1容器
虚拟化和容器关系
1主机级虚拟化
vmware 好像是物理机一样使用
Type-1 硬件->vmm
Type-2 vmware hostos->vmm

内核目的是资源分配和管理，用户应用都是在用户空间的进程中

2每次虚拟机都要装内核，代价太大了。那就直接在用户空间进行虚拟化，也就是容器虚拟化技术

减少中间层和中间环节就是提升效率

```
lxc：LinuX Container
        chroot，根切换；
        namespaces：名称空间进行隔离
        CGroups：控制组

        简单使用：
                lxc-checkconfig：
                        检查系统环境是否满足容器使用要求；
                lxc-create：创建lxc容器；
                        lxc-create -n NAME -t TEMPLATE_NAME
                lxc-start：启动容器；
                        lxc-start -n NAME -d

                        Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself
                lxc-info：查看容器相关的信息；
                        lxc-info -n NAME
                lxc-console：附加至指定容器的控制台；
                        lxc-console -n NAME -t NUMBER
                lxc-stop：停止容器；
                lxc-destory：删除处于停机状态的容器；

                lxc-snapshot：创建和恢复快照；


Docker安装方法：

    docker双发行版：
        docker-ee
        docker-ce
            moby

    1、CentOS Extras Repo
    2、Docker-CE

    下载：https://download.docker.com/


        仓库配置文件：https://download.docker.com/linux/centos/docker-ce.repo

Docker组件：



    docker程序环境：
        环境配置文件：
            /etc/sysconfig/docker-network
            /etc/sysconfig/docker-storage
            /etc/sysconfig/docker
        Unit File:
            /usr/lib/systemd/system/docker.service
        Docker Registry配置文件：
            /etc/containers/registries.conf

        docker-ce:
            配置文件：/etc/docker/daemon.json

    Docker镜像加速
        docker cn                {
        阿里云加速器                 "registry-mirrors": ["https://registry.docker-cn.com"]
        中国科技大学
        Client <--> Daemon <--> Registry Server
逻辑：
        Containers：容器
        Images：镜像、映像
        Registry：Image Repositories

容器的状态：
        created：
        runing：
        paused：
        stopped：
        deleted：



docker
        images
        pull
        run
        ps

查看docker相关的信息：
        version
        info

镜像：
        images
        rmi
        pull

容器：
        run：创建并运行一个容器；
        create：创建一个容器；
        start：启动一个处于停止状态容器；

        创建：
                create
                run

        启动：
                start

        停止：
                kill
                stop

        重启：
                restart

        暂停和继续：
                pause
                unpause

        删除容器：
                rm
                run --rm

创建容器：
        基于“镜像文件”，
                镜像文件有默认要运行的程序；

        注意：
                运行的容器内部必须有一个工作前台的运行的进程；
                docker的容器的通常也是仅为运行一个程序；
                        要想在容器内运行多个程序，一般需要提供一个管控程序，例如supervised。

        run, create
                --name CT_NAME
                --rm：容器运行终止即自行删除
                --network BRIDGE：让容器加入的网络；
                        默认为docker0；

                交互式启动一个容器：
                        -i：--interactive，交互式；
```

```
                                -t：Allocate a pseudo-TTY

                                从终端拆除：ctrl+p, ctrl+q

                attach：附加至某运行状态的容器的终端设备；

                exec：让运行中的容器运行一个额外的程序；

                查看：
                        logs：Fetch the logs of a container，容器内部程序运行时输出到终端的信息；

                        ps：List containers
                                -a, --all：列出所有容器；
                                --filter, -f：过滤器条件显示
                                        name=
                                        status={stopped|running|paused}

                        stats：动态方式显示容器的资源占用状态：

                        top：Display the running processes of a container


        Docker Hub：
                docker login
                docker logout

                docker push
                docker pull

        镜像制作：
                基于容器制作
                        在容器中完成操作后制作；
                基于镜像制作
                        编辑一个Dockerfile，而后根据此文件制作；

                基于容器制作：
                        docker commit
                                docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
                                        --author, -a
                                        --pause, -p
                                        --message, -m

                                        --change, -c

                将镜像文件导出为tar文件:
                        docker save
                                Save one or more images to a tar archive (streamed to STDOUT by default)

                                docker save [OPTIONS] IMAGE [IMAGE...]


                从tar文件导入镜像 ：
                        docker load
                                Load an image from a tar archive or STDIN

                                docker load [OPTIONS]

                                        --input, -i             Read from tar archive file, instead of STDIN
                                        --quiet, -q     false   Suppress the load output


    Docker参考手册：
        https://docs.docker.com/engine/reference/commandline/dockerd/

    配置docker守护进程的属性信息的方法：/etc/docker/daemon.json
        每一个可设置的键是dockerd的可用的选项，其值为选项的参数；但有些参数不可用于此文件中，例如add-registry,
insecure-registry；
            有些选项的参数是数组的格式，需要放置于[]；

        官方手册（完整的可用参数列表）：
            https://docs.docker.com/engine/reference/commandline/dockerd/#run-multiple-daemons

            {
                "authorization-plugins": [],
                "data-root": "",
                "dns": [],
                "dns-opts": [],
                "dns-search": [],
                "exec-opts": [],
                "exec-root": "",
                "experimental": false,
                "storage-driver": "",
                "storage-opts": [],
                "labels": [],
                "live-restore": true,
                "log-driver": "",
                "log-opts": {},
                "mtu": 0,
                "pidfile": "",
                "cluster-store": "",
                "cluster-store-opts": {},
                "cluster-advertise": "",
                "max-concurrent-downloads": 3,
                "max-concurrent-uploads": 5,
                "default-shm-size": "64M",
                "shutdown-timeout": 15,
                "debug": true,
                "hosts": [],
                "log-level": "",
                "tls": true,
                "tlsverify": true,
                "tlscacert": "",
                "tlscert": "",
                "tlskey": "",
                "swarm-default-advertise-addr": "",
                "api-cors-header": "",
                "selinux-enabled": false,
                "userns-remap": "",
                "group": "",
                "cgroup-parent": "",
                "default-ulimits": {},
                "init": false,
                "init-path": "/usr/libexec/docker-init",
                "ipv6": false,
                "iptables": false,
                "ip-forward": false,
                "ip-masq": false,
                "userland-proxy": false,
                "userland-proxy-path": "/usr/libexec/docker-proxy",
                "ip": "0.0.0.0",
                "bridge": "",
                "bip": "",
                "fixed-cidr": "",
                "fixed-cidr-v6": "",
                "default-gateway": "",
                "default-gateway-v6": "",
                "icc": false,
                "raw-logs": false,
                "allow-nondistributable-artifacts": [],
                "registry-mirrors": [],
                "seccomp-profile": "",
                "insecure-registries": [],
                "disable-legacy-registry": false,
                "no-new-privileges": false,
                "default-runtime": "runc",
                "oom-score-adjust": -500,
                "runtimes": {
                    "runc": {
                        "path": "runc"
                    },
                    "custom": {
                        "path": "/usr/local/bin/my-runc-replacement",
                        "runtimeArgs": [
                            "--debug"
                        ]
                    }
                }
            }
```

