SIG-Service Catalog Deep-Dive

Jonathan Berkhahn - jaberkha@us.ibm.com - @jberkhahn

Agenda

- Open Service Broker API
- Kube-Service Catalog Architecture
- Design Challenges
- Recent Features
- Future Plans

Open Service Broker API

- Specification of an API to allow automated deployment, management, and use of services
  - Cloud-native apps require resources such as stable storage, etc
  - App developers shouldn’t have to care about how the service is managed
  - OSB API abstracts all of this away

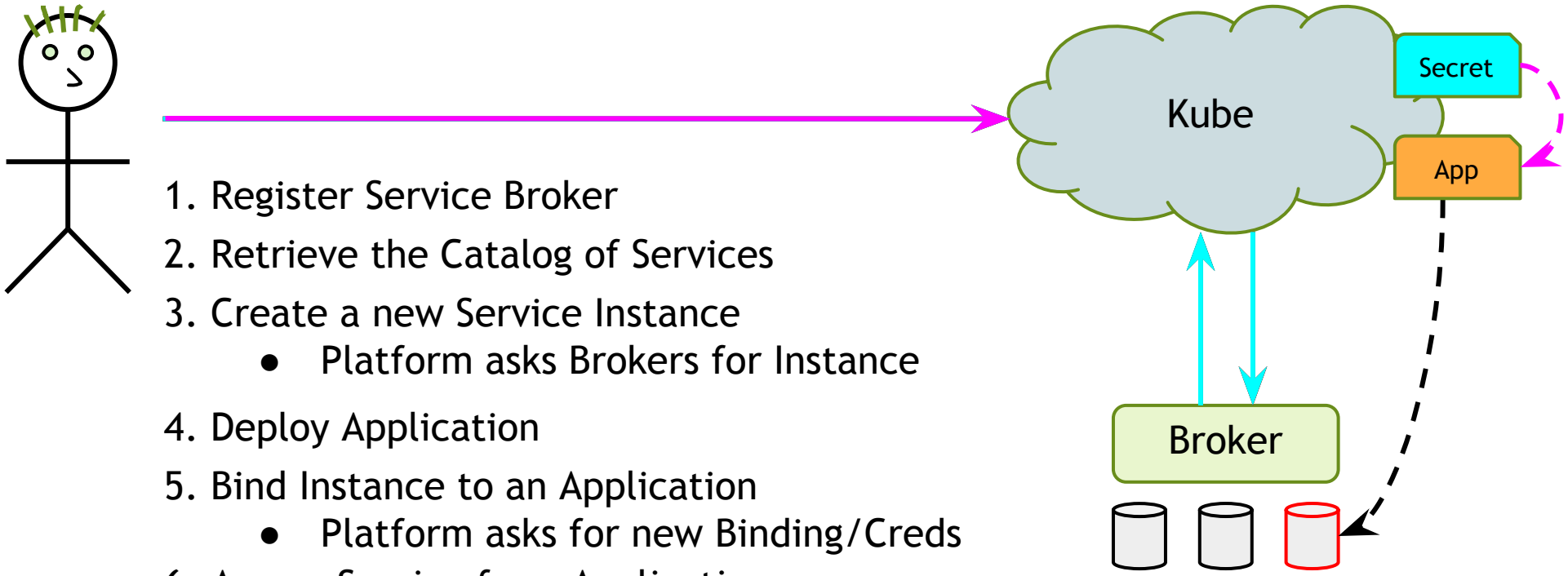
- Client side implemented by Service Catalog
  - managed through custom resource types

- Server side implemented by service provider as a ‘broker’
  - get catalog endpoint
  - provision service endpoint
  - bind service endpoint

Service Catalog Resource Types

- ClusterServiceBroker
  - A server running somewhere that offers various services, e.g. MySQL Broker
- ClusterServiceClass
  - A category of services offered by a Broker, e.g. MySQL Databases
- ServicePlan
  - A specific type of a Service that a Broker offers, e.g. 100 MB MySQL Databases
- ServiceInstance
  - A single instantiation of a Service/Plan, e.g. Jonathan's 100 MB MySQL Database
- ServiceBinding
  - A unique set of creds to access a specific Instance, e.g. username/password for Jonathan's 100 MB MySQL Database

