Kubernetes - CKAD (Limit Range & Resource Quota)

MSc. Carlos Avendaño Arango

Resources

Github Repository

https://github.com/carloselpapa10/kubernetes.git

Roadmap

Introduction

Limit Range

Resource Quota

Conclusion

Introduction

By default, containers run with unbounded compute resources on a Kubernetes cluster. With Resource quotas, cluster administrators can restrict the resource consumption and creation on a namespace basis. Within a namespace, a Pod or Container can consume as much CPU and memory as defined by the namespace's resource quota.

Limit Range

There is a concern that one **Pod or Container** could monopolize all of the resources. Limit Range is a policy to constrain resource by **Pod or Container** in a namespace.

A limit range, defined by a LimitRange object, provides constraints that can:

- Enforce minimum and maximum compute resources usage per **Pod or Container** in a *namespace*.
- Enforce minimum and maximum storage request per **PersistentVolumeClaim** in a namespace.
- Enforce a ratio between request and limit for a resource in a **namespace**.
- Set default request/limit for compute resources in a **namespace** and automatically inject them to Containers at runtime.

https://kubernetes.io/docs/concepts/policy/limit-range/

Overview of Limit Range

- The administrator creates one LimitRange in one namespace.
- Users create resources like ***Pods, Containers, and PersistentVolumeClaims*** in the namespace.
- The LimitRanger admission controller enforces defaults limits for all Pods and Containers that do not set compute resource requirements and tracks usage to ensure it does not exceed resource minimum, maximum and ratio defined in any LimitRange present in the namespace.
- If creating or updating a resource violates a limit range constraint, the request to the API server will fail with HTTP status code **403 FORBIDDEN**.
- If limit range is activated in a namespace for compute resources like CPU and MEMORY, users must specify requests or limits for those values; otherwise, the system may reject pod creation.
- LimitRange validations occurs only at Pod admission stage, not on Running pods.

Limit Range

Let us validate most of the points described in the previous slide!

Namespace

- Create a namespace
  $ kubectl create namespace limitrange-demo

- To avoid passing the target limitrange-demo in your kubectl commands, change the context using this command:
  $ kubectl config set-context --current --namespace=limitrange-demo

- Delete the namespace
  $ kubectl delete namespace limitrange-demo

Limit Range - Limiting Container compute resources

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-container
spec:
  limits:
  - max:
      cpu: “800m”
      memory: “1Gi”
    min:
      cpu: “100m”
      memory: “99Mi”
    default:
      cpu: “700m”
      memory: “900Mi”
    defaultRequest:
      cpu: “110m”
      memory: “111Mi”
    type: Container
```

Create a Limit Range in the namespace created in the previous slide.

- Create a limit range
  $ kubectl apply -f lab-limitrange-resourcequota/limit-mem-cpu-per-container.yaml

- Describe a Limit Range
  $ kubectl describe limitrange limit-mem-cpu-per-container

Limit Range - Case 1

Suppose you create a Pod with one container without specifying any compute resources such as CPU and MEMORY. Let's review the limits and requests values belonging to that container.

$ kubectl run busybox --image=busybox --restart=Never -- /bin/sh -c "echo hello world;sleep 3600"

```
nycmbw-nwx0087:lab1 cavendanoa$ kubectl run busybox --image=busybox --restart=Never --dry-run -o yaml -- /bin/sh -c 'echo hello world;sleep 3600'
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - echo hello world;sleep 3600
    image: busybox
    name: busybox
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Never
status: {}
```

Limit Range - Case 1

$ kubectl describe pod busybox

We can see that the compute resources such CPU and MEMORY were automatically assigned to the container as specified in the LimitRange configuration.

