CCloud对象

- // 写法：'@'对象； '.'属性； '#'操作； '//'注释；
- 基础对象
  - @CObject　　//实体对象，在后面继承的对象中，特别注意的属性才重新声明
    - .meta　　//基础属性。
      - .ID　　//唯一标志，系统生成
      - .名称　　//是否唯一根据对象的业务属性
      - .parent　　//归属对象，属性中文名在具体的对象定义
      - .创建时间　//系统生成
      - .标签　　//可选
      - .图标　　//可选，显示用
      - .描述　　//可选
    - .spec　　//特征属性，一般用来定义对象
    - // .spec.x[]： @Y //表示数组，数组对象类型为 @Y
    - // .spec.x： @Y // 表示这个属性对象类型为 @Y
    - // .spec.x(s) //该属性值有多组，每个值也是对象，但是没有特别定义这个对象
    - .status　　//状态属性 。 运行特征，系统生成
      - .状态
      - .更新时间　//对象初创建时和"创建时间"相等
    - .addition　　//附加属性。根据需要添加
      - .owner //用于一些对象需要分配权限
      - .log
      - .version //对象作为资源的版本信息
    - #功能操作
      - #输入
      - #输出
      - #流程
    - .show　　//根据显示对象封装数据
      - .@Create　//创建对象体现在界面上的过程
        - .@step1
        - .@step2
      - .@ListItem　//如果有被用于列表，单项承载信息
        - .属性
        - #操作
      - .@ListFilter　//列表搜索、过滤、筛选
        - .搜索 = [ ]　　//输入关键字，从哪些字段进行搜索
        - .筛选 = [ ]　　//可以用哪些字段进行筛选
      - .@Detail　　//查看对象详情时，界面希望呈现的内容
        - .属性
        - #操作
  - @C项目：@CObject： Inherited // 原来的 @主应用
    - .meta.项目归属　　// parent 归属的个人/企业
    - .spec.入口：@C服务入口　// 一个项目只设置一个，用于负载、分发
    - .spec.服务[]： @C服务　//服务关联实例，保证稳定访问
    - .spec.应用[]： @C应用 //应用->部署->实例
    - .status.状态
    - #创建项目
      - #流程
        - 1、输入项目及应用属性 //项目和应用基本属性是重复的
        - 2、创建
          - 创建一个 @C项目
          - 创建一个 @C应用

        - 创建一个 @C服务入口
    - .show
        - .@ListItem
            - .名称 = .meta.名称
            - .标签 = .meta.名称
            - .图标 = .meta.图标
            - .描述 = .meta.描述
            - .更新时间 = .status.更新时间
        - .@Detail
            - 基本信息　　//没有加"."，表示这只是属性分类。在前端页面分块显示
                - .名称 = .meta.名称
                - .标签 = .meta.名称
                - .图标 = .meta.图标
                - .描述 = .meta.描述
                - .创建时间 = .status.创建时间
            - 服务访问入口
                - .入口名称 =
                - .访问规则 = @C服务入口.spec.访问规则[]
                - #服务入口详情
            - 微服务列表
                - 列表项 = @C服务.show.@ListItem
            - 应用列表
                - 列表项 = @C应用.show.@ListItem
        - .@Create
            - .@step1 输入相关属性
                - .图标
                - .名称
                - .描述
                - .标签
                - .代码仓库　//提供给应用的。目前先提供Github仓库
                    - 先授权
                    - 再刷出授权账户下的代码仓库
                    - 用户选择一个仓库
            - .@step2 创建
    - .addition
- @C应用 : @CObject： Inherited
    - .meta.所属项目　//parent
    - .spec.代码仓库
        - .类型
            - Github　//当前支持这种
            - Bitbucket
            - Coding.net
            - GitLab
        - .授权　　//通常是一个Token
        - .代码仓库　//URL
    - .spec.镜像 [] : @C镜像
    - .spec.部署 [] : @C部署
    - .status.状态 = 初创建 | 正常运行　//镜像/部署不为空就不能删除
    - .status.更新时间
    - .show.@listItem
        - .应用图标 = .meta.图标
        - .应用名称 = .meta.名称

        - .描述 = .meta.描述
        - .更新时间 = .status.更新时间
    - .show.@detail
        - 应用信息
            - .应用图标 = .meta.图标
            - .应用名称 = .meta.名称
            - .所属项目 = .meta.所属项目
            - .描述 = .meta.描述
            - .标签 = .meta.标签
            - .代码仓库 = //原来界面上为“镜像来源” #updated
            - .更新时间 = .status.更新时间
        - 部署列表
            - 列表项 = @C部署.show.@listItem
        - 镜像及版本
            - 列表项 = @C镜像.show.@listItem
    - .show.@Create
        - .@step1 输入相关属性
            - .图标
            - .名称
            - .描述
            - .标签
            - .代码仓库 //目前先提供Github仓库
                - 先授权
                - 再刷出授权账户下的代码仓库
                - 用户选择一个仓库
        - .@step2 创建
- @C镜像 : @CObject： Inherited //和 @DockerImage 对应，抽象管理
    - .meta.所属应用 //parent，对象嵌套的，界面上一般显示名称
    - .spec.版本号 //构建时定义，对应 @DockerImage tag
    - .spec.镜像文件 // @DockerImage 和docker镜像对应
    - .spec.来源 = "内部构建" | "外部镜像"
    - .spec.存放路径 //在私有镜像仓库的路径
    - .spec.DockerFile : @DockerFile //配置文件，用于指导生成镜像
    - .status.状态 = 可用 | 不可用
    - .addition.日志 //需要记录构建过程
    - #构建镜像 //用DockerFile 生成 @DockerImage
        - #输入
            - .镜像名称
            - .版本号
            - .代码URL
            - .DockerFile
        - #输出
            - .镜像 //@C镜像
        - #流程
            - 1、填写名称、版本，选择代码分支
            - 2、编写DockerFile
            - 3、构建
                - 构建成功：到4
                - 构建失败：回到2
            - 4、保存镜像到仓库，结束
    - .show.@Create //构建镜像
        - .@step1 填写名称、版本，选择代码分支
            - .镜像名称 //输入

        - .版本号　//输入
        - .代码分支　//选择，把代码仓库刷出来选择
      - .@step2 编写DockerFile，构建镜像
        - #DockerFile语法参考　//点击，查看页面
        - #查看构建日志　//查看，构建构成实时显示；构建后可随时调出查看
        - .DockerFile文件　//文本编辑
          - #保存 (编辑状态显示) /#修改 (查看状态显示)
        - #开始构建 (首次进入该步骤) / #重新构建 (构建失败一次后)
          - 向服务端发起构建请求
          - 界面实时显示构建过程
          - 构建结束，提示构建结果：
            - 构建成功：进入应用详情页
            - 构建失败：停在本页面，由用户重新编辑文件后，再次发起构建请求
  - .show.@listItem　//该类型对象被放到列表中，列表的每项内容
    - .版本号 = .spec.版本号
    - .状态 = .status.状态
    - .更新时间 = .status.更新时间
    - .存放路径 = .spec.path
    - .镜像来源 = .spec.来源
    - #删除
  - .show.@detail
    - 基本信息
      - .版本号 = .spec.版本号
      - .状态 = .status.状态
      - .更新时间 = .status.更新时间
      - .存放路径 = .spec.path
      - .镜像来源 = .spec.来源
    - DockerFile
      - DockerFile文件内容 = .spec.DockerFile
    - 构建日志
      - //显示构建过程日志
- @C应用实例 : @CObject： Inherited //对应 @K8sPod
  - .meta.所属部署　//parent，作为它的控制器
  - .spec.容器(s)　//建议单容器，一实例一容器一进程
  - .spec.存储卷(s)　//Volume，用来持久化实例的存储资源
  - .spec.节点
    - .节点名称
    - .节点标签
    - .资源容量
    - .节点资源要求
      - cpu
      - 内存
      - 实例数量上限
  - .status.所处生命周期　//phase
    - 值及原因
      - 挂起(Pending)
        - Pod 已被 Kubernetes 系统接受，但有一个或者多个容器镜像尚未创建。等待时间包括调度 Pod 的时间和通过网络下载镜像的时间，这可能需要花点时间。
      - 运行(Running)
        - 该 Pod 已经绑定到了一个节点上，Pod 中所有的容器都已被创建。至少有一个容器正在运行，或者正处于启动或重启状态。
      - 成功(Succeeded)　//表示通过检测
        - Pod 中的所有容器都被成功终止，并且不会再重启。
      - 失败(Failed) //表示没有通过检测

