Design Doc: Syncing of Arbitrary Resources Between Provider and Consumer Clusters

Introduction

Our goal is making kube-bind support synchronization of arbitrary resources (identified via to-be-defined API fields) between provider and consumer cluster (potentially in any direction).

This is an informal doc to drive the discussion about the changes to the kube-bind API needed to reach the aforementioned goal.

By “arbitrary resources” we mean any API resources that aren’t themselves the provisioned services, but that are related to them. For example, a PostgreSQL resource exported by a PostgreSQL provider doesn’t qualify as it is itself the provisioned service. On the other hand, a K8s API secret that stores credentials to access that PostgreSQL resource does qualify.

Unless otherwise specified, throughout this doc we use “service” to refer to any software (e.g. a PostgreSQL database) that a provider provisions and operates on a remote cluster on behalf of a user, rather than a K8s API Service.

Use Cases

1. [Certain] Sync resources from the provider cluster to the consumer cluster.
    - Example: as part of provisioning a service *Svc*, the provider creates a secret *Sec* with the connection details to *Svc*. *Sec* must be synced to the consumer cluster so that apps can mount it to connect to *Svc*.
2. [Potential] Sync resources to the provider cluster from the consumer cluster.
    - Example: a consumer provisions a remote service *Svc* via a provider, and wants certain credentials to be present in *Svc* (e.g. she wants to authenticate with a password that she defines). She creates a secret *Sec* holding the credentials in her cluster, *Sec* must then be synced to the provider cluster so that the credentials can be configured in *Svc*.
3. [Potential] The consumer can choose between use cases 1 and 2.
    - Example: when provisioning a service *Svc*, the consumer can choose whether she provides the credentials for *Svc* via a secret, or whether the provider must define and share with her the credentials to *Svc*.
4. [Potential] Given a resource, sync its different subresources in opposite directions.
    - Example: a consumer creates a resource *R* which is related (in any possible way) to a provisioned service *Svc*. *R*’s spec should be synced from consumer cluster to provider cluster, while *R*’s status should be synced from provider cluster to consumer cluster. Or the vice versa: *R* is created in the provider cluster, its spec should be synced to the consumer while its status should be synced from consumer to provider.

5. [Potential] The service provider can restrict what the consumer is allowed to do in her own cluster.
    - Example 1: as part of provisioning a service *Svc*, the provider creates a secret *Sec* with the connection details to *Svc*. The provider can ask the consumer to relinquish her permission to delete *Sec*.
    - Example 2: a consumer has a contract to only create 5 PostgreSQL service instances via a provider. Therefore, the provider creates a resource quota enforcing the limit in the consumer's cluster, and asks the consumer to relinquish the right to update and delete that quota.
6. ADD MORE IF YOU HAVE THEM

Requirements/Constraints

1. [Certain] Only the resources a user agreed to sync are synchronised. This means that the user must have sole control over the accepted state of permission claims.
2. [Certain] The API fields that control the syncing of resources should be compatible with kcp, which already has API fields for that (here, here, and here).
3. [Certain] For every resource to sync, the API should express which verbs the service provider is claiming permission for (or is asking the user to relinquish). These verbs implicitly determine the syncing direction.
    - Example: if an APIExport claims the CREATE verb on a resource, the provider will create it in its cluster, and then it will be synced in the consumer cluster. kcp already started thinking about this although it's not in the API yet, so we have to be compatible with what they do in this regard.
4. [Potential/Nice to have] The konnector only sees the resources in the provider cluster that it needs to see.
    - Example: given a mangodb resource called X and a secret *X-mangodb-secret* with the connection details for the mangodb (created by the provider), the konnector needs to sync that secret. Ideally, it should have access only to that precise secret (and those of other mangodbs), and not to other secrets.

API Examples

"Provider" and "Consumer" both need write permissions

Mutual exclusivity on write use cases between consumer and provider must be enforced. The current model of "only consumer or provider can do writes" is too coarse grained, a more detailed model is needed.

