毕业实训进展汇报6

1752919 祁好雨

目录

- 理论进展
- 实践进展

理论进展

- configMap
- scheduler
- managing Resources
- kubernetes Qos
- monitoring Resources
- auto scaling

ConfigMap

- 作用：配置环境变量/configuration
- 替代方法:
  - passing command line arguments to containers
  - custom environment variables for container
  - mount config files as special volumes

替代方法1:passing command line arguments

- Dockerfile：
  - EntryPoint： 脚本/可执行文件/命令 形式： Shell(实际的命令运行在shell的子进程)/Exec
  - CMD： 参数 修改方法： docker run <imageName> arguments
- Kubernetes(yaml):

```
kind: Pod
spec:
  containers:
  - image: some/image
    command: ["/bin/command"]
    args: ["arg1", "arg2", "arg3"]
```

Docker	Kubernetes	Description
ENTRYPOINT	command	The executable that's executed inside the container
CMD	args	The arguments passed to the executable

不便的地方：
每次需要重新编辑和apply yaml/json文件

替代方法2:container 环境变量

```
kind: Pod
spec:
 containers:
 - image: luksa/fortune:env
   env:
   - name: INTERVAL
     value: "30"
   name: html-generator
```

Adding a single variable to the environment variable list

```
env:
- name: FIRST_VAR
  value: "foo"
- name: SECOND_VAR
  value: "$(FIRST_VAR)bar"
```

- args和command也可以引用$(VAR)环境变量

缺点:
- 不同的环境变量只能放在不同pod里
- pod的spec不能重用
- 更改环境变量需要重新apply spec

ConfigMap

- application并不需要知道ConfigMap的存在
- ConfigMap是以环境变量/挂载volume的方式作用在container上
- 不同的ConfigMap可以作用在同一批Pods上，实现不同目的
- 创建方法
  - kubectl create configmap <name> \
    --from-literal=key1=value1 --from-literal=key2=value2
  - kubectl create configmap <name> --from-file=xx.conf

ConfigMap

- 以环境变量的方式应用

```
metadata:
  name: fortune-env-from-configmap
spec:
  containers:
  - image: luksa/fortune:env
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name: fortune-config
          key: sleep-interval
...
```

You're setting the environment variable called INTERVAL.

Instead of setting a fixed value, you're initializing it from a ConfigMap key.

The name of the ConfigMap you're referencing

You're setting the variable to whatever is stored under this key in the ConfigMap.

传递单个变量

```
spec:
  containers:
  - image: some-image
    envFrom:
    - prefix: CONFIG_
      configMapRef:
        name: my-config-map
```

Using envFrom instead of env

All environment variables will be prefixed with CONFIG_.

Referencing the ConfigMap called my-config-map

传递configMap的所有变量

ConfigMap

- 以挂载volume的形式

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune-configmap-volume
spec:
  containers:
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    ...
    - name: config
      mountPath: /etc/nginx/conf.d
      readOnly: true
    ...
  volumes:
  ...
  - name: config
    configMap:
      name: fortune-config
  ...
```

You're mounting the configMap volume at this location.

The volume refers to your fortune-config ConfigMap.

