



Polymorphic Reconcilers in Kubernetes

snichols@vmware.com

@n3wscott ( ,  , )

mattmoor@vmware.com

@mattomata ()

@mattmoor ( , )

Background

Custom Resource Definitions (aka CRDs) are leading to an explosive expansion of the Kubernetes type system. Previously to reason about compute resources, you could juggle a handful of concepts.

Deployment

DaemonSet

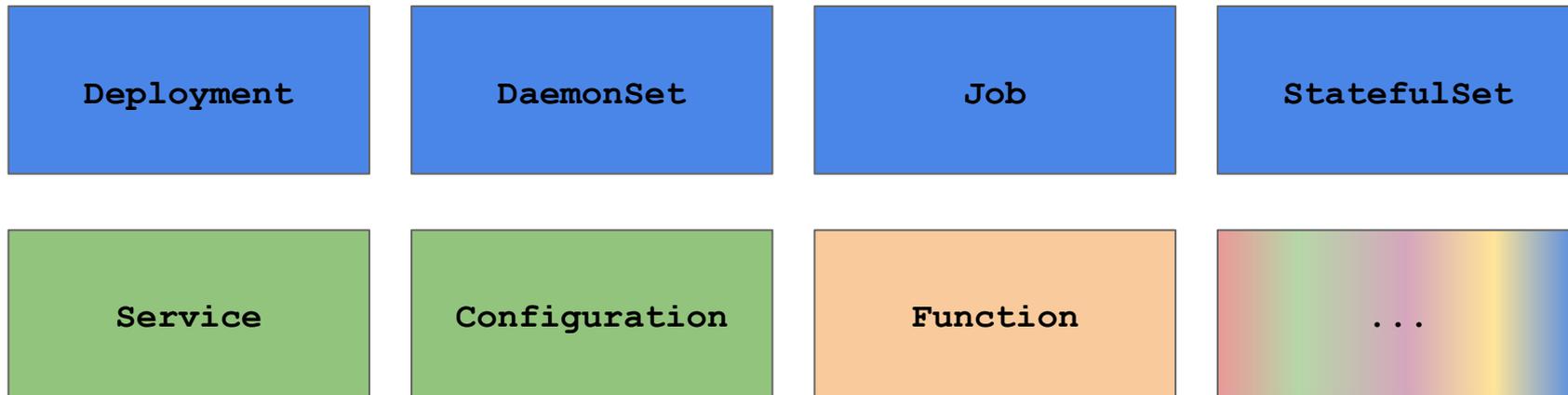
Job

StatefulSet



Background (cont'd)

However, with CRDs allowing folks to build their own higher-level compute abstractions, this list is and will continue to grow.



As a controller author, how do I keep up with this expanding set?

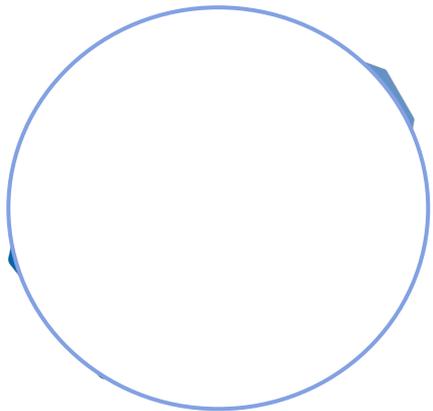


Option A) Bake it in!

```
package eventing

import (
    "github.com/knative/serving/..."
    "github.com/wesley/hutchinson/..."
    "github.com/colonel/mustard/..."
    "github.com/scarlet/speedster/..."
    "github.com/and/on/..."
    "github.com/and/on-forever/..."
    "github.com/please/no-more/..."
    "github.com/why/would/you/do/this/..."
    "github.com/omg/stop-it/..."
)
```





Option B)
untitled duck presentation

~~Polymorphic Reconcilers in Kubernetes~~

snichols@vmware.com

@n3wscott ( ,  , )

vaikas@vmware.com

@vaikas ( , )

