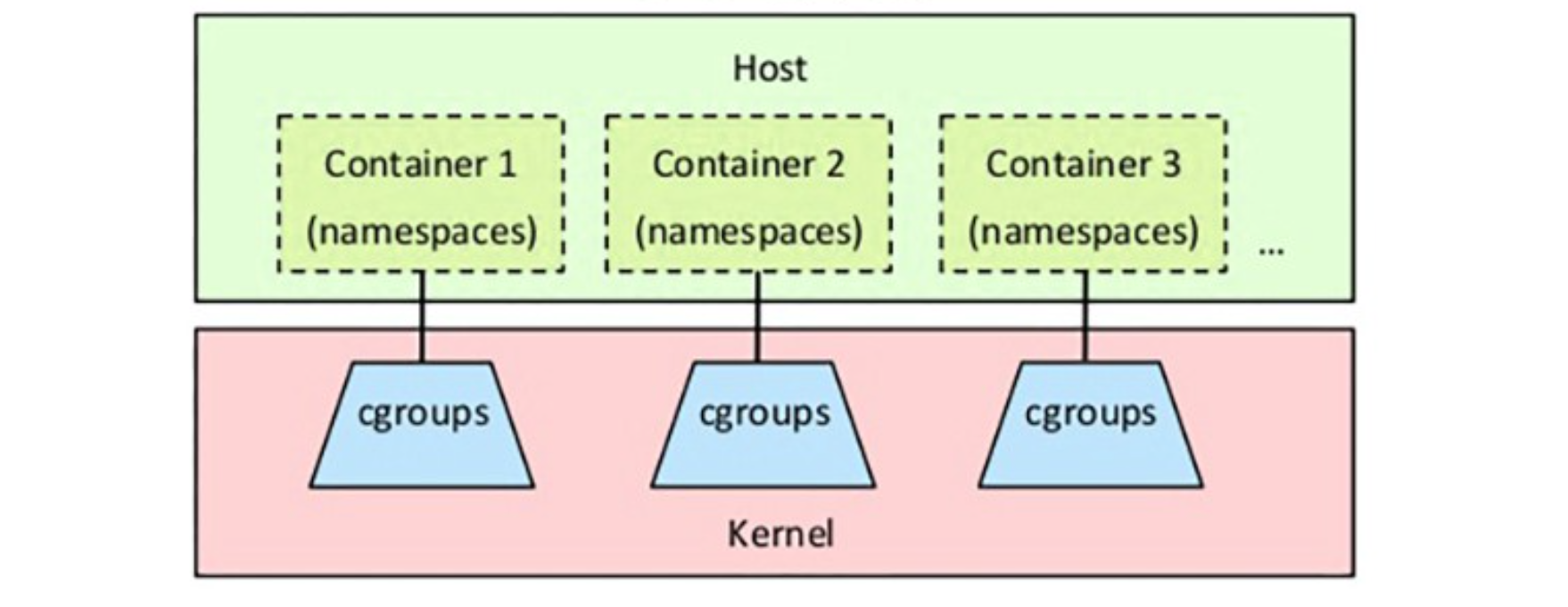
Deployment differs based on env - Dev, Staging, Prod

Envs, Configurations(Port, DB), Secrets, Resource limits, etc.

