7.22

> 确定学习的目标：
>
> 语言基础：https://github.com/unknwon/the-way-to-go_ZH_CN/blob/master/eBook/directory.md
> 容器基础：https://yeasy.gitbook.io/docker_practice
> 集群基础：https://www.bookstack.cn/read/kubernetes-handbook/SUMMARY.md
>
> 以集群基础为主线先了解平台能力，容器基础与语言基础附带进行了解。

开始学习：

k8s主要作用:**容器编排调度**

云计算的本质：**配置资源的方式**

本组的开发任务是开发PaaS平台。

k8s的service适合于微服务。

Cloud Native：云原生。

k8s是CNCF（Cloud Native Computing Foundation）最重要的组件之一，用户通过描述集群的架构，定义服务的最终状态，k8s将系统自动达到和维持在这个状态。

k8s中每个**pod都有独立的IP、存储**，每个运行在pod中的应用不必关系端口是否重复，只需在service中指定端口，集群内的service通过配置互相发现。

每个**容器**都是一个**进程**，通过增加容器的副本数实现并发。

k8s优秀的Pod生命周期控制，在k8s中可以创建多个namespace，使用相同的镜像可以很方便的复制一套环境，可以让开发与线上环境等价。

日志：把日志当作事件流，使用stdout输出并收集汇聚，可以通过如ES等进行查看

k8s的master components （如API server、controller manager、scheduler）可以根据**集群的大小**（）决定是放在一台机器上，还是分布在多台机器上。大型集群上master组件会分布在多台机器上，以提高性能和可靠性

集群：一组主节点和工作节点。集群大小：集群中包含的节点数量和运行的工作负载的多少。

基本概念

- 容器：独立运行的、轻量级的虚拟环境，通常使用Docker容器
- 节点：集群中的一台计算机，可以是物理机或虚拟机
- Pod：K8s中最小的部署单元，一个Pod可以包含一个或多个容器
- 集群(Cluster)：一组节点组成的集合，用于运行容器化应用程序

核心组件

Master Components

- API Server：集群的控制入口，负责接受和处理所有的API请求
- etcd（/etc 存放配置文件 和d distributed 分布的）：键值存储，保存集群的所有**配置**信息和**状态**数据
- Controller Manager：负责管理集群的各种控制器，如节点控制器、复制控制器

- Scheduler：负责根据预定的调度策略将容器调度到合适的节点上

Node Components

- kubelet：运行在每个节点上，负责管理该节点上的Pod和容器
- kube-proxy：实现Kubernetes服务的负载均衡和网络代理功能
- Container Runtime：例如Docker，负责运行容器

工作原理

1. 用户提交请求：通过**kubectl**(ctl：control)或者其他API客户端提交操作请求到API Server
2. API Server 处理请求：API Server 接收请求验证和更新etcd中的状态数据
3. Scheduler调度：Scheduler根据调度策略决定将Pod调度到哪个节点【 bind pod to node】
4. kubelet执行：目标节点上的kubelet接收指令，启动容器
5. Controller Manager：持续监控集群状态，确保实际状态和期望状态一致
6. kube-proxy 负载均衡：管理服务的网络规则，确保服务可以被访问。

实际操作

- 安装一个本地的kubernetes集群，例如使用“minikube”或者“kind”。
- 部署简单的应用程序，了解Pod、Service、Deployment等基本概念。
- 更多高级功能，如持久化存储、网络策略、自动扩展等。

7.23

kubernetes解决应用上云的问题，kubernetes中的应用将作为微服务运行，但是kubernetes本身没有给出微服务治理的解决方案，如服务的限流、熔断、良好的灰度发布支持等。

service mesh【服务网格】可以用来做什么

- Traffic Management
- Observability
- Policy Enforcment
- Service Identity and Security