@AikasVille ()

mattmoor@vmware.com

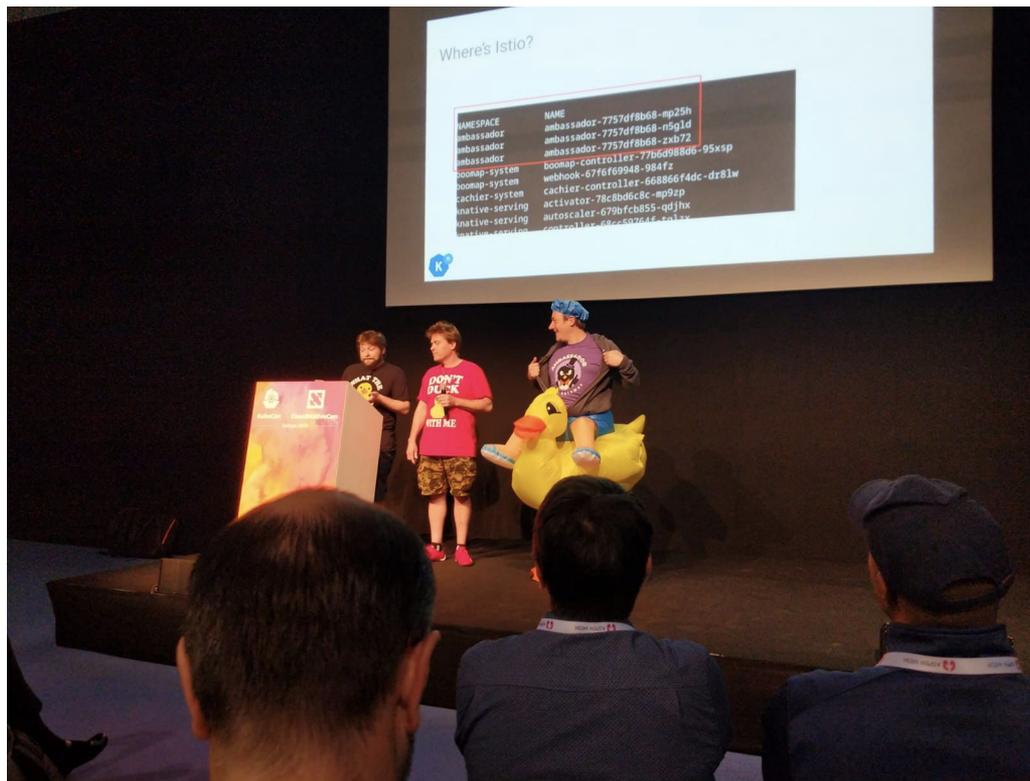
@mattomata ()

@mattmoor ( , )

Barcelona: www.youtube.com/watch?v=Mb8c5SP-Sw0

We gave a talk in Barcelona, which we'd encourage folks to watch for background. This talk is going to try to largely cover new content and demos.

... so you don't all fall asleep!



to do :

- quick recap of kubernetes duck typing*
- talk about bindings*
- demo binding*
- talk about dynamic type controllers*
- demo dynamic type controller*

Quick recap of Kubernetes duck typing

```
{  
  "foo": {  
    "bar": "..."  
  },  
  "bbb": "..."  
}
```