dockerd守护进程的C/S，其默认仅监听Unix SOcket格式的地址，/var/run/docker.sock；如果使用TCP套接字，

```
    /etc/docker/daemon.json:
        "hosts": ["tcp://0.0.0.0:2375", "unix:///var/run/docker.sock"]
```

也可向dockerd直接传递“-H|--host”选项；

自定义docker0桥的网络属性信息：/etc/docker/daemon.json文件

```
{
    "bip": "192.168.1.5/24",
    "fixed-cidr": "10.20.0.0/16",
    "fixed-cidr-v6": "2001:db8::/64",
    "mtu": 1500,
    "default-gateway": "10.20.1.1",
    "default-gateway-v6": "2001:db8:abcd::89",
    "dns": ["10.20.1.2","10.20.1.3"]
}
```

核心选项为bip，即bridge ip之意，用于指定docker0桥自身的IP地址；其它选项可通过此地址计算得出。

文档路径：
https://docs.docker.com/engine/userguide/networking/default_network/custom-docker0/

容器构建示例：
https://github.com/mysql/mysql-docker

容器的资源限制：

CPU:
RAM:
Device:

```
--device-read-bps value       Limit read rate (bytes per second) from a device (default [])
--device-read-iops value      Limit read rate (IO per second) from a device (default [])
--device-write-bps value      Limit write rate (bytes per second) to a device (default [])
--device-write-iops value     Limit write rate (IO per second) to a device (default [])
```

Docker private Registry的Nginx反代配置方式：

```
client_max_body_size 0;

location / {
    proxy_pass  http://registrysrvs;
    proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
    proxy_redirect off;
    proxy_buffering off;
    proxy_set_header        Host            $host;
    proxy_set_header        X-Real-IP       $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;

    auth_basic "Docker Registry Service";
    auth_basic_user_file "/etc/nginx/.ngxpasswd";
}
```

Docker-distribution配置文件格式详细信息：
https://docs.docker.com/registry/configuration/#list-of-configuration-options

Kubernetes

```
        架构：master/agent
                master主机：
                        kube-apiserver
                        kube-scheduler
                        kube-controller-manager

                agent主机（node）：
                        kubelet
                        container runtime(docker/rkt/...)
                        kube-proxy


容器编排三套解决方案：
    kubernetes
    mesos+marathon
    machine+swarn+compose

    Kubernetes：
        组件：master, nodes, database(k/v store)
            master：apiserver, controller-manager, scheduler
            nodes: kubelet, kube-proxy, container runtime
        核心术语：
            Pod, label, service, ingress
        网络插件：flannel, ...






Kubernetes-1.8安装：
    yum 仓库：
        https://yum.kubernetes.io/
        https://packages.cloud.google.com/yum/repos





 Kubernetes Cluster：
        环境：
                master, etcd：172.18.0.67
                node1：172.18.0.68
                node2：172.18.0.69
        前提：
                1、基于主机名通信：/etc/hosts；
                2、时间同步；
                3、关闭firewalld和iptables.service；

                OS：CentOS 7.3.1611, Extras仓库中；

        安装配置步骤：
                1、etcd cluster，仅master节点；
                2、flannel，集群的所有节点；
                3、配置k8s的master：仅master节点；
                        kubernetes-master
                        启动的服务：
                                kube-apiserver, kube-scheduler, kube-controller-manager
                4、配置k8s的各Node节点；
                        kubernetes-node

                        先设定启动docker服务；
                        启动的k8s的服务：
                                kube-proxy, kubelet


        deployment示例：

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  # Unique key of the Deployment instance
  name: deployment-example
spec:
  # 2 Pods should exist at all times.
  replicas: 2
  template:
    metadata:
      labels:
        # Apply this label to pods and default
        # the Deployment label selector to this value
        app: nginx
    spec:
      containers:
```

```
      - name: nginx
        # Run this image
        image: nginx:1.12

        service示例：

        kind: Service
        apiVersion: v1
        metadata:
        # Unique key of the Service instance
            name: nginx-example
        spec:
            ports:
                # Accept traffic sent to port 80
                - name: http
                  port: 80
                  targetPort: 80
            selector:
                # Loadbalance traffic across Pods matching
                # this label selector
                app: nginx
            # Create an HA proxy in the cloud provider
            # with an External IP address - *Only supported
            # by some cloud providers*
            type: LoadBalancer



Docker Compose

    MySQL:
        mysql: ### 容器名称
            image: mysql:5.7 ### 官方镜像 版本号5.7
            volumes:
                - mysql-data:/var/lib/mysql ### 数据卷，mysql数据就存放在这里
            ports:
                - "3306:3306" ###端口映射，主机端口:容器对外端口
            environment:
                - MYSQL_ROOT_PASSWORD=123456  ### 设置环境变量，这个变量名是官方镜像定义的。

        PHP:
        php-fpm:
            build:
                context: ./php ### 自定义PHP镜像的配置目录
            volumes:
                - ./www:/var/www/html ### 主机文件与容器文件映射共享，PHP代码存这里
            expose:
                - "9000" ### 容器对外暴露的端口
            depends_on:
                - mysql ### 依赖并链接Mysql容器，这样在PHP容器就可以通过mysql作为主机名来访问Mysql容器了

      Nginx:
        nginx:
            build:
                context: ./nginx ### 自定义Nginx镜像的配置目录
            volumes:
                - ./www:/var/www/html 主机文件与容器文件映射共享，PHP代码存这里
            ports:
                - "80:80" ### 端口映射，如果你主机80端口被占用，可以用8000:80
                - "443:443"
            depends_on:
                - php-fpm ### 依赖并连接PHP容器，这样在Nginx容器就可以通过php-fpm作为主机名来访问PHP容器了


 Kubernetes:

    master/node
    pod：network, uts, storage volumes
        PodIP

    master主机：
        apiserver、scheduler、controller-manager、etcd (CoreOS, raft, zab)
    node主机：
        kubelet(agent), kube-proxy(userspace/iptables/ipvs), container engine
    逻辑组件：
        Pod：容器集，
            原子调度单元：一个Pod的所有容器要运行于同一个节点；

                nmt：
                    tomcat <- nginx
                    mariadb <- tomcat application
                    nginx <- Client

            label

        Controller --> label selector --> Pod (label)
            管理Pod：确保Pod副本数量严格符合用户定义；
        Service --> label selector --> Pod (label)
            为Pod中的应用的客户端提供一个固定的访问端点：ClusterIP:ServicePort
                ServiceName --> ClusterIP
                DNS Addon

    NodeIP: Node Network
    ClusterIP: Cluster Network, Service
    Pod IP: Pod Network, Pod


    kubernetes rpm repo：
        https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64/

    kubeadm部署集群的文档：
        https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

    google containers registry：
        https://console.cloud.google.com/gcr/images/google-containers?project=google-containers


    在所有主机上执行：
        1、kubeadm的配置文件：
            # vim /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
                cgroups_driver=""
                docker info中显示的cgroup_driver一致；
        2、关闭swap
            swapoff -a

        3、设置docker和kubelet开机自启动
            systemctl enable docker.service kubelet.service

        4、启动docker

        5、load各镜像


    在master节点上执行：
        1、初始化master：
            kubeadm init --kubernetes-version=v1.10.0 --pod-network-cidr=10.244.0.0/16



        Your Kubernetes master has initialized successfully!

        To start using your cluster, you need to run the following as a regular user:

        mkdir -p $HOME/.kube
        sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
        sudo chown $(id -u):$(id -g) $HOME/.kube/config

        You should now deploy a pod network to the cluster.
        Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
        https://kubernetes.io/docs/concepts/cluster-administration/addons/

        You can now join any number of machines by running the following on each node
        as root:

        kubeadm join 172.18.0.80:6443 --token 7nn84i.vz7te46xm11bbjiq --discovery-token-ca-cert-hash
sha256:45920191c24cdbf496df9a3874421197aa1eab9d90021a5cdb18f5e2bb5183ef
```