Here, claimed verbs can refine more details on what the provider needs based on i.e. subresource, fields of the claimed resource, or on certain labels values.

```
permissionClaims:
 - resource: widgets
```

```
  group: kcp-dev
  resourceSelector:
    - name: foo
      namespace: bar
  verbs:
    claimed:
    - verb: update
    # these constraints are not very meaningful in restrictTo
    subresources:
    - status
    fields:
    - spec.nodename # field that can be updated
    labels:
    - kcp.dev/*
    - foo/bar
    restrictTo:
    - update # authorizer would reject updates on status subresource,
             # on spec.nodename, and updates on label values as
             # specified in claimed verbs
```

“Foo” sends secret from provider to consumer

The APIServiceBinding Object copies the permission claims from the APIServiceExport, but is the source of truth for the “accepted” state.

On the consumer cluster:

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceBinding
metadata:
  name: Foo
  namespace: "binding-test"
spec:
  # ... other, unrelated fields
  permissionClaims:
      - resource: Secret # required
        group: "" #default="" - core api
        all: false # true or false. If true, no resourceSelector can be
provided
        # required
        resourceSelector:
          name: "*"
          namespace: "*"
        # required
        # permissions that allow writing can only be in either "claimed"
```

```
or "restrictTo"
        Verbs:
          # verbs the provider can issue
          provider: ["*"]
          # verbs the consumer can issue
          consumer:
            - get
            - list
            - watch
        state: "Accepted"
```

On the provider cluster:

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceExport
metadata:
  name: Foo
  namespace: "binding-test"
spec:
  # ... other, unrelated fields
  permissionClaims:
      - resource: Secret #required
        group: “” #default=""
        all: false
        resourceSelector:
          name: "*"
          namespace: "*"
        verbs:
          # verbs the provider can issue
          provider: ["*"]
          # verbs the consumer can issue
          consumer:
            - get
            - list
            - watch
```

“Bar” sends single ConfigMap from Consumer to provider

On the consumer cluster

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceBinding
metadata:
  name: Bar
  namespace: "binding-test"
spec:
  # ... other, unrelated fields
  permissionClaims:
```

```
      - resource: ConfigMap # required
        group: "" #default="" - core api
        all: false # true or false. If true, no resourceSelector can be
provided
        # required
        verbs:
          provider: ["*"]
          consumer:
            - get
            - watch
        # mutually exclusive with "all: true"
        resourceSelector:
          - name: "provider-config"
            namespace: "binding-test"
        state: "Accepted"
```

On the provider cluster:

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceExport
metadata:
  name: Bar
  namespace: "binding-test"
spec:
  # ... other, unrelated fields
  permissionClaims:
      - resource: ConfigMap #required
        group: ""#default=""
        all: false
        verbs:
          consumer:
            - get
            - watch
          provider: ["*"]
        resourceSelector:
          - name: "provider-config"
            namespace: "binding-test"
```

Pattern-Based claims

“Baz” wants to pull a ConfigMap for each “Baz” created. It will only pull ConfigMaps in the same namespace that are named <metadata.name>-config.

In the consumer cluster:

```
kind: APIServiceBinding
metadata:
  name: Baz
  namespace: "binding-test"
```

```
spec:
  # ... other, unrelated fields
  permissionClaims:
      - resource: ConfigMap # required
        group: "" #default="" - core api
        all: false # true or false. If true, no resourceSelector can be
provided
        # required
        verbs:
          consumer: ["*"]
          provider:
            - get
            - watch
        # mutually exclusive with "all: true"
        resourceSelector:
          - name: "{.metadata.name}-config"
            namespace: "{.metadata.namespace}"
        state: "Accepted"
```

On the provider cluster:

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceExport
metadata:
  name: Baz
  namespace: "binding-test"
spec:
  # ... other, unrelated fields
  permissionClaims:
      - resource: ConfigMap #required
        group:"" #default=""
        all: false
        verbs:
          provider:
            - read
            - watch
          consumer: ["*"]
        resourceSelector:
          - name: "{.metadata.name}-config"
            namespace: "{.metadata.namespace}"
```

Associating permissionClaims with Resources

Introduction

In the effort of implementing synchronisation of secondary resources belonging to an APIServiceExport we need to associate the resources to be synced with an APIServiceExport, as well as the direction of the resources, which is identified by the verbs on the provider and consumer clusters.

Inventory

Currently APIServiceExports are created on demand by the backend in response to an APIServiceExportRequest. This is done by getting the CRD and converting it into an APIServiceExport spec, which is then created.
CRDs available for binding are marked via a label, <**kube-bind.io/exported: "true"**>. This label contains no information about the claimed resources. Extending this information via a label is not possible, because labels content is limited by kubernetes.
Annotations may have enough space to store that information, but would make for a poor user experience, because the data is unstructured, and would most likely need to be entered manually.

Proposal

Add a new resource APIServiceExportTemplate which contains a selector for the resource to be exported, as well as information about additional resources to be synchronised (claimed resources).

```
version: "kube-bind.io/v1alpha1"
kind: APIServiceExportTemplate
metadata:
  name: MangoDBs
  # should it be namespaced or cluster global?
spec:
  # required
  APIServiceSelector:
      group: mangobds.example.com
      kind: mangodbs
      # optional, if not provided, all served versions will be exported
      versions:
      - v1
  # optional, if not provided, no claimed resources will be synchronised
  claimedResources:
      - group: ""
        kind: "Secret"
        verbs:
```

```
          provider: ["*"]
          consumer:
            - get
            - list
            - watch
```

The backend will use the list of APIServiceExportTemplates to display the available Api services to the user, and use it as template for creating the APIServiceExports.

Scratch Pad

kube-bind
kcp consumer
kcp provider

Service Provider creates and updates

verbs:
  claimed: ["*"]
  restrictTo: ~~[“get”,”list”,”watch”]~~

Consumer creates, service provider reads Implies upsync
verbs: claimed: [“read”]

Consumer creates, service provider creates+updates Implies upsync or downsync, depending on who created the object
verbs: annotation: kube-bind.io/creator: provider permissionClaims:

Consumer creates, service provider creates+updates

Implies upsync or downsync, depending on who created the object

```
- resource: secrets
  resourceSelector:
    creator: user
  verbs:
    read: true # must be true if creator=user
    create: false # must be false if creator=user
    update:
    - subResource: status
    - field: spec.nodeName
    - annotations: *foo-gmbh.com/*
    delete: true # if no object on provider side, delete user object
- resource: secrets
  resourceSelector:
    creator: provider # implicit
  owner: …
  referenced: …
  verbs:
    read: true # must be true if creator=provider
    create: true # must be true if creator=provider
    update:
    - preserve: spec.nodeName
    delete: true

Provider Foo GmbH wants these permission:
   -  read and delete user-owned objects, and update status where
      name: foo*
      labels:
        a: b*
   -  create objects, delete and update preserving:
      field: spec.nodeName
      annotations: *.provider.com/*
Allow [N,y]?


permissionClaims:
- resource: secrets
  resourceSelector:
    creator: user
  verbs:
    read: true # must be true if creator=user
    create: false # must be false if creator=user
    update:
```

Consumer creates, service provider creates+updates

Implies upsync or downsync, depending on who created the object

```
    - subResource: status
    - field: spec.nodeName
    - annotations: *foo-gmbh.com/*
    deleteByProvider: true # if no object on provider side, delete user
object
- resource: secrets
  resourceSelector:
    creator: provider # implicit
  owner: …
  referenced: …
  read: true # read user object in addition to provider object
  create: false/true # provider can create objects
  userOwned: # upsync implicit
    preserve:
    - field: spec.nodeName
    deleteByProvider: true/false
    deleteByUser: false/true
```

```
permissionClaims:
  resourceSelector:

  - resource: secrets
    resoureSelector:
      annotations:
      - kube-bind.io/creator: user
    claimed:
      verb:
      - update
      subresources:
      - status

