KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2023

KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2023

Multi-Cluster Alerting: A Kubernetes-Native Approach

Junot Xiang & Wanjun Lei, QingCloud

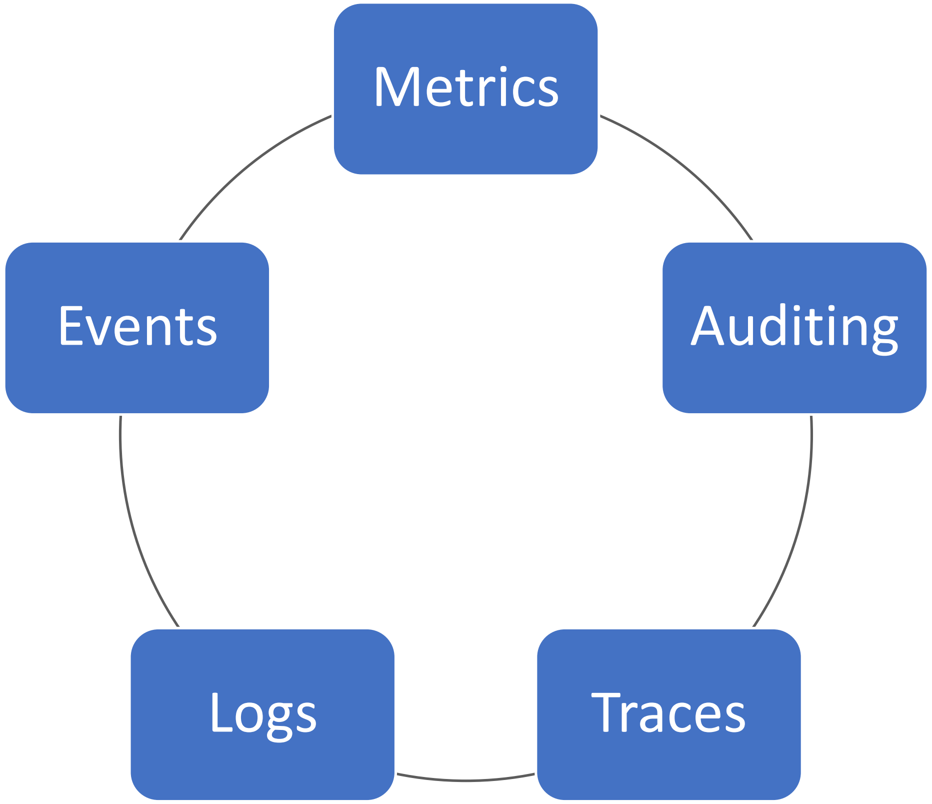
Agenda

KubeCon CloudNativeCon OPEN SOURCE SUMMIT China 2023

1. Observable Sources
2. Alerting Rules
3. Notification Configs

Observable Sources

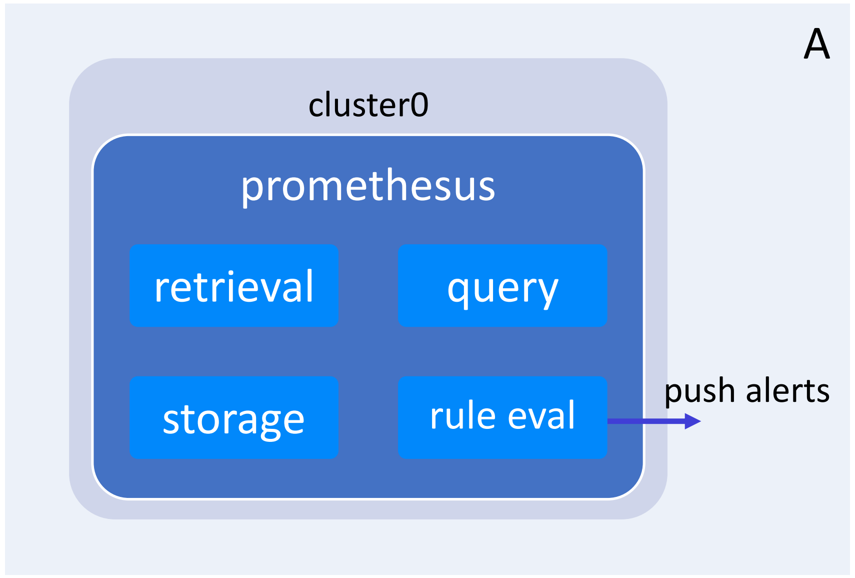
A variety of obsevable sources provide us detailed insights into the real-time performance of Kubernetes clusters and applications. Alerting based on them allows us to make timely and informed actions.