每个API对象都有3大类属性：元数据metadata、规范spec、状态status

metadata中必须有的三个数据：namespace、name、uid

规范spec：描述用户期望k8s集群中分布式系统达到的理想状态

状态status：描述系统实际当前达到的状态

例子：用户通过复制控制器Replication Controller设置期望的Pod副本数为3，如果当前状态为2，复制控制器的程序逻辑就是启动新的Pod，争取达到副本数为3。

k8s中所有的配置都是通过API对象的spec去设置的，用户通过配置系统的理想状态来改变系统。这是k8s的设计理念之一，所有的操作都是声明式。

Pod

Pod是在k8s集群中运行部署应用或服务的最小单元，它是可以支持多容器的。理念是在一个Pod中共享网络地址和文件系统，可以通过进程间通信和文件共享方式组合完成服务。pod根据业务不同有不同的控制器

Replication Controller，RC 副本控制器

副本（数量）控制器，只适应于长期伺服型的业务，控制pod提供高可用的Web服务

Replica Set RS 副本集

新一代的RC，支持多种类型的匹配模式。副本集对象一般不单独使用，作为Deployment的理想状态参数使用

Deployment 部署

RC,RS,Deployment保障Pod的数量，访问服务IP和端口号的问题，需要有服务发现和负载均衡能力。

Service 服务

服务发现：针对客户端访问的服务，找到对应的后端服务实例。

例子：客户端访问的是Service对象，每个Service会对应集群内部有效的虚拟IP，集群内部通过虚拟IP访问一个服务。

负载均衡：

kube-proxy是分布式代理服务器，每个节点都有一个

Job 任务

控制批处理任务的API对象，

DaemonSet 后台支撑服务集

保证每个节点都有一个此类的Pod执行，节点范围通过nodeSelector选定

适用于存储，日志，监控等服务

PetSet有状态服务集

每个pod 的名字很重要，pod故障时创建新的pod需要取相同的名字，并挂载相同的存储

适用MySQL和PostgreSQL，集群化管理服务Zookeeper、etcd等有状态服务

Federation集群联邦

在云计算中，服务的作用距离范围分为：同主机（Host,Node）、跨主机同可用区（Available Zone）、跨可用区同地区（Region）、跨地区同服务商（Cloud Service Povider）、跨云平台。

k8s是单一集群在同一地域内（Region），Federation 是提供跨Region跨服务商的k8s设计

存储卷 Volume

生命周期和作用范围是Pod，通过Persistent Volume来配置

持久存储卷 Persistent Volume PV和持久存储卷声明Persistent Volume Claim PVC

7.24

核心概念：

镜像创建容器，pod管理容器，通过Pause关联容器。

RS控制pod的副本数量

Deployment可以管理旧的RS转向新的RS（滚动部署）【管理Deployment，它负责RS和POD】

Service label:打标签。可以在pod，RS，Deployment打标签。在service上设置Selector(app=login)找到打标签的pod，

客户端可以通过ClusterIP访问。

架构设计：

Scheduler：选择运行节点

master-worker。

etcd、ApiServer、Scheduler、ControllerManager

kubelet

很多的服务器、很多的节点。

服务器分为：Master、Worker节点

k8s的存储(持久化)组件：etcd。

k8s的交互组件：ApiServer

Scheduler选择节点：资源内存cpu运行的服务，预选策略，优选策略，选择一个最优的节点，然后将节点和pod建立起关系。然后告诉apiServer这个pod可以运行在某一个节点上，pod和node的绑定关系会被持久化在etcd上。

启动pod的组件:ControllerManager【集群内部的控制中心】serviceController管理服务 endPodController管理pod列表，replicationController管理副本的，ResourceCoderController管理资源配额的。会时刻关注这些状态，并会时刻保证他们处于一个正确的状态

