DRA: Per-device Consumable Capacity

DRA: Per-device Consumable Capacity

Shared Publicly

Contact: Sunyanan Choochotkaew

This document is aimed for technical discussion about the KEP 5075: https://github.com/kubernetes/enhancements/pull/5104.

KEP-specific meeting notes can be found in the document tabs.

KEP Background

A motivating use case is to allocate a shared network device in CNI DRA driver which can be selected by more than one pods on demand (on claim). The original discussion is in this PR's comment thread. The limitation of current implementation has been addressed here. The virtual network device is created and configured once the CNI is called based on the information of the master network device. The configured information specific to the generated device cannot be listed in the ResourceSlice. A similar issue where the partial part (share) of the device is determined by the ResourceClaim in this comment of the KEP-4815 DRA Partitionable devices design update PR #5066.

Relations to related KEPs:
- KEP 4815: The partitioned devices can further be a sharable device.
- KEP 5007: The allocated share can be provisioned at the pre-bind step.

KEP Summary

The enhancement enables allocating sharable devices to more than one resource claims under consumable capacity of each per-device resource.
To achieve that, this KEP introduces
- a new device field to distinguish between “shareable” and “unshareable” devices,
- a capacity-aware scheduling mechanism that allows limiting or guaranteeing the shared device capacity,
- a new capacity requirement field in the device request of the resource claim,
- a new consumed capacity field in the allocation result of the resource claim,
- a distinct attribute constraint to prevent allocating the same shared device in the same claim.

Design

This enhancement introduces a `shared` field within the `BasicDevice` of the ResourceSlice to mark whether the device is a shared device.
The shared device can be assigned to more than one claim if it satisfies the request.
User needs to explicitly add the select condition device.shared to identify whether to select the shared device or not.

The enhancement also adds a `claimPolicy` field to `DeviceCapacity`.
This field is used to identify the capacity which has a limited quantity and to specify capacity claim policy.
If the cliam policy field is not empty. Either one of the policies must be defined with default value.

Users can define specific per-device resource requests through the newly added `CapacityRequest` field in the `DeviceRequest`.
`Requests` of `CapacityRequest` is used for verifying minimum device capacity.
If the capacity is consumable, the amount available for allocation is determined by subtracting the aggregated allocation results of current claims from the device's capacity as defined in the resource slice.
The subtracted amount will be used solely by the allocator and will not be reflected in the resource slice.
The subtracted amount is calculated based on the consumable conditions and will round the requested capacity up to the nearest valid amount under those conditions.
If users do not specify a capacity request for consumable capacity, the default consumed value will be applied.

A shared device can only be allocated once its consumability has been verified
and its attributes match the request's selectors and constraints.
The newly added `capacities` field in the `DeviceRequestAllocationResult` will be set when the allocation is successful.