```
Containers:
  busybox:
    Container ID:  docker://a47baa75c2791bdc0515a3f528f627611ce9b9709a9b8e0b67603a200e1133f3
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea85129b5b5983d0e0fb52d9ab70
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      echo hello world;sleep 3600
    State:          Running
      Started:      Wed, 24 Jul 2019 15:51:46 -0500
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     700m
      memory:  900Mi
    Requests:
      cpu:        110m
      memory:     111Mi
    Environment:  <none>
```

Limit Range - Case 2

Suppose you create a Pod with one container specifying requests but not limits as compute resources. Let's see what will happen.

$ kubectl create -f lab-limitrange-resourcequota/pod-only-with-requests.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - echo hello world;sleep 3600
    image: busybox
    name: busybox
    resources:
      requests:
        cpu: "105m"
        memory: "100Mi"
~
```

Limit Range - Case 2

$ kubectl describe pod busybox

We can see that cpu and memory values for limits compute resource were automatically assigned from the LimitRange configuration. So, only the requests values from the Pod creation were taking into account.

```
Containers:
  busybox:
    Container ID:  docker://63721a4669e0af9c619b525154b7e5afb8a9e2d251d79626dfe4b21c645f6131
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea85129b5b5983d0e0fb52d9ab70
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      echo hello world;sleep 3600
    State:          Running
      Started:      Wed, 24 Jul 2019 16:44:12 -0500
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     700m
      memory:  900Mi
    Requests:
      cpu:        105m
      memory:     100Mi
    Environment:  <none>
```

Limit Range - Case 3

Suppose you create a Pod with one container specifying limits but not requests as compute resources. Let's see what will happen.

$ kubectl apply -f lab-limitrange-resourcequota/pod-only-with-limits.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - echo hello world;sleep 3600
    image: busybox
    name: busybox
    resources:
      limits:
        cpu: "105m"
        memory: "105Mi"
```

Limit Range - Case 3

$ kubectl describe pod busybox

Unlike in previous case, the compute resources established for limits were assigned automatically for requests as you can see in the description of the pod.

```
Containers:
  busybox:
    Container ID:  docker://38fecdc4a767f0e3fb7ba9e12856b1afb3333a3d5e00bd027ca157dab3c986ca
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea85129b5b5983d0e0fb52d9ab70
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      echo hello world;sleep 3600
    State:          Running
      Started:      Wed, 24 Jul 2019 18:01:37 -0500
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     105m
      memory:  105Mi
    Requests:
      cpu:        105m
      memory:     105Mi
    Environment:  <none>
```

Limit Range - Case 4

Suppose you create a Pod with one container specifying limits and requests as compute resources. Remember that requests **MUST NOT** be greater than limits values. Let's see what will happen.

$ kubectl apply -f lab-limitrange-resourcequota/pod-with-compute-resources.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - echo hello world;sleep 3600
    image: busybox
    name: busybox
    resources:
      limits:
        cpu: "750m"
        memory: "950Mi"
      requests:
        cpu: "700m"
        memory: "800Mi"
```

Limit Range - Case 4

$ kubectl describe pod busybox

Without surprise, the compute resources such as limits and requests were assigned to the container.

```
Containers:
  busybox:
    Container ID:  docker://5a493510b116901cb6bd7387bdd0c8d28ee761422314f782289e994f2c4e0289
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:9f1003c480699be56815db0f8146ad2e22efea85129b5b5983d0e0fb52d9ab70
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      echo hello world;sleep 3600
    State:          Running
      Started:      Wed, 24 Jul 2019 18:26:05 -0500
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     750m
      memory:  950Mi
    Requests:
      cpu:        700m
      memory:     800Mi
    Environment:  <none>
```