# Namespaces & CGroups

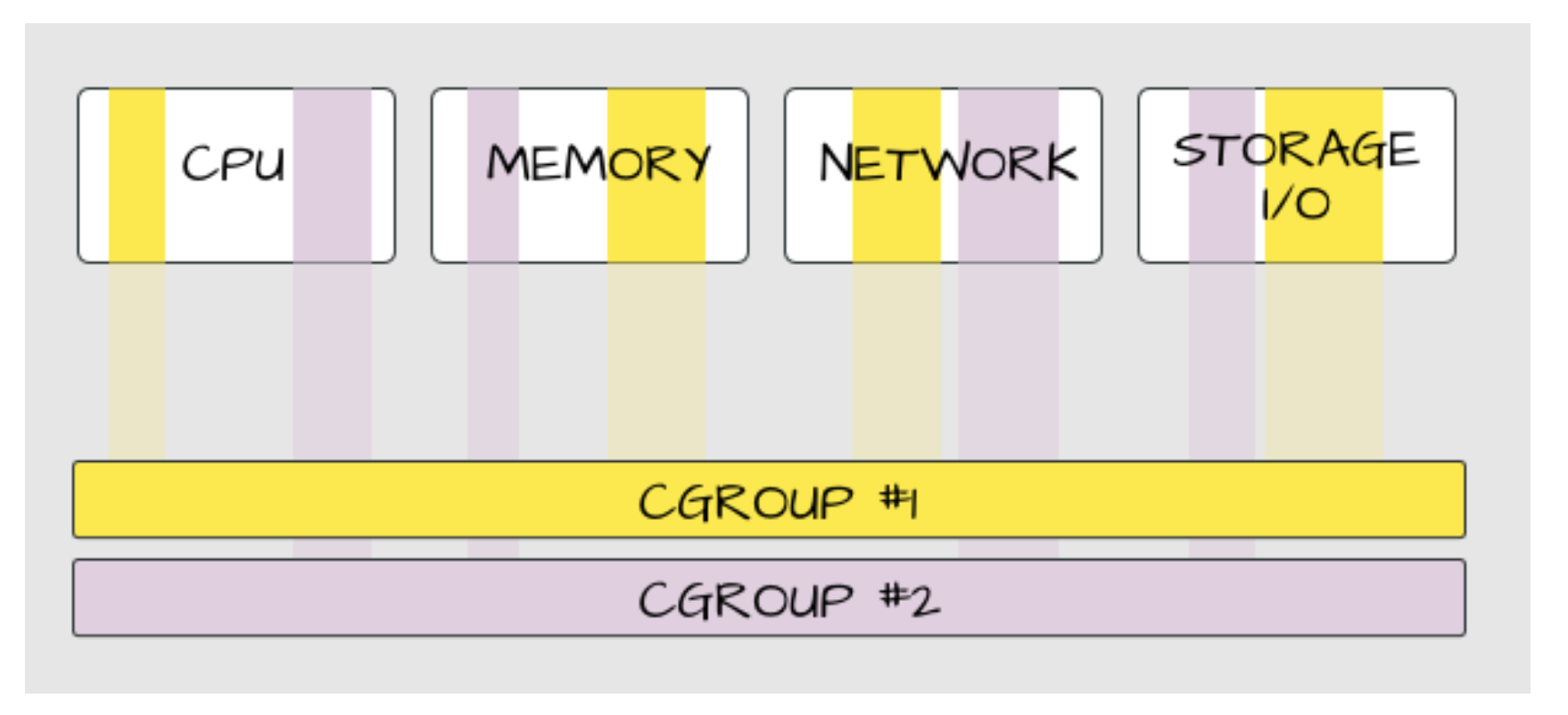


**Namespaces**



* Namespaces provide a layer of isolation for containers.
* Each aspect of a container runs in a separate namespace and its access is limited to that namespace.
* When you run a container, Docker creates a set of namespaces for that container.
* Namespace makes processes running inside that namespace believe they have their own instance of that resource.
* A namespace can limit visibility to certain process trees, network interfaces, user IDs, or filesystem mounts.

**CGroups**



* A control group (cgroup) is a Linux kernel feature that limits an application to a specific set of resource usage (CPU, memory, disk I/O, network, and so on)
* Control groups allow Docker Engine to share hardware resources to containers and optionally enforce limits and constraints.
* For example, you can limit the memory available to a specific container.

Cgroups involve resource metering and limiting:

Memory, CPU, Storage I/O, Network

# Docker Lab

#Images and containers

docker pull nginx

=> Pulls latest version, see all versions -<https://hub.docker.com/_/nginx?tab=tags>

docker run --name nginx -dt -p 80:80 nginx

docker ps

docker stop nginx

docker ps

docker ps -a

docker start nginx

docker ps

docker exec -it nginx /bin/sh

docker rm nginx

docker ps -a

docker images

docker rmi nginx

docker images

# Cleanup

## Containers

docker rm $(docker ps -qa)

## Images

docker image prune -a

vi app.js

# Copy the below content and paste it in app.js file

const http = require('http');

const hostname = '0.0.0.0';

const port = 80;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello Docker Chief\n');

});

server.listen(port, hostname, () => {

console.log('Server running at http://%s:%s/', hostname, port);

});

process.on('SIGINT', function() {

console.log('Caught interrupt signal and will exit');

process.exit();

});

# save the file with :wq

# vi Dockerfile

# Use an official Node runtime as the parent image

FROM node:6

# Set the working directory in the container to /app

WORKDIR /app

# Copy the current directory contents into the container at /app

ADD . /app

# Make the container's port 80 available to the outside world

EXPOSE 80

# Run app.js using node when the container launches

CMD ["node", "app.js"]

# save the file with :wq

# Docker Commands

docker build -t node-app:0.1 .

