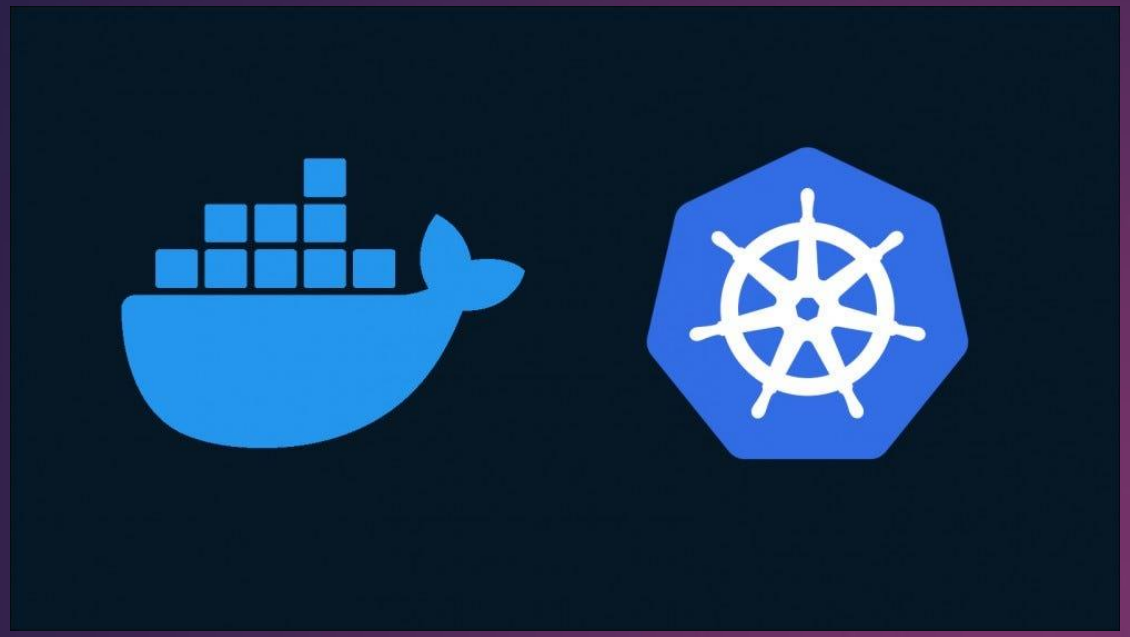
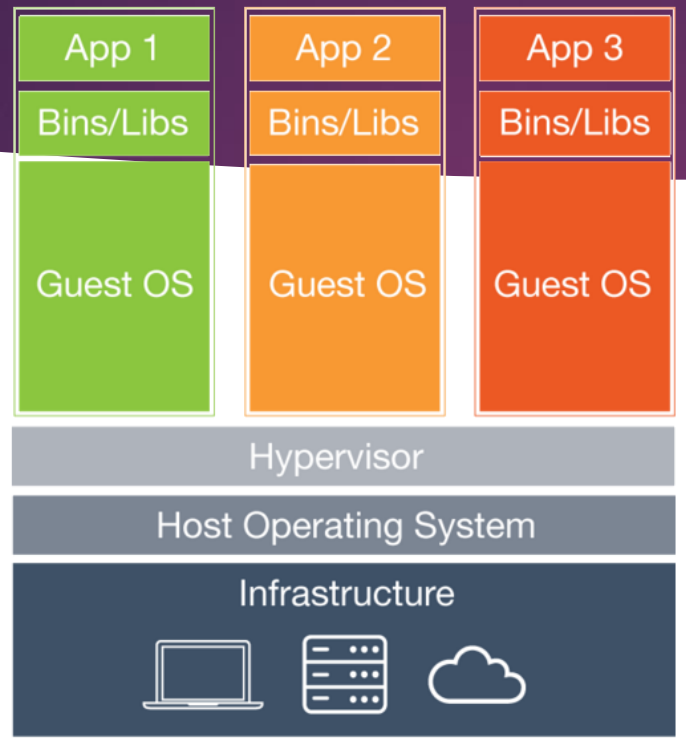
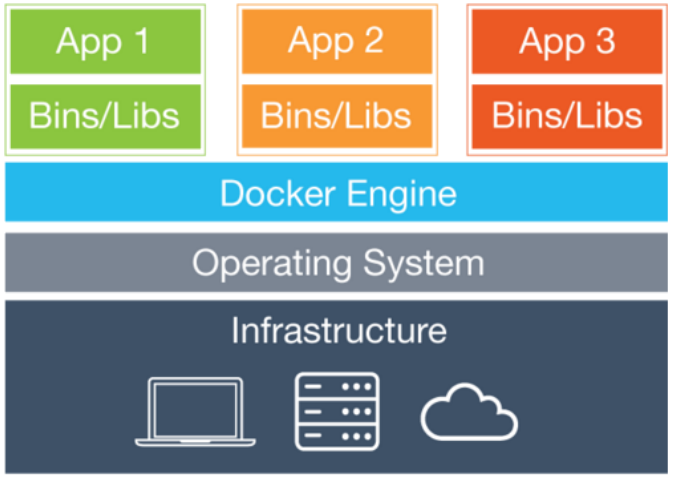