        - Pod 中的所有容器都已终止了，并且至少有一个容器是因为失败终止。也就是说，容器以非0状态退出或者被系统终止。
      - 未知(Unknown) //表示检测没有正常进行
        - 因为某些原因无法取得 Pod 的状态，通常是因为与 Pod 所在主机通信失败。
    - 生命周期
      - 0 start: accept by kube
      - 1 pending
      - 2 running goto
        - 3 succeed, goto end
        - 4 failed，重启后 goto 2
      - 5 end
    - //探针说明 //有三种类型的处理程序：
      - ExecAction：在容器内执行指定命令。如果命令退出时返回码为 0 则认为诊断成功。
      - TCPSocketAction：对指定端口上的容器的 IP 地址进行 TCP 检查。如果端口打开，则诊断被认为是成功的。
      - HTTPGetAction：对指定的端口和路径上的容器的 IP 地址执行 HTTP Get 请求。如果响应的状态码大于等于200 且小于 400，则诊断被认为是成功的。
  - .status.网络
    - .hostIP //节点IP
    - .popIP
  - .status.开始时间
- @C容器 : @CObject： Inherited //对应 @DockerContainer
  - //独立运行的一个或一组应用，以及它们的运行态环境
  - .meta.名称
  - .meta.UID
  - .spec.镜像： @C镜像
  - .spec.挂载卷[] //持久化数据
  - .spec.工作目录
  - .spec.网络
    - .端口
    - ...
  - .spec.环境变量
  - .status.状态
    - waiting
    - running
    - terminated
  - .status.重启次数
  - .status.上一次启动时间　　//上一次启动或者重启的时间
  - #启动
  - #终止
  - #删除
  - .show
- @C部署 : @CObject ： Inherited
  - .meta.所属应用　　//.meta.parent
  - .spec.部署配置　　//部署的配置文件，系统统一选择yaml格式
  - .status.状态　　//为检测状态
    - 进行中 // progressing
      - Deployment 正在创建新的控制器过程中。
      - Deployment 正在扩容一个已有的控制器。
      - Deployment 正在缩容一个已有的控制器。
      - 有新的可用的 pod 出现。
    - 已完成 //complete
      - Deployment 最小可用。最小可用意味着 Deployment 的可用 副本个数等于或者超过Deployment策略中的期望个数。
      - 所有与该 Deployment相关的副本都被更新到了您指定版本，也就说更新完成。
      - 该 Deployment 中没有旧的 Pod 存在。

    - 失败 // fail to progress
        - 无效的引用
        - 不可读的探针 //probe failure
        - 镜像拉取错误
        - 权限不够
        - 范围限制
        - 程序运行时配置错误
- .status.应用实例[]： @C应用实例 //每一个对应一个 @K8sPod
- .status.更新时间
- #部署
    - #输入
        - .部署名称
        - .部署节点[@C节点]
        - .部署的yaml文件
    - #输出
        - 一个 @C部署
        - N个 @C应用实例 //N为设置的实例数量
    - #流程
        - 1、填写部署名称，选择集群
        - 2、定义部署
        - 3、确认部署，开始部署
            - 部署成功：
                - 生成一个部署
                - 按配置，用部署控制运行应用实例
            - 部署失败：
                - 回到3，修改后，重新部署
- #更新
- #删除
- .show.@Create //创建部署
    - .@step1 填写部署名称，选择集群
        - .部署名称 //输入，默认：应用名称+“-”+日期+“-”+n；用递增数字来做后缀区分同一天的部署
        - .集群 //选择，从该账户拥有的集群中进行选择，即圈定节点
            - 集群列表项 = @C集群.show.@ListItem
    - .@step2 定义部署
        - #用YAML文件定义部署 //直接跳转到 .@step3 的编辑状态
        - 通过界面先配置一些通用、重要属性
            - .预设实例数量 //数字
            - .实例标签 //提示："用于关联服务，格式：key1=value1,key2=value2" //这里主要为匹配k8s规则
            - .容器[]
                - .名称 //赋值给 @C容器.meta.名称
                - .镜像 //将 .meta.所属应用 的镜像列表列出，供选择
                    - 界面取值： 镜像名称 镜像版本号
    - .@step3 确认，部署 //统一转化为YAML形式
        - #YAML文件如何写 //点击，查看页面
        - #查看部署日志
        - YAML部署配置文件 //文本编辑
        - #开始部署 (首次进入该步骤) / #重新部署 (部署失败一次后)
            - 部署过程显示部署日志
            - 部署成功
                - #查看部署 //调整至 部署详情页
            - 部署失败
                - 停留在本页，等待用户修复配置文件后，再发起部署
- .show.@listItem

        - .部署名称　//可链接到对应的部署
        - .所属应用　//可链接到对应的应用
        - .预设应用实例数量 = .spec.部署配置文件.预设实例数量
        - .实际运行实例数量 = .status.应用实例(s) 数量
        - .运行时间　//单位：60秒之内用s，60分钟之内用m，24小时之内用h
        - .更新时间
    - .show.@detail
        - 基本信息
            - .部署名称　//可链接到对应的部署
            - .所属应用　//可链接到对应的应用
            - .创建时间
        - 部署状态
            - .部署状态 = .status.状态
            - .状态说明 //可能导致该状态的原因
            - .更新时间 = .status.更新时间
            - .运行时间 = .status.运行时间
        - 实例情况
            - 总览
                - .预设实例数量 = .spec.部署配置文件.预设实例数量
                - .当前实例数量 = .status.应用实例[] 数量
                - .可用实例数量 =
            - 实例列表
                - 列表项 = @C应用实例.show.@ListItem
        - 部署定义
            - 部署yaml文件显示
            - #编辑 / #重新部署
        - 更新记录
            - 记录项 =
                - .更新时间
                - .操作账户
                - .行为 = 创建部署 | 更新部署 | 删除部署 | 应用部署
                - .命令
                - .内容 = yaml文件
        - #功能操作：
            - #更新部署
            - #删除部署
- @C服务 : @CObject： Inherited //对应 @K8sService
    - .meta.名称
    - .meta.标签
    - .meta.所属项目　//parent
    - .spec.功能类型 //创建服务时，需要选择，以便异常管理的性能监控
        - 集群服务
            - 存储服务
            - 调度服务
            - 控制器管理服务
            - 接入服务
        - 业务相关
        - 通用服务 //
            - CDN
            - 负载均衡
            - 日志服务
            - 消息服务
            - 数据库

        - 缓存服务
        - 对象存储服务
        - 反向代理服务
        - 搜索引擎
        - ...
- .spec.服务来源
    - 集群内，通过标签选择
    - 集群外，通过<IP:端口>映射服务
    - 集群外，通过别名映射服务
- .spec.服务关联　//关联的应用实例、外部服务，3种情况和上面的来源一一对应
    - .应用标签
        - 支持对标签的key或者key/value进行运算，逗号(表示AND)分割每组筛选，运算支持：
            - “=”，“==”和“!=”
            - in ， notin 和 exists(仅针对于key符号)
    - .IP和端口
        - .IP
        - .端口
    - .别名
        - .别名
        - .命名空间
- .spec.暴露方式
    - 通过集群指派IP，仅集群内访问
    - 开放节点端口，集群外部可访问
    - 通过负载均衡器，集群外可访问
- .spec.访问端口[]
    - .端口名称
    - .端口号：
    - .协议：TCP/UDP
    - 目标端口： //端口名称或端口号
- .status.状态
    - 正常
    - 异常
    - 未知
- .status.访问IP　//允许定义时指派，也允许系统指派。
    - //系统分配的IP需要运行起来才知道，所以把这个属性放在.status下
    - .IP类型 =
        - 集群分配给应用实例的IP
        - 节点IP
        - 外部IP
        - 负载均衡器IP
    - .IP(s)
- .status.服务后端IP和端口(s)　//服务真正到达的IP和端口，对应 k8s的 Endpoints
- .status.更新时间
- .addition
    - .owner //用于一些对象需要分配权限
    - .log
    - .version //对象作为资源的版本信息
- #创建服务
    - #Flow
        - 1 设置服务名称
        - 2 选择服务来源
            - 2.1 集群内应用，通过标签选择 //圈定pods
            - 2.2 集群外，通过<IP:端口>映射服务