```
volumes:
- name: config
  configMap:
    name: fortune-config
    items:
    - key: my-nginx-config.conf
      path: gzip.conf
```

Selecting which entries to include in the volume by listing them

You want the entry under this key included.

The entry's value should be stored in this file.

Scheduler

- component：
  - kubernetes： control plane+worker nodes
    - control plane： 1. API server 2. scheduler 3. controller manager 4. etcd分布式数据库(RAFT协议)
- function：
  - 分配新建的Pod到Node
  - Reschedule Pod(maximize utilization)
  - 通过API server更新状态(kubelet handle updates)
- 算法：
  - default scheduler lies between ML and simply placement

Scheduler

- default 算法：
  - 过滤acceptable nodes *
  - 打优先级，按照优先级筛选node
  - 同优先级node用round robin
- acceptable nodes

- Can the node fulfill the pod's requests for hardware resources? You'll learn how to specify them in chapter 14.
- Is the node running out of resources (is it reporting a memory or a disk pressure condition)?
- If the pod requests to be scheduled to a specific node (by name), is this the node?
- Does the node have a label that matches the node selector in the pod specification (if one is defined)?
- If the pod requests to be bound to a specific host port (discussed in chapter 13), is that port already taken on this node or not?
- If the pod requests a certain type of volume, can this volume be mounted for this pod on this node, or is another pod on the node already using the same volume?
- Does the pod tolerate the taints of the node? Taints and tolerations are explained in chapter 16.
- Does the pod specify node and/or pod affinity or anti-affinity rules? If yes, would scheduling the pod to this node break those rules? This is also explained in chapter 16.

Scheduler

- 更换算法的方式
  - default scheduler 用specialized configuration
  - 不部署scheduler，deploy an application to watch
  - multiple schedulers，specified chosen scheduler name in pods configuration

Managing resources

- requests
  - CPU
    - 200m(200millicore 20%CPU core)
    - 表示至少需要20%CPU core
    - 如果没有说明CPU request，pod在一个负载很多的node上面甚至可能长期分不到cpu资源
  - MEM
    - 10Mi
    - 表示最多分配10Mi的内存
- limits

Managing resources

- requests影响scheduling
  - scheduler计算acceptable nodes的时候使用pods的request资源计算，而不是当前使用的资源（即使当前使用的远远小于request）
  - scheduler用来计算node priority的函数需要用requests
    - LeastRequestedPriority：更喜欢当前pods需求少，可分配多的node
      - 适用于大多数情况，需要高性能
    - MostRequestedPriority：更喜欢当前pods需求多，可分配少的node
      - 适用于集群部署在云上，需要节省成本，making pods tightly packed

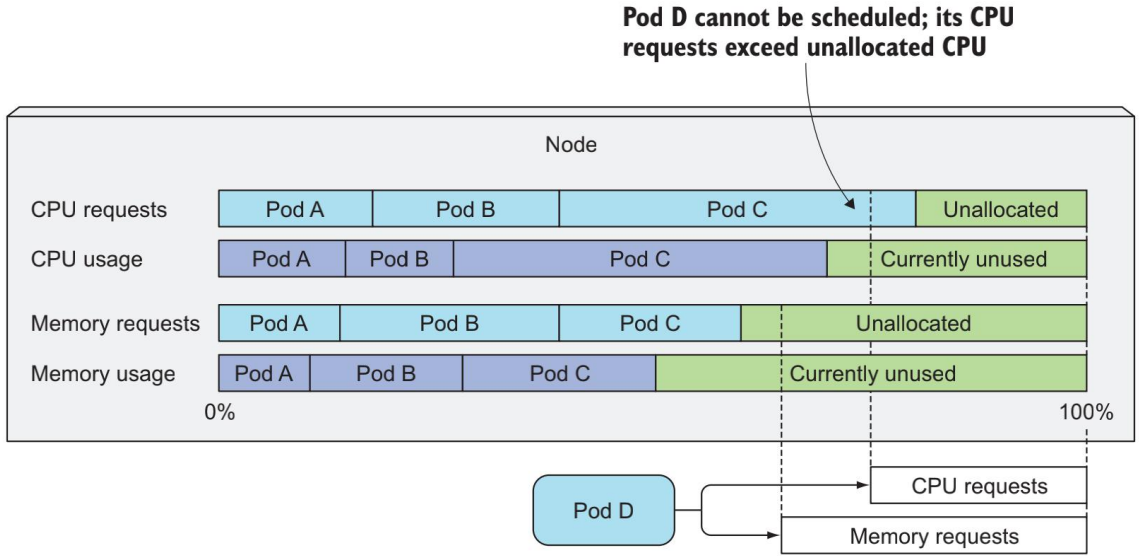


Figure 14.1 The Scheduler only cares about requests, not actual usage.

Managing resources