监听到，等待调度的状态，然后让pod运行起来。

kubelet：每一个worker节点上都存在一个kubelet，管理pod生命周期，网络。调用本机的docker，运行容器，运行pod

k8s认证和授权

k8s原生搭建复杂在**认证**和**授权**。

讲解了对称加密、非对称加密的流程

k8s认证方式：

1. 客户端认证
2. BearerToken

3. ServiceAccount 【k8s内部交流】

授权方式：

ABAC、WebHook、RBAC

RBAC（Role Based Access Control）：

User： 1.user 2.ServiceAccount【k8s集群内部】

Authority：1.Resource 2.Verbs：curd

Role：角色包含的信息：name，resource，verbs

RoleBinding:角色绑定。

将角色放到namespace中，角色只能访问当前namespace

集群角色：ClusterRole，ClusterRoleBinding，可以访问集群范围内的，不限定namespace

AdmisionControl：准入控制。alwaysAdmit、alwaysDeny、ServiceAccount、DenyEscolatingExec

kubernetes集群搭建方案

Kubeadm搭建、二进制Binary方案

高可用集群必须有三台master节点

问题：没有那么多机器，如何练习集群搭建

学习时：技术细节：1.怎么使用更规范2.使用陷阱3.需要使用时，注意什么...

Go语言学习-01

go是 类型安全、内存安全的编程语言。但不允许进行指针运算。goroutine线程和channel实现goroutine间通信。有垃圾回收（标记-清除算法）

可见性：大写字母开头为可导出、小写字母开头包内可用

函数：main包中没有main函数会报错，main函数既没有参数也不能有返回值

函数格式必须是**func main {**对大括号的使用必须是这样，或者是在一行内写完{}

普通函数：func functionName (parameter_list) (return_value_list) {

可导出函数：func FunctionName (parameter_list) (return_value_list){

类型：基本类型int、float、bool、string；结构化：struct、array、slice、map、channel；只描述类型的行为interface。结构化的类型默认值为nil。Go中没有类型继承。

go程序的一般结构

import ...

全局变量声明

func init ()(){}

func main()(){}

类型的函数

普通函数

没有隐式转换，要像函数一样进行类型转换

常量可以省略类型说明符

const b string = "abc" 等价 const b = "abc"

反斜杠作为常量表达式中作为多行连接符

常量中的iota用法，在常量枚举中使用

条件语句：多了select，不支持?:等三目运算符

&取变量的地址，*定义指针和将指针存的地址的变量取出。

循环语句：

- for 不带括号。分为基本循环、条件循环、无限循环。
- range循环 range关键字用于遍历数组、Slice、字符串、map和channel

Docker学习-01

Docker三大组件：镜像、容器、仓库

Go语言学习-02

goroutine的规则

7.25

安装docker、kind

7.26

docker进入容器的命令

1. docker exec -it container-id /bin/sh
2. 退出 exit

kind 的常用命令

1. 查看集群 kind get clusters
2. 删除集群 kind delete cluster --name 集群名字
3.

搭建一主三从集群

1. 编写集群配置文件

```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 31000  # 将主机 31000 端口映射到容器的 31000 端口
    hostPort: 31000
    listenAddress: "0.0.0.0" # Optional, defaults to "0.0.0.0"
    protocol: tcp # Optional, defaults to tcp
- role: worker
- role: worker
- role: worker
```

2. 通过配置文件搭建

```
kind create cluster --config cluster.yaml --name 1c3w
```

3. 创建service

   更换context、创建一个nginx的deploy测试服务

4. 搭建的集群每个节点都是一个docker中的容器。

   显示deployment的详细信息：kubectl describe deployment nginx

kubectl命令

1. 创建集群：kubectl create devlopment

入职一周总结

- GO
  - 基本结构和基本数据类型
  - 循环语句
- K8S
  - Pod的两种用法
- docker
  - 镜像：相当于一个root文件系统。
  - 容器：是进程，但有自己的命名空间。有自己的root文件系统、网络配置、进程空间、用户ID空间。容器的文件写入应当使用Volume或者绑定宿主目录

7.29

使用镜像

docker中**容器中是为了支持其中的主进程/前台进程**，当其结束时容器也就结束了。

docker commit在特殊情况下如被入侵情况下保存现场，**应当用Dockerfile来定制镜像**

镜像是多层存储，容器以镜像为基础层，在其上加一层容器运行时的存储层。

利用commit理解镜像构成

使用Dockerfile定制镜像

Dockerfile文件中的一个命令就是在镜像中添加一层，创建镜像的原则是**尽量少的增加层数**

Dockerfile常用指令

- FROM指定基础镜像

  FROM 基础镜像/scratch[空白镜像]