  - resource: secrets
    resoureSelector:
      annotations:
      - kube-bind.io/creator: user
    claimed:
      verb:
      - get
      - list
      - watch

  - resource: secrets
    name: provider-secret
    claimed:
```

Consumer creates, service provider creates+updates Implies upsync or downsync, depending on who created the object
verb: [update]
As a provider I want to be able to update a status subresource if a resource is created by a user and must be denied updates on any other fields. As a consumer I want to be able to create a resource and forbid updates on any other fields by the provider other than the status subresource. Conversely ... As a provider I want to be able to create a resource and forbid (override) updates on any field by the user. As a user I want to be able to read resources created by the provider.

Final Schema Scratch Pad

~~Proposal #1~~

~~Provider Foo GmbH wants these permission:~~
~~- read and delete user-owned objects, and update status where~~
~~name: foo*~~
~~labels:~~
~~a: b*~~
~~- create objects, delete and update preserving:~~
~~field: spec.nodeName~~
~~annotations: *.provider.com/*~~
~~Allow [N,y]?~~

~~spec:~~
~~claimedResources:~~
~~#group: ""~~
~~# version: v1~~
~~# resource: Y~~
~~selector:~~
~~- name: "foo*"~~
~~labels:~~
~~"a": "b*"~~
~~owner: consumer~~
~~# Exceptions to the resource synchronization. They can be used~~
~~# to make sure synchronization does not perform an action like delete.~~

```
    # Updates are more fine grained as synchronization may go in different
    # directions for different fields
    # If multiple claims target the same resource, only one can claim
    # an action. The others must exclude it.
    excludes:
      delete: true # consumer does not claim delete
      update:
        - field: "status" # consumer will not update the status
  - #group: ""
    # version: v1
    # resource: Y
    selector:
      - name: "foo*"
        labels:
          "a": "b*"
    owner: provider
    excludes:
      - create: true # provider will not create the resource in the user cluster
      - update:
          - field: "spec" # provider will not update the spec
  - # group: ""
    # version: v1
    # resource: X
    selector:
      name: "*"
    direction: down
    excludes:
      update:
        - field: spec.nodeName
        - annotation: "*.provider.com/*"
```

Superseded by proposal #5

Proposal #2

```
Provider Foo GmbH wants these permission:
   -  read and delete user-owned objects, and update status where
      name: foo*
      labels:
        a: b*
   -  create objects, delete and update preserving:
      field: spec.nodeName
      annotations: *.provider.com/*
Allow [N,y]?
```

Claimed resources are specified with a top-level “claims” field having the following fields:

- **group/resource**: specifies the target resource

- **selector**: narrows down the set of resources being claimed. Allowed filters are name, namespace, and labels

- **owner**: defines the authoritative source of truth for the claimed resource. Two cases:
    a. **owner**: **consumer**
        - implies syncing from consumer to provider
        - if consumer creates claimed resource: it is synced to the provider
        - if provider creates claimed resource: it is being ignored and potentially overwritten if consumer creates it.
    b. **owner**: **provider**
        - implies syncing from provider to consumer.
        - opposite to the above:
        - if provider creates claimed resource: it is synced to the consumer
        - if consumer creates claimed resource: it is being ignored and potentially overwritten if provider creates it.

- **overrides**: declare allowed mutations the *other* party can do:
    a. **deletion**: if set to true, the other party is allowed to delete the claimed resource.
    b. **fields**: fields that can be overwritten by the other party, i.e. spec.nodename
    c. **subresource**: subresources that can be overwritten by the other party, i.e. status
    d. **annotations**: annotations having keys specified here can be overwritten by the other party
    e. **labels**: labels having keys specified here can be overwritten by the other party