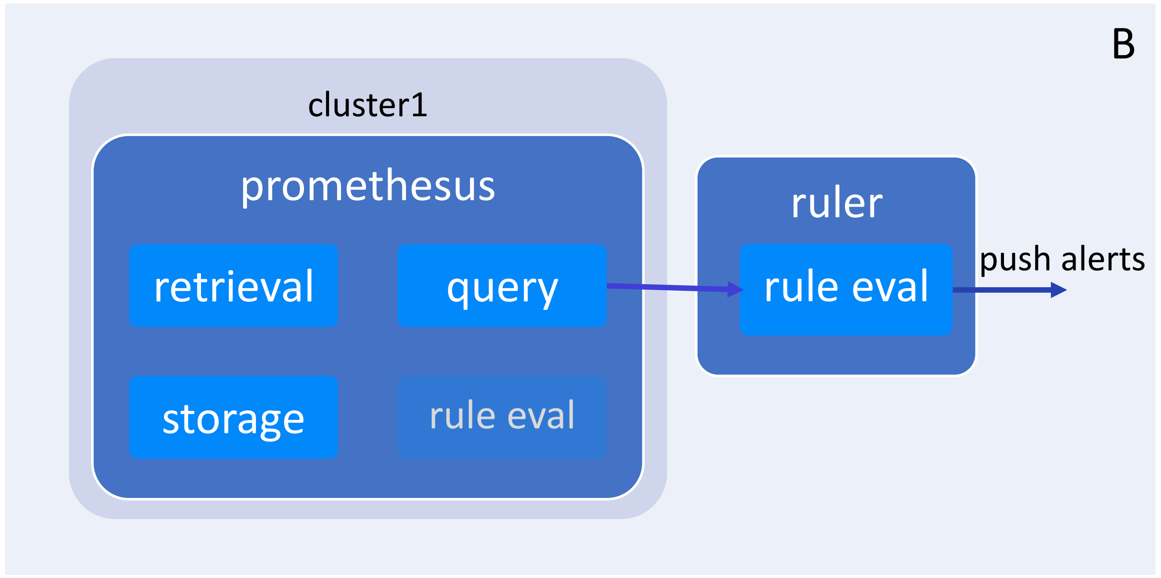


Alerting Rules - for metrics

Prometheus, as the de facto standard in clound native monitoring, is widely used.

For one separate cluster, just a prometheus can provide metrics retrieval, storage, query and alerting (here only refer to evaluation of alerting rules) (A). Maybe we'll add an extra ruler component to evaluate more alerting rules to take the pressure off Prometheus (B).

