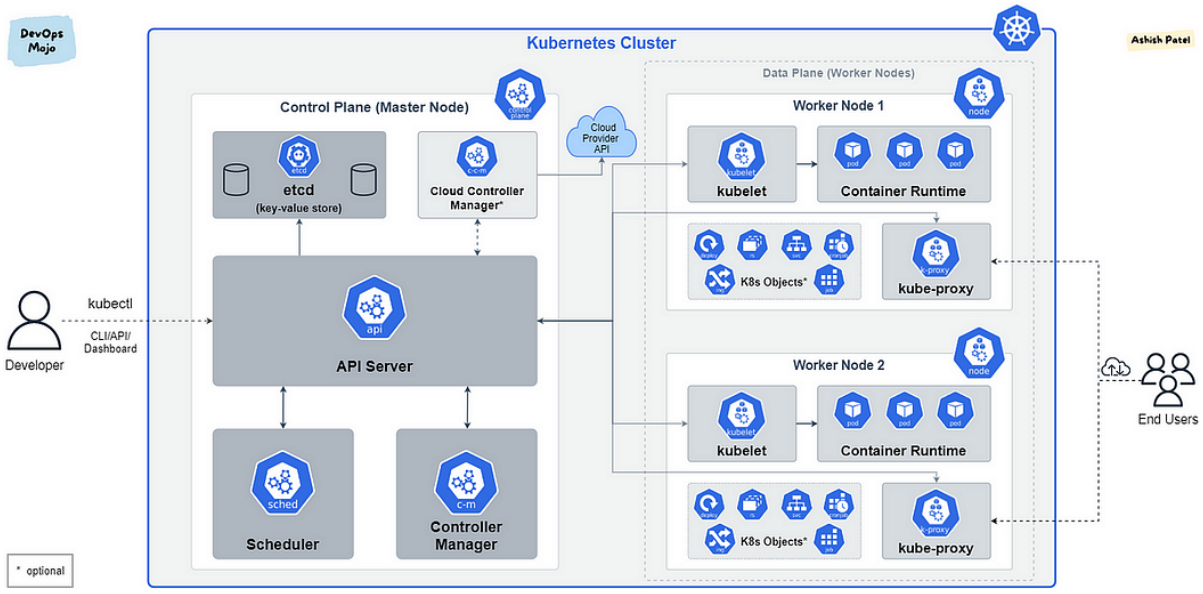
Kubernetes

> - Kubernetes
>
> Docker 可以在一台机器上快速的启动、关闭容器，但是缺乏对大量容器自动编排、管理和调度。k8s 就是一套管理系统，对容器进行更高级更灵活的管理。你只需要提供服务器，k8s 通过对容器的编排可以帮你最大化的利用这些服务器的资源。
>
> - Serverless
>
> 云服务可以让我们选择合适配置的虚拟的云主机，然后按时、按月、按年的收费，但不管你使用与否，费用都在那里，不增不减。Serverless 则可以让你只在使用时，根据使用时间、内存占用等一些你具体使用的东西来收费，不管你使用的频率高低都可以承受，就像水和电，用了多少就是多少费用。

- 10分钟看懂Docker和K8S的前世今生 - 知乎 (zhihu.com)
- google的容器集群管理器
  - 第一代Borg
  - 第二代Omega
  - 第三代Kubernetes，开源的业界标准
  - 曾经的宇宙中心是docker，但是如今k8s已经是事实上的业界标准
- docker淡出
  - cncf成立之初就决定对接oci
  - docker不再是装机必备
  - docker只作为容器运行时，行业基本直接调containerd，跳过了docker，节省了资源（docker本身也就是调用containerd）
  - 后来，许多产品直接脱离docker，不支持docker，留给docker的工作只剩下打包镜像（事实上打包镜像也有谷歌的kaniko作为替代方案）
  - 网络上，k8s的CNI成为主流，docker的CNM不再受生态支持
  - **docker的淡出也是k8s的逐渐强势，此消彼长**

concept



node

- 节点
- 物理主机

pod

- 创建和管理的最小**计算单元**，pod包含的多个容器总是被一同调度，运行在同一个host上
- （谷歌觉得需要有一个）pod模拟一台**逻辑主机**：运行在pod上的多个容器共享网络和磁盘，可以通过localhost直接彼此访问；但是不固定主机名和ip（serverless）
- [K8S实战基础知识之POD - 知乎 (zhihu.com)]