在每个一node上执行：

```
    # kubeadm join 172.18.0.80:6443 --token 7nn84i.vz7te46xm11bbjiq --discovery-token-ca-cert-hash
sha256:45920191c24cdbf496df9a3874421197aa1eab9d90021a5cdb18f5e2bb5183ef
```

4、基础应用命令

- kubectl run: 创建deployment控制器，并根据用户指定的镜像创建pod资源；
- kubectl scale：应用扩缩容；
- kubectl expose：创建service资源，用于为某些pod提供固定访问端点；
- kubectl set image: 升级应用

kubectl命令管理对象的方式有三种：

- 直接命令

```
 REPOSITORY                                   TAG                 IMAGE ID            CREATED             SIZE
k8s.gcr.io/kube-proxy-amd64                  v1.10.0             bfc21aadc7d3        13 days ago         97MB
k8s.gcr.io/kube-controller-manager-amd64     v1.10.0             ad86dbed1555        13 days ago         148MB
k8s.gcr.io/kube-scheduler-amd64              v1.10.0             704ba848e69a        13 days ago         50.4MB
k8s.gcr.io/kube-apiserver-amd64              v1.10.0             af20925d51a3        13 days ago         225MB
k8s.gcr.io/etcd-amd64                        3.1.12              52920ad46f5b        4 weeks ago         193MB
quay.io/coreos/flannel                       v0.10.0-amd64       f0fad859c909        2 months ago        44.6MB
k8s.gcr.io/pause-amd64                       3.1                 da86e6ba6ca1        3 months ago        742kB


node.tar
quay.io/coreos/flannel                       v0.10.0-amd64       f0fad859c909        2 months ago        44.6MB
k8s.gcr.io/pause-amd64                       3.1                 da86e6ba6ca1        3 months ago        742kB
k8s.gcr.io/kube-proxy-amd64                  v1.10.0             bfc21aadc7d3        13 days ago         97MB
```