API Design

Resource Slice BasicDevice

```
// BasicDevice defines one device instance.
type BasicDevice struct {
       ...
   // Shared marks whether the device is shared.
   //
   // The device with shared="true" can be allocated to more than one resource claim,
   //
   // +optional
   // +default=false
   // +featureGate=DRAConsumableCapacity
   Shared *bool `json:"shared,omitempty" protobuf:"bytes,9,opt,name=shared"`
}

// DeviceCapacity describes a quantity associated with a device.
type DeviceCapacity struct {
   // Value defines how much of a certain device capacity is available.
   //
   // +required
   Value resource.Quantity `json:"value" protobuf:"bytes,1,rep,name=value"`

   // ClaimPolicy specifies that this device capacity must be consumed
   // by each resource claim according to the defined consumption policy.
   //
   // +optional
   // +featureGate=DRAConsumableCapacity
   ClaimPolicy *CapacityClaimPolicy `json:"claimPolicy,omitempty"
protobuf:"bytes,2,opt,name=claimPolicy"`
}

// CapacityClaimPolicy defines consumption policy for consumable capacity.
// Either one of the consumption policies must be defined.
type CapacityClaimPolicy struct {
   // Set defines a set of acceptable quantities of consuming requests.
   //
   // +optional
   // +oneOf=CapacityClaimPolicy
   Set *CapacityClaimPolicySet `json:"set,omitempty" protobuf:"bytes,1,opt,name=set"`
```

```
    // Range defines an acceptable quantity range of consuming requests.
    // +optional
    // +oneOf=CapacityClaimPolicy
    Range *CapacityClaimPolicyRange `json:"range,omitempty"
protobuf:"bytes,2,opt,name=range"`
}

// CapacityClaimPolicySet defines a discrete set of allowable capacity values for
consumption.
type CapacityClaimPolicySet struct {
    // Default specifies the default capacity to be used for a consumption request
    // if no value is explicitly provided.
    //
    // +required
    Default resource.Quantity `json:"default,omitempty"
protobuf:"bytes,1,name=default"`

    // Options defines a discrete set of additional valid capacity values that can be
requested.
    // +optional
    // +listType=atomic
    Options []resource.Quantity `json:"options,omitempty"
protobuf:"bytes,2,rep,name=options"`
}

// CapacityClaimPolicyRange defines a valid range of consuming capacity.
type CapacityClaimPolicyRange struct {
    // Minimum specifies the minimum capacity required for a consumption request.
    // This field is required and serves as the default capacity to be consumed.
    //
    // +required
    Minimum resource.Quantity `json:"minimum,omitempty"
protobuf:"bytes,1,name=minimum"`

    // Maximum specifies the maximum capacity that can be requested in a consumption
request.
    //
    // +optional
    Maximum *resource.Quantity `json:"maximum,omitempty"
protobuf:"bytes,2,opt,name=maximum"`
```

```
   // Step defines the incremental block size by which capacity can increase from the
minimum.
   //
   // +optional
   Step *resource.Quantity `json:"step,omitempty" protobuf:"bytes,3,opt,name=step"`
}
```

Resource Claim DeviceRequest

```
type DeviceRequest struct {
       ...
   // Capacities define resource requirements against each capacity.
   //
   // +optional
   // +featureGate=DRAConsumableCapacity
   Capacities *CapacityRequirements `json:"capacities,omitempty"
protobuf:"bytes,9,opt,name=capacities"`
}

// CapacityRequirements defines the capacity requirements for a specific device
request.
type CapacityRequirements struct {
   // Requests specifies the requested quantity of device capacities for a device
request.
   // +optional
   Requests map[QualifiedName]resource.Quantity `json:"requests,omitempty"
protobuf:"bytes,1,rep,name=requests"`

   // Potentially enhancement field.
   // Limits define the maximum amount of per-device resources allowed.
   // This enables burstable usage when applicable.
}

// DeviceConstraint must have exactly one field set besides Requests.
type DeviceConstraint struct {
       ...
   // DistinctAttribute requires that all devices in question have this
   // attribute and that its type and value are unique across those devices.
   //
```

```
   // For example, specify attribute name to get virtual devices from distinct shared
physical devices.
   //
   // +optional
   // +oneOf=ConstraintType
   DistinctAttribute *FullyQualifiedName `json:"distinctAttribute,omitempty"
protobuf:"bytes,3,opt,name=distinctAttribute"`
}
```

Resource Claim DeviceRequestAllocationResult

```
type DeviceRequestAllocationResult struct {
 ...

   // Shared indicates whether the allocated device can be shared by multiple claims.
   //
   // +optional
   // +featureGate=DRAConsumableCapacity
   Shared *bool `json:"shared,omitempty" protobuf:"bytes,7,opt,name=shared"`

   // ConsumedCapacities is used for tracking the capacity consumed per device by the
claim request.
   // The total consumed capacity for each device must not exceed its corresponding
available capacity.
   //
   // +optional
   // +featureGate=DRAConsumableCapacity
   ConsumedCapacities map[QualifiedName]resource.Quantity
`json:"consumedCapacities,omitempty" protobuf:"bytes,8,rep,name=consumedCapacities"`
}
```

Alternatives

1. How to identify shared devices and how users can select/deselect/opt the shared property?

Current proposal: add a shared field to device property and use CEL as is.