kubelet

- 可以在本地电脑上安装，然后通过配置控制k8s集群
- 运行在每个节点上
- 负责运行pod，根据apiserver或者static pod
- 向apiserver汇报本地状态
- [深入理解kubernetes: kubelet - 知乎 (zhihu.com)]

apiserver

- 运行在master上，static pod
- 控制平面的前端，接受来自客户端的请求和k8s内部的请求
- 可以通过部署多个实例来进行扩缩

etcd

- 一致且高度可用的键值存储数据库（consistent and highly available key-value db），用作k8s所有集群数据的后台数据库
- 由apiserver作为统一前端

scheduler

- 监视apiserver上创建但尚未指定运行节点的pod，并选择节点来运行pod（即所谓调度）
- 考虑：资源、软硬件、策略、数据位置、工作负载间的干扰等
- 即接受**待调度pod**和**集群内node资源情况**为输入，输出调度结果
- 热知识
  - 资源限制定义在容器上，每个容器都可以有资源限制
  - pod可以有初始化容器（一般是one-shot任务），创建pod时会先**顺序执行**init containers
  - 因此计算资源用量时，考虑init containers用或，考虑containers用与
  - 如果`requests`和`limits`都没有设定，那么最后容器cgroup分得的`cpu.shares=2`（linux内核允许的最小值）

controller manager

- 运行多个controller，运行在master节点上
- node controller：负责在节点出现故障时进行通知响应
- job controller：监测代表一次性任务的job对象，创建pods来运行直至完成
- endpoint slice controller：填充端点分配对象，提供services和pod之间的链接
- 一文看懂 Controller Manager - 知乎 (zhihu.com)

kube-proxy

- 网络代理组件，运行在每个节点上
- 维护网络规则
- 一文看懂 Kube-proxy - 知乎 (zhihu.com)

ingress

- 对集群外部暴露服务
- k8s从入门到实战(九)：Ingress介绍与使用 - 掘金 (juejin.cn)

pause

- 主要：pause容器在pod启动时，作为配置namespace的placeholder，率先配置好namespace和network等配置，让其他容器加入
- 次要：ause容器的次要作用是收割子进程，作为PID=1的进程
  - pod之间共享pid namespace的话，pause容器的进程就会成为整个pod所有容器的父进程（pid=1）
  - 由于pod当前默认不共享pid namespace，pause无法再执行收割子进程的任务
- Kubernetes Pod 网络精髓：pause 容器详解-腾讯云开发者社区-腾讯云 (tencent.com)

sidecar

- 将本将属于应用程序的功能拆分成单独的进程，这个进程可以被理解为`Sidecar`。在微服务体系内，将集成在应用内的微服务功能剥离到了`sidecar`内，`sidecar`提供了微服务发现、注册，服务调用，应用认证，限速等功能。