kubectl
直接命令：run, expose, scale, set image,
资源配置文件：命令式（create）
资源配置文件：声明式（apply）

资源：（属性：值）

- apiVersion: groupname/version
- kind：种类，Pod/Service/Deployment/ReplicationController/...
- metadata：元数据， object
  - name：名称
  - namespace：名称空间，默认为default
  - labels：标签
  - annotations：注解

- spec: 定义期望的目标状态

  用户定义时使用的核心字段；

- status：当前状态

  是由kubernetes系统自动维护，管理员不能人为修改；

kubernetes的核心目标在于：让每个资源的当前状态无限接近于由用户定义的目标状态；

资源管理动作：CRUD

- kubectl
  - create
  - delete
  - get
  - edit, replace

- kubectl
  - apply：增、改
  - delete
  - patch
  - get

Pod的定义完整示例：

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: 2018-04-11T07:30:05Z
  name: mypod
  namespace: default
  resourceVersion: "17419"
  selfLink: /api/v1/namespaces/default/pods/mypod
  uid: 27a47a00-3d5a-11e8-84a2-000c296c3adf
spec:
  containers:
  - image: nginx:1.12-alpine
    imagePullPolicy: IfNotPresent
    name: nginx
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-sw47w
      readOnly: true
  dnsPolicy: ClusterFirst
  nodeName: server3.magedu.com
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: default-token-sw47w
    secret:
      defaultMode: 420
      secretName: default-token-sw47w

    每个属性的功用及格式都可以使用kubectl explain获取；


配置Pod资源：
    spec内嵌的字段（属性）：
        containers：对象列表；
            内建字段：
                name：容器名；
                image：启动容器使用的镜像；
                imagePullPolicy：获取镜像策略，下面是可用值列表
                    Always：总是重新到registry获取镜像文件；
                    Never：从不，仅使用本地镜像；
                    IfNotPresent：仅本地不存在时才去获取；
                ports：要暴露的端口，仅用标识，下面是可用的内建字段
                    containerPort：
                    name：
                    protocol：TCP/UDP
                command：自定义要运行的容器应用，字串列表；
                env：对象列表，可用到如下内建字段：
                    name：变量名；
                    value：变量值；

      标签及其选择器：
        metadat内建：
            labels：映射
                key：最长63个字符，字母、数字、下划线_、点号、连接线-
                value：最长63个字符，可以为空，字母、数字、下划线_、点号、连接线-

        显示资源标签：
            kubectl get --show-labels
            kubectl get -l KEY=VALUE

        标签选择器：
            基于等值关系的选择器：等值选择器；
                =, ==, !=
            基于集合的选择器：集合选择器；
                KEY in (VALUE1, VALUE2, ...)
                KEY notin (VALUE1, VALUE2, ...)
                KEY：存在此标签的所有资源；
                !KEY：不存在此标签的所有资源；

    每个资源都支持的三个核心字段：apiVersion、kind、metadata（name, namespace, labels, annotations）

    ReplicaSet的核心配置：
        期望的副本数量
        标签选择器
        Pod模板


    apiVersion: apps/v1
    kind: ReplicaSet
    metadata:
        name: rs-demo
        namespace: default
        labels:
            controller: rs-demo
    spec:
        replicas: 2
        selector:
            matchLabels:
                app: rs-demo-nginx
        template:
            metadata:
                name: rs-demo-pod
                labels:
                    app: rs-demo-nginx
            spec:
                containers:
                - name: nginx
                    image: nginx:1.12-alpine
                    imagePullPolicy: IfNotPresent
                    ports:
                    - name: http
                    containerPort: 80

    Deployment控制器：
        借助于ReplicaSet中间层来管理Pod资源；
            ReplicaSet name：deployname-HASH
                Pod Name: deployname-HASH-POD_HASH