        - 2.3 集群外，通过别名映射服务
      - 3 设置访问IP和端口
        - 设置端口 //可多个
        - 选择服务暴露方式
          - 通过集群IP，仅集群内部访问
          - 开放节点端口，集群外部可访问
          - 通过负载均衡器，集群外部可访问 //选择这种形式，创建服务后，还有在创建一个 @Ingress
      - 4 确认服务配置，创建服务
    - #Input
      - 服务定义转化成Yaml形式的文件
    - #Output
      - 一个新 @CService 和 对应的端点集合 @K8sEndpoint
- #更新服务
- #终止服务
- .show
  - .@创建服务
    - @Step1 设置服务名称
      - 服务名称：
      - 服务标签：//提示：“格式：key=value, key1=value1...“
    - @Step2 选择服务来源
      - 2.1 集群内应用，通过标签选择
        - 标签选择：//输入框，如frontend:environment=production, tier!=frontend
          - 下方提示：
            - 支持对标签的key或者key/value进行运算，逗号(表示AND)分割每组筛选，运算支持：
              - “=”，“==”和“!=”
              - in ， notin 和 exists(仅针对于key符号)
            - #查看集群内有哪些实例标签
              - [子应用名称] [实例数量] //[子应用名称]是链接，可跳转到 #子应用详情页
                - key1=value //标签
                - key2=value2
              - [子应用名称] [实例数量]
                - key1=value3
                - key3=value3
      - 2.2 集群外，通过<IP:端口>映射服务
        - IP地址：
        - 端口号：
        - //我们会自动帮忙创建Endpoints路由表
      - 2.3 集群外，通过别名映射服务 //type=ExtraName，直接跳到4
        - namespace：//没有输入，则为系统默认
        - externalName：输入CNAME，如：my.database.example.com
    - @Step3 设置访问IP和端口
      - 选择服务暴露方式
        - 3.1 通过集群IP，仅集群内部访问 //type=ClusterIP
          - ClusterIP：可系统指定
        - 3.2 开放节点端口，集群外部可访问 //type=NodePort
          - NodePort: 可系统分配
        - 3.3 通过负载均衡器，集群外部可访问 //type=LoadBalancer
          - 负载均衡器异步创建
      - 访问IP //提示：由系统指派，需要调整到下一步yaml文件中编辑
      - 设置端口 //可多个
        - 端口名称
        - 端口号：
        - 协议：

        - 目标端口：“请输入填写端口名称或端口号”
            - 输入框下列出2.1选择的pods对应的的[端口名称]/[协议]/[端口号]供参考
                - 如： https/TCP/9376
            - 如果2选择的是2.2 、2.3就不用列出参考了
    - @Step4 确认服务配置，创建服务
        - 以yaml形式确认，2.2包括Service和Endpoints的定义
    - @Step5 如果服务类型是LoadBalancer，创建服务成功，进入服务详情页后：弹出提示 “通过负载均衡器访问的服务需要配置Ingress访问规则。”
        - 用户可点按钮 #Ingress服务规则配置 进入相应页面
- .@ListItem
    - .服务名称
    - .服务来源　//.spec.服务来源.类型
    - .暴露方式　//.spec.服务访问.暴露类型
    - .检测状态　//.status.状态
- .@detail
    - 基本信息
        - .服务名称
        - .服务ID
        - .服务标签
        - .所属项目
        - .创建时间
    - 服务来源
        - .来源：
            - 集群内应用，通过标签选择
            - 集群外，通过<IP:端口>映射服务
            - 集群外，通过别名映射服务
        - .目标
            - .应用标签
            - .IP和端口
                - .IP
                - .端口
            - .别名
                - .别名
                - .命名空间
    - 当前状态　//动态变化，界面上可增加刷新按钮
        - .检测状态：
            - 成功
            - 失败
            - 未知
        - 服务访问：
            - //根据上面的暴露类型显示不同内容
            - 通过集群IP，仅集群内部访问
                - 集群内：
                    - <集群IP> : <端口号>
                    - <集群IP> : <端口号2> //多个端口就组合多个
            - 开放节点端口，集群外部可访问
                - 集群内：
                    - <集群IP> : <端口号>
                    - <集群IP> : <端口号2> //多个端口就组合多个
                - 集群外访问到节点：
                    - <节点IP1> : <节点端口>
                    - <节点IP2> : <节点端口>
                    - <节点IP3> : <节点端口>

- 通过负载均衡器，集群外部可访问
  - 集群内：
    - <集群IP> : <端口号>
    - <集群IP> : <端口号2> //多个端口就组合多个
  - 集群外访问到节点：
    - <节点IP1> : <节点端口>
    - <节点IP2> : <节点端口>
    - <节点IP3> : <节点端口>
  - 集群外通过负载均衡器访问：
    - <负载均衡器IP> : <端口号>
    - <Ingress IP> : <端口号>
- 服务后端 //服务真正到达的IP和端口，类似如下，(原来叫“路由表”不对)
  - //对应 k8s的 Endpoints
  - {
    - Addresses: [{"ip": "10.10.1.1"}, {"ip": "10.10.2.2"}],
    - Ports: [{"name": "a", "port": 8675}, {"name": "b", "port": 309}]
  - },
  - {
    - Addresses: [{"ip": "10.10.3.3"}],
    - Ports: [{"name": "a", "port": 93}, {"name": "b", "port": 76}]
  - },
  -]
- 服务定义 //转成yaml文件显示
- #更新服务
- #终止服务
- #删除服务

- @C服务入口 : @CObject： Inherited //对应 @K8sIngress
  - // Ingress Controller 负责实现路由需求， Ingress负责描述路由需求
  - .meta.入口名称
  - .spec.访问规则(s)
    - .host
    - .path
    - .service
      - .serviceName
      - .servicePort
  - .spec.默认后端服务 : @C服务
  - .spec.控制器： @C服务 //内含Nginx服务
  - .status.状态 //控制器的服务检测状态
    - 成功
    - 失败
    - 未知
  - .status.IP //作为负载均衡器的IP
  - #创建服务入口 //整个会在项目初始化的时候由系统自动完成
    - #Flow
      - 1 创建入口 //目前都做我们提供好的
      - 2
        - 2.1 生成两个应用：入口控制器 和 默认后端
        - 2.2 把上面两个应用进行部署，默认都部署2个实例
        - 2.3 给以上两个部署创建Service，控制器的服务暴露类型为NodePort或者LoadBalance类型
          - 前期没有用云服务商的负载均衡器，先用开放节点的方式
  - #配置访问规则
  - .show.@listItem
    - .名称