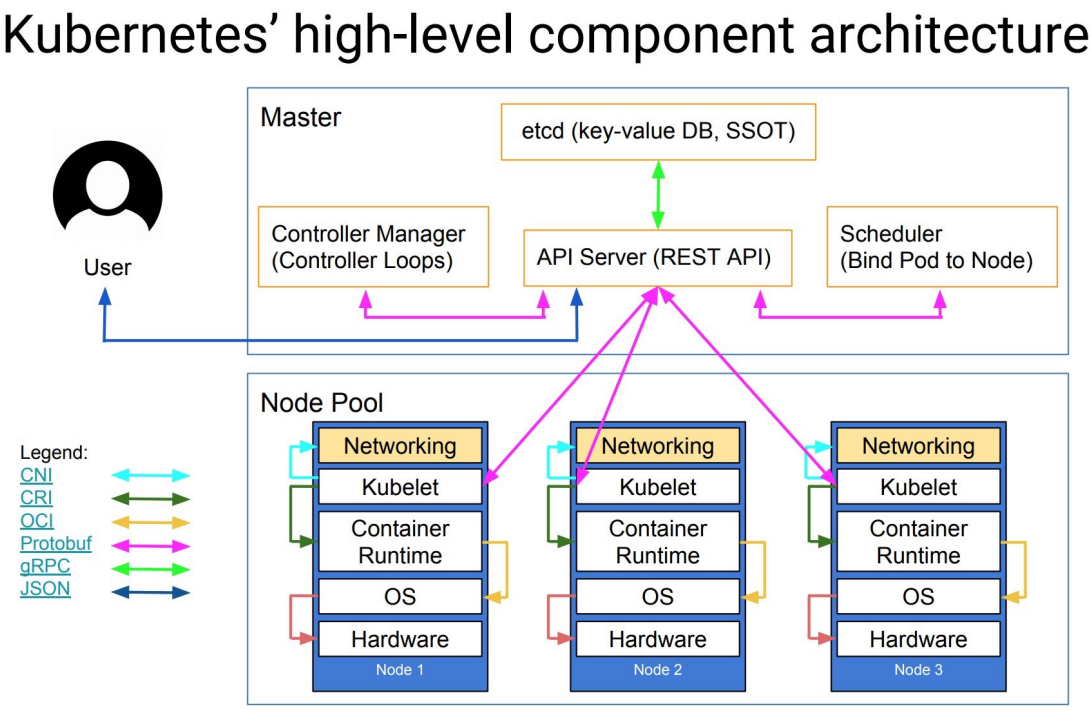
- 在阿里云上，通过`kubectl label namespace xxx istio-injection=enabled`即可自动开启
- why istio/service mesh?
  - istio概述，与微服务、云原生、k8s的关系-CSDN博客
  - 为什么在使用了 Kubernetes 后你可能还需要 Istio？ · Jimmy Song
- Istio入门：什么是Istio？Istio的4个主要功能和实现原理 - 知乎 (zhihu.com)
- Istio入坑指南（二） Istio的安装与简单的使用 docker desktop安装istio-CSDN博客

安装

- 安装-生产环境
  - kubeadm
    - 最常用
    - 能扩容升级管理k8s环境
  - kops
  - kubespray
- 安装-学习、开发、iot
  - kind(kubernetes in docker)
    - 用容器代替虚拟机作为节点
    - 节点之间共享内核，不适合需要修改内核的场景
  - minikube
  - MicroK8S
  - k3s

- 使用kind快速搭建本地k8s集群 - 万德福儿 - 博客园 (cnblogs.com)
- 使用kubeadm快速部署一套K8S集群 - Double冬 - 博客园 (cnblogs.com)

架构



- cs架构
  - client：kubectl
  - server：k8s集群
- 集群：分为两类节点
  - master节点：运行控制逻辑
  - worker节点：运行用户应用
- kubectl基于`~/.kube/config`找到server
- master节点的进程
  - systemd
  - containerd
  - kubelet：节点agent，可以在master节点内通过`systemctl status kubelet`查看
  - kube-开头的进程：控制面组件
  - etcd：kv存储，存储集群状态
  - coredns：集群内部dns服务器，由kubeadm创建
  - cni插件，可以分配节点ip
- master节点上如何启动k8s?
  - 通常情况下，kubelet要监听来自apisercer的事件，然后根据指示在自身节点启动pod；master节点上运行kube-apiserver的pod，这个pod也应该由kubelet启动，kubelet从哪里获取启动任务的指示呢?
  - static pod：在指定的节点上由kubelet直接管理，不需要等待apiserver的指示和配置
  - systemd启动kubelet时，可以看到`--config`传入了配置文件路径，这个配置文件路径里面有指定`staticPodPath`
  - worker节点上没有static pod的定义