        嵌套字段：
            replicas
            selector
            template

            revisionHistoryLimit <integer>：保留的replicaset资源历史版本数；用于回滚；
            strategy     <Object>：更新策略
                type：策略类型，Recreate, RollingUpdate

                rollingUpdate：为滚动更新机制定义其更新控制逻辑
                    maxSurge：更新期间，存在的由当前控制器控制的总Pod数量可超出期望值多少：
                        数值：0-N
                        百分比：0-100%
                    maxUnavailable：更新期间，存在的由当前控制器控制的总Pod数量可少于期望值多少；
                        数值：0-N
                        百分比：0-100%



            paused       <boolean>：当前控制器是否为暂停状态；


        apiVersion: apps/v1
        kind: Deployment
        metadata:
            name: deploy-demo
            namespace: default
            labels:
                controller: deploy-demo
        spec:
            replicas: 2
            selector:
                matchLabels:
                app: nginx-demo
            template:
                metadata:
                name: pod-demo
                labels:
                    app: nginx-demo
                spec:
                containers:
                - name: nginx
                    image: nginx:1.12-alpine
                    imagePullPolicy: IfNotPresent
                    ports:
                    - name: http
                    containerPort: 80
```

```
  数据类型：
      string
      boolean
      list:
          表示方式：["item1","item2",...]
          表示方式：
              - "item1"
              - "item2"
      object：
          内嵌其它字段；
      []object：对象列表
          - field1: value
            field2: value
            field3: value
          - field1: value
            field2:  value
      map：
          关联数组：以key:value依次给出；


  Service：
      Endpoint：端点
          PodIP, Pod Port：Endpoint

  Service Type：
      ClusterIP
      NodePort
      LoadBalancer
      ExternalName

  Kubernetes Cluster：
      核心组件类别：master/node
          Addons：附件
              dns：
                  skydns
                  kube-dns
                  coreDNS
```

```
kubectl run client --image=cirros --rm -it -- /bin/sh
```

```
Pod状态监控：
  liveness probe：存活性探测；
      控制器可基于存活性探测来判定pod资源是否为健康状态，是否需要重启或重构；
```

readiness probe：就绪性探测；

为某service资源将某后端Pod资源添加至service之上时，要事先进行pod资源的就绪状态检测，以避免把未初始化完成的Pod调度给请求者。

假如：service, deployment

```
{
  "port": "8080",
  "use_auth": false,

  "jwt": {
    "admin": {
      "key": "admin"
    },
    "user": {
      "key": "heketi"
    }
  },

  "glusterfs": {
    "executor": "ssh",

    "sshexec": {
      "keyfile": "/etc/heketi/heketi_key",
      "user": "root",
      "port": "22",
      "fstab": "/etc/fstab"
    },

    "_db_comment": "Database file name",
    "db": "/var/lib/heketi/heketi.db",

    "loglevel" : "debug"
  }
}
```

```
heketi-cli cluster info fe78e94bcac68d0acde3ad1cbc9067d1
```

Dynamic Provision: 动态供给；

PV动态创建；

heketi+glusterfs：

1、各节点安装glusterfs客户端：glusterfs-client；

2、heketi启用认证时，定义存储类时必须给定其用户名和密码；

restuser:

restuserkey: 不应该以明文方式直接给出；

通过k8s的另一个标准资源secret给出；

ConfigMap, Secret:

配置容器中的应用的方法：

自定义命令及其参数；

通过环境变量传递参数；

对于不支持通过环境变量加载配置信息，或者仅支持有限的配置通过环境变量获取时需要entrypoint脚本；

通过存储卷额外提供配置文件；

标准的k8s资源；

ConfigMap：包含提供给应用的配置信息；

用户账号的相关信息：

user, group, API, Requestpath, API request verbs

HTTP: GET, HEAD, POST, PATCH, PUT, DELETE
kubectl: get, describe, edit, patch, create, apply, delete

Resources, subresource
namespace

认证：basic, https证书、http token、JWT

授权：Node，ABAC（Attribute-Based Access Control), RBAC(Role-Based Access Control)

RBAC
Role：仅生效于名称空间
ClusterRole：生效于集群级别

RoleBinding：
ClusterRoleBinding：