```
$ kubectl describe nodes
Name:       minikube
...
Capacity:
  cpu:            2
  memory:         2048484Ki
  pods:           110
Allocatable:
  cpu:            2
  memory:         1946084Ki
  pods:           110
...
```

The overall capacity of the node

The resources allocatable to pods

- 查看node的状态
  - describe nodes <nodeName>
  - capacity:包含一些系统占用，并不全分给pods
  - allocatable:scheduler需要的信息，根据已分配requests计算
  - 一般来说node并不能被application”用满”,kube-system会占用一部分

Managing resources

- CPU requests影响cpu time sharing
  - cpu requests不是limits，也就是说当有更多可用的时候，pods会使用更多cpu资源
  - 但cpu requests会按比例使用总共的cpu资源
  - 当然，当其他进程idle时，running的进程可以使用全部的cpu资源

Managing resources

- 用户可以自定义resources
  - 自定义resources需要先添加到node的capacity一项中
  - kubernetes会自动将capacity中的属性复制到allocatable中
  - 自定义resources的量度单位需要是整数

实验思考

- RL-based模型特征选取requests还是实时资源耗费量呢?
- 如果是实时资源耗费量，如何落地根据实时资源耗费量进行调度的scheduler?
- 1.filter acceptable nodes->2. priority function->3.optimize
- 目前第一步遵循requests rule，如果根据实时资源耗费量是否需要取消第一步?

- 特征原始数据：CPU，MEM，HTTP请求(?)
- HTTP请求需要自定义resources，细节操作需要更深入的学习

Managing resources

- limits
  - CPU：可以压缩，分配给container的cpu资源并不是不能收回的
  - MEM：不可压缩，一旦分配只能等待process释放才能回收
  - 为了避免出现container占用过多Memory，可以通过limits规定最多使用资源
  - Node的全部limits总和可以超过100%
  - 实际运行时占用资源若超过100%，会kill container
  - 如果container想使用超过limit的memory，会被kill
  - container并不知道limits的存在，它能“看到”node上所有mem和cpu core，所以如果application需要利用cpu和mem信息来进行分配或者开多个线程/进程，会造成灾难性的结果

Managing resources

- 针对namespace下所有pods：
  - LimitRange：设置Min,Max CPU/Mem和default requests/limits
  - ResourceQuota：设置total limit of available resources

kubernetes Qos

- 使用场景：
  - 如果一个node上面运行了多个pod，podA占用了90%的资源，podB突然需要更多的资源，出现了node资源分配不够的情况，需要根据Qos来决定杀死哪一个container