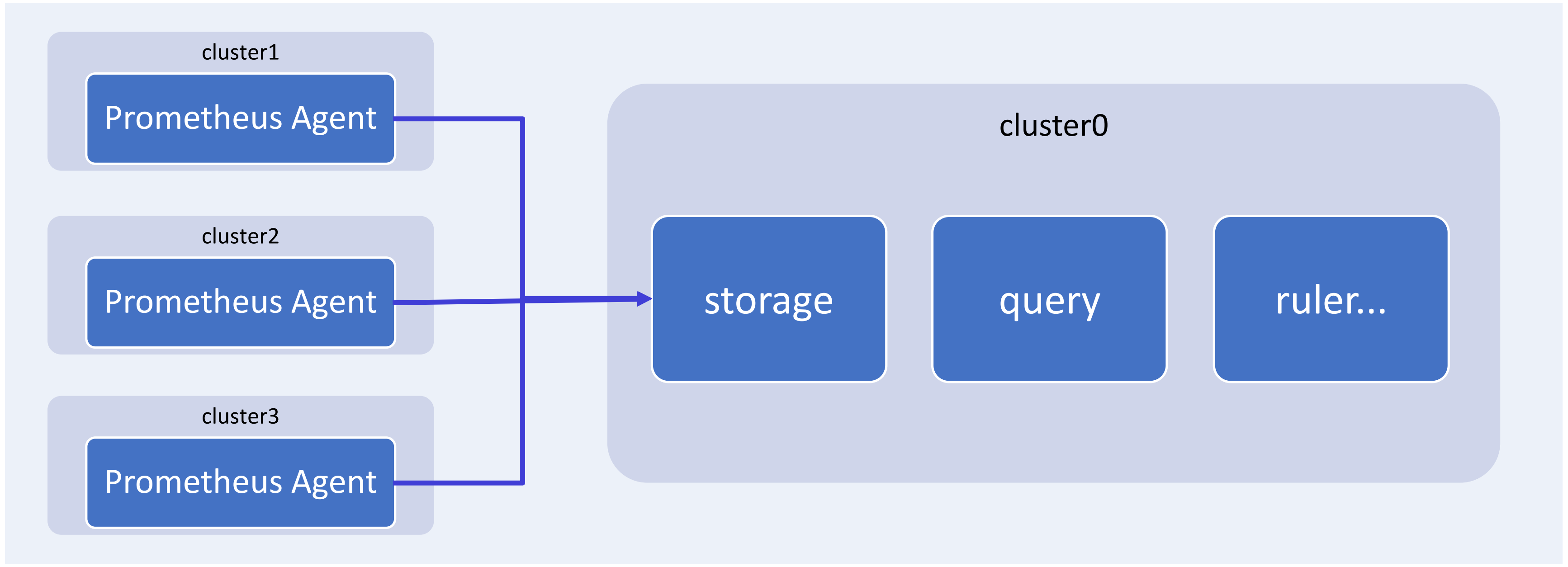




The separate cluster is for every cluster manages its own metrics storage

Alerting Rules - for metrics

The metrics storage management mode changes to that every cluster write its scraped metrics to a hosting cluster or a cloud cluster, which put higher requirements on the ruler for alerting. The rulers have to support alerting both each cluster and multiple associated clusters.



Every cluster metrics storage is centralized to a hosting cluster.
A *cluster* label has to be added to metric data to distinguish culsters.

Alerting Rules - for metrics

Prometheus Operator as a Kubernetes native way to manage Prometheus, provides a lot of convenience for deploying and configuring Prometheus.

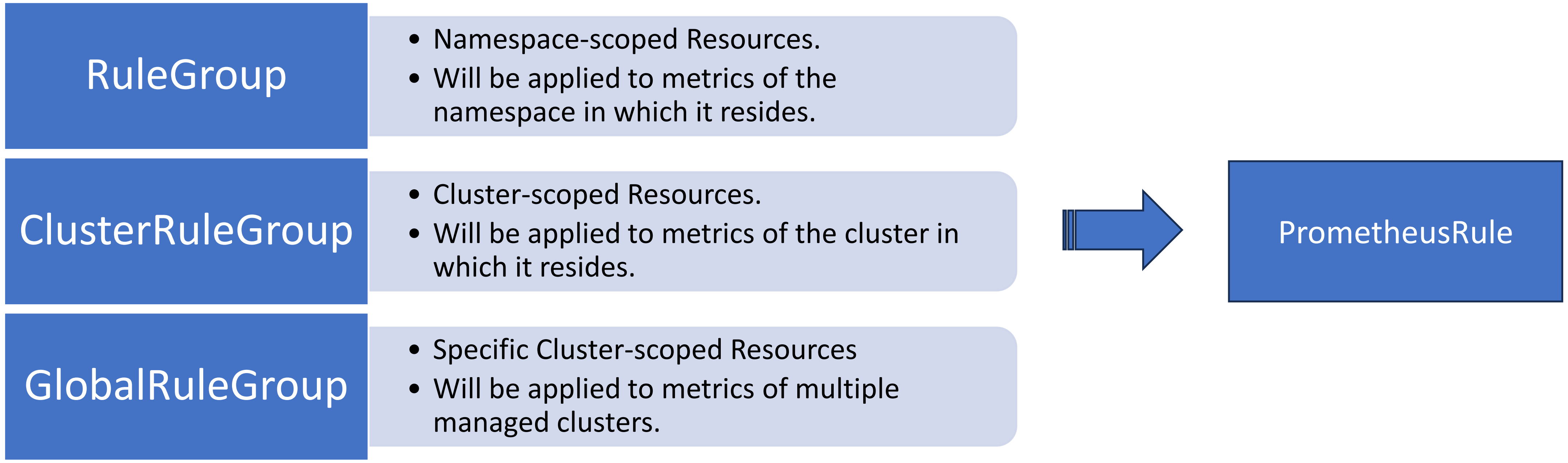
There are still difficulties to configure alerting rules by PrometheusRule defined by it.