docker images

docker run -p 4000:80 --name my-app -dt node-app:0.1

curl [http://localhost:4000](http://localhost:4000/)

docker logs my-app

docker stop my-app && docker rm my-app

docker run -p 4000:80 --name my-app02 -d node-app:0.1

docker ps

docker logs [container\_id]

follow the log's output as the container is running, use the -f option,

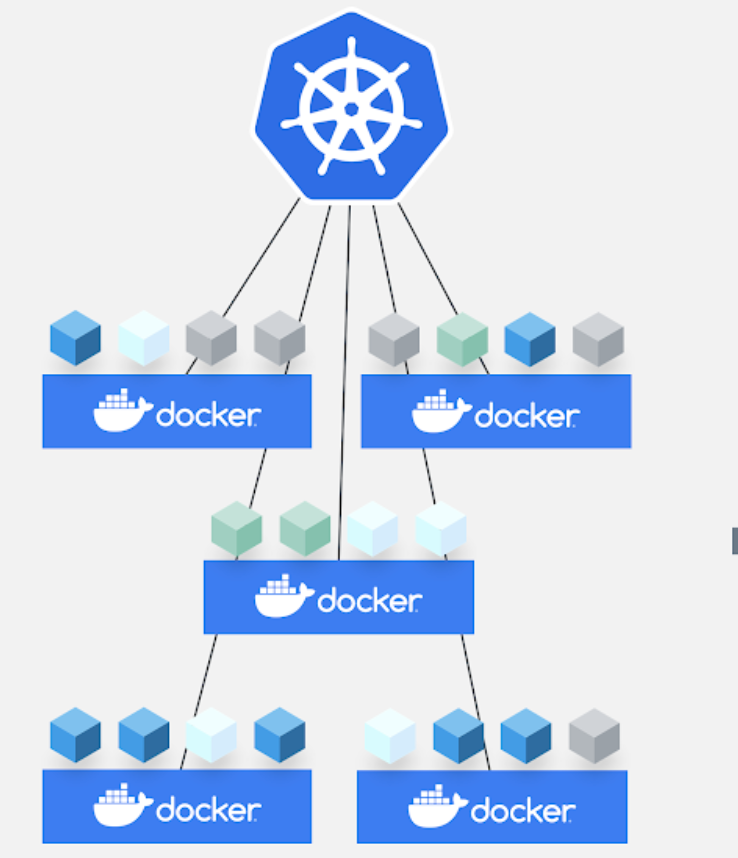
docker logs -f [container\_id]

examine a container's metadata in Docker by using Docker inspect:

docker inspect [container\_id]

docker exec -it my-app02 '/bin/bash'

How do we manage containers in different nodes?



Core Concepts

Kubernetes Architecture

Master Node initialisation

Worker Nodes initialization - Worker1 & 2

Basic commands of Kubernetes

# Kubernetes Architecture

Master

API Server

Scheduler

Controller Manager

Cloud Controller Manager(CCM)