The Magic



1. Register Service Broker
2. Retrieve the Catalog of Services
3. Create a new Service Instance
    - Platform asks Brokers for Instance
4. Deploy Application
5. Bind Instance to an Application
    - Platform asks for new Binding/Creds
6. Access Service from Application
    - Using Creds from Binding Secret

Design Issues - API Aggregation

- CRDs didn’t exist yet, TPRs were buggy
- Didn’t want Service Catalog to have access to the main etcd in Kube for security reasons
- Solution: implement our own apiserver, use API aggregation to hook it in
- Allows normal interaction, i.e. `kubectl create -f serviceinstance.yml`

Design Issues - GUIDs vs. Names

- Kube names are fixed
- OSB API resources have mutable names, and fixed GUIDs
- Service Catalog types use OSB API GUID as the name, and have a mutable ExternalName field
- svcat cli tool alleviates this pain by referencing human-readable ExternalNames as much as possible

```
metadata:
   name: 12kbac-adad12kbasd // from the broker; immutable
   uid: affd6f9b-defe-11e8-87bb-0242ac110007 // generated by Kube
spec:
   externalName: mysql // from the broker; can change
   externalID: 12kbac-adad12kbasd // same as metadata.name
```

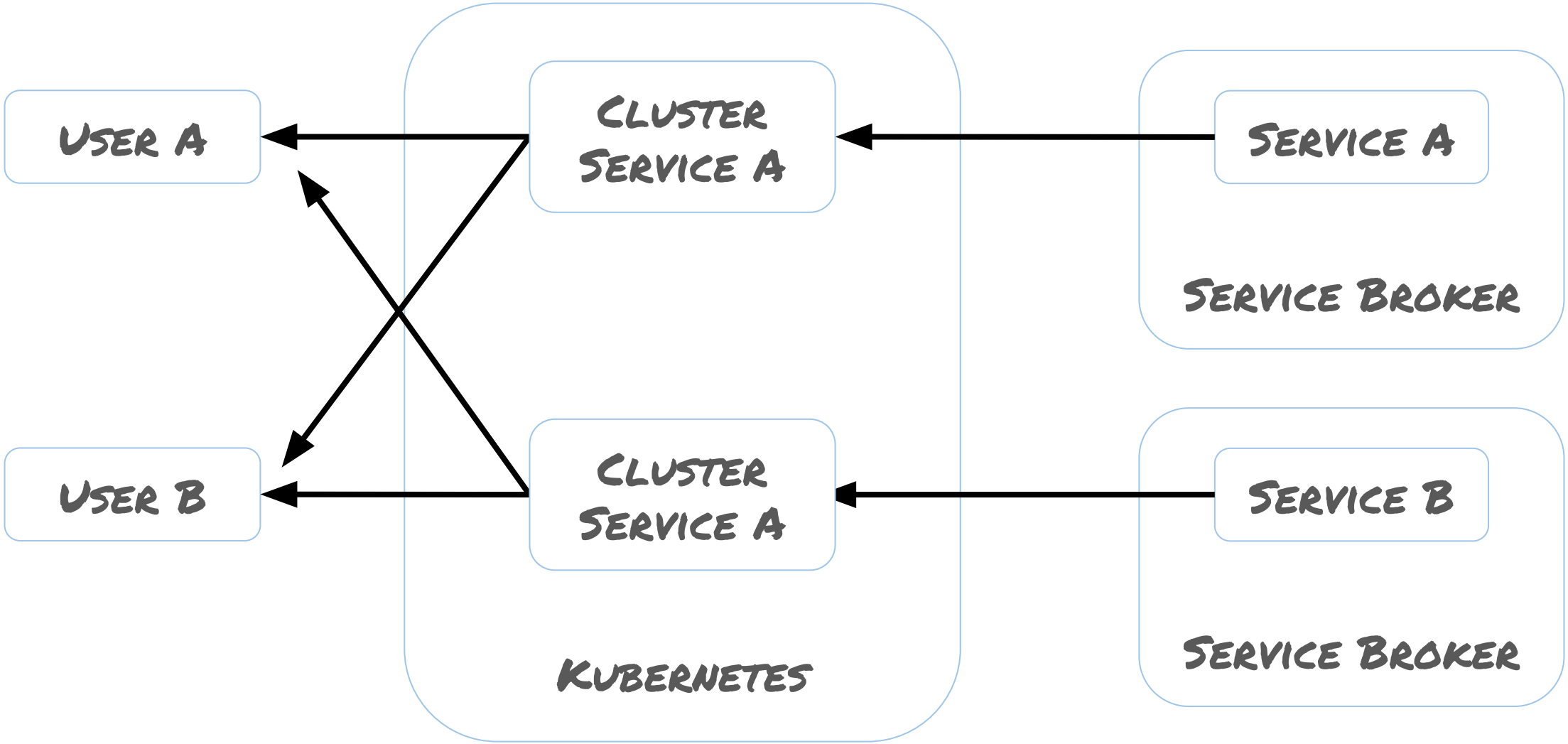
Design Issues - Broker Synchronization

- Kube isn’t the sole source of truth
- Declarative control flow allows users to manipulate Service Catalog resources at-will
- Broker can still reject these changes
- Ongoing work to fix these sync issues

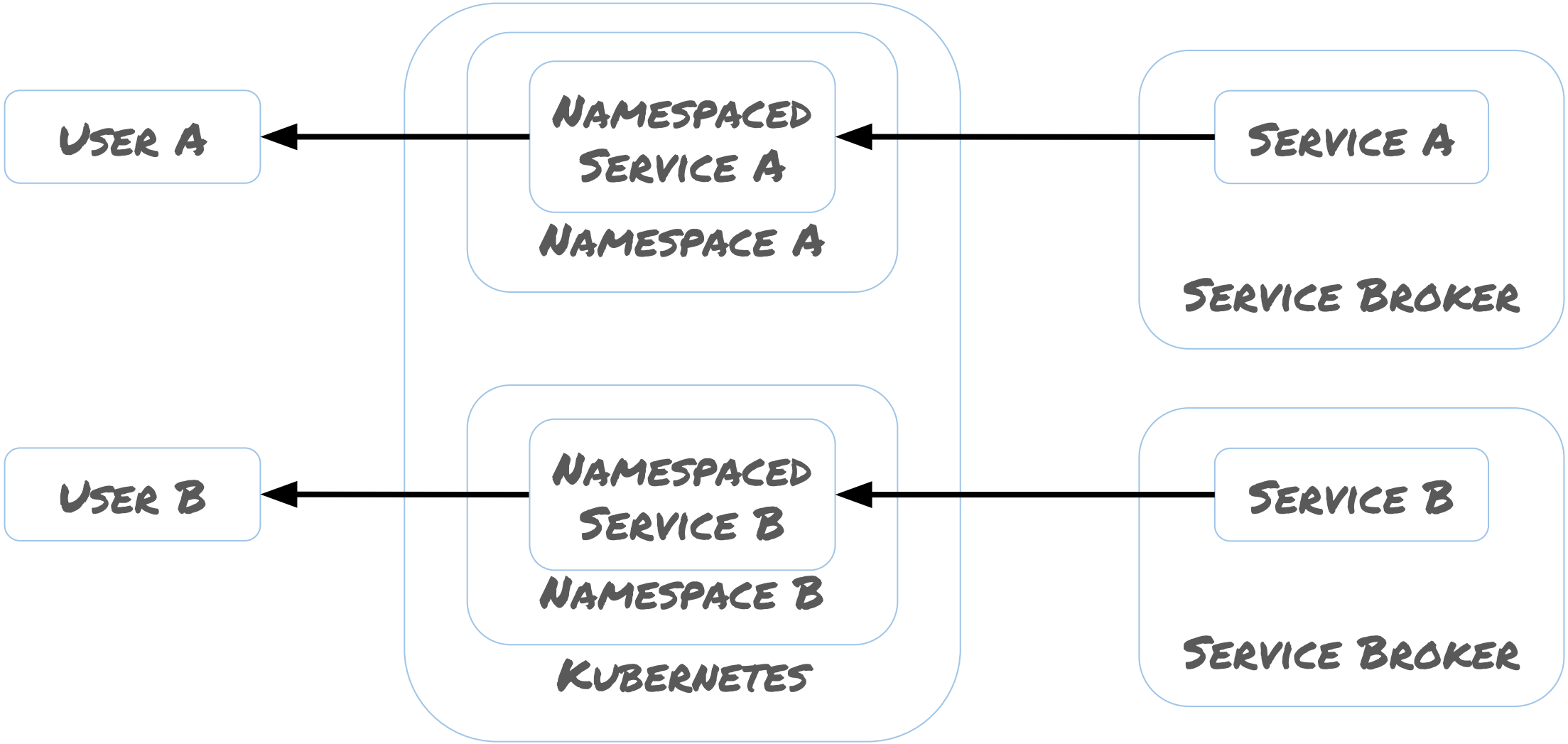
Recent Features - Service Curation

- Original resource types available cluster-wide
- Allow Kube operators and users to grant selective access to service brokers/classes/plans
    - Namespaced brokers
    - Catalog Restrictions