- Steps:
    - Add “shared” field to device property
    - Use “CEL” to select (shared=true), deselect (shared=false), or opt (shared=any, shared=true||shared=false)
- Pros: small changes
- Cons: can cause human (user) error to include (opt) shared device without awareness by not explicitly defining shared=false

Alternatives:

1. add a shared field to device property and implicitly switch default value to deselect instead of opt.
    - Pros: small changes (in user API but not in allocation process)
    - Cons: not trivial to switch default value (when shared is not defined in CEL from any to deselect) and can cause confusion to users.

2. add shared field to device property and explicitly define a field in resource request to select or opt the shared property
    - Steps:
        - Add “shared” field to device property
        - Add “AllowShared” to opt and “RequireShared” to select the shared device
    - Pros: easy to understand and use, reduce human (user) error on making a request
    - Cons: additional field in resource request and may need abstract layer later when the device has future features added.

3. add capabilities field to device property (same layer as attributes and capacities) and these fields under capability are handled separately like capacities.
    - Steps:
        - Add “capabilities” field to device property and define “shared” in the list
        - Add “capabilities” field to resource claim request and add “require”, “optional” to list capabilities to select or opt.
    - Pros: flexible for new future device feature, which should be deselected by default, added to select or opt the device with that feature
    - Cons: significant changes to the API

2. How to handle shared device allocation on the same Pod?

Current proposal: scheduler will not allocate the same shared device to the same pod

- Steps:
    - Add device even if it is the shared device into allocatedDevice list

- Pros: small changes to the allocator logic, practical for virtual functions
- Cons: not trivial to explain why the pod cannot get multiple fractions of resource from the same shared device (use case 12)
    - However, use case 12 can be addressed by future aggregated capacity requests when the user can define total number of capacity regardless of the number of devices allocated.

Alternatives:

Scheduler can allocate the same shared device to the same pod more than one time

- Steps:
    - Define another memo for shared devices to keep not only the list of allocated devices but also the amount of allocated capacity to compute availability within the same pod allocation process.
    - Need a flag to prevent allocating the same device (use case 2)
    - Need a field to define an algorithm to allocate over shared devices such as bin packing or equal distribution.
- Pros: customizability
- Cons: complexity

3. How to define shareable property of device?

https://github.com/kubernetes/enhancements/pull/5104#discussion_r2100686267

Summary from KEP comments:

It is a corresponding decision from 1. to have a property field in device basic API to identify whether the device is shareable or not.
Then, there is a comment/discussion → my KEP update decision as below:
- boolean flag is not recommended in API convention → should consider enum
- since there is no other mode comes out except Once or ManyTimes → keep boolean
- ManyTimes = infinite, we may want to limit → we can use count capacity to limit
- ManyTimes = infinite, we don't want infinite → we can change to Allocatable: N to force to have a limit
- In current network behavior, we do have infinite sharing of NIC by default (macvlan, ipvlan) as long as pod can be created. → keep boolean and add alternatives (rename boolean to `allowMultipleAllocations`)

4. How should the default request (no capacity request) behave?

Current:
- resourceslice defines default behavior. For graceful migration, the default should be set to maximum consumable capacity value.

Potential enhancement (expect different KEP):

lionel.jouin@est.tech raises a new use case where a device has bandwidth capacity that could be consumable or not consumable. User 1 request with default supposing no consuming. User 2 can request in the same way but cannot request for consumable (guaranteed) bandwidth on the same device

Rough proposed API extension:

SharingPolicy:
  strategy: AlwaysConsumed|ConsumedOrNever|BlockOrShare

AlwaysConsumed: default behavior to always apply specified default value if no capacity request
ConsumedOrNever: cover the use case that Lionel mentioned above. If the first consumer defines a capacity request, this capacity is marked as consumable. If not (capacityResults: nil), it can't be consumable until the first consumer releases.
BlockOrShare: opposite of consumed of never. If the first consumer requests none of this capacity, it tooks a whole device (full capacity). If the first consumer defines a request, this device can be shared until guaranteed value.

This allows extension to control resources by users.
However, it can introduce a significant change so I think it deserves a different KEP.