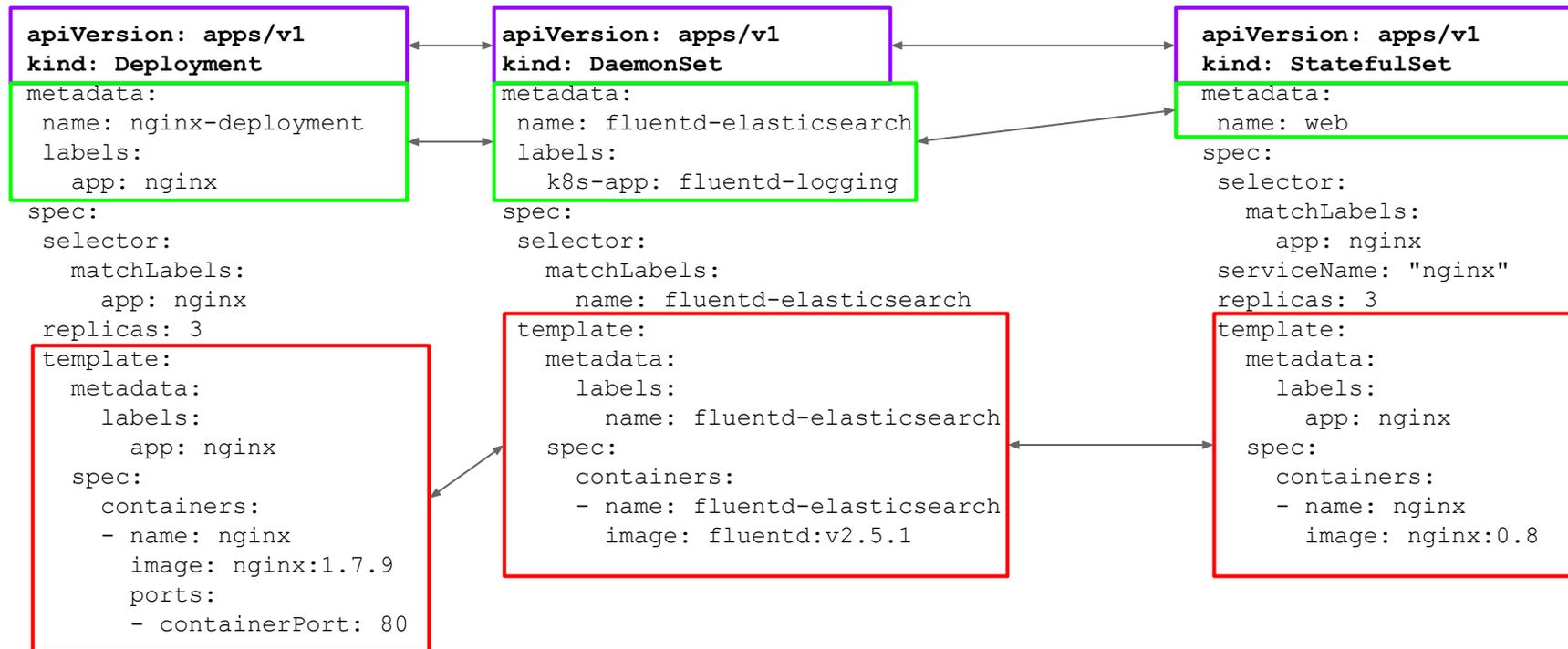
```
{  
  "aaa": "...",  
  "foo": {  
    "bar": "..."  
  }  
}
```

```
{  
  "ccc": "...",  
  "foo": {  
    "bar": "..."  
  },  
  "ddd": "..."  
}
```

} Partial Schema



Kubernetes' happy accident: the apps "duck type"



All the app resources share *this* partial schema, and we can read from and write to this as shown previously.



to do :

- ~~quick recap of kubernetes duck typing~~
- talk about bindings
- demo binding
- talk about dynamic type controllers
- demo dynamic type controller

Problem Statement

There are many instances where users want to late-“bind” things into their applications:

- Secrets / ConfigMaps
- Sidecars

Let's take a look at a simple binding that illustrates a proposed direction for Knative event sources...



“SinkBinding”

```
apiVersion: bindings.mattmoor.dev/v1alpha1
kind: SinkBinding
metadata:
  name: foo-bar
spec:
  target:
    # The K8s resource(s) that want to send
    # events somewhere.

  sink:
    # The “somewhere” (K8s resource) to
    # send the events.
```



How do we inject the “sink” into all of these?

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
          - containerPort: 80
```

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
```

```
template:
  metadata:
    labels:
      name: fluentd-elasticsearch
  spec:
    containers:
      - name: fluentd-elasticsearch
        image: fluentd:v2.5.1
```

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
```

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:0.8
```

... and how do we extract the “sink” from all the possible destinations?



Option A) Bake it in!

```
package eventing

import (
    "github.com/knative/serving/..."
    "github.com/wesley/hutchinson/..."
    "github.com/colonel/mustard/..."
    "github.com/scarlet/speedster/..."
    "github.com/and/on/..."
    "github.com/and/on-forever/..."
    "github.com/please/no-more/..."
    "github.com/why/would/you/do/this/..."
    "github.com/omg/stop-it/..."
)
```



Option B) Ducks!

```
apiVersion: bindings.mattmoor.dev/v1alpha1
kind: SinkBinding
metadata:
  name: foo-bar
spec:
```

```
  target:
```

```
    apiVersion: apps/v1
    kind: Deployment
    name: bar
```

} Anything with a PodSpec (aka “PodSpec”-able)

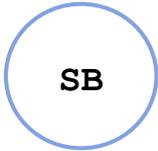
```
  sink:
```

```
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: foo
```

} Our “Addressable” duck type.