        - .访问规则(s)
            - host:
                - path: {serviceName : servicePort}
    - .show.@detail
        - 基本信息
            - .入口名称
            - .入口IP
        - 访问规则　//按host分组显示
            - .host //如: foo.bar.com
                - /.path　=>　<.serviceName>:<.servicePort>
                - /.path　=>　<.serviceName>:<.servicePort>
            - .host2
                - /path1　=>　<serviceName1>:<servicePort1>
                - /path2　=>　<serviceName2>:<servicePort2>
        - 默认后端
            - .服务：<serviceName>:<servicePort>　//链接到服务详情 @C服务
            - .子应用：Ingress-default-backend //链接到对应子应用
        - 控制器
            - .服务：<serviceName>:<servicePort>　//链接到服务详情
                - //如果没有创建service，就不用显示，nginx可以直接通过容器映射到宿主机，然后提供对外服务
                - //有创建服务，为NodePort类型
            - .子应用：Nginx-Ingress-controller //链接到对应子应用
- @C集群 : @CObject： Inherited
    - .spec.类型
        - 高可用集群
        - 负载均衡集群
        - 高计算能力集群
    - .spec.节点[]：@C节点
    - .status.状态
        - 正常
        - 异常
    - #创建
    - #删除集群 //有节点不能删除
    - #删除节点 //可以删除集群中的某几个节点
    - .show.@listItem
        - .集群名称 = .meta.名称
        - .集群类型 = .spec.类型
        - .节点列表
            - .列表项 = .spec.节点[].节点名称　　//可链接到节点详情页
    - .show.@Detail
        - 基本信息
            - .集群名称 = .meta.名称
            - .类型 = .spec.类型
            - .图标 =　.meta.图标
            - .状态 =　.status.状态
            - .更新时间 = .status.更新时间
        - 节点列表
            - .列表项 = @C节点.show.@ListItem
        - 资源监控
- @C节点 : @CObject： Inherited
    - .meta.所属集群　//parent
    - .spec.主机：@C主机
    - .status.地址信息

        - .主机名称　// HostName
        - .对外IP　// ExternalIP，可以被集群外部路由到的IP。
        - .对内IP　// InternalIP，只能在集群内进行路由的节点的IP地址。
    - .status.状态
        - 节点健康
        - 内存过低
        - 磁盘容量低
        - 未知　//40秒内没有收到检测到节点报告
    - .status.资源容量
        - .CPU　//个数
        - .内存
        - .最大应用实例数量
    - .status.资源可用
        - .CPU　//个数
        - .内存
        - .最大应用实例数量
    - .status.系统信息
        - .操作系统镜像
        - .Docker版本
        - .Kubernetes版本
    - .show.@ListItem
        - 节点名称
        - 运行时间
        - 创建时间
    - .show.@Detail
        - 基本信息
        - 节点标签
        - 节点状态
            - 地址信息
            - 状态信息
            - 节点容量
            - 可用配置
            - 系统信息
- @C主机 : @CObject： Inherited
    - .meta.名称
    - .meta.所属集群
    - .meta.服务商： @服务商
    - .spec.所在服务商平台的实例ID
    - .meta.区域
    - .meta.标签
    - .spec.IP地址
    - .spec.资源情况
        - 平均负载
            - 1分钟
            - 5分钟
            - 15分钟
        - CPU使用率
        - 内存使用率
        - 磁盘利用率(%Util)
        - 磁盘占用率
        - 入带宽
        - 出带宽
    - .spec.配置信息

- .规格
  - .CPU核数
  - .内存容量
- .操作系统
  - //如：CentOS 7.4 64位
- .存储
  - .类型
  - .容量
- .网络
  - .类型
  - .付费方式
  - .带宽限制
- .安全
  - 安全产品名称
- .其它镜像　　//和主机同时购买并安装在主机上
- .spec.费用信息
  - .spec.@服务商
  - .spec.付费方式 = 预付费　//根据 @C服务商 的定价选择结果，= 后面为示例
  - .spec.计费周期 = 按月
  - .status
    - .状态 = 正常使用 / 即将到期 / 已停止使用
    - .到期时间　　//预付费模式
    - .创建时间　　//从云平台上创建的时间，区分于加入我们体系的"创建时间"
  - .addition
    - .消费记录
      - .消费时间
      - .消费类型 = 购买 / 续费 / 后付费
      - .资源ID
      - .资源名称
      - .时长
      - .应付金额
      - .是否支付
      - .支付方式 =
        - 在线支付
        - 账号余额
        - 代金券或优惠券
        - ...
- .status.状态
- .status.当前系统时间
- .status.系统连续运行时长
- .status.当前登录用户数
- #关机 / #开机 / #重启
- #查杀进程
- #远程访问 //严格控制权限
- #查看文件 / #修改文件　　//linux系统一切皆文件
- .addition
- .show.@ListItem
  - .主机名称
  - .主机IP
  - .服务商
  - .状态
  - .资源监控
    - CPU

        - 内存
        - 磁盘
        - 网络
- .show.@Detail
    - 基本信息
        - .名称 = .meta.名称
        - .所属集群 = .meta.parent
        - .标签 = .meta.标签
        - .服务商 = .spec.@服务商.meta.名称
        - .区域 = .spec.区域
        - .IP地址 = .spec.IP地址
        - .状态 = .status.状态
        - .添加时间 = .meta.创建时间
    - 资源监控
        - 总览
            - 当前系统时间
            - 系统连续运行时长
            - 当前登录用户数量
            - 平均负载
                - 1分钟：
                - 5分钟：
                - 15分钟：
                - 说明：
                    - 平均负载数据是每隔5秒钟检查一次活跃的进程数，然后按特定算法计算出的数值。如果这个数除以逻辑CPU的数量，结果高于5的时候就表明系统在超负荷运转了。
                    - 平均负载的值越小代表系统压力越小，越大则代表系统压力越大。通常，我们会以最后一个数值，也就是15分钟内的平均负载作为参考来评估系统的负载情况。
                    - 对于只有单核cpu的系统，1.0是该系统所能承受负荷的边界值，大于1.0则有处理需要等待。
                    - 一个单核cpu的系统，平均负载的合适值是0.7以下。如果负载长期徘徊在1.0，则需要考虑马上处理了。超过1.0的负载，可能会带来非常严重的后果。
            - 几个重要指标 //对比临界值
            - CPU使用率：当前/容量
            - 内存使用率：
            - 磁盘利用率(%Util)：
            - 磁盘占用率
            - 入/出带宽 ：
                - 入/Max
                - 出/Max
        - 计算
            - CPU使用情况 //多个CPU就多组。下面为字段及其值示例
                - 运行用户进程(un-niced)的CPU占比(user)： 5.0 // 用之前的那个cpu图
                    - //- 显示实时(1小时内至少6个点)、24小时、1周
                        - //正常情况下只要服务器不是很闲，那么大部分的 CPU 时间应该都在此执行这类程序
                        - //排查：当 user 占用率过高的时候，通常是某些个别的进程占用了大量的 CPU，这时候很容易通过 top 找到该程序；此时如果怀疑程序异常，可以通过 perf 等思路找出热点调用函数来进一步排查；
                - 运行内核进程的CPU占比(system)： 1.7
                    - //通常情况下该值会比较小，但是当服务器执行的 IO 比较密集的时候，该值会比较大
                    - //排查：当 system 占用率过高的时候，如果 IO 操作(包括终端 IO)比较多，可能会造成这部分的 CPU 占用率高，比如在 file server、database server 等类型的服务器上，否则(比如>20%)很可能有些部分的内核、驱动模块有问题；
                - 运行niced用户进程的CPU占比(nice)：0.0 //改变过优先级的进程
                    - //CPU 在高 nice 值(低优先级)用户态以低优先级运行占用的时间(nice>0)。默认新启动的进程 nice=0，是不会计入这里的，除非手动通过 renice 或者 setpriority() 的方式修改程序的nice值
                    - //排查：当 nice 占用率过高的时候，通常是有意行为，当进程的发起者知道某些进程占用较高的 CPU，会设置其 nice 值确保不会淹没其他进程对 CPU 的使用请求；

- 内核空闲的CPU占比(idle)：93.0
- 等待IO完成的CPU占比(iowait)：0.0
  - //排查：当 iowait 占用率过高的时候，通常意味着某些程序的 IO 操作效率很低，或者 IO 对应设备的性能很低以至于读写操作需要很长的时间来完成；
- 处理硬中断的CPU占比(Hardware Interrupts)：0.3
- 处理软中断的CPU占比(Software Interrupts)：0.0
- 虚拟机CPU占比(steal): 0.0 // 在虚拟机情况下才有意义
- //说明
  - Linux系统中，CPU被两类程序占用：一类是进程（或线程），也称进程上下文；另一类是各种中断、异常的处理程序，也称中断上下文。
  - 进程的存在，是用来处理事务的，如读取用户输入并显示在屏幕上。而事务总有处理完的时候，如用户不再输入，也没有新的内容需要在屏幕上显示。此时这个进程就可以让出CPU，但会随时准备回来（如用户突然有按键动作）。同理，如果系统没有中断、异常事件，CPU就不会花时间在中断上下文。