- RUN执行命令
  - shell格式`RUN <命令>`

    ```
    RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
    ```

  - exec格式`RUN ["可执行文件", "参数1", "参数2"]`

    ```
    RUN ["apt-get", "install", "-y", "curl"]
    ```

- 【例子】创建DIY镜像
  1. 编写Dockerfile文件

     首先创建空文件夹mynginx，然后创建dockerfile文件。写入

     ```
     FROM nginx
     RUN echo '<h1>Hello, Dockers! </h1>' > /usr/share/nginx/html/index.html
     ```

  2. 构建镜像

     ```
     docker build -t nginx:v3 .
     ```

构建镜像命令参数详解

docker build

1. -t 指定构建的镜像名
2. . 指定上下文，将上下文的所有文件发送到 docker service中

Dockerfile指令详解

- COPY 复制文件

  两种写法：命令行、函数调用

  `COPY [--chown=<user>:group] <源路径>...<目标路径>`

  `COPY [--chown=<user>:<group>] ["<源路径>, ..<目标路径>"]`

  改变文件所属组：--chown=用户:组，没有改变文件wrx权限的命令吗

  例子：将当前上下文中的hello.txt 文件复制为镜像中的app文件

  ```
  COPY hello.txt /usr/src/app
  ```

- ADD

  仅在复制文件需要自动压缩场合使用

- CMD

使用容器

常用命令：docker run -it 容器 /bin/bash

- 分配一个文件系统，并在只读的镜像层外面挂载一层可读写层
- 从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去
- 从地址池配置一个 ip 地址给容器

-it 交互式运行，-d 后台运行

docker 容器 logs： 查看容器输出信息

docker 容器 stop，docker 容器 start ，docker 容器 restart

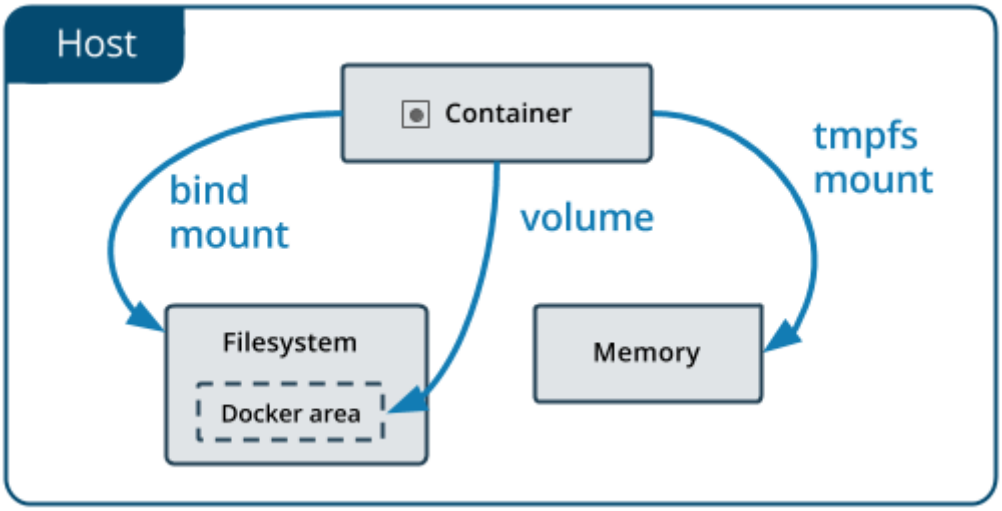
查看类型的命令：

- 查看容器：docker container ls [-a] 、docker ps [-a]
- 查看镜像：docker image ls 、docker images