- 类别
  - BestEffort
    - 没有任何limit或者request约束的container
    - 在node负载过多的时候可能分配不到资源
    - 因为没有limit，在负载较少的情况可以用尽available的资源
  - Burstable
  - Guaranteed

kubernetes Qos

- 使用场景:
  - 如果一个node上面运行了多个pod，podA占用了90%的资源，podB突然需要更多的资源，出现了node资源分配不够的情况，需要根据Qos来决定杀死哪一个container
- 类别
  - BestEffort
  - Burstable
  - Guaranteed
    - pod上面所有container都设定了cpu和mem的requests和limits
    - limits和requests是相同的
    - 如果只设定了limits，默认requests和limits相同，默认pod等级为guaranteed
    - 只能消费符合limits的资源，不能消费多余的

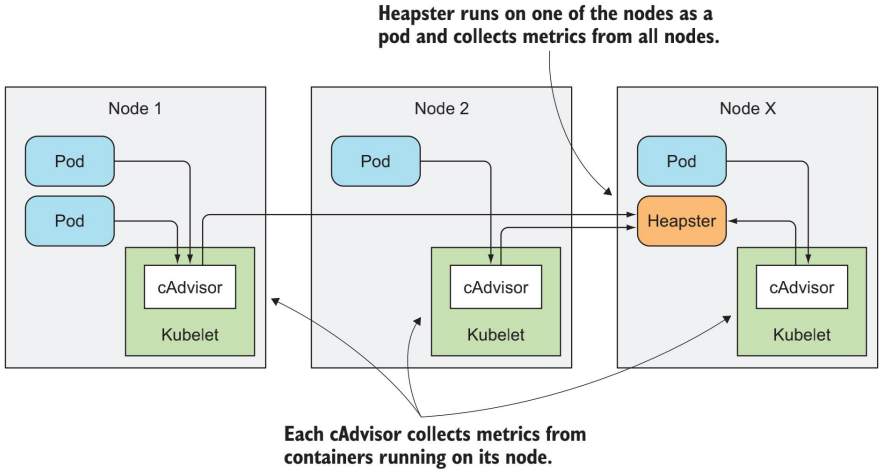
kubernetes Qos

- 使用场景：
  - 如果一个node上面运行了多个pod，podA占用了90%的资源，podB突然需要更多的资源，出现了node资源分配不够的情况，需要根据Qos来决定杀死哪一个container
- 类别
  - BestEffort
  - Burstable
    - 既不是bestEffort又不是guaranteed的pod就是burstable的
    - burstable的pod允许消费多余requests的资源，最多达到limits(if available)
    - burstable的pod确保能被分配requests的资源
  - Guaranteed

kubernetes Qos

- kubectl describe pod在status.qosClass可以查看pod的qos类别
- 实际情况：
  - BestEffort---->Burstable------>Guaranteed(杀死Pod时间线)
  - Qos同级别：
    - 计算OOM score，得分最高的pod被杀死
    - OOM score计算方式
      - 当前pod占用mem在可用mem中的百分比
      - 当前pod的Qos Level和request mem
    - 总结：同级别的pod，占用自己request memory百分比最多的被kill

Monitoring resources



- cAdvisor
  - kubernetes提供的监控agent
  - 运行在每一个node的kubelet中
  - 负责实时收集metrics并汇总到heapster
- heapster
  - 像pod一样运行在node上的插件
  - 汇总所有监控数据
  - 像外界暴露一个查询的ip address（service）

Monitoring resources

- influxDB
  - 常用时序数据库
  - heapster和cAdvisor只能保存短期数据
  - further analysis需要先把数据存到数据库
- Grafana
  - 搭建可视化大盘
  - 以pod的形式运行在集群中
  - 暴露一个ip地址能够打开Grafana window

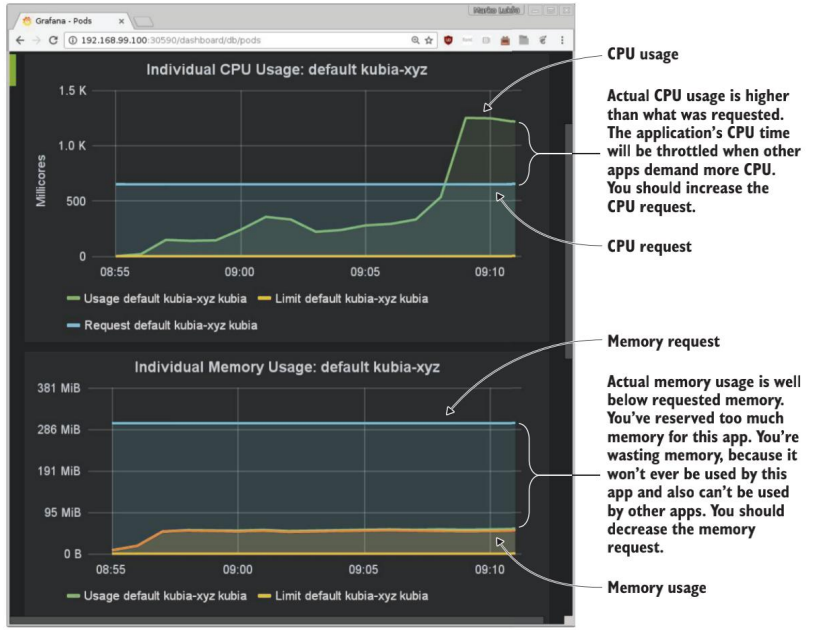


Figure 14.10 CPU and memory usage chart for a pod

Auto Scaling