```
claims:
- resource: secrets
 group: v1 # optional
 owner: consumer # implies syncing from consumer to provider

 selector:
 - namespace: foo # optional, if ommitted, all secrets in namespace "foo" are claimed
   name: consumer-secret # optional, if ommitted, all secrets named "consumer-secret" are
claimed
   labels: # optional
   - a: b # all secrets having the label "a=b" are being claimed

 overrides: # optional, if ommitted, any changes done by the PROVIDER are ignored
            #
   deletion: true # default: false, provider is allowed to delete foo/consumer-secret
secrets having the label "a=b"
```

```
   labels:
   - key.provider.com # provider can set and override labels with the key
"key.provider.com"
   annotations:
   - key.provider.com # provider can set and override annotations with the key
"key.provider.com"
   subresources:
   - status # provider can set and override the subresource "status"

- resource: secrets
 group: v1 # optional
 owner: provider # implies syncing from provider to consumer

 selector:
 - namespace: foo # optional, if ommitted, all secrets in namespace "foo" are claimed
   name: provider-secret # optional, if ommitted, all secrets named "provider-secret" are
claimed
   labels: # optional
   - b: c # all secrets having the label "b=c" are being claimed

 overrides: # optional, if ommitted, any changes done by the CONSUMER are ignored
   deletion: false # default: false, consumer is not allowed to delete foo/provider-secret
secrets having the label "b=c".
                   # if the consumer deletes, the resource will be synchronized from
provider to consumer.tp
   fields:
   - spec.nodename # consumer can set and override the field spec.nodename
```

Proposal #3 (Simplified version)

```
spec:
 claimedResources:
   - target:
       <resource-identifier>
     selector:
       <selector-logic>
     provider:
       <provider-logic>
     consumer:
       <consumer-logic>
```

Example: to downsync a secret created by the provider

```yaml
spec:
 claimedResources:
   - target:
       apiVersion: v1
       kind: Secret
     selector:
       matchLabels:
         app.kubernetes.io/instance: '{.metadata.name}'
         app.kubernetes.io/managed-by: kubedb.com
         app.kubernetes.io/name: mongodbs.kubedb.com
     provider:
       isOwner: true
       verbs:
         - "*"
     consumer:
       downSync: true
       verbs:
         - get
         - list
         - watch
```

~~Proposal 4~~

~~Provider Foo GmbH wants these permission:~~
~~- read and delete user-owned objects, and update status, spec.nodeName, annotations *.provider.com/* where name: foo* labels: a: b*~~
~~- create objects, delete and update preserving field spec.nodeName with annotations *.provider.com/*~~
~~Allow [N,y]?~~

```yaml
permissionClaims:
- resource: secrets
  group: “”
  version: v1
  selector:
    # creator decides which objects this claim is about: “User” means
    # objects that are pre-existing, “Provider” means objects that are
    # created (down-synced) from the provider. Defaults to “User”.
    creator: User # defaulted
    name: foo*
```

```
    labels:
    - a: b*
  verbs:
  # read is always true. It means that the provider will be able to read
    # these objects. The kube-bind konnector will sync these objects to
    # the provider side. Defaults to true.
    read: true # must always be true, is defaulted if verbs is missing
    # create determines whether the kube-bind konnector will sync matching
    # objects from the provider side down to the consumer cluster. Must be
    # false if selector.creator is set to User.
    create: false # true doesn't make sense, hence this is enforced
    # update lists a number of claimed permissions for the provider.
    # "field" and "preserving" are mutual exclusive.
    update:
      # subResource must be either unset or empty, or "status". If unset
      # or empty and annotation is not set, this item applies to
      # non-metadata, non-status fields. If set to "status", this item
      # applies to status fields. The permission can further restricted
      # via "field" and "preserving".
      subResource: status
    - # field is a JSON path to a field that the provider will be able to
      # mutate. This means any value on the consumer side is overwritten
      # with the provider intent. The JSON path must not reference
      # metadata or status fields.
      field: spec.nodeName
    # delete set to true means that the provider can delete a previously
    # synced object and the kube-bind konnector then will also delete the
    # object in the consumer cluster. Deletion on the provider side is
    # delayed until the konnector makes sure the consumer object is also
    # deleted.
    delete: true
- resource: secrets
  group: ""
  apiVersion: v1
  selector:
    # creator decides which objects this claim is about: "User" means
    # objects that are pre-existing, "Provider" means objects that are
    # created (down-synced) from the provider. Defaults to "User".
    creator: Provider
  # annotations specifies keys or key glob patterns with * matching
  # everything other than "/" that are synced to the provider side (via
  # "read") or synced to the consumer side (via "write").
  annotations: # TODO: maybe need more flexible semantics
  - read: ...
  - write: *foo-gmbh.com/*
  # equivalently to annotations
  labels: ...
```