Kubernetes (K8s) § Docker

Virtual machines Containers

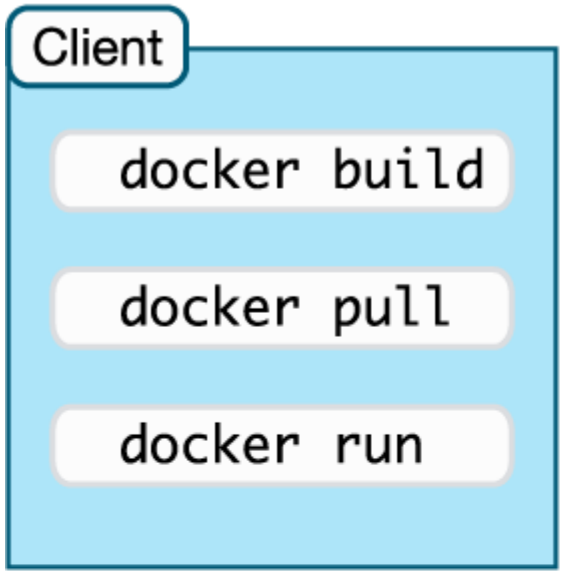
Traditional Deployment

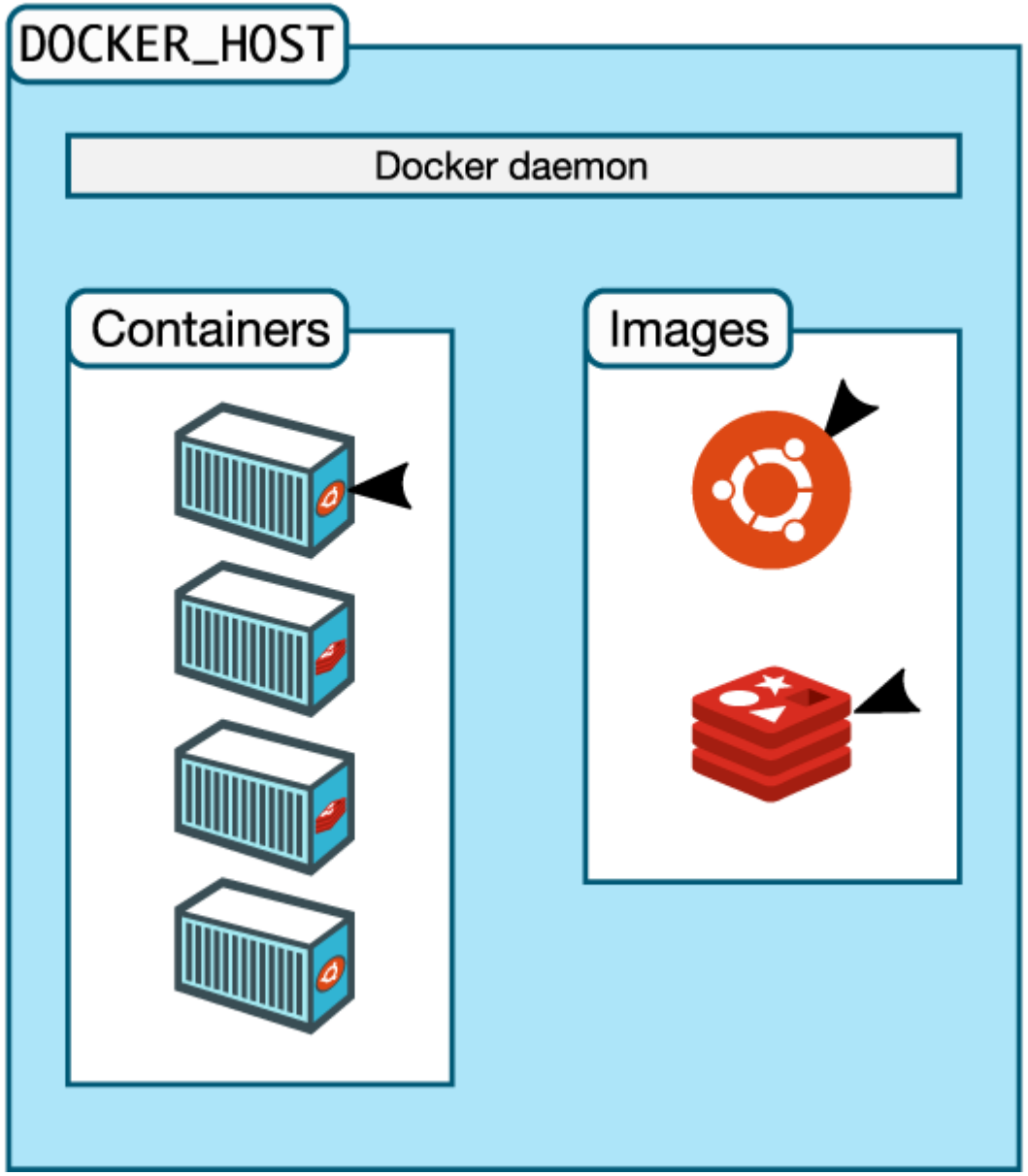




Всяка виртуална машина включва приложение, необходимите му библиотеки, както и цялата операционна система

Контейнерите включват приложенията и всичко, което им е необходимо, но делят ядрото с останали контейнери. Те работят като изолирани процеси в потребителско пространство на сървъра. Също така не са прикачени към определена инфраструктура – Docker контейнерите работят на всеки компютър, на която и да е инфраструктура и във всеки облак.





Контейнерите са нов вид виртуализация, където само се виртуализира горната половина на операционната система, така наречения shell и се използва ядрото на основната операционна система kernel (така не се налага да се виртуализира/емулира и хардуера)

Стандарт за контейнери - OCI

Създадена през юни 2015 г. от Docker и други лидери в контейнерната индустрия, OCI в момента съдържа три спецификации: спецификация за време на изпълнение (runtime-spec), спецификация на изображението (image-spec) и спецификация за разпространение (distribution-spec). Спецификацията за време на изпълнение очертава как да стартирате „пакет на файлова система“, който е разопакован на диска. На високо ниво внедряването на OCI ще изтегли OCI изображение, след което ще разопакова това изображение в пакет на OCI Runtime файлова система.

Сигурност ???

- Класическа архитектура
- VM
- Контейнери

Тенденции в съвместимостта на технологиите ???

OCI -
OCI е органът, отговорен за определяне на стандартите за контейнери. Неговата работа е допринесла за улесняване на оперативната съвместимост между различните компонентни технологии. Спецификацията на изображението на OCI определя как да изглежда контейнерът. Спецификацията за изпълнение осигурява интерфейс за работещи контейнери.