Limit Range - Limiting Pod compute resources

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-pod
spec:
  limits:
  - max:
      cpu: “2”
      memory: “1Gi”
    type: Pod
```

Create a Limit Range in the namespace named *limitrange-demo*.

- Create a limit range
  $ kubectl apply -f lab-limitrange-resourcequota/limit-mem-cpu-per-container.yaml

- Describe a Limit Range
  $ kubectl describe limitrange limit-mem-cpu-per-pod

Limit Range - Case 5

Suppose you create a Pod with two containers. What will occur if the sum of any compute resource in this Pod is greater than one specified in the limit range named ***limit-mem-cpu-per-pod***?

$ kubectl apply -f lab-limitrange-resourcequota/pod-with-two-containers.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: busybox
  name: busybox
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - echo hello world;sleep 3600
    image: busybox
    name: busybox1
    resources:
      limits:
        cpu: "1"
        memory: "1500Mi"
      requests:
        cpu: "1"
        memory: "1000Mi"
  - args:
    - /bin/sh
    - -c
    - echo hello world2;sleep 3600
    image: busybox
    name: busybox2
    resources:
      limits:
        cpu: "1"
        memory: "700Mi"
      requests:
        cpu: "1"
        memory: "500Mi"
```

Limit Range - Case 5

As the sum of one compute resource (i.e. limits) in the Pod violates the limit range constraint, the request to the API server failed with HTTP status code **403 FORBIDDEN** as seen in the picture below.
Container 1 => 1500 Mi + Container 2 => 700 Mi = 2200 Mi = 2.2 Gi > 2Gi which is the maximum permitted.

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl apply -f lab-limitrange-resourcequota/pod-with-two-containers.yaml
Error from server (Forbidden): error when creating "lab-limitrange-resourcequota/pod-with-two-containers.yaml": pods "busybox"
is forbidden: maximum memory usage per Pod is 2Gi, but limit is 2306867200
nycmbw-nwx0087:kubernetes cavendanoa$
```

Limit Range - Limiting Storage Resources

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storage-limits
spec:
  limits:
  - max:
      storage: “2Gi”
    min:
      storage: “1Gi”
    type: PersitentVolumeClaim
```

Create a Limit Range in the namespace named *limitrange-demo*.

- Create a limit range
  $ kubectl apply -f lab-limitrange-resourcequota/limit-storage.yaml

- Describe a Limit Range
  $ kubectl describe limitrange limit-storage.yaml

Limit Range - Case 6

Suppose you create a PersistentVolumeClaim with 500 Mi as requests storage. Will it be allowed in the current namespace?

$ kubectl apply -f lab-limitrange-resourcequota/pvc-limit-lower.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-limit-lower
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

Limit Range - Case 6

The Kubernetes API server won't allow the PersistentVolumeClaim be created since the requests storage is lower than the value defined in the Limit Range.
500 Mi < 1Gi which is the minimum permitted.

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl apply -f lab-limitrange-resourcequota/pvc-limit-lower.yaml
Error from server (Forbidden): error when creating "lab-limitrange-resourcequota/pvc-limit-lower.yaml": persistentvolumeclaims "pvc-limit
-lower" is forbidden: minimum storage usage per PersistentVolumeClaim is 1Gi, but request is 500Mi
nycmbw-nwx0087:kubernetes cavendanoa$
```

Limit Range - Limits/Requests ration

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-memory-ratio-pod
spec:
  limits:
  - maxLimitRequestRatio:
      memory: 2
    type: Pod
```

If **LimitRangeItem.maxLimitRequestRatio** is specified in the **LimitRangeSpec**, the named resource must have a request and limit that are both non-zero where limit divided by request is less than or equal to the enumerated value.

This LimitRange enforces memory limit to be at most twice the amount of the memory request for any pod in the namespace.

Create a Limit Range in the namespace named *limitrange-demo*.

- Create a limit range
  $ kubectl apply -f lab-limitrange-resourcequota/limit-memory-ratio-pod.yaml

- Describe a Limit Range
  $ kubectl describe limitrange limit-memory-ratio-pod

Limit Range - Case 7

Suppose that you create a Pod with compute resources in which requests memory divided by limits does not fit with the value defined in the Limit Range.