Controller Architecture



```
apiVersion: bindings.mattmoor.dev/v1alpha1
kind: SinkBinding
metadata:
  name: foo-bar
spec:
  target:
    apiVersion: apps/v1
    kind: Deployment
    name: bar
  sink:
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: foo
```

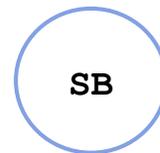


We register a **mutating webhook** for these types so that they are not committed to etcd without their binding injected. *(This is critical for immutable resources, e.g. Job)*



Controller Architecture (cont'd)

```
apiVersion: bindings.mattmoor.dev/v1alpha1
kind: SinkBinding
metadata:
  name: foo-bar
spec:
  target:
    apiVersion: apps/v1
    kind: Deployment
    name: bar
  sink:
    apiVersion: serving.knative.dev/v1
    kind: Service
    name: foo
```



We run a **controller** over SinkBinding that tracks referenced “sinks” so that if the sink address changes the new address can be patched into the binding target.



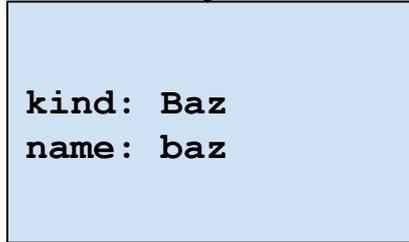
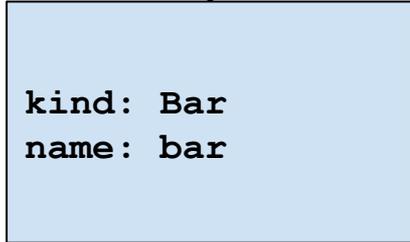
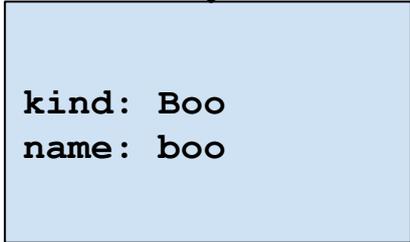
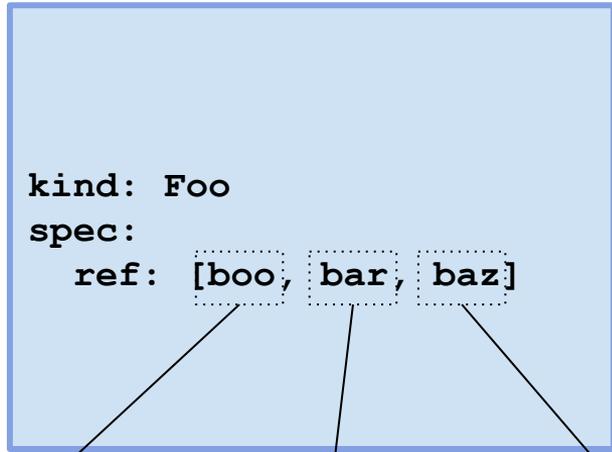
to do :