Questions?
- Can each capacity have a different strategy? How will it behave?
  → Each capacity can be consumable and non-consumable independently as the current design.
- What users expect when their default has no capacity?
    - Full device (all full capacity)
    - Default consumed capacity
    - Freely share and make the capacity cannot be guaranteed anymore (ConsumedOrNever)

  (currently, the expected behavior is the default consumed capacity if select the shareable device class)
  → To get deterministic behavior, we may need to add more CEL interpretation on the sharing policy mode on capacity domain.

Concrete Use Cases

Concrete Use Cases of Shared Device

(Please find example manifests for each case below.)

No.	Shorthand	Details	Targets	Dependent KEP(s)
1	Give me one shared network device	- [Device(driver)] Advertise shared with no consumable capacity - [Claim] Request one shared device without specifying resource request. - [Scheduler] Assign any shared device from any node in the list. - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims.	Alpha: 1.34	
2	Give me two distinct shared network devices	- [Device(driver)] Advertise shared with no consumable capacity - [Claim] Request exact two shared devices without specifying resource request and give a distinct constraint. - [Scheduler] Assign any two distinct shared devices from any node in the list without specifying resource requests. - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] Both assigned devices can be assigned to other claims.	Alpha: 1.34	
3	Give me a specific shared network device by name (applicable to other attributes such as switch port)	- [Device(driver)] Advertise shared with no consumable capacity - [Claim] Request one shared device	Alpha: 1.34	

		without specifying resource request with selector with name field. - [Scheduler] Assign the shared device with the specified name from the node that has the match device. - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims.		
4	Give me a shared network device with guarantee bandwidth 1Gi	- [Device(driver)] Advertise shared with a consumable capacity of bandwidth. - [Claim] Request one shared device with specifying resource request bandwidth=1Gi and select consumable field of bandwidth to be true. -- [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims but with less availability of capacity (-1Gi).	Alpha: 1.34	
5	Give me a shared network device with bandwidth more than 1Gi.	- [Device(driver)] Advertise shared with a non-consumable capacity of bandwidth. - [Claim] Request one shared device with 1Gi minimum bandwidth. -- [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims.	Alpha: 1.34	

6	Give me a shared network device with bandwidth more than 1 Gi. The shared device has a limit of sharing.	- [Device(driver)] shared with a consumable capacity of claims (minimum consumeCondition= 1), and a non-consumable capacity of bandwidth. - [Claim] Request one shared device with 1Gi minimum bandwidth. One claim capacity is deducted by one. - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims but with less availability of claim capacity (-1).	Alpha: 1.34	
7	Give me a shared device and block other claims to share the device	*- [Device] Has taint* *- [Claim] Has toleration* - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] Ensure that this is also applicable for shared device	Alpha: 1.34	*DRA: device taints and tolerations (1.33)*
8	Consumable capacity has specific refinement of consuming conditions (e.g., predefined blocks, minimum/maximum amount). - GPU with minimum memory chunk - Control single pod cannot reserve more than 5 GiB of bandwidth in guaranteed CNI driver	- [DeviceCapacity] Has consumeConditions. - [Claim] requests an amount of resource. The request amount will round up to allocate enough resources. -- [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims but with less availability of claim capacity (- allocated block size).		

9	Give me a shared network device informing the driver that the bandwidth usage can be burstable up to 10Gi.	- [DeviceCapacity] Advertise shared with a non-consumable capacity of bandwidth. - [Claim] requests maximum bandwidth (limit) 10Gi. - [Scheduler] Assign a shared device that has bandwidth at least 10Gi. - [Scheduler] Ensure that no non-sharable device is selected. - [Scheduler] The same device can be assigned to other claims.		
10	Give me any device regardless of shared property.			
11	Give me any number of network devices with aggregated bandwidth is 10Gi.			
12	Give me two chunks of vGPU memory slices from any physical devices (same GPU is acceptable).			
13	Device has bandwidth that could be consumable or not consumable. User 1 request with default supposing no consuming. User 2 can request in the same way but cannot request for consumable (guaranteed) bandwidth on the same device			

Example Manifests (Use cases to be copied to KEP)

User Stories (Optional)

See [all concrete use cases](https://docs.google.com/document/d/1U0u2uErpYcf-RooPEws5oDMiJ9kT2uDoWNjcWrLH224/edit?tab=t.f3ylp1uxsq1c).