- Large configuration granularity, resulting in no insufficient support for concurrent updates.
- Lack of configurability, such as enabling or disabling an alerting rule.
- Poor support in multi-tenant and multi-cluster scenarios.

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: <name>
  namespace: <namespace>
spec:
  groups:
  - name: <group_name>
    interval: <eval_interval>
    rules:
    - name: <rule_name>
      expr: <PromQL>
      labels: {}
      for: <duration>
      ...
```

Alerting Rules - for metrics

Three fine-grained and enhanced custom resources are defined to make rule configuration more flexible and easily applied to multi-cluster and multi-tenant scenarios, and compatible with upstream PrometheusRule resources.



RuleGroup
- Namespace-scoped Resources.
- Will be applied to metrics of the namespace in which it resides.

ClusterRuleGroup
- Cluster-scoped Resources.
- Will be applied to metrics of the cluster in which it resides.

GlobalRuleGroup
- Specific Cluster-scoped Resources
- Will be applied to metrics of multiple managed clusters.

https://github.com/kubesphere/kubesphere

Alerting Rules - for metrics

- Each rule group is configuration unit that contains multiple associated rules.
- Retain the original rule structure to ensure compatibility.
- Enable or disable a single rule or an entire group.
- A **exprBuilder** is provided to automatically build a **expr**(PromQL) for general scenarios.

```yaml
kind: RuleGroup
apiVersion: alerting.kubesphere.io/v2beta1
metadata:
  name: nginx-alert
  namespace: demo
  labels:
    alerting.kubesphere.io/enable: 'true'
  annotations:
    kubesphere.io/creator: admin
spec:
  interval: 1m
  rules:
    - alert: ReplicasAvailabilityLow
      expr: ''
      exprBuilder:
        workload:
          kind: Deployment
          names:
            - nginx
          comparator: '>'
          metricThreshold:
            replica:
              unavailableRatio: 0.3
      for: 1m
      severity: critical
      annotations:
        summary: Deployment has a low availability of replicas
      disable: false
```

Alerting Rules – for metrics

The exprBuilder provides a convenient way to configure a Prometheus query expression by covering various metrics alerting for workloads and nodes.

- **workload**:
  - kinds cover Deployment,StatefulSet,DaemonSet.
  - metrics cover cpu, memory, network, replicas of the workload.
- **node**: (not available for RuleGroup instances)
  - metrics cover cpu,memory,network,disk,pod_rate, etc. of the node.

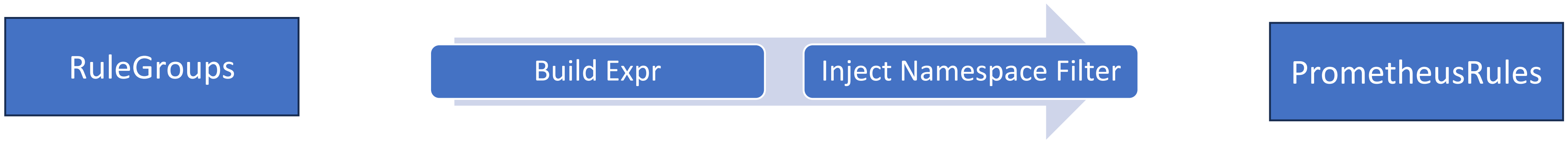
```yaml
exprBuilder:
  workload:
    kind: Deployment
    names:
      - nginx
    comparator: '>'
    metricThreshold:
      replica:
        unavailableRatio: 0.3
```