```
  annotations:
  verbs:
    read: true # must always be true, is defaulted if verbs is missing
    create: true # new objects can be created (there could be old objects
                 # too even with false here)
    update:
      - # preserving is a JSON path of a non-metadata field. If subResource
        # is not specified, this means that consumer-side non-metadata,
        # non-status values can be mutated by overwriting them with the
        # provider values. If subResource is "status", the same applies for
        # consumer-side non-metadata, non-status values. Preserved and
        # explicitly specified field values must not overlap.
      preserving: spec.nodeName
    # delete set to true means that the provider can delete a previously
    # synced object and the kube-bind konnector then will also delete the
    # object in the consumer cluster. Deletion on the provider side is
    # delayed until the konnector makes sure the consumer object is also
    # deleted.
    delete: true
```

Supersed by proposal #6

Proposal #5

Provider Foo GmbH wants these permission:
read and delete user-owned objects, read spec.nodeName and update status where
name: foo*
labels:
  a: b*
create objects, delete and update preserving:
    field: spec.nodeName
    annotations: *.provider.com/*
Allow [N,y]?

```
spec:
  claimedResources:
    - #group: ""
      # version: v1
      # resource: Y
      selector:
        - name: "foo*"
        - labels:
            "a": "b*"
      owner: consumer
      # includes are the operations that the claimed resource reserves. Updates are more fine
grained
      # and allow for selecting individual fields and subresources.
```

```
    # each action can only be performed in one direction. E.g if status updates are allowed
both
    # up and down, we have an invalid configuration
    # selectors are ORed, meaning if one selector matches, the selected portion is synced
    includes:
      create: true # consumer does not claim delete
      update:
        - field: "spec" # consumer will not update the status
  - #group: ""
    # version: v1
    # resource: Y
    selector:
      - name: "foo*"
      - labels:
          "a": "b*"
    owner: provider
    includes:
      delete: true # provider will delete the resource
      update:
        - subresource: "status" # provider will not update the spec
  - # group: ""
    # version: v1
    # resource: X
    selector:
      name: "*"
    direction: down
    includes:
      update:
        - field: "!spec.nodeName"
        - annotation: "!*.provider.com/*"
  - # group: ""
    # version: v1
    # resource: X
    selector:
      name: "*"
    direction: up
    includes:
      update:
        - field: "spec.nodeName"
```

Proposal #6

```
permissionClaims:
- resource: secrets

  group: “”
  version: v1

  # selector selects which resources are being claimed.
  # If unset, all resources across all namespaces are being claimed.
  selector:An entry of "*" anywhere in the list means all object names of
the group/resource within the "namespaces" field are claimed.
    # Wildcard entries other than "*" and regular expressions are
currently unsupported.
    # If a resource matches any value in names, the resource is considered
matching.
    # Default is "*".
    names:
    - foo*
    labelSelectors:
    - a: b*
      c: d
    - ...
    fieldSelectors:
    - ...TODO: specify what this means technically: annotations,
finalizers
    # +optional
    owner: Provider/Consumer  required: true