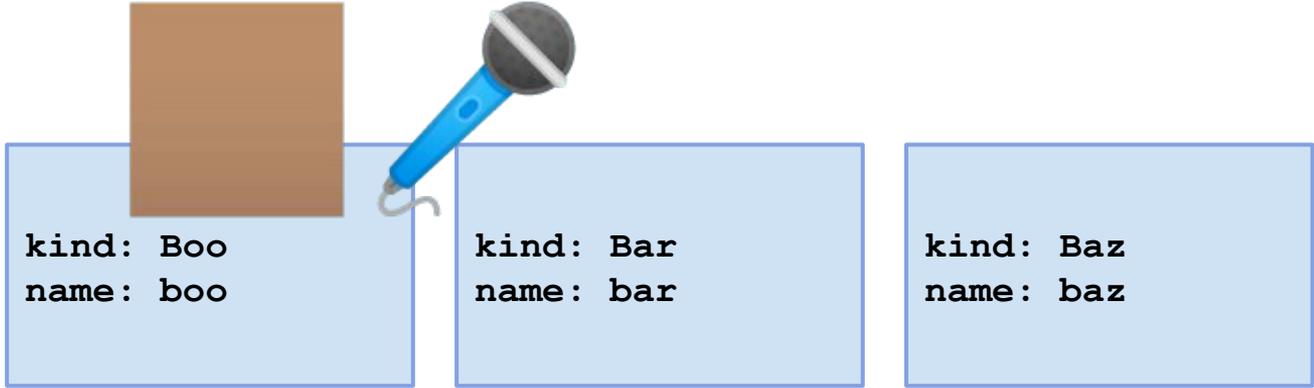
- ~~quick recap of kubernetes duck typing~~
- ~~talk about bindings~~
- demo binding
- talk about dynamic type controllers
- demo dynamic type controller

to do :

- ~~quick recap of kubernetes duck typing~~
- ~~talk about bindings~~
- ~~demo binding~~
- talk about dynamic type controllers
- demo dynamic type controller

Foo

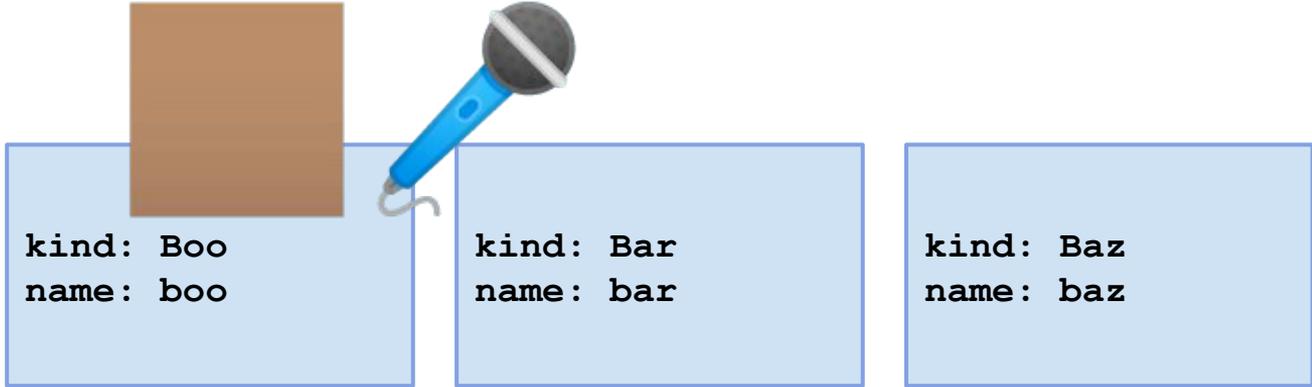




`kind: Boo`
`name: boo`

`kind: Bar`
`name: bar`

`kind: Baz`
`name: baz`



ConfigMap

 : [Boo, Bar, Baz]



Boo

```
kind: Boo  
name: boo
```

Bar

```
kind: Bar  
name: bar
```

Baz

```
kind: Baz  
name: baz
```



```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  labels:
    duck.knative.dev/addressable: "true"
  name: services.serving.knative.dev
spec:
  group: serving.knative.dev
  names:
    categories:
      - all
      - knative
      - serving
    kind: Service
    listKind: ServiceList
    plural: services
    shortNames:
      - kservice
      - ksvc
    singular: service
  scope: Namespaced
  subresources:
    status: {}
  version: v1alpha1
  versions:
    - [name: v1alpha1, served: true, storage: true]
    - [name: v1beta1, served: true, storage: false]
    - [name: v1, served: true, storage: false]
```

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  labels:
    duck.knative.dev/addressable: "true"
  name: brokers.eventing.knative.dev
spec:
  group: eventing.knative.dev
  names:
    categories:
      - all
      - knative
      - eventing
    kind: Broker
    listKind: BrokerList
    plural: brokers
    singular: broker
  scope: Namespaced
  subresources:
    status: {}
  version: v1alpha1
  versions:
    - [name: v1alpha1, served: true, storage: true]
```



crd