Cloud и контейнери

Облачните термини “Platform As A Service” - PaaS или VE (Virtual Environment), става дума за контейнери!

IaaS

DevOps е подход, който комбинира разработката на софтуер (Development) и операциите (Operations) с цел подобряване на скоростта, ефективността и качеството на доставката на софтуерни приложения. Ето някои от основните характеристики на DevOps:

1. Култура на сътрудничество: DevOps насърчава сътрудничеството и комуникацията между разработчици, оператори и други заинтересовани страни. Това включва споделяне на знания, работа в екип и създаване на общи цели.
2. Автоматизация: DevOps се базира на автоматизацията на процесите, като например автоматизирано тестване, сбор и доставка на приложения. Това помага да се намали човешката грешка, да се ускори доставката и да се подобри качеството на софтуера.
3. Инфраструктура като код: DevOps насърчава използването на инструменти и практики за управление на инфраструктурата като код. Това позволява дефиниране и управление на инфраструктурни ресурси чрез код, което улеснява автоматизацията и повтаряемостта.
4. Контейнеризация: DevOps използва контейнери, като например Docker, за пакетиране и доставка на приложения. Контейнерите осигуряват консистентност и изолация на приложенията, което улеснява разработката и доставката на софтуера.
5. Непрекъсната интеграция и доставка: DevOps насърчава практиките на непрекъсната интеграция (CI) и непрекъсната доставка (CD), които включват автоматизирано тестване, сбор и доставка на приложения. Това позволява бърза и честа доставка на нови функции и поправки на грешки.
6. Мониторинг и логване: DevOps включва мониторинг и логване на приложенията и инфраструктурата, за да се осигури видимост и разбиране на тяхното поведение и производителност. Това помага да се идентифицират проблеми и да се предприемат бързи мерки за отстраняването им.

Kubernetes

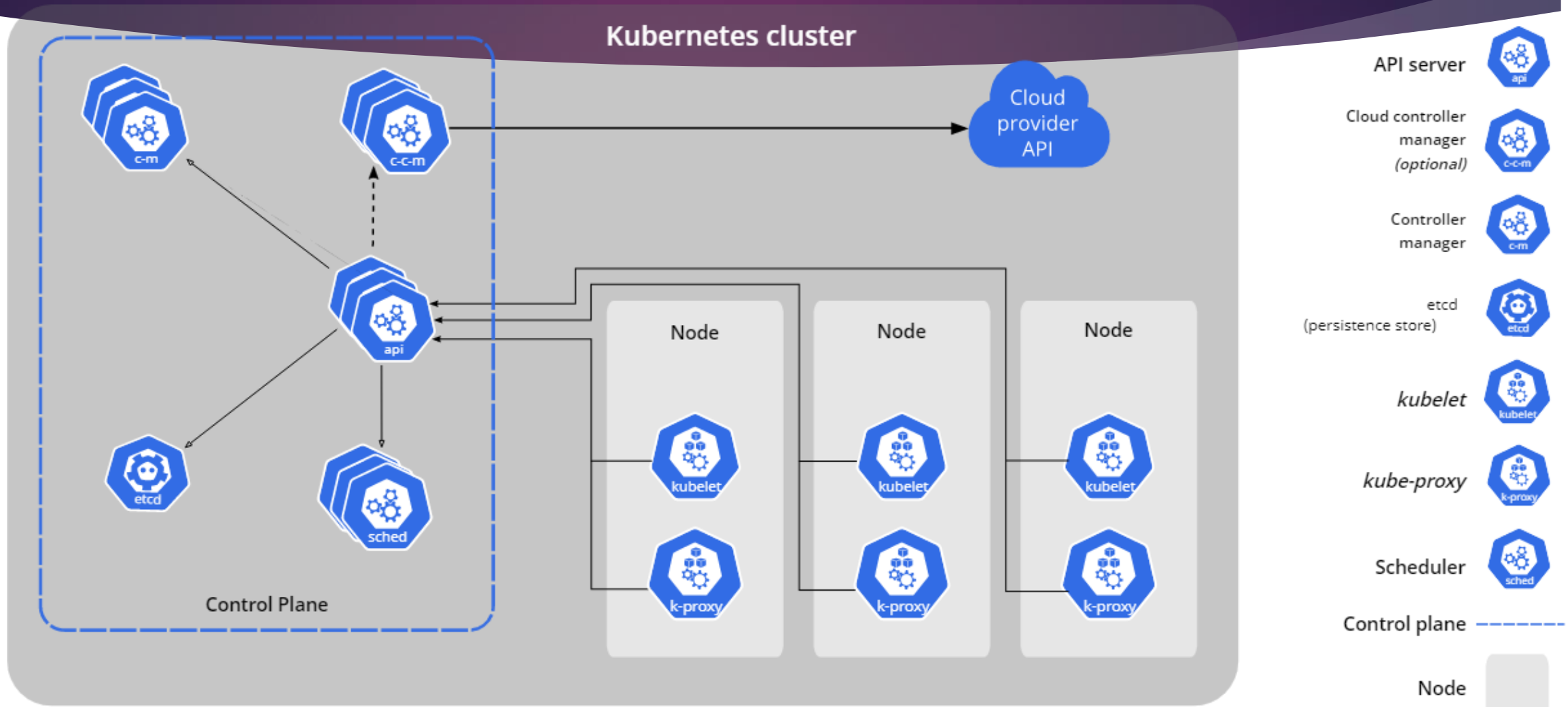
Kubernetes е оркестрационна и автоматизационна платформа, която може да използва Doker, lxc или Kata контейнери като основа. Kubernetes е разработена от Google за вътрешна употреба, но вече стандартизирана и може да се използва от всеки. Kubernetes служи за оркестрирането и контрола на много контейнери, например може за секунди да се вдигнат 100 контейнера с уеб сървъра или 100 контейнера с бази данни и дори с проби да следи дали уеб услугата работи върху контейнерите и ако има проблем контейнера ще бъде пресъздаден.