$ kubectl apply -f lab-limitrange-resourcequota/limit-range-pod-3.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: busybox3
spec:
  containers:
  - name: busybox01
    image: busybox
    resources:
      limits:
        memory: "300Mi"
      requests:
        memory: "100Mi"
```

Right values should be 200Mi for limits memory and at most 100Mi for requests memory.

Limit Range - Case 7

The Kubernetes API server won't allow the Pod be created since the requests memory divided by limits is greater than the value defined in the Limit Range.
300Mi / 100Mi = 3 > 2 which is the limit request ratio permitted.

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl apply -f lab-limitrange-resourcequota/limit-range-pod-3.yaml
Error from server (Forbidden): error when creating "lab-limitrange-resourcequota/limit-range-pod-3.yaml": pods "busybox3" is forbidden:
memory max limit to request ratio per Pod is 2, but provided ratio is 3.000000
nycmbw-nwx0087:kubernetes cavendanoa$
```

Resource Quotas

A resource quota, defined by a ***ResourceQuota*** object, provides constraints that limit the **aggregate resource consumption** per namespace. It can limit the quantity of objects that can be created in a **namespace** by type, as well as the total amount of compute resources that may be consumed by resources in that project.

https://kubernetes.io/docs/concepts/policy/resource-quotas/

Overview of Resource Quotas

- Different teams work in different namespaces. Currently this is voluntary, but support for making this mandatory via ACLs is planned.
- The administrator creates one Resource Quota for each namespace.
- Users create resources (Pod, Services, etc) in the namespace, and the quota system tracks usage to ensure it does not exceed hard resource limits defined in a ResourceQuota.
- If creating or updating a resource violates a quota constraint, the request will fail with HTTP status code 403 FORBIDDEN.
- If quota is enabled in a namespace for compute resources such as CPU and MEMORY, users must specify requests or limits for those values, the quota system may reject pod creation.

HINT: Use the LimitRanger admission controller to force defaults for pods that make no compute resources requirements.

Resource Quotas - Object count

Here is an example set of resources users may want to put under object count quota:

- count/persistentvolumeclaims
- count/services
- count/secrets
- count/configmaps
- count/replicationcontrollers
- count/deployments.apps
- count/replicasets.apps
- count/statefulsets.apps
- count/jobs.batch
- count/cronjobs.batch
- count/deployments.extensions

Case 8: Resource Quotas per PriorityClass

Pods can be created at a specific priority. You can control a pod's consumption of system resources based on a pod's priority, by using the ***scopeSelector*** field in the quota spec.

A quota is matched and consumed only if ***scopeSelector*** in the quota spec selects the pod.

PriorityClass

A priority class is a **non-namespaced** object that defines a mapping from a priority class name to the integer value of the priority. The name is specified in the name field of the PriorityClass object's metadata. The value is specified in the required value field. The higher value, the higher the priority.

PriorityClass has two optional fields: ***globalDefault*** and ***description***.

The *globalDefault* field indicates that the value of this *PriorityClass* should be used for Pods without a *priorityClassName*. Only one PriorityClass with *globalDefault* set to true can exist in the system. If there is no PriorityClass with globalDefault set, the priority of Pods with no priorityClassName is zero.

PriorityClass sample: high, medium, low

$ kubectl apply -f lab-limitrange-resourcequota/priority.yaml

```
apiVersion: v1
kind: List
items:
- apiVersion: scheduling.k8s.io/v1
  kind: PriorityClass
  metadata:
    name: high
  value: 1000000
  globalDefault: false
  description: "This priority class should be used for high priority pods only."
- apiVersion: scheduling.k8s.io/v1
  kind: PriorityClass
  metadata:
    name: medium
  value: 800000
  globalDefault: false
  description: "This priority class should be used for medium priority  pods only."
- apiVersion: scheduling.k8s.io/v1
  kind: PriorityClass
  metadata:
    name: low
  value: 500000
  globalDefault: true
  description: "This priority class should be used for low priority  pods only."
```

Case 8: Resource Quotas per PriorityClass

This example creates a quota object and matches it with pods at specific priorities.

