NOT considered anymore EventDefinition metadata for improved Event Discovery

Instead read here:

- [public] Use Event Type for generic discovery without workloads
- [public] Allow Event Type discovery based on improved integration for CloudEvent …

PUBLIC DOCUMENT

Contributor(s)	Matthias Wessendorf	
Proposal Date	Apr 25, 2023	
Status		
Main Github Issue	https://github.com/knative/eventing/issues/6890 https://github.com/knative/eventing/pull/6999	

[Section filled out by WG Lead; NB: these only need to be “approvers”, ***not*** WG leads]

Approver(s)	**Approver Area**	**Approver Email**
Pierangelo Di Pilato	Eventing WG	

Motivation / Abstract

Currently in Knative the EventType API describes types of events that can be consumed from a given broker. However this assumes that there are sources connected to a running instance of a broker.

It is not possible to tell what event type definitions are generally available in Knative Eventing, based on the installed sources/event emitters, **independent** of a running broker.

In the Knative Eventing we do have a few built-in sources, see the following query to identify those:

```
None
k get crds -l 'duck.knative.dev/source'
NAME                                   CREATED AT
apiserversources.sources.knative.dev   2023-03-15T12:49:47Z
containersources.sources.knative.dev   2023-03-15T12:49:47Z
pingsources.sources.knative.dev        2023-03-15T12:49:47Z
sinkbindings.sources.knative.dev       2023-03-15T12:49:47Z
```

Some of those sources, like the ApiServerSource or the PingSource do have a fixed set of event types they are emitting:

Source	Event Types
ApiServerSource	• dev.knative.apiserver.resource.add • dev.knative.apiserver.resource.delete • dev.knative.apiserver.resource.update • dev.knative.apiserver.ref.add • dev.knative.apiserver.ref.delete • dev.knative.apiserver.ref.update
PingSource	• dev.knative.sources.ping

On an empty installation of Knative it is not possible to say what event types are generally available in the system for future consumption, only when a source object is created and points to a broker:

```
None
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  name: ping-source-broker
spec:
  schedule: "*/1 * * * *"
  data: '{"message": "Hello world!"}'
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1
      kind: Broker
      name: my-broker
```

After this creation of the Source, the events for a specific broker are discoverable:

```
None
k get eventtypes.eventing.knative.dev
TYPE                       SOURCE
REFERENCE NAME   READY
dev.knative.sources.ping   /apis/v1/.../pingsources/ping-source-broker
my-broker        True
```

Another limitation from the above scenario is that EventTypes are only discoverable when used in combination with a broker. Applications based on Channels or regular sinks, are not able to make use of the EventType.

New CRD for Event Type Definitions

Knative Eventing should have a new (Cluster) Event Type Definition metadata CRD that can capture this information independently of running applications.

The idea behind this new proposed CRD is influenced by the CNCF CloudEvents "Message Definition Groups" API, which basically groups event type definitions into groups.

Take a look at the table above, which provides a grouping of Knative sources and their event types. With this table developers know what they can expect from the different sources, regardless of the application implementation depending on a broker or not.

Besides the simple grouping of events type definitions to a specific group (the Knative source) the new CRD will also give additional information about the event payload and its metadata.

Background

User Stories

As system integrator
I want to register my source with information what events it emits
so that other teams can discover and subscribe to my system events

As a application developer
I want to declare what events my service is returning after processed incoming events
so that other teams can discover and subscribe to my app events

As application developer
I want to have a simple catalog* to find events
so that I can easily consume events available in the system

Scope

- In Scope
  - EventDefinition CRD / type
  - Cluser EventDefinition CRD / type
  - Relation to EventType
- Out of Scope
  - The discovery of “custom events” (e.g., events generated within the cluster by a Service, by a reply to the Broker, etc.) We believe EventType could be used in those cases, although we haven’t fully fleshed out the details yet.

    - A registry implementation of the CE registry spec (which could help with the point above)
    - Security policies in the Broker. For example to allow only registered EventTypes to flow through the system, or to register EventTypes upon first-seen, etc.
    - EventType usage on other components like Channels

Proposal Design / Approach

Provide a new metadata CRD to be installed with the build-in sources, called EventDefinition / ClusterEventDefinition.

Design

(Cluster) Event Definition CRD

We provide a “metadata CRD", (Cluster)EventDefinition, that offers generic information about event definitions, rather than for a specific workload.

Below is are a few examples:

1. Cluster-scoped EventDefinition for the Knative “ping source”:

```
None
apiVersion: eventing.knative.dev/v1alpha1
kind: ClusterEventDefinition
metadata:
  name: dev.knative.sources.ping
spec:
  group: pingsource.sources.knative.dev
  description: Knative PingSource CloudEvent type definition
  metadata:
    attributes:
      - name: type
        value: dev.knative.sources.ping
        required: true
      - name: specversion
        value: "1.0"
      - name: id
        required: true
      - name: source
        value:
"/apis/v1/namespaces/{namespaceName}/pingsources/{souceName}"
      - name: time
        required: true
  schemaUrl: ""
  format: CloudEvents/1.0
```

2. Custom, namespaced, event definition for 3rd party integration

```
None
apiVersion: eventing.knative.dev/v1alpha1
kind: EventDefinition
metadata:
  name: com.my.shop.neworder
spec:
  group: oder.system.com
  description: Abstract CloudEvent type definition for new orders
  metadata:
    attributes:
      - name: type
        value: com.my.shop.neworder
        required: true
      - name: specversion
        value: "1.0"
      - name: id
        required: true
      - name: source
        value: "/someURI"
      - name: time
        required: true
  schemaUrl: "https://..."
  format: CloudEvents/1.0
```

