K8S 20: Kubernetes. Resources. Quota

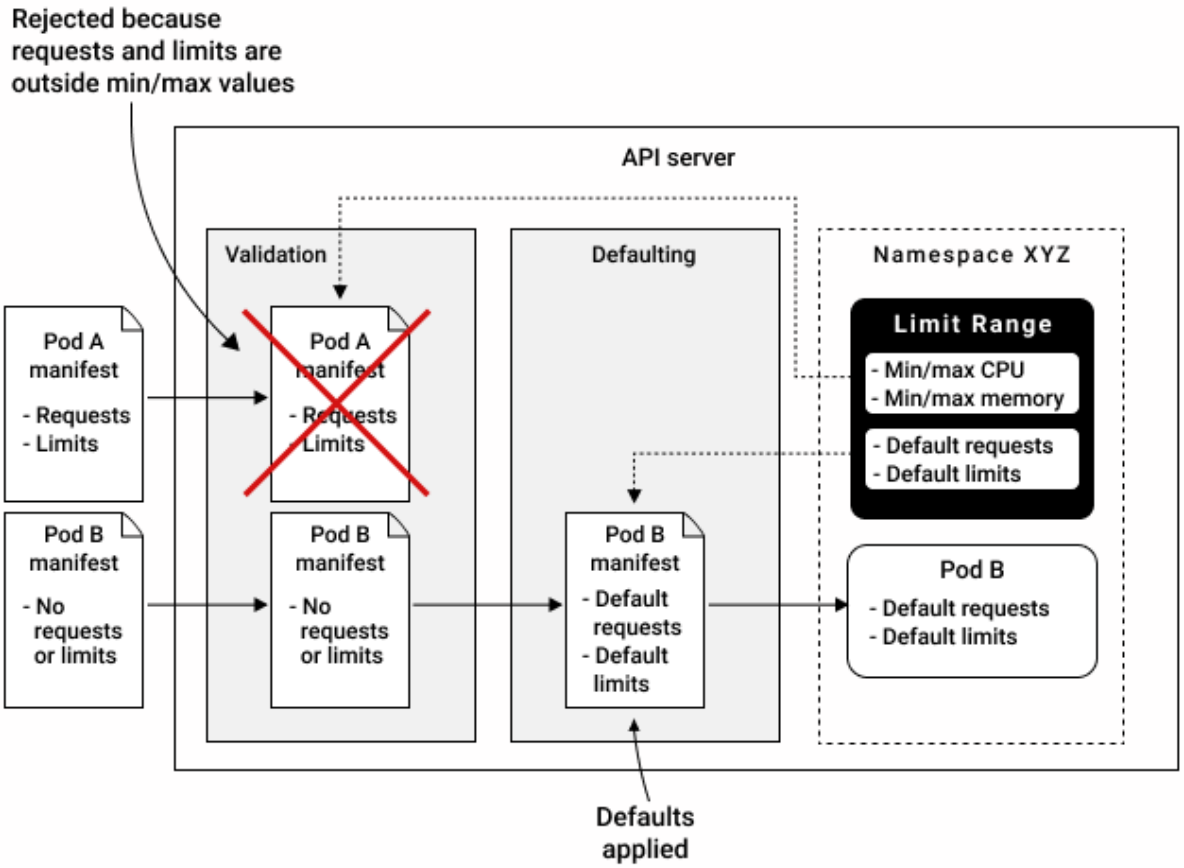
Описание:

[Setting Resource Requests and Limits in Kubernetes (Kubernetes Best Practices)]
Для задания лимитов в Kubernetes, кроме уже разобранных ограничений по ресурсам на уровне контейнера, существует еще 2 сущности — LimitRange и ResourceQuota.
Действуют данные сущности на уровне Namespace, а не всего кластера.
LimitRange позволяет задать рамки и значения по умолчанию для запросов и лимитов на ресурсы, тем самым позволяя не указывать их в контейнерах. Записанные ограничения на уровне конкретных ресурсов имеют больший приоритет, тем самым LimitRange позволяет перевести автоматически все ресурсы в класс Burstable, если это требуется.

Пример:

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: example
spec:
  limits:
  - type: Pod
    min:
      cpu: 50m
      memory: 5Mi
    max:
      cpu: 1
      memory: 1Gi
  - type: Container
    defaultRequest:
      cpu: 100m
      memory: 10Mi
    default:
      cpu: 200m
      memory: 100Mi
    min:
      cpu: 50m
      memory: 5Mi
    max:
```

```
    cpu: 1
    memory: 1Gi
  maxLimitRequestRatio:
    cpu: 4
    memory: 10
- type: PersistentVolumeClaim
  min:
    storage: 1Gi
  max:
    storage: 10Gi
```



Таким образом, мы ограничили ресурсы для Pod, контейнеров и PersistenVolumeClaim, а также задали значения по умолчанию для контейнеров.
Для ограничения суммарных ресурсов, разрешенных для использования в Namespace нужен другой объект — ResourceQuota.

Пример:

```
apiVersion: v1
```

```yaml
kind: ResourceQuota
metadata:
  name: cpu-and-mem
spec:
  hard:
    requests.cpu: 400m
    requests.memory: 200Mi
    limits.cpu: 600m
    limits.memory: 500Mi
```

Точно также можно задать квоту на PVC, на каждый StorageClass отдельно:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storage
spec:
  hard:
    requests.storage: 500Gi
    ssd.storageclass.storage.k8s.io/requests.storage: 300Gi
    standard.storageclass.storage.k8s.io/requests.storage: 1Ti
```

И даже ограничить количество объектов в namespace:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: objects
spec:
  hard:
    pods: 10
    replicationcontrollers: 5
    secrets: 10
    configmaps: 10
    persistentvolumeclaims: 4
    services: 5
    services.loadbalancers: 1
    services.nodeports: 2
    ssd.storageclass.storage.k8s.io/persistentvolumeclaims:
```

Кроме того, ResourceQuota позволяет более гранулярно управлять квотами в рамках QOS класса:

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: besteffort-notterminating-pods
spec:
  scopes:
  - BestEffort
  - NotTerminating
  hard:
    pods: 4
```

Данный пример задает квоту на 4 пода класса BestEffort в состоянии NonTerminating. То есть только 4 пода такого QOS класса могут существовать одновременно в данном namespace.

Полезные ссылки:

- Limit Ranges (official docs)
- Resource Quotas (official docs)
- Quotas and Limit Ranges (openshift official docs)

Задание:

1. Напишите манифест для LimitRange со следующими параметрами:
- Параметры для Pods:
    - min cpu - 10m,
    - min memory - 5Mi,
    - max cpu - 500m,
    - max memory - 500Mi,
- Параметры для Container:
    - default request cpu - 10m,
    - default request memory - 5Mi.
2. Напишите манифест для ResourceQuota со следующими параметрами:
- Жесткие (hard) ограничения:
    - requests cpu - 500m,
    - requests memory - 500Mi,
    - limit cpu - 1000m,
    - limit memory - 1000Mi,
    - pods - 50.

3. С учетом запросов на ресурсы у Deployments из предыдущего задания посчитайте, сколько подов каждого класса можно будет запустить (с учетом, что только один тип нагрузки запускается единовременно), и проверьте это на практике.
4. В ответе пришлите подсчеты, манифесты каждого типа с конечным количеством реплик и вывод списка контейнеров при проверке на практике.