Kubernetes е оркестратор на контейнери с отворен код за автоматично инсталиране на приложен софтуер и по-нататъшното му мащабиране и управление.
Дата на пускане: 9 септември 2014 г.
Първоначален автор: Гугъл
Език за програмиране: Go

Go

```
package main
import ("fmt")

func main() {
  for i:=0; i < 5; i++ {
    fmt.Println(i)
  }
}
```

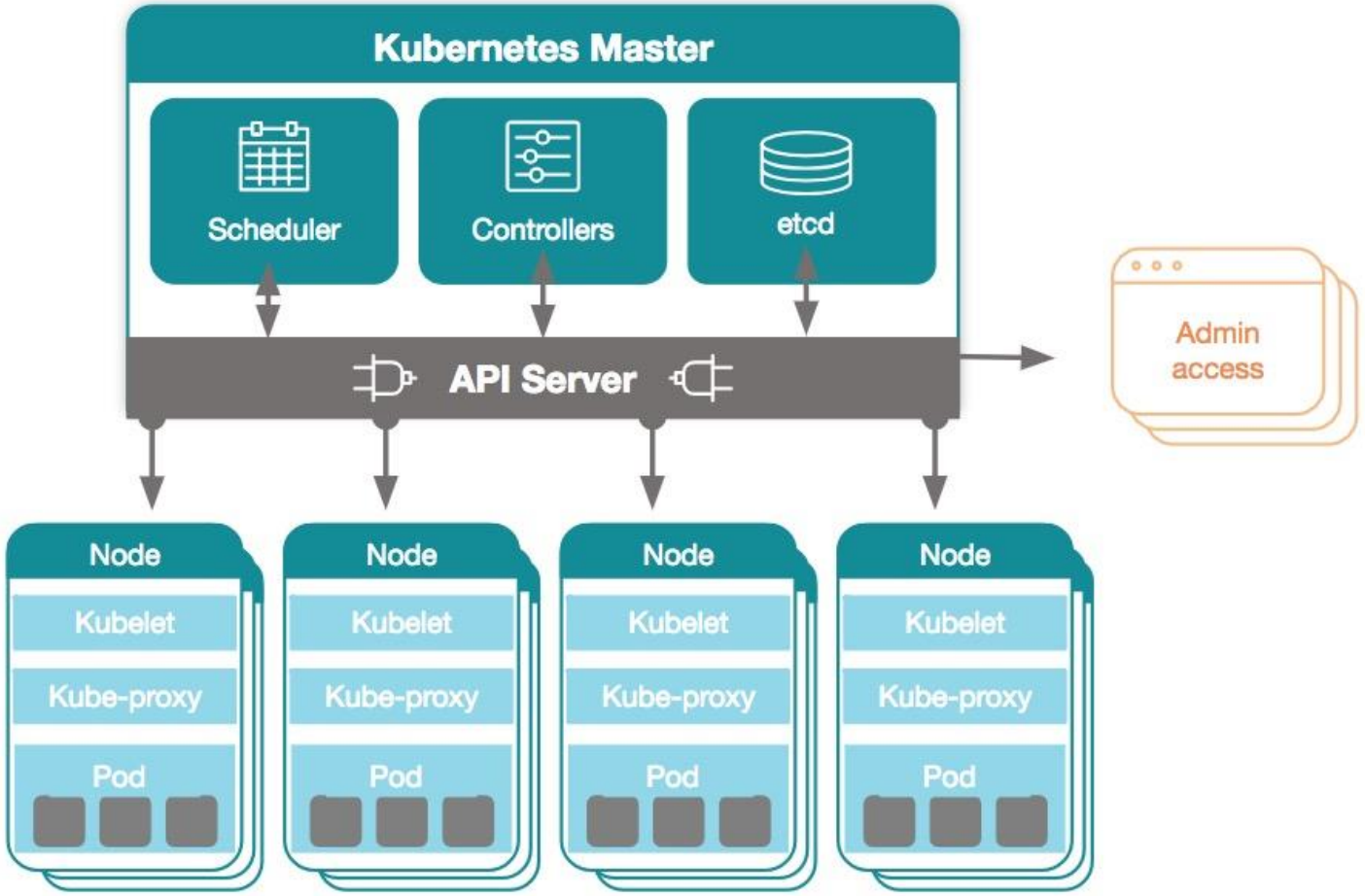

Kubernetes cluster
c-m
c-c-m
Cloud provider API
api
etcd
sched
Control Plane
Node
kubelet
k-proxy
Node
kubelet
k-proxy
Node
kubelet
k-proxy
API server
Cloud controller manager (optional)
Controller manager
etcd (persistence store)
kubelet
kube-proxy
Scheduler
Control plane
Node

Kubernetes Components

Когато внедрите Kubernetes, получавате клъстер. Клъстерът на Kubernetes се състои от набор от работни машини, наречени възли, които изпълняват приложения в контейнери. Всеки клъстер има поне един работен възел. Работният възел хоства Pods съдържащ контейнери.

