- Kubernetes
  - 核心组件
    - kube-proxy
      - 每个Node上都会运行一个kube-proxy服务进程
      - 将某个Service的访问请求转发到后端的多个Pod实例上
      - kube-proxy在运行过程中动态创建与Service相关的iptables规则，实现将访问转发到后端对应的Pod
    - kube-Schedule
      - 在整个系统中承担了“承上启下”的作用。“承上”是指它负责接收Controller Manager创建的新Pod，为其进行调度。启下”是指安置工作完成后，通知目标Node上的kubelet服务进程接管后继工作
    - kube-Controller Manager
      - 实时监控集群中特定资源的状态变化，当发生各种故障时，会尝试将其状态调整为期望的状态
      - 是K8s中各种操作系统的管理者，是集群内部的管理控制中心，也是K8s自动化功能的核心
    - kube-apiserver
      - K8s的访问、控制入口
      - 除了提供CRUD接口，还提供了一类很特殊的REST接口---K8s Proxy API接口。与SVC，Ingress不同，会把资源对象直接暴露在URL
    - kubectl
      - K8s系统的命令行工具
    - kubelet
      - 处理Master下发到本节点的任务，管理静态Pod及Pod中的容器
      - 定期向API Server汇报节点资源的使用情况，通过cAdvisor监控容器和节点资源
      - kubelet监听etcd，所有针对Pod的操作都会被kubelet监听，并会做出相应的修改
    - etcd
      - 保存整个集群系统的所有信息
  - node
    - 集群节点，也分为master节点、work节点两种
    - 可以是一台物理机，也可以是一个虚拟机
    - 每个节点都运行着kubelet、Docker Engine。master节点还有etcd、一些kube-开头的组件
  - namespace
    - 用于实现用户隔离、资源隔离（配合RBAC）
  - Pod
    - K8s中能够定义管理的最小单位
    - 分为普通Pod和静态Pod
    - 里面包含pause容器，init容器，应用容器三类
  - ReplicationController
    - Pod基础控制器之一
    - 控制Pod的副本数量、Pod升级、回滚等
  - ReplicaSet
    - 与ReplicationCntroller基本相同，增加了支持标签集合选取的方式
  - Deployment
    - 更好的实现Pod编排的控制器之一，不仅可以控制Pod副本数量，回滚、更细粒度的控制升级过程。还支持历史记录
    - 内部是以ReplicaSet为基础
  - StatefulSet
    - 针对有状态Pod的控制器
    - 使用前提，需要有一个StorageClass，因为StatefulSet会自动为每个应用Pod申请PVC。一个Headless Service
  - DaemonSet
    - 会在每个Node上仅运行一份Pod的副本实例的控制器
    - DaemonSet也能执行滚动升级
  - Job
    - 定义并启动一个批处理任务。批处理任务通常并行启动多个计算进程去处理一批工作项，处理完成后，整个批处理任务结束
    - Job又分为3种类型，Non-parall Jobs、Parallel Jobs with a fixed conpletion count，Parallel Jobs with a work queue
    - 也分为3种工作模式，Queue with Pod Per Work Item、Queue with Pod Per Work Item、Queue with Variable Pod Count
  - Cronjob
    - 它基本照搬了Linux Cron的表达式，区别是第1位是分钟而不是秒
  - ConfigMap
    - 可以使应用所需的配置信息与程序进行分离，使应用程序可以更好的复用
    - k-v形式，其中v可以是单个值，也可以是全部文件内容
  - Service
    - 实现外部访问的关键。通过Endpoints将请求负载分发到后端的各个容器应用上
    - 3种类型。ClusterIP（默认）、NodePort、LoadBalancer
    - 包含2种均衡分发策略。oundRobin、SessionAffinity
    - Headless Servic，去中心化。对其进行访问将获得所有包含标签app=nginx的Pod列表，而不是单个筛选后的Pod，长与StatefulSet联用
    - coreDNS等网络组件，是Service的基础
  - Ingress
    - 将不同的URL转发到后端不同的Service，实现HTTP层的业务路由机制
  - RBAC
    - K8s默认控制方式
    - RBAC完全由几个API对象完成，可以用kubectl或API进行操作。可以在运行时调整
    - 4个新的顶级资源对象：Role、ClusterRole、RoleBinding和ClusterRoleBinding
    - Role只能对命名空间内的资源进行授权
    - ClusterRole可以对集群范围内的资源进行授权，并且可以授权访问非资源路径，例如“/healthz”
    - RoleBinding把一个角色绑定到一个目标上，绑定目标可以是User（用户）、Group（组）或者Service Account。为某个命名空间授权
    - ClusterRoleBinding绑定中的角色只能是集群角色，用于进行集群级别或对所有命名空间都生效的授权
  - Service Account
    - 也是一种账号，但它并不是给K8s集群的用户使用的，而是给运行在Pod里的进程使用的，它为Pod里的进程提供了必要的身份证明
  - Secret
    - 保管私密数据
    - 三种使用方式。通过为Pod指定SA来自动使用该Secret、通过挂载该Secret到Pod来使用、Docker镜像下载时使用，通过指定Pod 的spec.ImagePullSecrets来引用
  - NetworkPolicy
    - 实现细粒度的容器间网络访问隔离策略
    - 对Pod之间的网络通信进行限制和准入控制，设置方式为将Pod的Label作为查询条件
    - 要与策略控制器（Policy Controller）一起使用
  - PersistentVolume