This KEP focuses on the request with a selection of shared devices (`device.shared == "true"`).

```yaml
selectors:
 - cel:
     expression: |-
       device.shared["resource-driver.example.com"] == "true"
```

Story	Consumable Capacity	Selector	Resource Request	AllocationMode	Context
1	no	no	no	Exact(1)	network
2	no	no	no	Exact(2)	network
3	no	yes	no	Exact(1)	network
4	yes	no	yes	Exact(1)	network
5	no	no	yes	Exact(1)	network
6	both	no	yes	Exact(1)	network

Story 1

A DRA driver advertises a shared device
and a user simply requests a shared device.

```yaml
kind: ResourceSlice
...
spec:
 driver: simple-cni.dra.networking.x-k8s.io
 devices:
 - name: eth1
   basic:
     shared: "true"
     attributes:
```

```
      name:
        string: "eth1"
 - name: eth2
   basic:
     shared: "true"
     attributes:
       name:
         string: "eth2"
```

A common shared device class is defined below.

```yaml
kind: DeviceClass
metadata:
 name: simple-shared.networking.x-k8s.io
 selectors:
 - cel:
     expression: |-
       device.driver == "simple-cni.dra.networking.x-k8s.io" &&
       device.shared == "true"
```

Then, a user defines the following resource claim.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: macvlan
     deviceClassName: simple-shared.networking.x-k8s.io
   config:
   - requests:
     - macvlan
     opaque:
       driver: simple-cni.dra.networking.x-k8s.io
       parameters: # CNIParameters with the GVK, interface name and CNI Config (in
YAML format).
         apiVersion: cni.networking.x-k8s.io/v1alpha1
         kind: CNI
```

```
          ifName: "net1"
          config:
            cniVersion: 1.0.0
            name: net1
            plugins:
            - type: macvlan
              mode: bridge
              ipam:
                type: host-local
                ranges:
                - - subnet: 10.10.1.0/24
```

> The device config is out of the KEP scope. From this point, the request config will be omitted for simplicity.

Story 2

With the same resource as story 1, a user requests two devices.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: macvlan
     deviceClassName: simple-shared.networking.x-k8s.io
     allocationMode: ExactCount
     count: 2
```

Or with distinctAttribute

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: macvlan-1
     deviceClassName: simple-shared.networking.x-k8s.io
     allocationMode: ExactCount
```

```
      count: 1
    - name: macvlan-2
      deviceClassName: simple-shared.networking.x-k8s.io
      allocationMode: ExactCount
      count: 1
    constraints:
    - requests:
      - macvlan-1
      - macvlan-2
      distinctAttribute: name
```

Story 3

With the same resource as story 1, a user specifies a sharted device by some attributes such as name.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: net1
     deviceClassName: simple-shared.networking.x-k8s.io
     selectors:
     - cel:
         expression: |-
           device.attributes["simple-cni.dra.networking.x-k8s.io"].name == "eth1"
```

Story 4

A DRA driver specifies a consumable capacity with a minimum consume condition, and a user requests the consumable resource.
A scheduler selects devices according to the availability.

```yaml
kind: ResourceSlice
...
spec:
 driver: guaranteed-cni.dra.networking.x-k8s.io
```

```
 devices:
 - name: eth1
   basic:
     shared: "true"
     attributes:
       name:
         string: "eth1"
     capacity:
       bandwidth:
         consumable:
           minimum: "1Mi"
         value: "10Gi"
```

A shared device class with guaranteed bandwidth is defined below.

```yaml
kind: DeviceClass
metadata:
 name: bandwidth-guaranteed-cni.networking.x-k8s.io
 selectors:
 - cel:
     expression: |-
       device.driver == "guaranteed-cni.dra.networking.x-k8s.io" &&
       device.shared == "true" &&

device.capacities["guaranteed-cni.dra.networking.x-k8s.io"].bandwidth.consumable ==
"true"
```

Then, a user defines the following resource claim.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: net1
     deviceClassName: bandwidth-guaranteed-cni.networking.x-k8s.io
     resources:
       requests:
```

```
        bandwidth: "1Gi"
```

Story 5

A DRA driver specifies a non-consumable capacity,
and a user specifies a minimum bandwidth requested on a shared device.