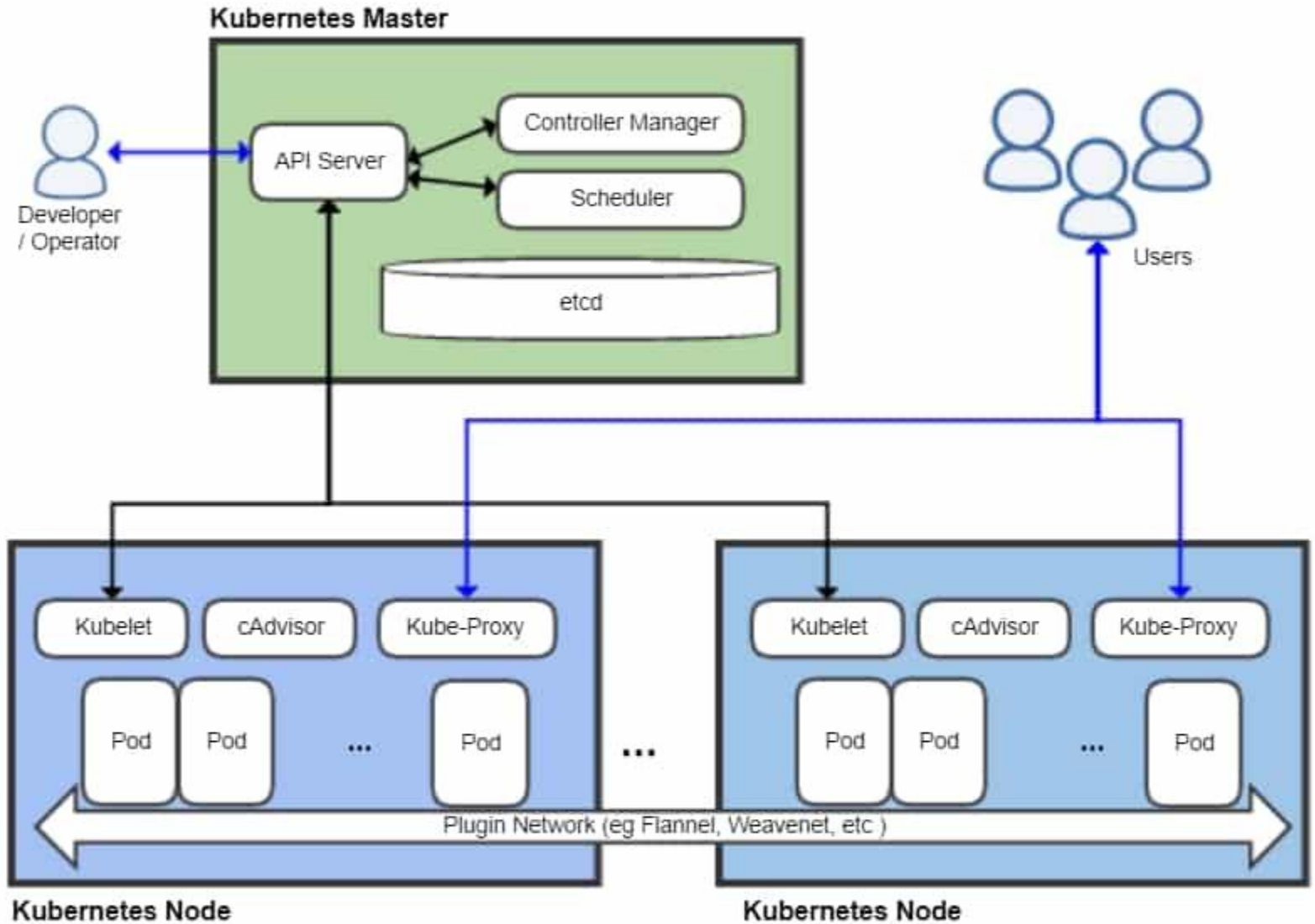
Линукс сървърите, които могат да са виртуални или физически, участващи в клъстара, се наричат в контекста **nodes**

Kubernetes Architecture



Кубернетес (Kubernetes) е отворен източник система за оркестриране и управление на контейнеризирани
приложения. Тя се състои от няколко основни компонента, които работят заедно, за да осигурят автоматизирано и мащабируемо управление на контейнерите. Ето някои от основните компоненти на Кубернетес:

1. Мастър компоненти (Master Components):
   - kube-apiserver: Предоставя API интерфейс за комуникация с Кубернетес клъстъра и управление на ресурсите.
   - kube-controller-manager: Управлява различни контролери, които следят състоянието на клъстъра и вземат действия при необходимост.
   - kube-scheduler: Определя на кой работен възел да бъде пуснат контейнерът, взимайки предвид ресурсите и изискванията на приложението.
2. Работни възли (Worker Nodes):
   - kubelet: Агент, който работи на всеки работен възел и управлява контейнерите, стартира ги, спира и следи техните състояния.
   - kube-proxy: Отговаря за мрежовата комуникация между контейнерите и извън клъстъра.
3. Еталонни компоненти (Add-on Components):
   - DNS: Предоставя вътрешно разпознаваеми имена на контейнерите в клъстъра.
   - Dashboard: Графичен интерфейс за управление на Кубернетес клъстъра.
   - Ingress Controller: Управлява входящия трафик към приложенията в клъстъра.
   - Мониторинг и логване: Различни компоненти, които предоставят мониторинг и логване на приложенията и инфраструктурата.


Kubernetes Master
API Server
Controller Manager
Scheduler
etcd
Developer
/ Operator
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Pod обектът е създаденото приложение и един pod може да има един или няколко контейнери, като обикновено е само един контейнер в pod, само ако искаме контейнерите да си споделят папки, тоест например единия контейнер да пише в една папка, а другия да чете написаното, то тогава ще сложим 2 или повече контейнери в pod . **Краткотрайни** логически единици.