```
kind: CustomResourceDefinition
metadata:
  labels:
    ...
    displayName: "true"
    ...
```

Boo

Bar

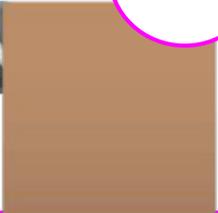
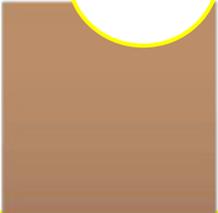
Baz



```
kind: Boo
name: boo
```

```
kind: Bar
name: bar
```

```
kind: Baz
name: baz
```



to do :

- ~~quick recap of kubernetes duck typing~~
- ~~talk about bindings~~
- ~~demo binding~~
- ~~talk about dynamic type controllers~~
- ~~demo dynamic type controller~~

Knative Context

Service.serving.knative.dev

- A containerized service that can scale way up, down to zero.

Broker.eventing.knative.dev

- A stream of events inside of a named mesh.

Trigger.eventing.knative.dev

- An active query on a stream of events inside of a Broker.



Ducktypes in play

Service.serving.knative.dev

- Addressable

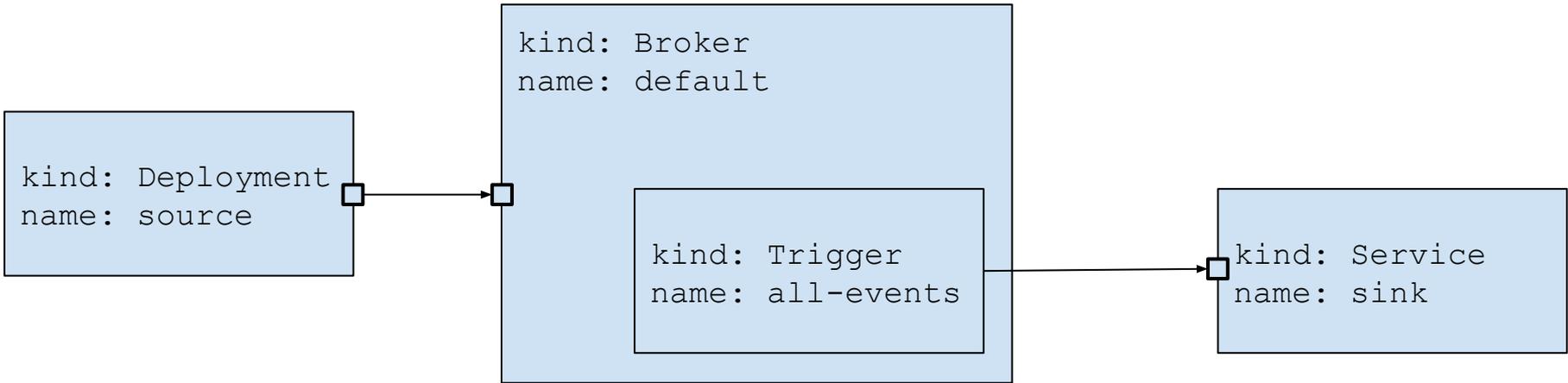
Broker.eventing.knative.dev

- Addressable

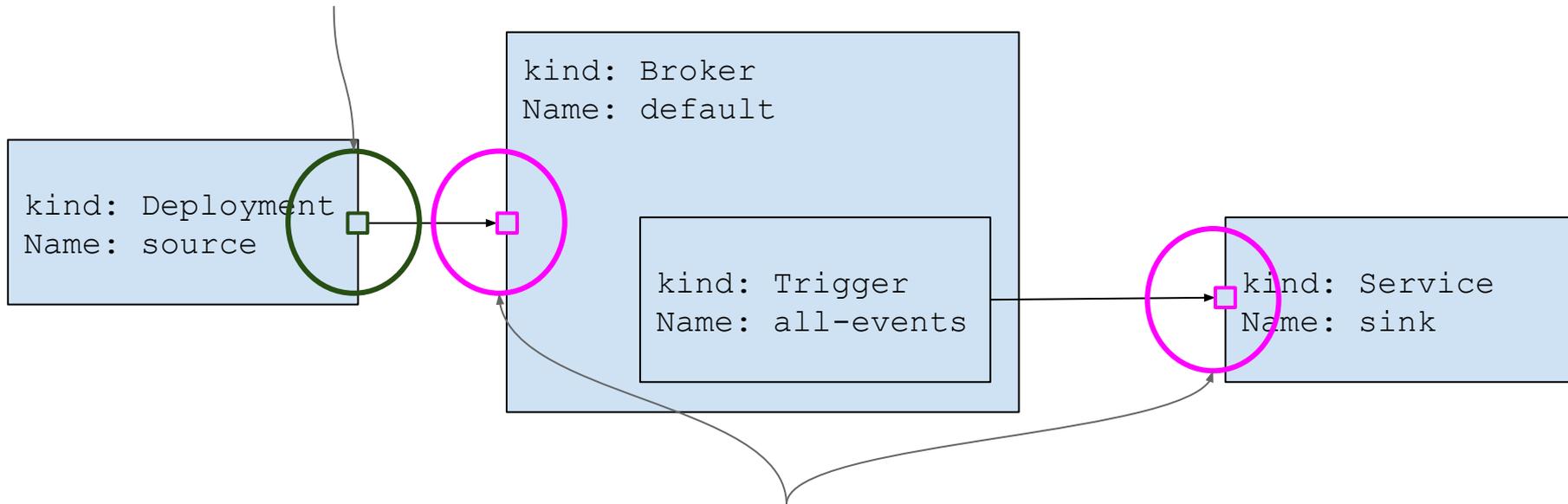
Trigger.eventing.knative.dev

- None*



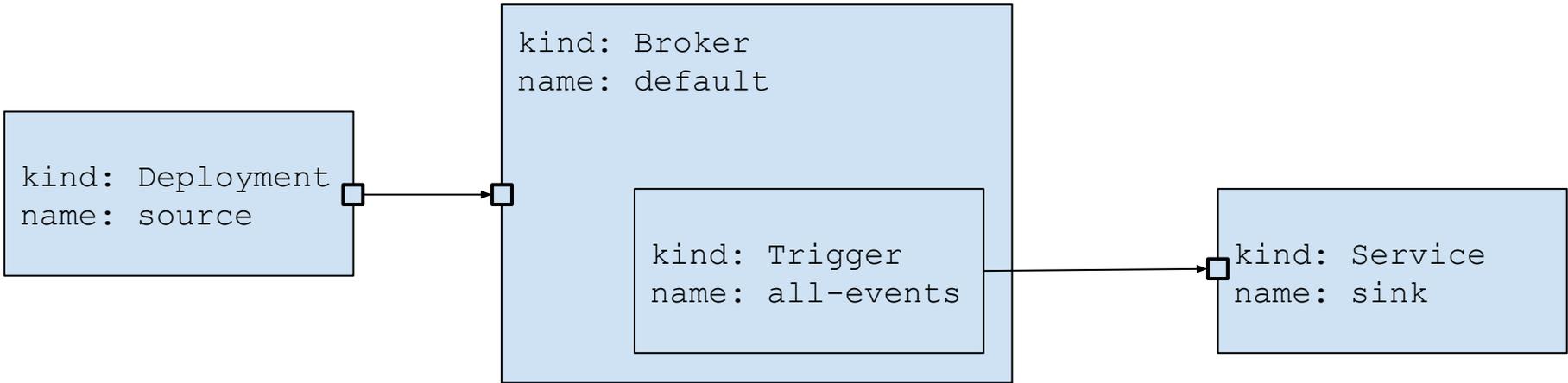


SinkBinding



Addressable







```
kind: Deployment  
name: source
```

```
kind: Broker  
name: default
```

```
kind: Trigger  
name: foo-events  
spec:  
  filter: foo
```

Triggerable (🦆)

```
kind: Service  
name: sink  
labels:  
  trigger: foo
```



to do :

- ~~quick recap of kubernetes duck typing~~
- ~~talk about bindings~~
- ~~demo binding~~
- ~~talk about dynamic type controllers~~
- ~~demo dynamic type controller~~

TODO: Wall of links