- 内存使用情况
  - 使用中的内存：7775876k（7.7GB）// used
    - //- 显示实时(1小时内至少6个点)、24小时、1周，=》用之前的那个图
  - 物理内存总量：8306544k（8GB）// total (=used + free)
  - 空闲内存总量：530668k （530M）//free
  - 缓存的内存量：79236k （79M）//buffers ; yun: buff/cache
  - //说明
    - 使用中的内存总量（used）指的是现在系统内核控制的内存数，
    - 空闲内存总量（free）是内核还未纳入其管控范围的数量。纳入内核管理的内存不见得都在使用中，还包括过去使用过的现在可以被重复利用的内存，内核并不把这些可被重新使用的内存交还到free中去，因此在linux上free内存会越来越少，但不用为此担心
    - 可用内存数，近似的计算公式：free + buffers + cached
- 进程情况
  - 正在运行：2 //running //图表，服务端可用模拟数据
  - 总进程数：322 //total
  - 休眠进程数：320 //sleeping
  - 停止进程数：0 //stoped
  - 僵尸进程数：0 //zombie
  - //说明
    - 其它的只要显示当前状态的数值，可以用饼状，总进程数=后面几个之和
- swap交换分区情况
  - 使用的交换区总量：2556k （2.5M） //used
    - //- 显示实时(1小时内至少6个点)、24小时、1周，图表，服务端可用模拟数据
  - 交换区总量：2031608k（2GB）// total (=used + free)
  - 空闲交换区总量：2029052k （2GB） //free
  - 缓冲的交换区总量：4231276k （4GB） //cached yun: avail mem
  - //说明：
    - used：如果这个数值在不断的变化，说明内核在不断进行内存和swap的数据交换，这是真正的内存不够用了。

- 网络
  - I/O情况
  - 宽带占有情况
    - 统计
      - 传输总量 //TOTAL
      - 发送数据 //TX
      - 接收数据 //RX
    - 传输列表
      - .列表项 ////参考：https://tower.im/projects/25209cdb945b4b128e222a373a5cea05/uploads/64db19aab4d94248bfcd253cd9822852/?version=1
        - 源IP
        - 目标IP

        - 是否有数据传输，及其传输方向： => | <=
        - 传输速率： //2s、10s、40s时
- 重要协议统计信息 // //netstat -s，在系统连续运行时长内累计的值
    - 重要协议统计信息 //netstat -s，在系统连续运行时长内累计的值
    - //参考：https://tower.im/projects/25209cdb945b4b128e222a373a5cea05/uploads/f017a29572d34d9a95f7b4ee36712455/?version=1
    - Tcp: //下面为参考示例
        - 90 active connections openings
        - 23 passive connection openings
        - 1 failed connection attempts
        - 3 connection resets received
        - 8 connections established
        - 9991 segments received
        - 6302 segments send out
        - 16 segments retransmitted
        - 0 bad segments received.
        - 98 resets sent
    - Udp:
        - 189 packets received
        - 1 packets to unknown port received.
        - 0 packet receive errors
        - 189 packets sent
    - Ip:
        - 10194 total packets received
        - 2 with invalid addresses
        - 0 forwarded
        - 0 incoming packets discarded
        - 10192 incoming packets delivered
        - 6489 requests sent out
    - Icmp:
        - 11 ICMP messages received
        - 2 input ICMP message failed.
        - ICMP input histogram:
            - destination unreachable: 2
            - echo requests: 5
            - echo replies: 4
        - 10 ICMP messages sent
        - 0 ICMP messages failed
        - ICMP output histogram:
            - destination unreachable: 1
            - echo request: 4
            - echo replies: 5
    - --其它的暂时不显示，以后再说---
        - IcmpMsg:
            - InType0: 4
                - InType3: 2
                - InType8: 5
                - OutType0: 5
                - OutType3: 1
                - OutType8: 4
        - UdpLite:
        - TcpExt:

- 1 resets received for embryonic SYN_RECV sockets
- 12 TCP sockets finished time wait in fast timer
- 94 delayed acks sent
- Quick ack mode was activated 8 times
- 22 packets directly queued to recvmsg prequeue.
- 22 bytes directly received in process context from prequeue
- 8055 packet headers predicted
- 489 acknowledgments not containing data payload received
- 184 predicted acknowledgments
- 1 times recovered from packet loss by selective acknowledgements
- 3 congestion windows recovered without slow start after partial ack
- 1 fast retransmits
- 4 other TCP timeouts
- TCPLossProbes: 13
- 8 DSACKs sent for old packets
- 5 DSACKs sent for out of order packets
- 10 DSACKs received
- 20 connections reset due to unexpected data
- 3 connections reset due to early user close
- TCPDSACKIgnoredNoUndo: 4
- TCPSackShiftFallback: 10
- TCPRcvCoalesce: 1355
- TCPOFOQueue: 180
- TCPOFOMerge: 6
- TCPAutoCorking: 112
- TCPSynRetrans: 2
- TCPOrigDataSent: 973
- TCPKeepAlive: 2

- IpExt:
  - InOctets: 84617275
  - OutOctets: 537187
  - InNoECTPkts: 60101

- //http tcp udp ip 间的关系
  - ip 对应于网络层
    - tcp 和 udp 对应于传输层
  - http 对应于应用层
    - socket 属于api ，是对tcp/ip的封装。
  - 其中，应用层存在的意义是使tcp / ip传输过来的数据内容能够识别出来。
  - 通过socket 我们才能使用tcp/ip协议
  - CSDN上有个比较形象的描述：HTTP是轿车，提供了封装或者显示数据的具体形式;Socket是发动机，提供了网络通信的能力。

- 存储
  - 磁盘占用率 周期内监控
  - 磁盘利用率 周期内监控
  - 磁盘空间信息 ////df 列表
    - .列表项
      - 文件系统 //Filesystem
      - 容量
      - 已用 //Used
      - 可用 //Available
      - 已用% //Use%
      - 挂载点 //Mounted on
      - //示例

- 文件系统 容量 已用 可用 已用% 挂载点
- /dev/sda7 19G 871M 18G 5% /
- /dev/sda9 195G 89G 96G 49% /opt
- /dev/sda8 4.8G 557M 4.0G 13% /var
- /dev/sda6 19G 1.9G 17G 11% /usr
- /dev/sda3 965M 24M 892M 3% /boot
- tmpfs 16G 0 16G 0% /dev/shm
- total 21G 4.2G 16G 22% -
- //参考图：https://tower.im/projects/25209cdb945b4b128e222a373a5cea05/uploads/16ab0b750621494b82ce1ab78abb1876/?version=1

- 配置信息
  - .规格
    - .CPU核数
    - .内存容量
  - .操作系统
    - //如：CentOS 7.4 64位
  - .存储
    - .类型
    - .容量
  - .网络
    - .类型
    - .付费方式
    - .带宽限制
  - .安全
    - //安全产品名称罗列
  - .其它镜像 //和主机同时购买并安装在主机上
- 费用信息
  - .状态 = .spec.费用.status.状态
  - .付费方式= [.spec.费用.计费周期][.spec.费用.付费方式] //如：按月预付费
  - .到期时间 = .spec.费用.status.到期时间
  - #详情