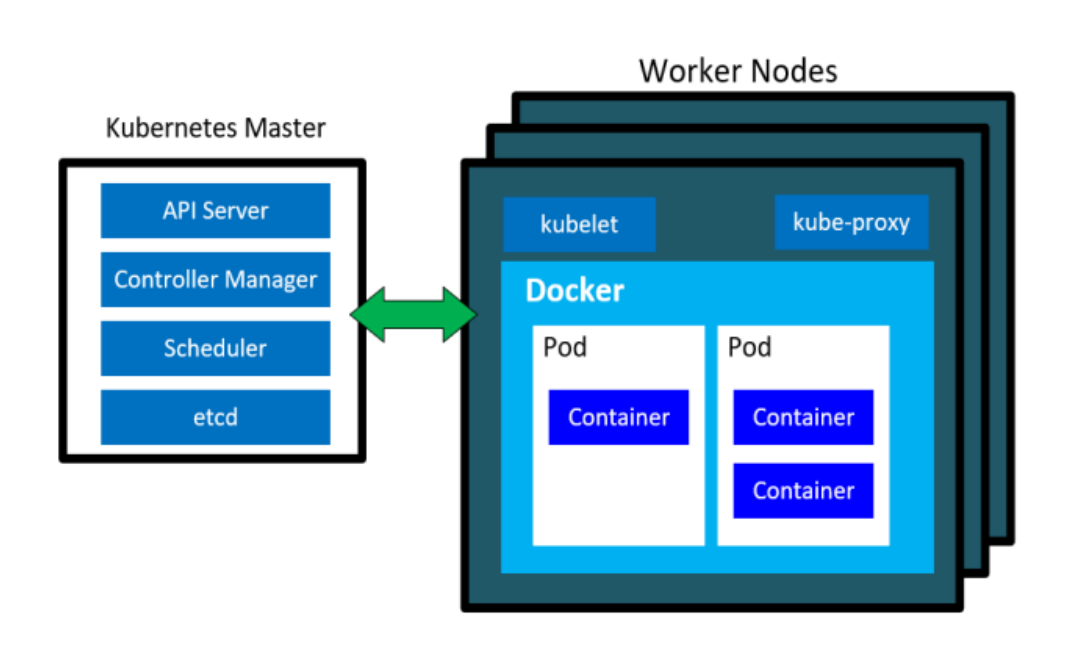
Конролер, който се създава да следи създадената група от pod обекти наречени "deployment" и ако има проблем с някой pod от deployment групата, контролера ще унищожи pod/контейнера и отново ще го направи. Когато създаваме deployment групата се създава и replication set обект, който следи дали броя на контейнерите отговаря на желания брой

(Kube-proxy) обект, който служи за резпределяне на трафик между обектите и чрез този обект можем да направим така, че на външен публичен интернет адрес, проксито да отговаря и пренасочва трафика към групата от pod обекти, наречени deployment, чрез използване на service обекта. Кубе проксито служи за комутатор /маршрутизатор във Kubernetes средата.

kube-apiserver е проектиран да се мащабира хоризонтално – т.е. мащабира се чрез внедряване на повече инстанции. Можете да стартирате няколко инстанции на kube-apiserver и да балансирате трафика между тези инстанции.

kube-scheduler Компонент, който следи за новосъздадени подове (Pods) без присвоен възел и избира възел, върху който да работят. Факторите, взети под внимание при решенията за планиране, включват: индивидуални и колективни изисквания за ресурси, хардуерни/софтуерни ограничения, спецификации за местоположение на данните, смущения между работните натоварвания и крайни срокове.

Kubernetes service обектът можем да свържем групата от pod обекти в deployment с външния свят или просто да направим виртуален клъстерен адрес, с цел други deployment групи да говорят с този deployment група, тоест вътрешна комуникация между pod обекти от различни deployment групи. Kubernetes service е един вид правила, които kube-proxy използва.



etcd - хранилище на ключови стойности, използвано като резервно хранилище на Kubernetes за всички клъстерни данни.

kubelet - агент, който работи на всеки възел в клъстера. Той гарантира, че контейнерите работят в Pod.

YAML (Yet Another Markup Language) е език за описание на данни, който се използва широко в Kubernetes и други инструменти за конфигурация и оркестриране на инфраструктура. YAML е предназначен да бъде лесно четим и разбираем от хора, както и лесно обработваем от компютри.

Ето някои основни характеристики на YAML:

1. Синтаксис: YAML използва интуитивен синтаксис, базиран на отстъпи и ключови думи, за да организира и структурира данните. Това прави YAML много четим и лесен за разбиране.
2. Структурирани данни: YAML позволява дефинирането на структурирани данни, като списъци, речници и вложени обекти. Това позволява да се представят сложни конфигурации и данни.
3. Коментари: YAML поддържа коментари, които могат да бъдат използвани за документиране на кода или за добавяне на пояснения.
4. Разширяемост: YAML позволява на потребителите да дефинират свои собствени типове данни
и да ги използват в конфигурациите си. Това прави YAML много гъвкав и разширяем.

Ето пример за YAML файл, който дефинира прост под в Kubernetes:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    ports:
    - containerPort: 80
```

Този YAML файл дефинира под с име “my-pod”, който съдържа контейнер с име “my-container”,
базиран на образа “nginx” и отваря порт 80 за достъп. Когато този файл се подаде на Kubernetes,
той ще създаде под съгласно дефиницията в YAML файла.

YAML е мощен инструмент за конфигуриране и оркестриране на инфраструктура и е широко
използван в Kubernetes и други инструменти за DevOps.

Ето пример за по-сложна YAML дефиниция на под в Kubernetes:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
  - name: container1
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: container2
    image: redis:latest
    env:
    - name: REDIS_PASSWORD
      value: mypassword
  volumes:
  - name: shared-data
    emptyDir: {}
  - name: config-volume
    configMap:
      name: my-config
  nodeSelector:
    disk: ssd
```

В този пример имаме по-сложна YAML дефиниция на под. Подът има име “my-pod” и е маркиран с етикет “app: my-app”.

Подът съдържа два контейнера - “container1” и “container2”. “container1” използва образ на Nginx и експортира порт 80. “container2” използва образ на Redis и има дефинирана променлива на средата REDIS_PASSWORD със стойност “mypassword”.

Подът също така има два обема (volumes). “shared-data” е празен обем, създаден с помощта на emptyDir. “config-volume” е обем, свързан с ConfigMap с име “my-config”.

Този под също така има nodeSelector, който указва, че подът трябва да се изпълни на възел с диск от тип SSD.

Тази по-сложна YAML дефиниция позволява по-голяма гъвкавост и конфигурация на пода в Kubernetes, като включва етикети, променливи на средата, обеми и други параметри.