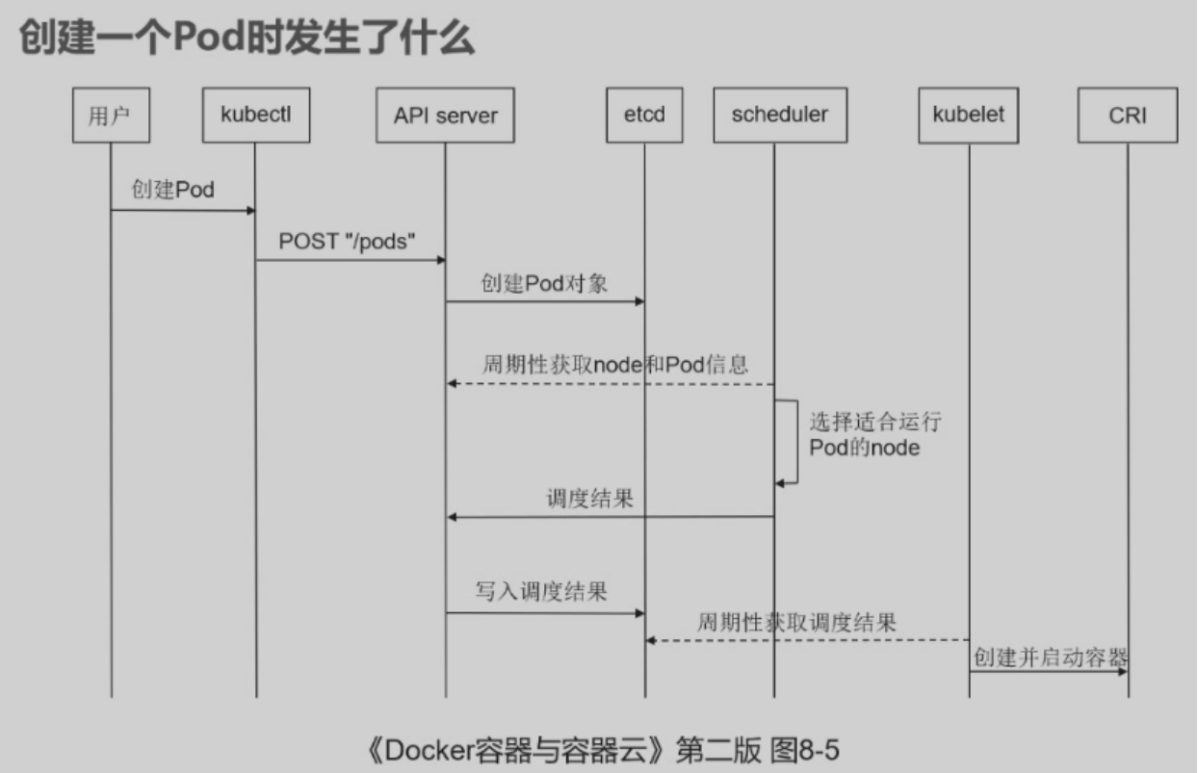
- master节点上，CNI插件尚未给容器配置网络时，controller manager、scheduler和kubeadm如何访问apiserver?
  - apiserver、controller manager和scheduler的配置制定了使用hostnetwrok，没有单独的network namespace
- 资源限制
  - container level
    - `requests` & `limits`
  - pod level
    - pod本身的运行也会带来一部分overhead，比如katacontainer等hypervisor类容器中的sandbox，一个包含超过100MB的LInux内核；这类资源消耗在runtime并不属于任何容器，需要pod level的资源限制把这部分overhead管理起来
  - qos level
    - quality of service
    - 分类
      - Guaranteed: 每个pod都制定了CPU和内存的`requests`和`limits`，且设定的`requests`和`limits`相等
      - Burstable: 介于上下二者之间
      - BestEffort: pod所有容器都没有设置CPU和内存的`requests`或`limits`
    - 作用
      - 做出pod eviction（pod驱逐）的决策
      - 决定整个class内所有pod资源分配的优先级
  - node level
    - allocatable: 可以被k8s用来运行pod的资源量
    - kube-reserved: 预留给kubelet、container runtime等k8s组件
    - system-reserved: 预留给ssh和udev等守护进程，也包括为kernel预留
    - eviction-threshold: 资源即将耗尽时，开始安装qos优先级驱逐pod

部署

- worker上的container runtime需要用户自行安装
- 如何定义一个容器：写一个yaml
  - 然后`kubectl applt -f nginx.yaml`
  - `apply`的含义：不再指定创建还是删除等动作，只是声明用户需要什么样的理想状态，至于怎么实现到达（需要创建/删除/停止还是whatever）这个状态，交给k8s就完了，这也符合k8s的设计
  - 根据`requests`设置`cpu.share`，决定争抢cpu时的优先级，`spu.share`的相对权重决定了分得多少cpu时间
  - cpu的单位：m=millicore=千分之一个逻辑CPU核
- 资源记账
  - `requests`不等于保证或者预留