Resources bundled with Knative Eventing

For built-in sources that emit a fixed set of events Knative Eventing will bundle a set of ClusterEventDefinitions, with the needed details. On a fresh Knative Eventing installation, with no workload running, the expected state would be a set of predefined Cluster Event Definitions:

```
None
kubectl get clustereventtypedefinitions.eventing.knative.dev
NAME                                       GROUP
dev.knative.apiserver.ref.add              apiserversources.sources.knative.dev
dev.knative.apiserver.ref.delete           apiserversources.sources.knative.dev
dev.knative.apiserver.ref.update           apiserversources.sources.knative.dev
dev.knative.apiserver.resource.add         apiserversources.sources.knative.dev
dev.knative.apiserver.resource.delete      apiserversources.sources.knative.dev
dev.knative.apiserver.resource.update      apiserversources.sources.knative.dev
dev.knative.sources.ping                   pingsources.sources.knative.dev
```

3rd party integrations with custom EventDefinitions

For 3rd party integrators or application developers that have sources, jobs, deployments etc that are emitting events the recommendation is to also provide a set of EventDefinitions for improved discoverability. See the above example.

Relationship to existing EventType API

Each EventType object, representing a “living” instance of a type of an event that can be directly consumed and will have a relationship to its broader Event Type definition. During reconciliation of EventTypes they should be updated with an extra annotation indicating to which EventDefinition the EvenType belongs. This annotation reference must be set during the reconciliation process of the EventType itself.

Implementation

The Golang API “eventdefinition_types.go”:

```
None
type EventDefinition struct {
      metav1.TypeMeta `json:",inline"`
      // +optional
      metav1.ObjectMeta `json:"metadata,omitempty"`

      // Spec defines the desired state of the EventDefinition.
      Spec EventDefinitionSpec `json:"spec,omitempty"`

}
```

```
type EventDefinitionSpec struct {
      // SchemaURL is a URI, it represents the payload schemaurl
      // It may be a JSON schema, a protobuf schema etc. It is optional
      // +optional
      SchemaURL *apis.URL `json:"schemaUrl,omitempty"`

      // Describes the format of the events
      //but for us it is generally only CE...?
      Format string `json:"format,omitempty"`

      // The group where the EventDefinition is defined.
      Group string `json:"group,omitempty"`

      // Event metadata, such as attributes, extensions
      // from CloudEvents spec.
      Metadata EventDefinitionMetadata `json:"metadata,inline"`

      // Description is an optional field used to describe the
      // definition, in any meaningful way.
      // +optional
      Description string `json:"description,omitempty"`

}

type EventDefinitionMetadata struct {
      Attributes map[string]EventDefinitionAttribute `json:"attributes"`
}

type EventDefinitionAttribute struct {
      Required bool        `json:"required"`
      Value    interface{} `json:"value,omitempty"`
}
```

The new API is seen as pure metadata and therefore the CRD will have no status in the first iteration of the API version.

Prerequisites / Dependencies

[Are there any issues / tech that need to be in place for this to work?]

Integration Checklist

Operations

The feature will be part of the regular eventing distribution and is supported for the built-in source like PingSource or ApiServerSource.

Observability

[Will this feature need instrumentation or measures that are exposed to specific personas? If so, which personas and optics are needed?]
Developers will be able to use the new API without any tweaking.

CLI `kn`

Developers will be able to use `kn` subcommand groups to manipulate instances of EvenTypeDefinition. Following the same UX design as for other resource types, there will be several options flags to define fields of ETD with opinionated defaults when suitable.

Test Plan

[How is the feature tested for use? i.e unit testing, E2E, isolated or in conjunction with other components? that conformance tests need to be in place?]

- Unit tests for the feature
- Reconciler e2e for the feature

Documentation

[What personas will use this feature and which documented use-cases does this affect? Are there new use-cases that need to be written or existing ones edited?]

The feature will help developers discovering event type definitions better and will need a solid set of documentation and samples

Exit Criteria

[What are the requirements to exit each stage]

Alpha

- API will start in v1alpha1
- Tests need to be implemented
- Documentation needs to be there
- Iterate if the API has missing pieces?
- (optional) CLI integration

Beta

- After phase of stabilization the API will move to v1beta1
- Tests need to be updated
- Documentation needs to be updated
- CLI integration

GA

- Similar to the above: Once the v1beta API is stable, move to v1

Alternatives Considered

Modify EventType and do not require the broker

Instead of the new additional type one option would be to change the EventType and make the broker field optional so that an event type would be "ready" without a broker, and generic type definition information could be submitted as well.
This approach has the following advantages:
- Less new APIs / types

This approach has the following disadvantages:
- The Event Type API would do too much as it would represent types of events that can be consumed from a given broker and in the cluster

- This would confuse users as it is not directly clear when an event type is ready and why, given the different meanings.
- Multi purpose type is offering a bad or harder to understand UX

Use just EventDefinition for “cluster” and “namespace” scope

Instead of introducing the “cluster scoped” nuance for “ClusterEventDefinition”, we use the EventDefinition for both: custom event definition (true namespaced) and “cluster wide” definitions, like default source for knative.
While on the surface this seems easier, b/c has one CRD less, there are downsides that the actual differentiation between “cluster” and “namespaced” are NOT given. Leading to different confusion