Go

Езикът Go (или Golang) е широко използван в сферата на Kubernetes. Go е избран като език за разработка на Kubernetes, защото езикът е компактен, бърз и ефективен за конкурентно програмиране. Всъщност, Kubernetes е написан на Go и използва някои от неговите ключови функции и библиотеки.

Ето някои начини, по които Go се използва в Kubernetes:

1. Клиенти на Kubernetes API: Go предоставя библиотека за клиенти на Kubernetes API, която позволява на разработчиците да взаимодействат с Kubernetes кластър от своя Go код. Това включва създаване, четене, обновяване и изтриване на ресурси в Kubernetes.
2. Контролери и оператори: В Kubernetes, контролерите и операторите са компоненти, които управляват и поддържат състоянието на ресурсите в кластъра. Много от тези контролери и оператори са написани на Go, използвайки библиотеки като “client-go” и “controller-runtime”.
3. Разработка на приложения: Go се използва широко за разработка на приложения, които се изпълняват върху Kubernetes. Това включва създаване на микросервизи, RESTful API и др.
4. Разработка на инструменти и разширения: Разработчиците използват Go за създаване на инструменти и разширения за Kubernetes. Това може да включва инструменти за мониторинг, логване, мащабиране, деплоймент и други.

Go е предпочитан език в сферата на Kubernetes, тъй като е лесен за използване, надежден и ефективен. Благодарение на тези причини, Go продължава да бъде предпочитан избор за разработчиците, които работят с Kubernetes.

Ето прост пример за работа с Kubernetes **стъпка по стъпка**:

Стъпка 1: Инсталиране на Kubernetes

- Инсталирайте Kubernetes върху вашия сървър или локална машина.

Можете да използвате инструменти като Minikube за локална инсталация или Kubernetes на публични доставчици на облачни услуги като Google Cloud Platform или Amazon Web Services.

Стъпка 2: Създаване на конфигурационни файлове

- Създайте конфигурационни файлове за вашите приложения и ресурси в Kubernetes.

Това може да включва файлове с разширение .yaml, които описват подове, услуги, репликации и други.

Стъпка 3: Пускане на приложение

- Използвайте командата kubectl за пускане на вашето приложение в Kubernetes като

изпълните командата **kubectl apply -f <име_на_файла>.yaml**. Това ще създаде необходимите ресурси в клъстъра.

Стъпка 4: Проверка на състоянието на клъстъра

- Използвайте командата kubectl get pods за да видите текущото състояние на подовете във

вашия клъстър. Това ще покаже списък с подове, техния статус и други подробности.

Стъпка 5: Мащабиране на приложението

- За да мащабирате приложението, използвайте командата kubectl scale deployment

<име_на_приложението> --replicas=<брой_реплики>. Това ще увеличи броя на репликите на приложението и ще гарантира, че се изпълн