    - 没有机制给容器预留资源量
    - 也没有机制保证了容器一定能得到 `requests` 的资源量
    - 没有资源竞争时，可以使用超过 `requests` 但不超过 `limits` 的资源量；有资源竞争时，有可能连 `requests` 的资源量都得不到
- 调度
    - 记账和调度即发生在创建pod图中的**选择适合运行pod的node**
    - 分为**调度周期scheduling cycle**和**绑定周期binding cycle**
    - 调度周期是串行的，避免重复或冲突；绑定周期可能是并行的
    - 调度：过滤（能不能跑）和打分（有多适合跑）
- 更新策略与扩容策略
    - k8s的部署策略配置：k8s-部署策略 - 努力头小白 - 博客园 (cnblogs.com)
    - 部署策略：灰度发布、蓝绿部署、金丝雀都是啥？ - 知乎 (zhihu.com)
    - 基于CPU利用率或其他指标的k8s扩容
        - K8s弹性伸缩-HPA（Horizontal Pod Autoscaler ） - 简书 (jianshu.com)
        - Pod 水平自动扩缩 | Kubernetes
        - K8S-kubectl scale（静态）扩缩容pod，K8S-kubectl autoscale（动态）扩缩容pod，K8S的HorizontalPodAutoscaler（HPA）机制自动扩容缩容-CSDN博客
- daemon set
    - 污点和容忍度 | Kubernetes
    - DaemonSet | Kubernetes

```yaml
# nginx.yaml
apiVersion: v1
kind: Pod
# 给pod取个名字
metadata:
  name: nginx
# specification/expectation, desired state
spec:
  containers:
  # 指定pod包含的容器
  - name: nginx
    image: nginx:1.23.3
    resources:
    # cpu和内存都存在overcommit，调度时看requests，实际运行允许最高到limits
      limits:
        cpu: 500m
        memory: "200Mi"
      requests:
        cpu: 250m
        memory: "100Mi"
     ports:
     - containerPort: 80
```



API

> 一切都是OOP

- k8s采用REST风格API
- API也可以分版本管理
  - 版本名称包含alpha
    - alpha级别的API默认被禁用
    - 软件可能有bug
    - 可能随时删除
    - 不保证向后兼容
    - 不建议生产使用
  - 版本名称包含beta
    - 内置的bet级别API默认被禁用
    - 特性很好的测试过
    - 将会被长期支持（9个月或三个次要版本）
    - 可能不向后兼容，但会提供迁移说明
    - 不建议生产使用
- REST风格推荐使用一组有限的方法（verb/method）来操作资源的增删改查
  - 采用了OpenAPI（前身为swagger）
  - k8s将自身对资源的操作映射到http协议的verb上，即`curl -X GET xxx`和`kubectl get ...`效果一样

实战

> 以下基于阿里云上的k8s集群服务展开
>
> 假设在部署一个具有以下微服务的后端，镜像保存在私有仓库中：
>
> - 后端本身
> - mysql
> - memcache

准备

- k8s
  - **阿里云买一个**或者**本地用kind搭建一个**
  - 至少要三个节点
- 代码
  - 配置通过环境变量的形式进入容器，因此代码要通过环境变量来获取配置的变量

kubectl

1. 根据教程安装工具安装kubectl
2. 查看阿里云中集群信息 -> 连接信息，根据指引把内容复制到`~/.kube/config`
3. `kubectl cluster-info`查看信息，能输出控制面地址等信息即为成功

namespace

- 创建一个新的`namespace`，而不使用默认的`default`

```yaml
kind: Namespace  #类型为Namespace
apiVersion: v1
metadata:
  name: bli-deployment  #命名空间名称
  labels:
    name: bli-deployment-v1
```

secret

- 顾名思义，用k8s来处理敏感数据
- yaml中出现的value均为base64加密过的密文，密文可以通过`echo -n 'xxx' | base64`处理明文得到
- 这里保存mysql的验证信息和docker私有仓库的验证信息
- 使用 kubectl 管理 Secret | Kubernetes
- 使用 Secret 安全地分发凭据
- 从私有仓库拉取镜像