- 步骤
  - 获得metrics
  - 计算需要的pod数量
  - 更新replicas field

Figure 15.1 Flow of metrics from the pod(s) to the HorizontalPodAutoscaler(s)

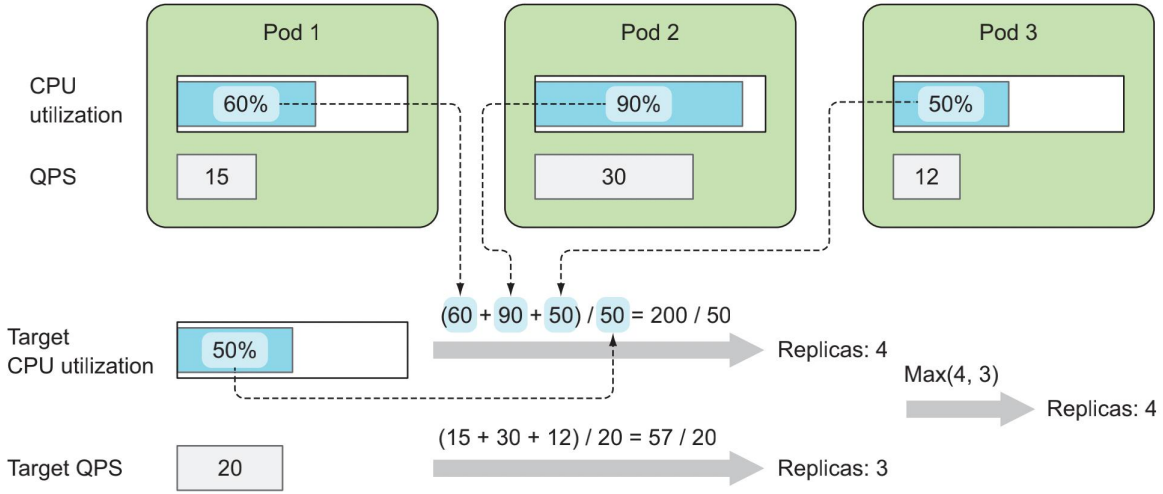


Figure 15.2 Calculating the number of replicas from two metrics

- kubernetes1.8及以上版本可以选择 --horizontal-pod-autoscaler-use-rest-clients=true flag从controller manager拿metric
- 注意：这里计算的百分比单位是pod requested CPU不是node available CPU

Auto scaling

- 目前kubernetes的HPA(horizontal pod autoscaler)只能用cpu作计算metrics
- memory based auto scaling需要解决内存释放的问题(K.I.A 15.1.3)
- custom metric的配置方式非常复杂(K.I.A 15.1.4)

Auto scaling

- 在K.I.A第一版时期(kubernetes 1.9)不能实现vertical scaling
- 在kubernetes1.11以后，HPA从heapster读metrics已废弃
- 目前kubernetes版本提供对vertical autoscaling的支持
  - https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler

Auto scaling

- node粒度的扩容：
  - 仅在云服务器上支持
  - 物理机扩node需要加机器，没有意义

Advanced scheduling

- Node Taints
- Pod Tolerations
- Node selector
- Node affinity
- Pod affinity
- Pod anti-affinity

Node Taints

- taint
  - key node-role.kubernetes.io/master
  - value null
  - effect NoSchedule

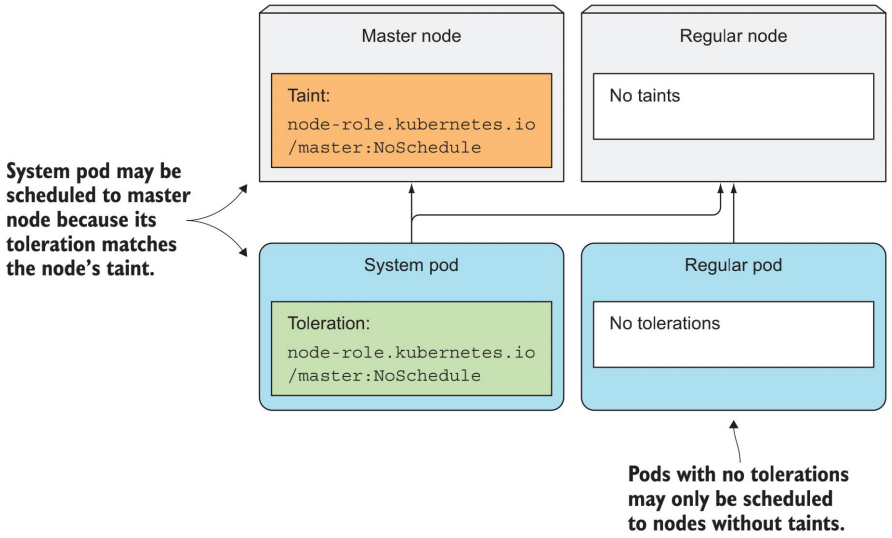
```
$ kubectl describe node master.k8s
Name:           master.k8s
Role:
Labels:         beta.kubernetes.io/arch=amd64
                beta.kubernetes.io/os=linux
                kubernetes.io/hostname=master.k8s
                node-role.kubernetes.io/master=
Annotations:    node.alpha.kubernetes.io/ttl=0
                volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:         node-role.kubernetes.io/master:NoSchedule
...
```

The master node has one taint.

Node Taints & Pod Toleration

- 只有含有相应pod toleration的pod能被分配到含对应taint的node上