Стъпка 6: Изтриване на приложение

Ето **примерен манифест в Kubernetes**, който дефинира приложение с два контейнера:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: container-1
      image: image-1:latest
      ports:
        - containerPort: 8080
    - name: container-2
      image: image-2:latest
      ports:
        - containerPort: 9000
```

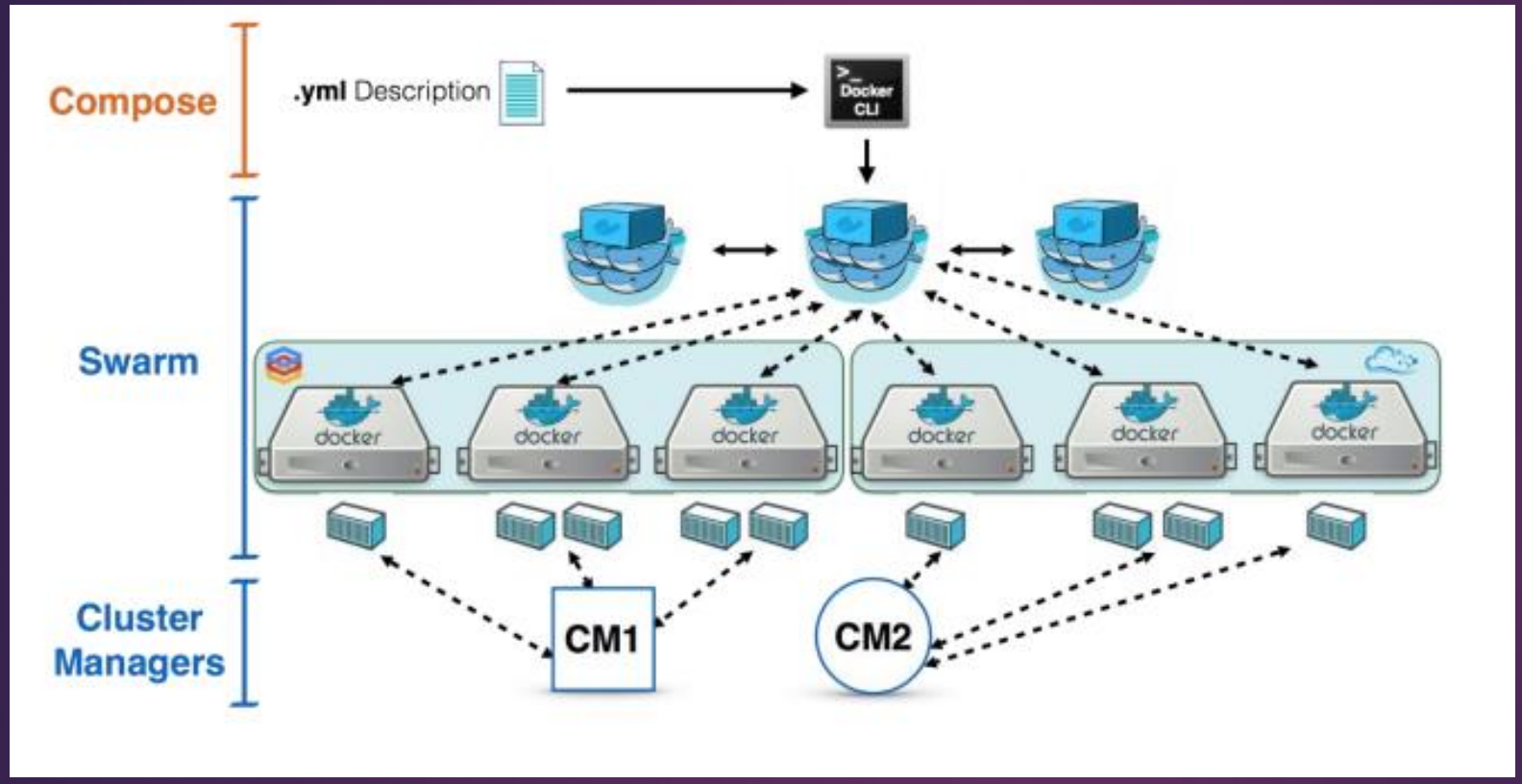
В този примерен манифест Pod с име “my-app” съдържа два контейнера. Първият контейнер с име “container-1” използва образ с име “image-1:latest” и слуша на порт 8080. Вторият контейнер с име “container-2” използва образ с име “image-2:latest” и слуша на порт 9000.

Този манифест може да бъде разширен с допълнителни опции и ресурси за всяка отделна контейнери, като например монтиране на томове, дефиниране на ресурсни ограничения и други. Също така, може да се използва и манифест за мрежова служба, ако контейнерите трябва да бъдат

Процесът на оркестрация с Kubernetes включва следните стъпки:

1. Дефиниране на клъстър: Първата стъпка е да се дефинира и настрои Kubernetes клъстър. Това включва инсталиране на Kubernetes мастър и работни възли, които ще изпълняват контейнерите.
2. Дефиниране на приложение: Следващата стъпка е да се дефинира приложението, което ще бъде разгърнато върху Kubernetes. Това включва създаване на конфигурационни файлове, наречени манифести, които описват контейнерите, службите, мрежите и другите ресурси, необходими за приложението.
3. Разгърняване на приложение: След като манифестите са създадени, те се изпращат към Kubernetes мастъра, който ги обработва и започва процеса на разгърняване на контейнерите. Kubernetes мастър създава и стартира контейнерите върху работните възли, като спазва дефинираните правила и ограничения.
4. Управление на приложение: След разгърняването на приложението, Kubernetes постоянно наблюдава състоянието на контейнерите и при нужда взима действия за поддържането на желаното състояние. Това включва мащабиране на броя на репликите, автоматично възстановяване при грешки, премахване на контейнери и други операции.
5. Мониторинг и логове: Kubernetes предоставя вградени инструменти за мониторинг и събиране на логове от контейнерите и приложенията. Това позволява на операторите да следят и анализират работата на приложението и да вземат действия при нужда.
6. Актуализация на приложение: Когато е необходимо да се актуализира приложението, Kubernetes предоставя механизми за нулево-до-минимално прекъсване на услугата. Това включва стъпки като създаване на нови контейнери с актуализиран код, постепенно прехвърляне на трафика към новите контейнери и спиране на старите контейнери.

Docker имат конкурент на **Kubernetes**, **Docker Swarm**. Чрез Docker Swarm можем да обединим много много Линукс сървъри в клъстер и да стартираме много контейнери върху тях



Чрез Yaml файловете лесно можем да опишем един deployment обект, без да пишем дълги kubectl команди.
(много подобен на Python езика за програмиране).

Red Hat Openshift разработен от Red Hat Linux и е надграждане над Kubernetes Red Hat Openshift има същите опции като Kubernetes и може да се правят deployments, pods, services, Ingress и тн Red Hat Openshift обаче доста надгражда над Kubernetes с няколко основни възможности:

- Source to image е функция, която позволява директно да дръпнем код от git repository и Openshift сам избира какъв темплейт да използва и да вкара кода в контейнера направен от този темплейт. Идеята е, че ако приложението ни използва Питон, то Openshift ще избере темплейт с инсталиран Питон. Openshift автоматично открива кой език използва кода ни в Git.

Сравнение Docker Swarm § Kubernetes

•Docker Swarm е **по-лесно** за конфигуриране поради неговата простота. Освен това е по-лесно да се интегрира в екосистемата на Docker.
•Вместо това **Толерантност към повреди** Kubernetes е по-висока, което може да бъде по-положително в среди като високодостъпни сървъри.
•Docker Swarm е **по-бързо** относно разполагането и разширяването на контейнери.
•Kubernetes от своя страна предлага **по-големи гаранции** към състоянията на клъстера.
•El **балансиране на натоварването** в Kubernetes позволява по-добър баланс, въпреки че не е автоматичен, както в Docker.
•Kubernetes предлага **по-добра гъвкавост**, дори в сложни приложения.
•Docker Swarm ще поддържа до 2000 **възли**, в сравнение с 5000 за Kubernetes.
•Kubernetes е **оптимизиран** за много малки клъстери, докато Dockers е за голям клъстер.
•Kubernetes е **по-сложен**, По-опростен докер.
•Kubernetes може да позволи **споделяне на места за съхранение** между всеки контейнер, докато Docker е по-ограничен и се споделя само между контейнери в една и съща под.
•Docker Swarm позволява да се използва **софтуер на трети страни** за регистриране и наблюдение Kubernetes включва свои собствени вградени инструменти.
•Docker Swarm е ограничен до 95000 **контейнери**, докато Kubernetes може да поддържа до 300000.
•Docker има подкрепа от голяма **общност,** Kubernetes също има подкрепата на компании като Microsoft, Amazon, Google и IBM.
•Docker се използва от **фирми** като Spotify, Pinterest, eBay, Twitter и др. Докато Kubernetes предпочитат 9GAG, Intuit, Buffer, Evernote и т.н.