- .show.@Create //添加单台机器
  - .@Step1：输入基本信息
    - .主机名称
    - .服务商 // 从已有中选择：
      - 阿里云
      - 腾讯云
      - 亚马逊AWS
      - 微软Azure
      - ...
      - 其它 //选该项需要用户输入具体的服务商
    - .IP地址
    - .用户名 //有root权限的用户
    - .密码
  - .@Step2： 开始添加
    - 提示进度：“添加主机中...，请稍后！”
    - 显示添加过程 //类似终端打印显示，保存过程日志
    - 添加成功：
      - 界面提示：“主机添加成功！”
      - 显示按钮 #查看主机 #继续添加
      - 推送消息：
        - 类型 = 添加主机 https://workflowy.com/s/COUx.Z76WJdJmJg

        - 添加失败：
            - 界面提示：“主机添加失败！”
            - 显示按钮： #重新添加
            - 推送消息：
                - 类型 = 添加主机 https://workflowy.com/s/COUx.Z76WJdJmJg
    - .show.@批量导入机器
        - .@Step1 : 选择云服务商，添加访问密钥 //各云密钥获取过程请参考帮助文档
            - 阿里云：
                - Access Key ID：
                - Access Key Secret：
            - 腾讯云
                - API密钥 SecretId：
                - API密钥 SecretKey：
            - 亚马逊AWS
                - Access Key ID：
                - Access Key Secret：
            - 微软Azure
                - 订阅（Subscription）ID：
                - 租户（Tenant）ID：
                - Client ID：
                - Client Secret：
        - .@Step2 : 选择云主机，导入
            - 列出对应账户下的主机
            - 选择需要导入的主机
            - 确认，开始导入到平台
- 安全
    - 要务
        - 现在安全吗？问题在哪？我该怎么办?
    - 做法
        - 接入第三方安全产品和企业，包括态势感知、防护策略、应急方案
            - 监控来自外部的攻击
            - 设置好防护策略，主动防御
            - 定期做安全检测，报告漏洞，及时修复 //定期报告结果；可设置检测项和周期；消息和用户短信可以接收到安全告警
        - 异常情况，告警通知：
            - 攻击/漏洞情况
            - 防御/修复情况
            - 如需应急，应急方案
    - 防护策略
        - //每个策略
            - 策略名称及标签 (针对防御层)
            - 策略说明
            - 配置信息 #设置
            - 使用情况 #详情
            - 防御效果：很棒、 好 、一般、 效果不佳
        - ==示例
            - 1 阿里云SCDN #网络安全
                - 策略说明：
                    - 防DDos攻击，防CC攻击，安全加速
                - 配置规格 #修改配置
                    - 业务带宽：如100Mbps
                    - DDos基础防护带宽： 如20Gbps
                    - CC防护：如60,000QPS
                    - 弹性防护：20Gbps

        - 近日使用 #详情
            - 时间周期：今天 昨天 近7天 近30天 自定义
            - 业务带宽：如15.8Mbps
            - 业务流量：如1.15GB
            - DDos攻击带宽：如下
                - 电信：13.00 kbps
                - 联通：25.00 kbps
        - 防御效果
    - 2 阿里云DDos高防IP #网络安全
    - 3 腾讯云镜 #主机安全
    - 4 AWS Web应用程序防火墙
- @CSecurityAttack //攻击，外部行为，一个对象为一种攻击
    - .攻击名称
    - .攻击表现 //可以有简单的攻击路径等文本描述
    - .攻击事件[] //一次事件，表示攻击一次
        - .发生时间
        - .攻击对象[] //攻击网络、某主机、某应用或容器，列出
        - .攻击强度
            - 弱
            - 一般
            - 强
            - 很强
    - .危害程度
        - 没影响
        - 提醒
        - 可疑
        - 紧急
    - .数据采集[] //根据攻击模型库，去采集、监控相关信息
        -
        - //比如DDOS攻击攻击，会采集下面的数据：
        - 流量情况
            - 攻击带宽
            - 总带宽
        - 攻击子类型
        - 被攻击IP TOP5
        - 被攻击URL TOP5
        - 攻击浏览器 TOP5
- @CSecurityBug //漏洞，内部自检行为，一个对象为一种漏洞
    - .漏洞名称
    - .漏洞表现
    - .影响对象[] //漏洞发生在哪里，会影响哪些对象
    - .紧急程度
        - 可不修复
        - 可延后修复
        - 需尽快修复
        - 需紧急修复
    - .检查项[] //都检测了哪些信息，得到出现该漏洞的结论
- @CSecurityDetector //安全检测
    - .检测类别 //日志、基线配置等
    - .检测项[]
        - .检测内容
        - .检测结果
- @CSecurityDefend //安全防护

- .策略名称
  - //如阿里云SCDN、阿里云DDos高防IP、腾讯云镜、AWS Web应用程序防火墙
- .策略标签
  - 网络层
  - 主机层
  - 应用层
  - 容器层
- .策略说明 //如，阿里云SCDN：防DDos攻击，防CC攻击，安全加速
- .配置信息 #设置 //不同的策略，配置属性不一样
  - //如：
    - 业务带宽：100Mbps
    - DDos基础防护带宽：20Gbps
    - CC防护：60000QPS
    - 弹性防护：20Gbps
- .使用情况 #详情 //不同的策略不一样
  - //如：
    - 今天
      - 业务带宽：15.8Mbps
      - 业务流量：1.15GB
      - DDos攻击带宽：
        - 电信：13.00kbs
        - 联通：25.00kbs
- .防御效果
  - 很好
  - 好
  - 一般
  - 不佳

- 监控
  - #流程
    - 监控对象的定义和注册
      - 服务监控 //如业务逻辑、CDN、数据库、负载均衡、集群调度服务等
        - 健康状态
        - 服务性能 //APM
        - 影响因素
      - 资源监控
        - 资源状态
        - 资源使用情况
        - 影响因素
    - 数据采集
      - 按监控对象、监控项、采集周期，定期采集数据，并计算和上报属性值
    - 告警通知
      - 计算并上报的属性值如果触发告警策略，则根据预先设置好的告警对象以及沉默时间，进行告警。
      - 告警方式：站内信、短信
      - 告警通知中会包含修复建议
  - @C告警等级 //值为1、2、3触发告警
    - 0 //正常
    - 1 //提醒(有异常，但不严重或不紧急)
    - 2 //严重(不紧急)
    - 3 //严重紧急
  - @C告警通知 //监控值触发策略则进行告警
    - .监控对象[] //@CMonitor 给不同的监控分组，然后通知到对应的组成员
    - 1级:
      - 通知对象：[账号名称1],[账号名称2],[账号名称3]... //用户权限

        - 沉默时间：[间隔时间]
    - 2级:
        - 通知对象： [账号名称x1],[账号名称x2],[账号名称x3]...
        - 沉默时间：[间隔时间]
    - 3级：
        - 通知对象： [账号名称x1],[账号名称x2],[账号名称x3]...
        - 沉默时间：[间隔时间]
    - //梯度告警，触发策略先通知第一级，如果在间隔时间内，还是异常，则会再次发起通知。如果达到下一级的通道沉默时间，通知下一级
- @CMonitor
    - .对象 //1:服务/集群/日志/事件；2: 实例/部署/节点(主机)
    - .监控项[]
        - 【监控属性 】:
            - 采集周期
            - 策略[]
                - 【属性值】：
                    - 等级
                    - 通知文本
    - .影响因素[@CMonitor]
        - //同类型的合并显示
- 服务
    - @集群调度服务 ：@CMonitor
        - .对象 = @kube-scheduler //k8s集群原生，可以是其它的
        - .监控项 =
            - "状态":
                - 采集周期=30,
                - 策略：
                    - "正常" :
                        - 等级：0
                        - 异常通知文本：无
                    - "异常" :
                        - 等级：2
                        - 通知文本：“[集群名称]的调度服务[服务名称]状态异常，原因：[状态码说明]，请及时处理！”
                    - "未知" :
                        - 等级：3
                        - 通知文本：“[集群名称]的调度服务[服务名称]状态异常，需要人工干预，请及时处理！”
    - @集群控制器管理服务 ：@CMonitor
        - .对象 = @kube-controller-manager //如k8s集群的管理服务
        - .监控项 =
            - "状态":
                - 采集周期=30
                - 策略：
                    - "正常" :
                        - 等级：0
                        - 异常通知文本：无
                    - "异常" :
                        - 等级：2
                        - 通知文本：“[集群名称]的控制器管理服务[服务名称]状态异常，原因：[状态码说明]，请及时处理！”
                    - "未知" :
                        - 等级：3
                        - 通知文本：“[集群名称]的控制器管理服务[服务名称]状态异常，需要人工干预，请及时处理！”
    - @集群接入服务 ：@CMonitor //APIServer
        - .对象 = @kube-controller-manager //如k8s集群的管理服务