进入容器：

- docker exec -it 容器 bash

数据管理



docker的三种文件挂载:bind

数据卷 volume

规则:以数据卷内容优先，挂载后镜像目录内原先有的文件会被替换，只有volume中为空时会复制容器内文件到volume。

- 创建一个数据卷

  ```
  docker volume create my-vol
  ```

  查看数据卷

  ```
  docker volume ls
  ```

  查看数据卷配置

  ```
  docker volume inspect my-vol
  ```

  数据卷配置 my-vol

  ```
  [
      {
          "CreatedAt": "2024-07-29T06:16:54Z",
          "Driver": "local",
          "Labels": null,
          "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
          "Name": "my-vol",
          "Options": null,
          "Scope": "local"
      }
  ]
  ```

- 挂载数据卷到容器

  在创建容器的时候进行设置。-d后台运行容器，-P随机分配端口

```
docker run -d -P --name web \
#-v my-vol:/usr/share/nginx/html
--mount source=my-vol,target=/usr/share/nginx/html nginx
```

通过docker inspect web 可以看到容器的配置，其中mounts中

```
"Mounts": [
          {
              "Type": "volume",
              "Name": "my-vol",
              "Source": "/var/lib/docker/volumes/my-vol/_data",
              "Destination": "/usr/share/nginx/html",
              "Driver": "local",
              "Mode": "z",
              "RW": true,
              "Propagation": ""
          }
      ],
```

这两个目录的文件就会自动同步。

- 删除数据卷

```
# 删除指定volume
docker volume rm my-vol
# 在删除容器时，删除挂载的volume
docker rm -v
# 删除无主的volume。prune-修剪
docker volume prune
```

挂载主机目录

使用网络

外部访问容器

外部访问容器：通过端口映射外部和容器。

-P 大P时，docker会随机映射一个端口到内部容器开放的网络端口

例子 `docker run -d -P nginx`

-p 小p时，使用**hostPort:containerPort**进行配置

例子`docker run -d -p 80:80 nginx`

容器互联

1. 创建网络 -d 类型有bridge、overlay【适用于集群】。

```
docker network create -d bridge my-net
```

2. 连接容器

   创建容器，并连接到`my-net`

```
docker run -it --rm --name busybox1 --network my-net busybox
docker run -it --rm --name busybox2 --network my-net busybox
```

   然后可以在容器busybox1中ping busybox2

```
ping busybox2
```

3. 多个容器之间需要互联，推荐使用Docker Compose

配置DNS

自定义容器的主机名和DNS，利用**虚拟文件**来挂载容器的三个**相关配置文件**

Docker Compose

通过dockers-compose.yml定义一组关联的应用容器

- 服务 service
- 项目 project

一个项目由多个服务(容器)关联，compose面向项目进行管理

举例:一个项目包含web应用和redis缓存。

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! 该页面已被访问 {} 次。\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

编写Dockerfile文件【构建一个镜像】

```
FROM python:3.6-alpine
ADD . /code
# 指定工作路径，当使用相对路径时，其绝对路径是和上一个workdir相关
WORKDIR /code
# RUN pip install redis flask 设置镜像安装
RUN pip install -i https://pypi.tuna.tsinghua.edu.cn/simple redis flask
# 容器启动 python app.py 进程
CMD ["python", "app.py"]
```

编写compose.yaml

```
services:
    web:
        # build 指定Dockerfile所在的路径，可以是绝对路径或相对docker-compose的路径
        # compose会自动构建这个镜像，然后使用这个镜像
        build: .
        # 通过context指定dockerfile所在文件夹路径
        # dockerfile指定dockerfile指定文件名
        # 使用args指定构建镜像时的变量
        # 使用cache_from指定构建镜像的缓存
        # build:
        #   context: .
        #   dockerfile: Dockerfile
        #   args:
        #       buildno:1
        #   cache_from
        #       - alpine:latest
        #       - corp/web_app:3.14
        ports:
            - "5000:5000"
    redis:
        image: "redis:alpine"
```

使用compose进行构建

在指定服务上执行一个命令：

> docker compose **run** 服务 命令

想在外部访问，需要

```
Invalid HTTP_HOST header: '192.168.75.128:8000'. You may need to add
'192.168.75.128' to ALLOWED_HOSTS.
```

docker compose up

docker compose down

docker compose logs

docker compose ps