  # read is always claimed.
  # By default no labels and annotations are read.
  # Reading of labels and annotations can be claimed optionaly by adding
  # labels and annotations items.
  # If labels of consumer owned objects that are set by the consumer are
  # read, labelsOnProviderOwnedObjects and
annotationsOnProviderOwnedObjects can be set.
  # +optional
  # +kubebuilder:default={}
  read:
    # labels is a list of claimed label key wildcard patterns
    # that are synchronized from the consumer cluster to the provider on
    # objects owned by the consumer.
    labels:
    - pattern: x-provider-*
      prefix: foo- # +optional, future
    # labelsOnProviderOwnedObjects is a list of claimed label key wildcard
    # patterns that are synchronized from the consumer cluster
```

```
  # to the provider on objects owned by the provider.
  labelsOnProviderOwnedObjects:
  - pattern: x-provider-*
    regex: # +optional, future
    prefix: foo- # +optional, future

  annotations: # analogous
  annotationsOnProviderOwnedObjects: # analogous

# only for owner Provider
#
# create determines whether the kube-bind konnector will sync matching
# objects from the provider cluster down to the consumer cluster.
#
# +optional
create:
  # replaceExisting means that an existing object owned by the consumer
  # will be replaced by the provider object.
  #
  # If set to false, and a conflicting consumer object exists, it is not
  # touched.
  replaceExisting: true

# donate set to true means that a newly created object by the provider
# is immediately owned by the consumer. If false, the object stays in
# ownership of the provider.
#
# create.autoDdonate and autoAdopt are mutually exclusive.
autoDonate: true

# adopt set to true means that objects created by the consumer are
# adopted by the provider, i.e. the provider will become the owner.
#
# create.autoDonate and autoAdopt are mutually exclusive.
autoAdopt: true

# onConflict determines how the conflicts between objects on
# the consumer cluster and provider cluster will be resolved.
onConflict:
  # recreateWhenConsumerSideDeleted set to true (the default) means
  # the provider will recreate the object in case the object
  # is missing on the consumer side, but has been synchronized before.
  #
  # If set to false, deleted provider-owned objects get deleted
  # on the provider side as well.
  #
  # Even if the consumer mistakenly or intentionally deletes the object,
```

```
    # the provider will recreate it. If the field is set as false,
    # the provider will not recreate the object
    # in case the object is deleted on the RecreateWhenConsumerSideDeleted
    # cluster.
    #
    # Defaults to true.
    recreateWhenConsumerSideDeleted: true # deletion conflict

  # update lists which updates to objects on objects on the consumer
  # side are claimed.
  #
  # +optional
  update:

    fields:
    - spec.status
    preservings:
    alwaysRecreate: true
labels:
    - pattern: …
      stripPrefix: foo- # +optional, future

    # labelsOnConsumerOwnedObjects is a list of claimed label key wildcard
patterns
    # that are synchronized from the provider to the consumer for objects
    # owned by the consumer.
    #
    # By default, no labels are synched.
    labelsOnConsumerOwnedObjects:
    - pattern: …
      stripPrefix: foo- # +optional, future
      cel: # +optional, future future

    annotations: # analogous
    annotationsOnConsumerOwnedObjects: # analogous
```

Open Questions/General Thoughts

Expressiveness of resource claims

How do we express resource claims? So far in kcp namespace and name templates are used. Might we want label selectors or even field selectors (e.g. sync the secrets whose ownerRef points to this MangoDB instance).

Have minimal permission claims?

Users can accept/deny each permission claim individually. But denying some claims might make the service unusable/broken. Should the API express what claims MUST be accepted to have a functional service instance vs those that are optionals?

Restricting konnector's access to resources to be synced

- RBAC restrictions: probably they are not expressive enough
- List&Watch filtering capabilities: they are also not expressive enough

Given the aforementioned restrictions, can we actually implement this? We'd have to write a proxy that lives in the provider cluster and implements list and watch with more fine-grained RBAC and filtering capabilities.

Namespace restrictions:

Kube-Bind maps namespaces in the consumer cluster to namespaces in the provider cluster. Should permissions only be granted to mapped namespaces?

Upsides

- stronger isolation between consumers

Downsides

- Does not work for non-namespaced resources
- we could still filter on the konnector instead
- needs one watch per namespace instead of a global watch