- .监控项 =
  - "状态":
    - 采集周期=30
    - 策略:
      - "正常" :
        - 等级：0
        - 异常通知文本：无
      - "异常" :
        - 等级：2
        - 通知文本：“[集群名称]的控制器管理服务[服务名称]状态异常，原因：[状态码说明]，请及时处理！”
      - "未知" :
        - 等级：3
        - 通知文本：“[集群名称]的控制器管理服务[服务名称]状态异常，需要人工干预，请及时处理！”
- @CX服务 : @CMonitor //某个服务
  - .对象 = 具体的服务
  - .监控项 =
    - "状态":
      - 采集周期=30,
      - 策略:
        - "正常" :
          - 等级：0
          - 异常通知文本：无
        - "异常" :
          - 等级：2
          - 通知文本：“服务[服务名称]状态异常，原因：[状态码说明]，请及时处理！”
        - "未知" :
          - 等级：3
          - 通知文本：“服务[服务名称]状态异常，需要人工干预，请及时处理！”
    - 【性能指标】//不同类型的服务指标不同，在创建服务时进行分类。我们对不同类型及其性能指标进行枚举
  - 影响对象 =
    - 应用实例 // @CMonitor pod
      - 影响对象 = 容器监控
- 集群
  - @集群监控 : @CMonitor
    - .对象 = 某集群
    - .监控项 =
      - "状态":
        - 采集周期=30,
        - 策略:
          - "正常" :
            - 等级：0
            - 通知文本：无
          - "集群网络故障" :
            - 等级：2
            - 通知文本：“[集群名称]网络故障，一段时间后如果未恢复，请及时跟踪处理！”
          - "可用节点数量 < 集群节点数的一半" :
            - 等级：3
            - 通知文本：“[集群名称]可用节点太少。可用节点数量：[数量],集群节点总数：[数量]，请及时处理！”
      - "CPU使用率":
        - 采集周期=30,
        - 策略:
          - x<10%:
            - 等级：1

- 通知文本：“[集群名称]CPU使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
- 10%<x<70%:
  - 等级：0
  - 通知文本：无
- 90%>x>=70% //70是临界值
  - 等级：1
  - 通知文本：“[集群名称]CPU使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
- 100%>x>=90%
  - 等级：1
  - 通知文本：“[集群名称]CPU使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”

- "内存使用率":
  - 采集周期=30,
  - 策略:
    - x<10%:
      - 等级：1
      - 通知文本：“[集群名称]内存使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
    - 10%<x<80%: //80是临界值
      - 等级：0
      - 通知文本：无
    - 90%>x>=80%
      - 等级：1
      - 通知文本：“[集群名称]内存使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
    - 100%>x>=90%
      - 等级：1
      - 通知文本：“[集群名称]内存使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”

- 影响因素 =
  - 节点[] // @CMonitor ,有多少节点列出

- @节点监控 ：@CMonitor
  - .对象
  - .监控项 =
    - "状态":
      - 采集周期=30,
      - 策略:
        - "正常" :
          - 等级：0
          - 通知文本：无
        - "未知":
          - 等级：3
          - 通知文本：“[集群名称]的[节点名称]无法获取节点状况，一段时间后，如果集群未自修复，需要人工干预！”
        - "网络故障" :
          - 等级：2
          - 通知文本：“[集群名称]的节点[节点名称]网络故障，一段时间后如果未恢复，请及时跟踪处理！”
        - "内存过低":
          - 等级：2
          - 通知文本：“[集群名称]-[节点名称]内存过低，一段时间后，如果集群未自修复，请主动跟踪处理！”
        - "磁盘容量低":
          - 等级：2
          - 通知文本：“[集群名称]-[节点名称]磁盘容量低，一段时间后，如果集群未自修复，请主动跟踪处理！”
    - "CPU使用率":
      - 采集周期=30,
      - 策略:
        - x<10%:
          - 等级：1

            - 通知文本：“[集群名称]的节点[节点名称]CPU使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
        - 10%<x<70%:
            - 等级：0
            - 通知文本：无
        - 90%>x>=70% //70是临界值
            - 等级：1
            - 通知文本：“[集群名称]的节点[节点名称]CPU使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
        - 100%>x>=90%
            - 等级：1
            - 通知文本：“[集群名称]的节点[节点名称]CPU使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”
    - "内存使用率":
        - 采集周期=30,
        - 策略:
            - x<10%:
                - 等级：1
                - 通知文本：“[集群名称]-[节点名称]内存使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
            - 10%<x<80%: //80是临界值
                - 等级：0
                - 通知文本：无
            - 90%>x>=80%
                - 等级：1
                - 通知文本：“[集群名称]-[节点名称]内存使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
            - 100%>x>=90%
                - 等级：1
                - 通知文本：“[集群名称]-[节点名称]内存使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”
    - "应用实例数量":
        - 采集周期=30,
        - 策略:
            - "<=实例数量最大值":
                - 等级：0
                - 通知文本：无
            - "实例数量最大值+2>=x>=实例数量最大值"
                - 等级：2
                - 通知文本：“[集群名称]-[节点名称]当前应用实例数量：[数量]，最大应用实例数量：[数量]，一段时间后，如果集群未自修复，请主动跟踪处理！”
            - ">实例数量最大值+2"
                - 等级：3
                - 通知文本：“[集群名称]-[节点名称]的应用数量过多。当前应用实例数量：[数量]，最大应用实例数量：[数量]，请主动跟踪处理！”
    - "磁盘利用率": //(%Util)
        - 采集周期=30,
        - 策略: //临界值：70%
    - "磁盘占用率":
        - 采集周期=30,
        - 策略: //临界值：80%
    - "网络入带宽": // (当前/配置max，单位kbs/s)
        - 采集周期=30,
        - 策略: //临界值：60%
    - "网络出带宽": // (当前/配置max，单位kbs/s)
        - 采集周期=30,
        - 策略: //临界值：60%
- 影响因素
    - 主机 //node对应的主机可能资源情况和node不一样，可能node只是隔离出的一个虚拟机

- 容器 //跑在这个节点上的容器资源情况，方便核对node资源使用情况

- @主机监控 ：@CMonitor
  - .对象
  - .监控项 =
    - "状态":
      - 采集周期=30,
      - 策略:
        - "正常" :
          - 等级：0
          - 通知文本：无
        - "网络故障" :
          - 等级：2
          - 通知文本：“主机[主机名称]网络故障，一段时间后如果未恢复，请及时跟踪处理！”
        - "已关机":
          - 等级：1
          - 通知文本：“主机[主机名称]当前为关机状态，如果不是主动操作，请及时查看原因！”
        - "重启中":
          - 等级：1
          - 通知文本：“主机[主机名称]在重启中，如果不是主动操作，请及时查看原因！”
        - "费用到期":
          - 等级：3
          - 通知文本：“主机[主机名称]费用到期，需要及时续费，以免影响使用！”
    - "CPU平均负载": //通常取1分钟、5分钟、15分钟数值参考。以15分钟为评估
      - 采集周期=30,
      - 策略:
        - //单核0.7为临界值，长期徘徊在1.0需要处理
    - "CPU使用率":
      - 采集周期=30,
      - 策略:
        - x<10%:
          - 等级：1
          - 通知文本：“[集群名称]的节点[节点名称]CPU使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
        - 10%<x<70%:
          - 等级：0
          - 通知文本：无
        - 90%>x>=70% //70是临界值
          - 等级：1
          - 通知文本：“[集群名称]的节点[节点名称]CPU使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
        - 100%>x>=90%
          - 等级：1
          - 通知文本：“[集群名称]的节点[节点名称]CPU使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”
    - "内存使用率":
      - 采集周期=30,
      - 策略:
        - x<10%:
          - 等级：1
          - 通知文本：“[集群名称]的节点[节点名称]内存使用率低，可以适当观察一段时间，确认是否存在资源浪费情况。”
        - 10%<x<80%: //80是临界值
          - 等级：0
          - 通知文本：无
        - 90%>x>=80%
          - 等级：1
          - 通知文本：“[集群名称]的节点[节点名称]内存使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”