```
$ kubectl describe po kube-proxy-80wqm -n kube-system
...
Tolerations:    node-role.kubernetes.io/master=:NoSchedule
                node.alpha.kubernetes.io/notReady=:Exists:NoExecute
                node.alpha.kubernetes.io/unreachable=:Exists:NoExecute
...
```



Node Taints

- 典型Node Taints Effect
  - NoSchedule
    - 如果pod不容忍这个taint就不会被调度到该node上
  - PreferNoSchedule
    - scheduler会尽量避免调度不含该toleration的pod到tainted node上
    - 如果实在没有其他node available才调度
  - NoExecute
    - 不仅影响调度还影响执行
    - 正在运行的pod如果不容忍该toleration则会被驱逐出tainted node

Node Taints

- Adding taint
  - kubectl taint node <nodeName> <key=value:effect>

```
$ kubectl taint node node1.k8s node-type=production:NoSchedule
node "node1.k8s" tainted
```

- Adding toleration

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prod
spec:
  replicas: 5
  template:
    spec:
      ...
      tolerations:
      - key: node-type
        Operator: Equal
        value: production
        effect: NoSchedule
```

This toleration allows the pod to be scheduled to production nodes.

Node Affinity

- 类似Node Selector
  - pod的spec
  - 选择适合该pod调度的node
  - 通过node的label选择
- 跟Node affinity应用场景相关的label示例
  - geographical region info
  - availability zone
  - hostname

These three labels are the most important ones related to node affinity.

```
beta.kubernetes.io/os=linux
cloud.google.com/gke-nodepool=default-pool
failure-domain.beta.kubernetes.io/region=europe-west1
failure-domain.beta.kubernetes.io/zone=europe-west1-d
kubernetes.io/hostname=gke-kubia-default-pool-db274c5a-mjnf
```

Node Affinity

- Node Affinity > Node Selector
  - 除了match提供了prefer的模式
  - 提供了scheduling和executing两种影响方式