Etcd

Worker

Kubelet <-> Dockerd

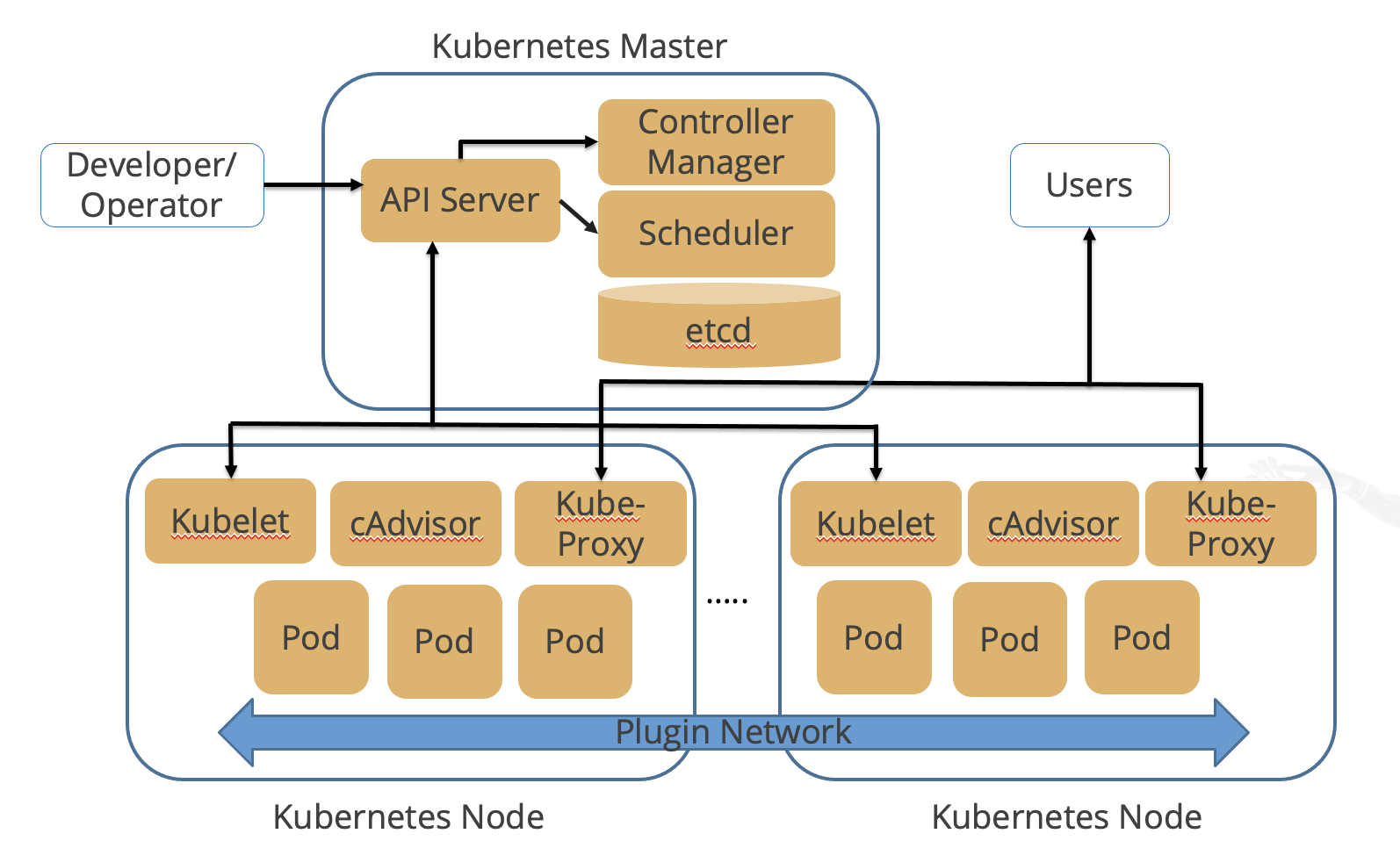
Kube-proxy

Pods - Containers - App running

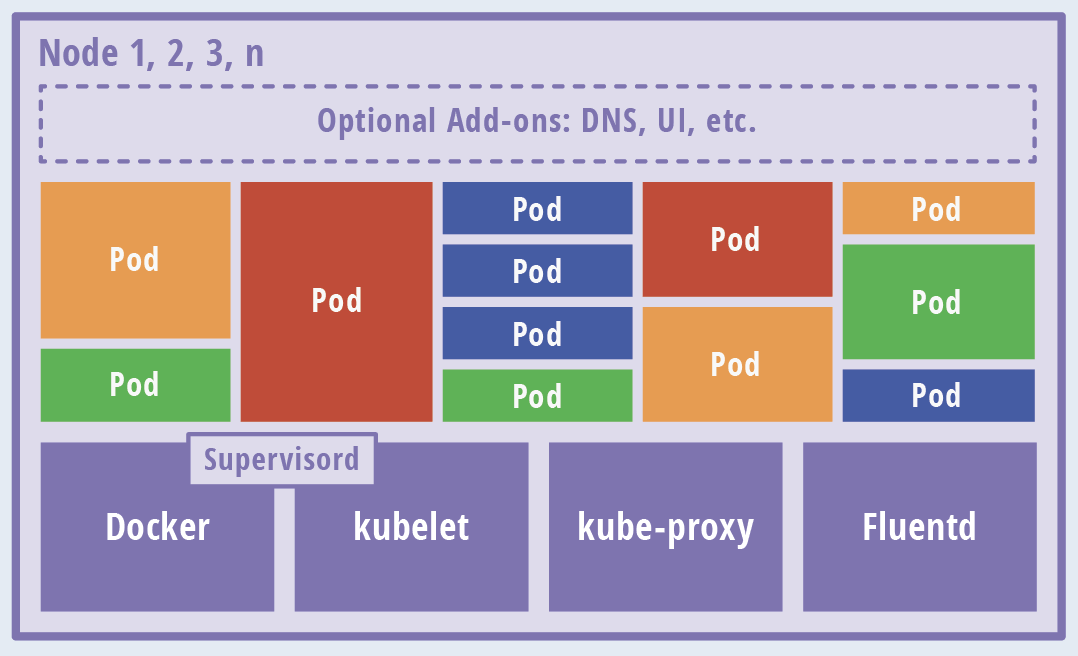
Operator - Access API-Server for k8s calls - Create, Update, Read, Delete resources

K8s client - UI, API, CLI - kubectl

User - App users - Access App deployed in k8s



Node View - Many sizes of pods deployed



# 