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: bli-deployment
type: Opaque
data:
  db-user: ====
  db-name: ====
  db-password: ====
---
apiVersion: v1
kind: Secret
metadata:
  name: docker-secret
  namespace: bli-deployment
data:
  .dockerconfigjson: ====
type: kubernetes.io/dockerconfigjson
```

memcache

- 一个微服务的deployment和service写在一个yaml里面
- 指定一个`namespace`，不然就会放在默认的`default`空间
- `imagePullSecrets`即指明了如何通过私有仓库的验证
- 这里使用`kind: Deployment`而不是`kind: Pod`是为了更方便地扩容，同时部署多个相同的pod，数量通过`spec.replicas`指定

```
# memcache -- memcache
apiVersion: apps/v1
kind: Deployment
metadata:
  name: memcache-deployment
  namespace: bli-deployment
spec:
  replicas: 2
  selector:
```

```
    matchLabels:
      app: memcache-pod
  template:
    metadata:
      labels:
        app: memcache-pod
    spec:
      imagePullSecrets:
      - name: docker-secret
      containers:
      - name: memcache-container
        image: registry.cn-hangzhou.aliyuncs.com/xxxx
        resources:
          requests:
            cpu: "250m"
            memory: "512Mi"
          limits:
            cpu: "500m"
            memory: "1Gi"
        ports:
        - containerPort: 8911
---
apiVersion: v1
kind: Service
metadata:
  name: memcache-service
  namespace: bli-deployment
  labels:
    app: memcache-service
spec:
  selector:
    app: memcache-pod # 这应该与pod的标签匹配，以便Service能找到对应的Pod
  ports:
    - name: memcache-mapping
      # 集群内部的service的端口
      port: 8911
      # 容器暴露的端口
      targetPort: 8911
  type: ClusterIP
```

backend

- 这里的环境变量，当需要传入一个ip地址/域名时，改为填写service的名字，k8s会自动完成转换填充；后端就可以通过环境变量获取到`memcache`的地址并连接上
- 敏感数据通过`valueFrom.secretKeyRef`获取

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  namespace: bli-deployment
spec:
```

```yaml
  replicas: 2
  selector:
    matchLabels:
      app: backend-pod
  template:
    metadata:
      labels:
        app: backend-pod
    spec:
      imagePullSecrets:
      - name: docker-secret
      containers:
      - name: backend-container
        image: registry.cn-hangzhou.aliyuncs.com/xxxx
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
          limits:
            cpu: "1000m"
            memory: "1Gi"
        env:
          - name: DEBUG
            value: "false"
          - name: PRODUCTION
            value: "true"
          # for mysql
          - name: DB_HOST
            value: "xx.xx.xx.xx"
          - name: DB_PORT
            value: "8000"
          - name: DB_USER
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: db-user
          - name: DB_NAME
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: db-name
          - name: DB_PASS
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: db-password
          # for memcache
          - name: MEMCACHE_HOST
            value: memcache-service
          - name: MEMCACHE_PORT
            value: "8911"
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
```

```yaml
metadata:
  name: backend-service
  namespace: bli-deployment
  labels:
    app: backend-service
spec:
  selector:
    app: backend-pod # 这应该与pod的标签匹配，以便Service能找到对应的Pod
  ports:
    - name: backend-mapping
      port: 5000
      targetPort: 5000
  type: ClusterIP
```

ingress

- 在部署service时，可以选择指定`ClusterIP`或者`NodePort`类型（还有其他类型）
  - 前者指的是，仅把容器的端口映射到service的某个端口上，那么这个端口一般来说只有集群内部可以访问
  - 后者指的是，还把容器的端口映射到节点（物理机）的某个端口上，可以根据节点ip来访问服务
  - k8s-集群里的三种IP（NodeIP、PodIP、ClusterIP）-CSDN博客
- 阿里云的节点都只有局域网ip，而购买k8s时还会分配一个属于集群的公网ip。如果使用`NodePort`，那么无法访问服务（因为没有节点的公网ip），因此要让外界能够访问部署的服务，需要借助ingress来完成域名到集群这一步
- 阿里云默认装好了nginx-ingress
- 可以一个公网ip上同时配置多个暴露服务
- Kubernetes K8S之Ingress详解与示例-腾讯云开发者社区-腾讯云 (tencent.com)