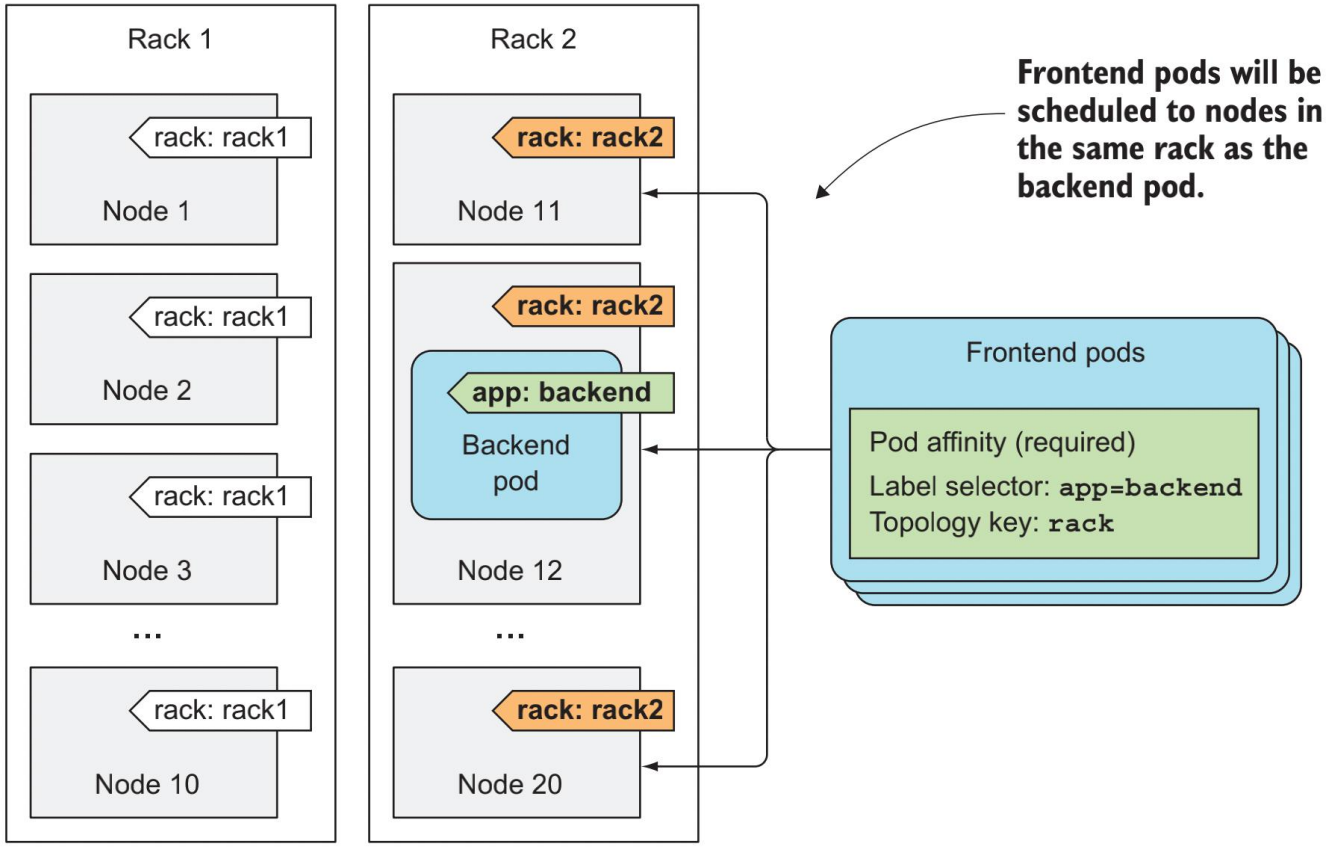
```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: gpu
            operator: In
            values:
            - "true"
```

Pod Affinity

```
spec:
  affinity:
    podAffinity:                                                    <-
      requiredDuringSchedulingIgnoredDuringExecution:
      - topologyKey: kubernetes.io/hostname
        labelSelector:
          matchLabels:
            app: backend
...
```

- pods之间的调度偏好
- topologyKey
  - label key
- labelSelector
  - matchLabels/matchExpression
- 逻辑
  - pod会被调度到label selector matched的node的相同topology key上
  - 例如：topologyKey=hostName pod会被调度到刚好match上的那个node
  - 例如：topologyKey=labelName pod可被调度到match上的node相同labelName的所有node上

Pod Affinity



实验思考

- advanced scheduling的这些rules用不用？
- 如果用的话，是作为核心决策(违背RL模型)还是模型外部输入在最后一层使用？
- 目前的kubernetes提供的scheduling method用处不大，需要看自定义scheduler相关的资料

实践进展

- 动态调度模块前端
- 动态调度模块后端

下周计划

- 马尔可夫决策过程+Q learning
- kubernetes custom scheduler
- kubernetes vertical scaling
- 预测模块整合(?)