$ kubectl apply -f lab-limitrange-resourcequota/quota.yaml

```
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: pods-high
  spec:
    hard:
      cpu: "1000"
      memory: 200Gi
      pods: "10"
    scopeSelector:
      matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values: ["high"]
```

```
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: pods-medium
  spec:
    hard:
      cpu: "10"
      memory: 20Gi
      pods: "10"
    scopeSelector:
      matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values: ["medium"]
```

```
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: pods-low
  spec:
    hard:
      cpu: "5"
      memory: 10Gi
      pods: "10"
    scopeSelector:
      matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values: ["low"]
```

Case 8: Resource Quotas per PriorityClass

This example creates a Pod with high as priority class name.

$ kubectl apply -f lab-limitrange-resourcequota/high-priority-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: high-priority
spec:
  containers:
  - name: high-priority
    image: ubuntu
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo hello; sleep 10; done"]
    resources:
      requests:
        memory: "10Gi"
        cpu: "500m"
      limits:
        memory: "10Gi"
        cpu: "500m"
  priorityClassName: high
```

Case 8: Resource Quotas per PriorityClass

Here we can see how this new pod is taken by the high priority group.

$ kubectl describe quota

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl describe quota
Name:       pods-high
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         500m  1k
memory      10Gi  200Gi
pods        1     10


Name:       pods-low
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         0     5
memory      0     10Gi
pods        0     10


Name:       pods-medium
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         0     10
memory      0     20Gi
pods        0     10
```

Case 8: Resource Quotas per PriorityClass

What will happen if we create a new Pod without specifying a priorityClassName?

$ kubectl apply -f lab-limitrange-resourcequota/no-priority-defined-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: no-priority
spec:
  containers:
  - name: no-priority
    image: ubuntu
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo hello; sleep 10; done"]
    resources:
      requests:
        memory: "1Gi"
        cpu: "50m"
      limits:
        memory: "5Gi"
        cpu: "50m"
```

Case 8: Resource Quotas per PriorityClass

Here we can see how this new pod is taken by the low priority group since the *defaultGlobal* field was defined as true in the low priority configuration.

$ kubectl describe quota

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl describe quota
Name:       pods-high
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         500m  1k
memory      10Gi  200Gi
pods        1     10


Name:       pods-low
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         50m   5
memory      1Gi   10Gi
pods        1     10


Name:       pods-medium
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         0     10
memory      0     20Gi
pods        0     10
```

Case 8: Resource Quotas per PriorityClass

According to the Resources Quota defined in lab-limitrange-resourcequota/quota.yaml, we must specify compute resources such as CPU and Memory for each new Pod. Next error is generated when trying to create a pod without specifying compute resources.

$ kubectl apply -f lab-limitrange-resourcequota/no-resources-defined-pod.yaml

```
nycmbw-nwx0087:kubernetes cavendanoa$ kubectl apply -f lab-limitrange-resourcequota/no-resources-defined-pod.yaml
Error from server (Forbidden): error when creating "lab-limitrange-resourcequota/no-resources-defined-pod.yaml": pods "no-resources" is forbidden: failed quota: pods-low: must specify cpu,memory
nycmbw-nwx0087:kubernetes cavendanoa$
```

Setting Quotas

$ kubectl create namespace myspace

$ kubectl apply -f lab-limitrange-resourcequota/compute-resources-quota.yaml -n myspace

$ kubectl apply -f lab-limitrange-resourcequota/object-counts-quota.yaml

$ kubectl create quota test --hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secrets=4 --namespace myspace

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
spec:
  hard:
    configmaps: "10"
    persistentvolumeclaims: "4"
    replicationcontrollers: "20"
    secrets: "10"
    services: "10"
    services.loadbalancers: "2"
```

Conclusion

Resource Quotas are used to limit the aggregate compute resources as well as the objects (Pods, Services, Deployments, etc) we may have in our system. Using Resource Quotas together with Limit Range we can control the resources consumption of our applications.