```yaml
kind: ResourceSlice
...
spec:
 driver: extended-cni.dra.networking.x-k8s.io
 devices:
 - name: eth1
   basic:
     shared: "true"
     attributes:
       name:
         string: "eth1"
     capacity:
       bandwidth:
         value: 10Gi
```

A shared device class is defined below.

```yaml
kind: DeviceClass
metadata:
 name: extended-cni.networking.x-k8s.io
 selectors:
 - cel:
     expression: |-
       device.driver == "extended-cni.dra.networking.x-k8s.io" &&
       device.shared == "true" &&
       device.capacities["extended-cni.dra.networking.x-k8s.io"].bandwidth.consumable
!= "true"
```

Then, a user defines the following resource claim.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: net1
     deviceClassName: extended-cni.networking.x-k8s.io
     resources:
       requests:
         memory: "1Gi"
```

Story 6

A DRA driver specifies both consumable count to limit number of claim to share and non-consumable bandwidth capacity,
and a user specifies a minimum bandwidth requested on a shared device.

```yaml
kind: ResourceSlice
...
spec:
 driver: complex-cni.dra.networking.x-k8s.io
 devices:
 - name: eth1
   basic:
     shared: "true"
     attributes:
       name:
         string: "eth1"
     capacity:
       bandwidth:
         value: 10Gi
       count:
         consumable:
           default: 1
           sets: [1]
         value: 1000
```

A shared device class is defined below.

```yaml
kind: DeviceClass
metadata:
 name: complex-cni.networking.x-k8s.io
 selectors:
 - cel:
     expression: |-
       device.driver == "complex-cni.dra.networking.x-k8s.io" &&
       device.shared == "true" &&
       device.capacities["complex-cni.dra.networking.x-k8s.io"].bandwidth.consumable
!= "true"
```

Then, a user defines the following resource claim.

```yaml
kind: ResourceClaim
...
spec:
 devices:
   requests:
   - name: net1
     deviceClassName: extended-cni.networking.x-k8s.io
     resources:
       requests:
         memory: "1Gi"
         count: 1
```

Meeting Notes

Feb 25

Attendees:
Sunyanan (Pang), Chiba-san, Ming, John, Morten

Agenda
Related to ResourceSlice (DeviceCapacity)

- `Consumable` field in Capacity reverted back?
    - Related comment: https://github.com/kubernetes/enhancements/pull/5104#discussion_r1961990235
    - May set in unresolve question
        - What is the better user experience here?
        - User requests non-consumable capacity.
        - Each resource type may have a minimum request?
        - What to do if a request or limit is not defined?
    - Describe concrete use cases.
- ConsumedCondition field for refinement such as predefined blocks, minimum/maximum amount per request? Should it be in this KEP scope?

Related to ResourceClaim and Allocation

- AllowShared (request mixing from both shared and non-sharable, request `none`?), RequireShared (need explicit request only for this case?)
    - Related comment:
        - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1962046081
        - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1962051468
        -
    - Instead of the allocation mode, use an additional field not (like allowSharedDevice?).
- Request by amount in allocation mode. (aggregation)
    - New allocation: range, max, min
- `All` field
    - Related comment:
        - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1962053976
    - Alternative to block the share device
- `Limit` field in resource request for this KEP or out of scope?
    - Related comment:
        - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1968393186
- Request, select capacity more than X

Others

- Comments that I would need further clarification whether it is resolved or modification needed
    - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1960400744
    - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1960749767
    - https://github.com/kubernetes/enhancements/pull/5104#discussion_r1962044641
    -

Feb 19

Attendees:
Sunyanan (Pang), John, Morten, Lionel, Eran, Ming, Tatsuhiro (Chiba)