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-https
  namespace: bli-deployment
spec:
  ingressClassName: "nginx"
  tls:
    - hosts:
      - ss.bli.cn
      - xx.bli.cn
      secretName: tls-secret
  rules:
    - host: ss.bli.cn
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
```

```
                name: backend-service
                port:
                  number: 5000
    - host: xx.bli.cn
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: sk-service
                port:
                  number: 1991
```

https

- 用lets encrypt
- 主要参考k8s中级篇-cert-manager+Let's Encrypt自动证书签发_cert-manager let's encrypt-CSDN博客
- 其次看了下Kubernetes之Ingress自动化https-腾讯云开发者社区-腾讯云 (tencent.com)

1. 安装cert-manager：`kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.14.4/cert-manager.yaml`
2. 写issuer

```
# kubectl apply -f https://github.com/jetstack/cert-
manager/releases/download/v1.14.4/cert-manager.yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-issuer
  namespace: bli-deployment
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: xxxx@qq.com
    privateKeySecretRef:
      name: letsencrypt-secret
    solvers:
      - http01:
          ingress:
            class: nginx
```

apply后检查：

```
$ kubectl get issuer -A
NAMESPACE            NAME                  READY   AGE
bli-deployment   letsencrypt-issuer   True    5m32s
```

3. 写certificate

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: bli-deployment-cert
  namespace: bli-deployment
spec:
  secretName: tls-secret
  issuerRef:
    name: letsencrypt-issuer
    kind: Issuer
  duration: 2160h
  renewBefore: 360h
  # keyEncoding: pkcs1
  dnsNames:
  - ss.bli.cn
  - xx.bli.cn
```

检查:

```
$ kubectl get pods -n bli-deployment -w
NAME                                  READY   STATUS    RESTARTS   AGE
backend-deployment-----------------   1/1     Running   0          15h
cm-acme-http-solver------             1/1     Running   0          25s
cm-acme-http-solver------             1/1     Running   0          26s
memcache-deployment-----------------  1/1     Running   0          39h
memcache-deployment-----------------  1/1     Running   0          39h
sk-deployment-----------------        1/1     Running   0          4d11h

$ kubectl get certificate -n bli-deployment
NAME                     READY   SECRET       AGE
bli-deployment-cert   True    tls-secret   82s
```

4. 修改ingress，使用tls-secret，开启https

cache

> 服务端高并发分布式架构演进之路 - 个人文章 - SegmentFault 思否

- 也就是说，一个比较合适的做法是，一个deployment的每个pod里面部署memcache+backend（第三次演进）
- 一个deployment部署多个pod的redis
- mysql可以使用平台的或者分布式的
- backend查数据时，先查pod本地的memcache，再查分布式的redis，最后去问mysql
- stateful set和headless service部署redis集群
  - 在Kubernetes上部署一套 Redis 集群 - 知乎 (zhihu.com)
  - K8S 快速入门（十五）实战篇：Headless Services、StatefulSet 部署有状态服务_statefulset headless service-CSDN博客

reference

- Docker，containerd，CRI，CRI-O，OCI，runc 分不清？看这一篇就够了 - 知乎 (zhihu.com)
- Kubernetes 架构 · Kubernetes 中文指南——云原生应用架构实战手册 (jimmysong.io)
- 研发工程师玩转Kubernetes——使用Deployment进行多副本维护-腾讯云开发者社区-腾讯云 (tencent.com)
- Kubernetes(K8S)（六）——service（ClusterIP、NodePort、无头服务、LoadBalancer、ExternalName等） 无头服务和clusterip-CSDN博客
- Init 容器 | Kubernetes
- 浅谈 K8S QoS(服务质量等级)-腾讯云开发者社区-腾讯云 (tencent.com)
- 一文带你搞懂 Kubernetes 容器边车模式 - 知乎 (zhihu.com)
- Rethink：为什么微服务没有sidecar不行？ *开源*蔡芳芳 InfoQ精选文章
- 从零开始入门 K8s | GPU 管理和 Device Plugin 工作机制 - 知乎 (zhihu.com)