    - K8s的存储资源对象
    - 可以设置accessMode、Reclaim Policy等参数
    - 支持CephFS、local、NFS、hostPath等多种类型
    - 没有定义StorageClass的PV是静态的PV，有则是动态PV
  - PersistentVolumeClaim
    - 对PV资源的需求申请，需先创建PV
    - 动态供应模式下，删除了PVC，跟它绑定的PV会根据回收策略“Delete”进行处理
  - StorageClass
    - 由系统自动完成PV的创建和绑定，实现动态的资源供应
  - LimitRange
    - 对集群内的Requests和Limits配置基于Namespace的全局限制
    - 创建LimitRange后，Pod等资源必须设置Requests或Limits，否则系统会拒绝Pod的创建
    - LimitRange只会在pod创建或更新时执行检查
  - ResourceQuota
    - 限制命名空间中某种类型的对象的总数目，也可以设置命名空间中Pod可以使用的计算资源的总上限
    - 与LimitRange类型，ResoruceQuota也被设置在Namespace中
  - PodDisruptionBudget
    - PodDisruptionBudget资源对象用于指定一段时间内Pod存活的实例最小数量或百分比
    - 一个PodDisruptionBudget作用于一组被同一控制器管理的Pod（RC或RS等）
  - VPA
    - Pod垂直调度
  - 网络
    - 要求所有节点上的所有容器，都可以在不用NAT的方式下与别的容器进行通信，反之亦然
    - 需要Flannel、coreNDS等开源组件来实现网络需求
    - 容器与容器之间公用同一个网络协议栈，所以它们可以通过localhost+port访问彼此
    - 相同Node上的2个Pod，自己的一端通过veth连接到Docker网桥即可实现相互通信
    - 不相同Node上的2个Pod。自己手动配置每个Node的路由项，或借助第三方软件实现不相同Node上的Pod的通信
    - CNI，容器运行环境与网络插件之间的简单接口规范
    - Network Policy网络策略。Policy API接收NetworkPolicy定义，Policy Listener进行监听，并将策略通过每个Node的Agent进行实际设置
  - 存储
    - PV，是对底层网络共享存储的抽象，将共享存储定义为一种“资源”
    - PVC是用户对存储资源的一个“申请”。PVC能够“消费”PV资源。PVC可以申请特定的存储空间和访问模式
    - StorageClass，实现PV的动态存储
  - Node管理
    - 隔离，使其脱离Kubernetes集群的调度范围。可以在yaml中定义再replace，也可以使用kubectl cordon demo
    - 恢复，将Node纳入调度范围。kubectl uncordon demo
  - 资源管理
    - LimitRange和ResourceQuota（都属于Admission-controller的一种），前者解决request和limit参数的默认值和取值范围等问题，后则解决约束Namespace的资源配额问题
    - Resource QoS（资源服务质量管理）
      - 保证高可靠性的Pod可以申请可靠资源，而一些不需要高可靠性的Pod可以申请可靠性较低或者不可靠的资源
      - 3个QoS Classes。Guaranteed、Burstable、BestEffor（等级由高到低）
    - Resource Quotas（配额管理）
      - 为每个Namespace都提供一个总体的资源使用的限制
      - 限制命名空间中某种类型的对象的总数目，也可以设置命名空间中Pod可以使用的计算资源的总上限
      - 资源配额与集群资源总量是完全独立的，并且是通过绝对的单位来配置的
  - API Server管理
    - 3种级别的客户端身份认证方式。HTTPS证书认证、HTTP Token、HTTP Base
    - 通过认证后，还需要通过授权策略来决定本次操作是否合法。授权策略主要有Webhook、RBAC等
  - Admission Control
    - 通过了认证和鉴权后，客户端的调用请求还需要通过准入控制
    - 允许用户自定义扩展
    - 有丰富的访问控制策略，通常用在安全方面，如PodSecurityPolicy等
  - Pod驱逐机制
    - kubelet能利用一些信号作为决策依据来触发驱逐行为
    - kubelet可以定义驱逐阀值，一旦超过阀值，就会触发kubelet进行资源回收操作。驱逐阀值又可以通过软阀值或硬阀值两种方式进行设置。
    - 驱逐监控频率。kubelet每隔一个这样的时间间隔就会对驱逐阀值进行评估
    - 节点抖动。节点的状况在软阀值的上下抖动，但是又没有超过宽限期，则会导致该Node的状态在True和False之间不断变换
    - kubelet在驱逐Pod之前，会尝试回收Node级别的资源。基于是否为容器定义了独立的imagefs，会导致不同的资源回收过程
    - 如果kubelet无法通过节点级别的资源回收获取足够的资源，就会驱逐用户的Pod
    - 资源最少回收量
      - 驱逐Pod可能只回收了很少的资源，这就导致了kubelet反复触发驱逐阀值。为了缓和这种状况，kubelet对每种资源都定义minimun-reclaim
    - 节点资源紧缺时的系统行为
  - Pod主动驱逐保护
    - 可能对Pod进行主动驱逐的操作主要有两种。节点在维护和升级时（kubectl drain）、应用的自动缩容操作（autoscaling down）
    - 保持存活的Pod的数量，保证不会在同一时间停掉太多Pod，从而导致服务中断
  - 亲和性调度
  - 集群监控
    - Metrics Server
    - Prometheus+Grafana
  - 日志管理
    - 推荐采用Fluentd+Elasticsearch+Kibana
  - 审计
    - 加强对集群操作的安全监管。主要体现为审计日志（Audit Log）
    - 高级审计特性（Advanced Auditing
  - Helm
    - Kubernetes应用包管理工具
    - 由HelmClient和TillerServer两个组件组成
    - 3个概念。Chart、Release、Repository