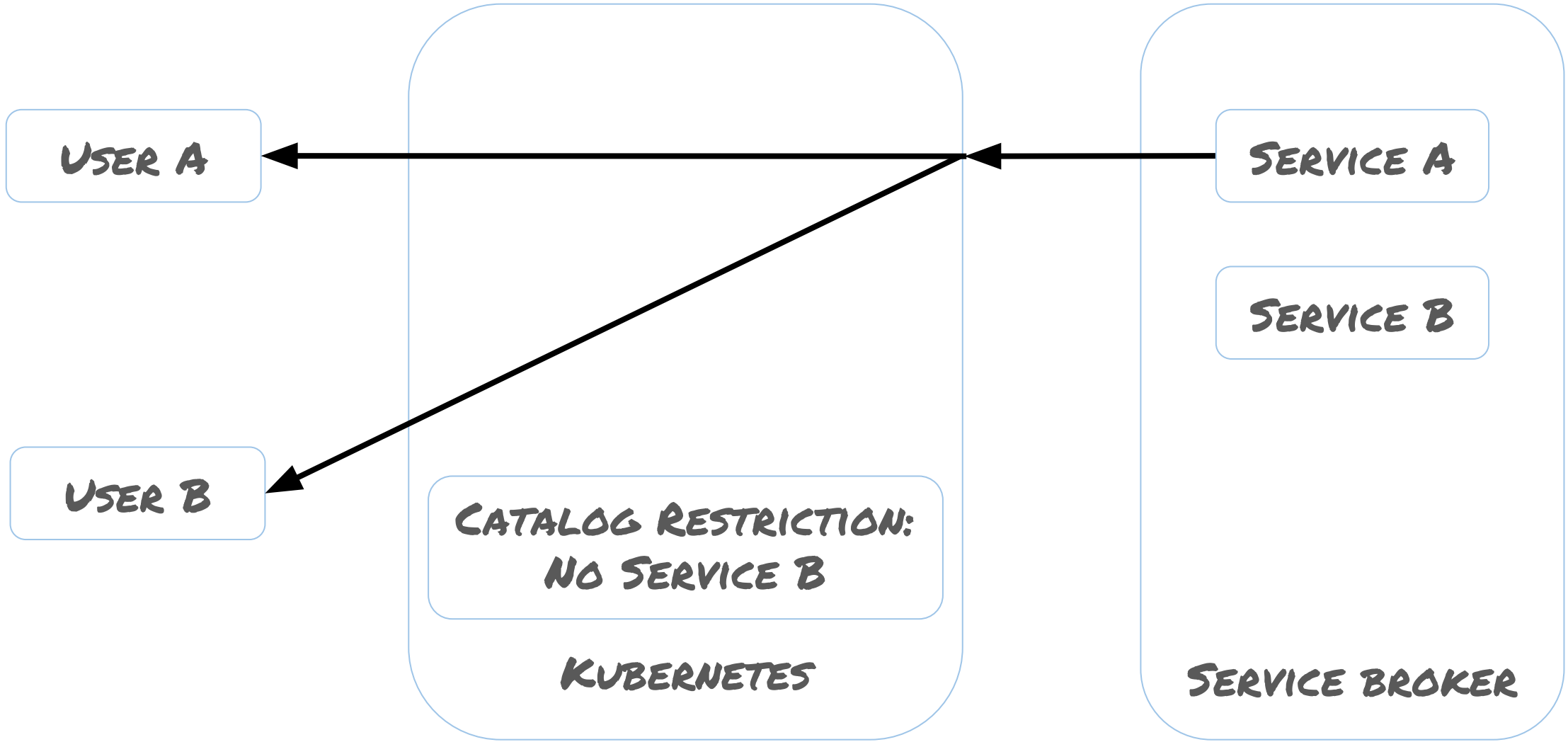
Cluster Service Brokers



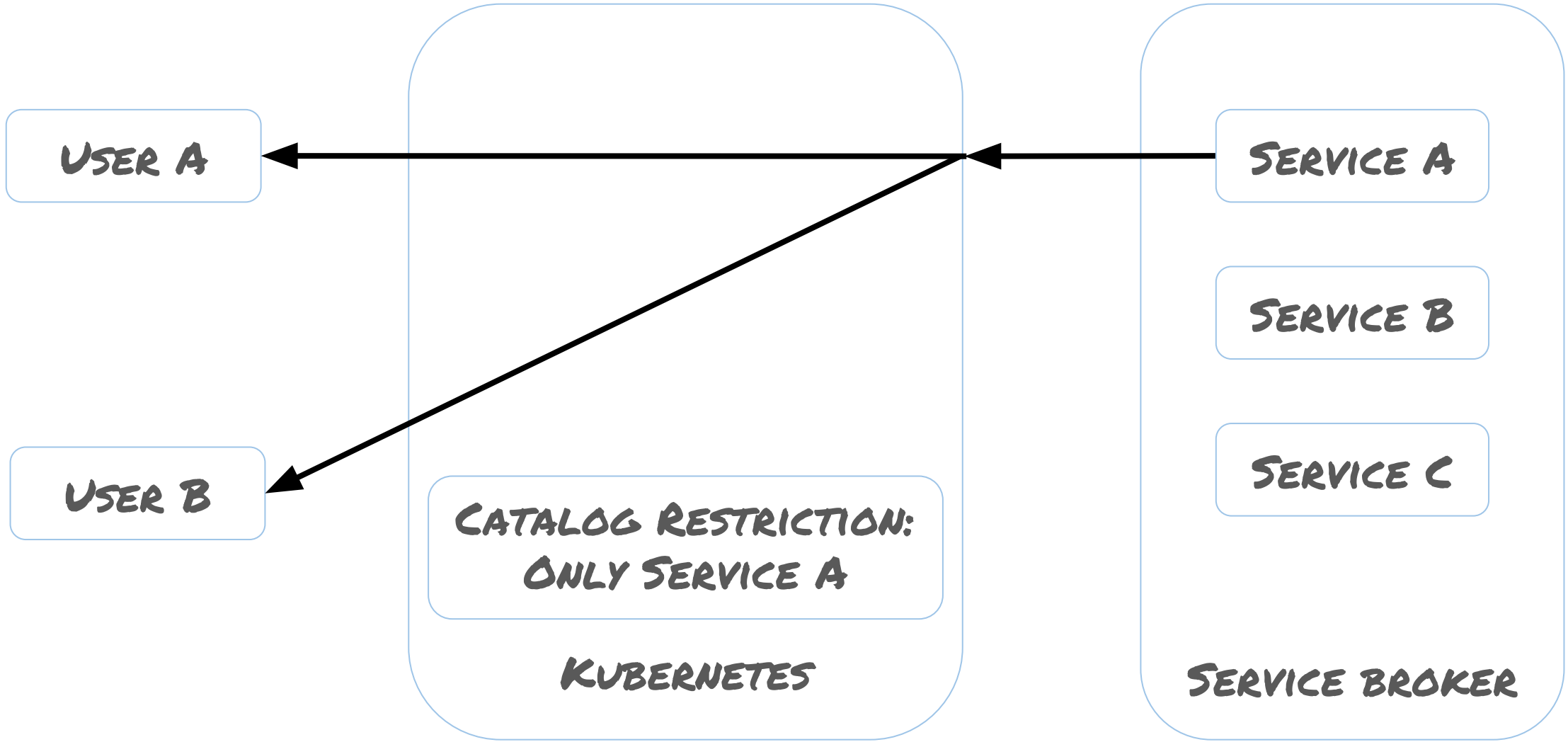
Namespaced Service Brokers



Catalog Restrictions



Catalog Restrictions



svcat

- CLI to allow for CRUD of Brokers, Classes, Plans, Instances, Bindings
- Additional commands for non k8s-like features, such as viewing the marketplace of services

Future Plans

- Investigating use of CRDs to replace aggregated API server
- Improve synchronization between Kube and brokers
- Default parameters to allow for operator creation of custom classes/plans
- Pod presets
- Coming up on 1.0.0

Questions

More information:

- https://svc-cat.io
- https://github.com/kubernetes-incubator/service-catalog
- https://www.openservicebrokerapi.org/