- 100%>x>=90%
  - 等级：1
  - 通知文本：“[集群名称]的节点[节点名称]内存使用率过高，可以适当观察一段时间，确认是否存在需要增加资源。”
- "磁盘利用率": //(%Util)
  - 采集周期=30,
  - 策略: //临界值：70%
- "磁盘占用率":
  - 采集周期=30,
  - 策略: //临界值：80%
- "网络入带宽": // (当前/配置max，单位kbs/s)
  - 采集周期=30,
  - 策略: //临界值：60%
- "网络出带宽": // (当前/配置max，单位kbs/s)
  - 采集周期=30,
  - 策略: //临界值：60%

- @容器监控 ：@CMonitor
  - .对象
  - .监控项 =
    - "状态":
      - 采集周期=30,
      - 策略:
        - "正常运行":
          - 等级：0
          - 通知文本：无
        - "网络故障":
          - 等级：2
          - 通知文本：“[集群名称]-[节点名称]-[容器名称]网络故障，一段时间后如果未恢复，请及时跟踪处理！”
    - "CPU使用率":
      - 采集周期=30,
      - 策略:
        - x<70%:
          - 等级：0
          - 通知文本：无
        - 90%>x>=70% //70是临界值
          - 等级：1
          - 通知文本：“[集群名称]-[节点名称]-[容器名称]CPU使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”
        - x>=90%
          - 等级：1
          - 通知文本：“[集群名称]-[节点名称]-[容器名称]CPU使用率过高。”
    - "内存使用率":
      - 采集周期=30,
      - 策略:
        - x<80%: //80是临界值
          - 等级：0
          - 通知文本：无
        - x>=80%
          - 等级：1
          - 通知文本：“[集群名称]-[节点名称]-[容器名称]内存使用率高，可以适当观察一段时间，确认是否存在资源超负载情况。”

- 用户
  - @C用户 ： @CObject： Inherited
    - @C账户

- @身份 = 个人 / 企业员工
- @C角色
- @C权限
- @C服务商 ： @CObject： Inherited //先把相关第三方服务的内容都整理到这里
  - .spec.提供的产品类型 =
    - 计算
    - 存储
    - 数据库
    - CDN
    - 安全
    - ...
  - .spec.产品定价[] // 不同类型产品，不同定价
    - .产品类型 //如 云服务器
    - .付费方式 = 预付费 | 后付费
    - .计费周期 = 10分钟 | 1小时 | 月| 年 ... //根据服务商和产品类型
      - //前两种这种一般为后付费选择，后两种为预付费选择
      - 有些服务商有用
    - .单价
      - 元/年
      - 元/月
      - 元/时
      - 元/分
      - 元/秒 //AWS,按每秒计费
      - ... //其它按量的单位，如网络...
  - .spec.@C账户 //我们在服务商的账户，整理进来，方便管理
    - .spec.账号
    - .spec.授权[] //AccessKey\token等登录、访问等授权
      - 如阿里云的访问密钥：
        - AccessKeyId用于标识用户。
        - AccessKeySecret是用来验证用户的密钥。AccessKeySecret必须保密。
    - .spec.余额 //通常现在所在服务平台都会设置
  - .addition
    - .充值记录
      - .充值时间
      - .充值额度
      - .充值后余额： //根据原来的账户余额进行计算
- 支撑对象
  - @C消息： @VCObject： Inherited
    - .发送时间
    - .接收对象类型 = 个人 / 企业
    - .接收对象[] //某个具体的用户，或者某一群体
      - 个人
        - .类型 = 个人
        - .用户ID =
      - 企业
        - .类型 = 企业
        - .企业ID =
        - .用户ID[] = //发送消息给企业了里符合条件的员工
    - .类型
    - .标题 //可以用标题就把内容显示完整，内容可带链接，可跳转
    - .详情 //内容为空，在前端呈现上，就不需要有专门的详情页面
    - .状态 = 已读 / 未读
    - .config //模版,系统中各种消息，整理到这里方便查看，以下示例

- 加入企业 - 邀请
  - .类型 = "加入企业"
  - .接收对象类型 = 个人
  - .标题 = "【邀请人】邀请你加入【企业名称】， #邀请链接 "
- 加入企业 - 拒绝
  - .类型 = "加入企业"
  - .接收对象类型 = 个人
  - .标题 = "【审核人】拒绝了你加入【企业名称】的申请，你可以核对信息后重新提交申请， #邀请链接 "
- 加入企业 - 审核通过
  - .类型 = "加入企业"
  - .接收对象类型 = 个人
  - .标题 = "【审核人】审核同意了你加入【企业名称】的申请，你可以进入企业了。 #进入审核 "
- 加入企业 - 有申请
  - .类型 = "加入企业"
  - .接收对象类型 = 企业
  - .标题 = "【姓名】【手机】申请加入【企业名称】，请及时审核。 "
- 企业变更 -
  - .类型 = "企业变更"
  - .接收对象类型 = 个人
  - .标题 = "你的企业被【修改人姓名，没姓名显示手机】修改了名称，修改前【旧企业名称】，修改后【新企业名称】。 #查看企业"
- 企业变更 - #copy
  - .类型 = "企业变更"
  - .接收对象类型 = 个人
  - .标题 = ""
- 企业变更 - #copy
  - .类型 = "企业变更"
  - .接收对象类型 = 个人
  - .标题 = ""
- 企业变更 - #copy
  - .类型 = "企业变更"
  - .接收对象类型 = 个人
  - .标题 = ""
- .show
  - .@ListItem //如果有被用于列表，单项承载信息
    - .发送时间
    - .类型
    - .标题
    - .状态 = 已读 / 未读
    - #查看详情 //如果详情内容为空(简短消息)，没有该操作
  - .@ListFilter //列表搜索、过滤、筛选
    - .搜索 = [.标题, .详情] //输入搜索的key，会从这些字段进行查找
    - .筛选 = [.类型] //可以按这个属性去筛选
  - .@Detail //查看对象详情时，界面希望呈现的内容
    - .属性
    - .发送时间
    - .类型
    - .标题
    - .详情
    - .状态 = 已读 / 未读
- @C日志
- @C事件
- @C任务

- 显示对象
  - @C应用镜像 tmp //相同应用的镜像整理在一起
    - .meta.所属应用 = @C镜像.spec.所属应用 //parent
    - .spec.镜像(s) [@C镜像] //镜像列表，列表里每一个项为 @C镜像
    - .showObject.listItem
      - .图标 = .meta.应用.图标
      - .名称 = .meta.应用.名称
      - .标签 = .meta.应用.标签
      - .描述 = .meta.应用.addition.desc
      - .最新版本 = @C镜像.spec.版本号 //更新时间最近的那个镜像
      - .更新时间 = @C镜像.spec.更新时间 //最新版本的更新时间
    - .showObject.detail
      - .图标 = .meta.应用.图标
      - .名称 = .meta.应用.名称
      - .标签 = .meta.应用.标签
      - .描述 = .meta.应用.addition.desc
      - .@版本列表 // 数据来源：.spec.镜像(s)
        - .showObject.listItem = @C镜像.@showObject.listItem
  - @C镜像仓库 tmp //不同应用的镜像进行管理
    - .spec.应用(s)
    - .showObject
      - .listItem
        - 应用名称 = @C镜像.spec.所属应用.meta.名称
        - 标签 = @C镜像.spec.所属应用.meta.标签
      - .detail