Discussed Points

- ConsumableCapacity field
    - Confusing Capacity and ConsumesCapacity fields and the infinity flag in attributes is confusing too. Idea:
        - We may a flag to determine the device is sharable or not
        - Then capacity further have a flag to tell whether it is consumable or not (only sharable device can have capacity that have this flag set to true)
        - With the above change, no need for infinity flag to help determine the sharable device
    - Add field to refine the capacity as blocks, field to define minimum capacity to request for share (some device may not allow too small piece of share)
- ResourceClaim's resource request
    - Burstable resource request? Shall we make it available for both request and limit? Idea:
        - If we are going to support, we check consumable with the request field. If the request field is not set but the limit is set, use the limit. The task of handling burstability of the device should be for an individual driver (as what the CPU manager works on). Same as node resource request, limit and request are usually set as the same number for the devices that do not support a burstable use case.
    - Shared allocation mode may not be explicitly needed. Idea:
        - Admin can predefine shareable and non-sharable devices explicitly with the device class.
          Still, if admin defines both in the same device class. Shall we allow users (who create the resource claim) to specify a selector or a constraint to choose sharable or non-sharable devices.
            - If sharable is an attribute, users can specify it with a selector.
            - If sharable is a device defined field, we may have a specific constraint flag in the request to specify accordingly.
- Aggregation features across devices such as to have aggregated memory from the same NUMA (John is working on in a separate enhancement)

Feedbacks for Beta

Bugs

- https://github.com/kubernetes/kubernetes/issues/133705
    - CDI spec not defined
    - Cannot schedule next device
      WIP: https://github.com/kubernetes/kubernetes/issues/133705
    -

Enhancements

- Move structured type to framework pkg:
  https://github.com/kubernetes/kubernetes/issues/133161 [From KEP alpha discussion]
  → Assigned to @nmn3m https://github.com/kubernetes/kubernetes/pull/134404
- Advance scheduling
    - If any resource reaches the cap, allocate the rest resource with the full remaining resource
      For example, core remains 20% and memory remains 40Gi, request for 20% core, 20Gi, should the allocation allocates (20%, 40Gi) or (20%, 20Gi) as-is
      [by Pang, Aug 31]
    - Sort device from less resource available first or sort from most resource available first [From KEP alpha discussion]
    - For infinite device (allow multiple allocation, no capacity), there should have a logic to select pick or not pick to the device that has been already allocated once [by John Sep12]
        - *[Additional thought] we may further consider the condition to pick the device with more or less than N times allocation* (if checked only allocated once, it could be binary flag but if check N number, it may utilize CEL selector)

- Distinct more than one attribute, if any attribute in the list is the same, reject or if all attributes in the list are same, reject? (currently former)

- Increase test coverages: (ref: https://github.com/kubernetes/enhancements/issues/5075#issuecomment-3195511060) [by Patrick]
    - WIP: https://github.com/sunya-ch/kubernetes/tree/consumable-capacity-testing-update
    - `consumable-capacity-multi-allocatable-device-without-capacity-without-capacity-request [suggested by John]`

Unit Testing

Allocating Test

(structured/internal/allocating_test.go)

Prefix: *consumable-capacity-*

Slice

- Single device
  - AllowMultipleAllocations=true (*multi-allocatable-device*)
    - With capacity
      - With sharing policy (*with-policy*)
      - Without sharing policy (*without-policy*)
    - Without capacity (*without-capacity*)
  - AllowMultipleAllocations=false (*dedicated-allocatable-device*)
- Multiple devices
  - All AllowMultipleAllocations=true (*multi-allocatable-devices*)
  - Mixed (*mixed-allocatable-devices*)

Claim

- Single claim
  - No previous allocation
  - With previous allocation
- Multiple claims
  - No previous allocation
  - With previous allocation
- With capacity request
  -
- Without capacity request

Test case	Description	Slices	Class	Claims	Expected
`consumable-capacity-multi-allocatable-device-without-capacity-without-capacity-request`	Shared	AllowMultipleAllocations = true No capacity			