```yaml
exprBuilder:
  node:
    names:
      - node0
    comparator: '>'
    metricThreshold:
      memory:
        utilization: 0.9
```

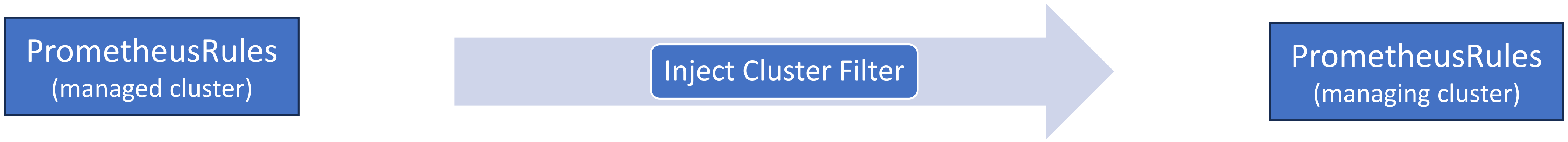
Alerting Rules - for metrics

The instances of RuleGroup/ClusterRuleGroup/GlobalRuleGroup can be combined into instances of PrometheusRule, and some restrictions will be added during the process.

Limit namespace scope of RuleGroups to provide multi-tenant alerts.



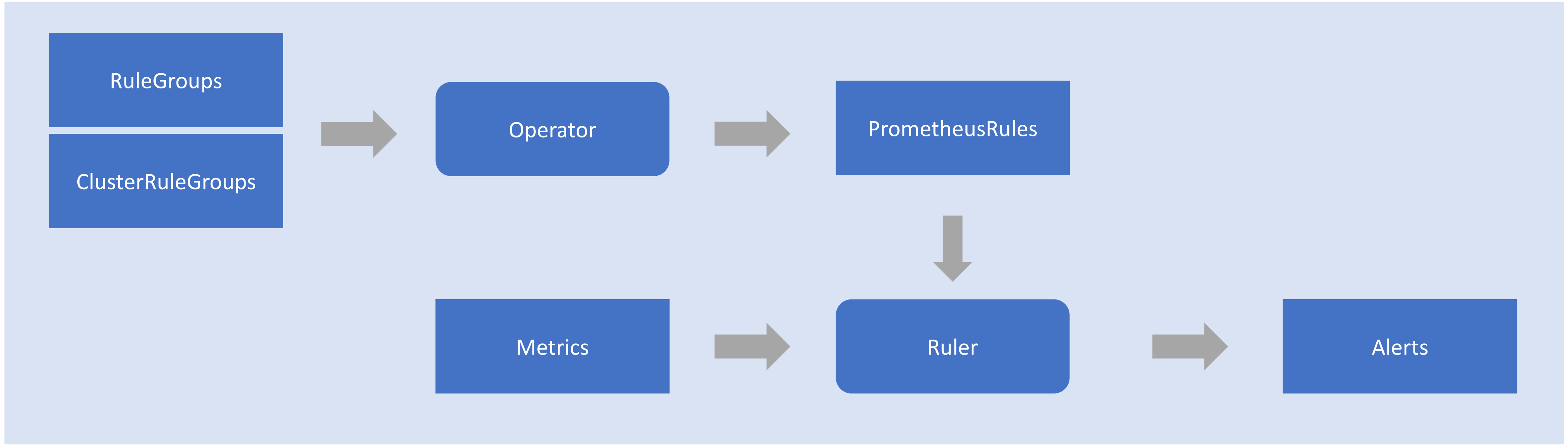
Limit affect scope of PrometheusRules to provide unified alerts for multiple clusters.


PrometheusRules (managed cluster)
Inject Cluster Filter
PrometheusRules (managing cluster)

Alerting Rules - for metrics

Regardless of the metrics storage management mode, the Rule evaluation (Ruler) is better on the data side.

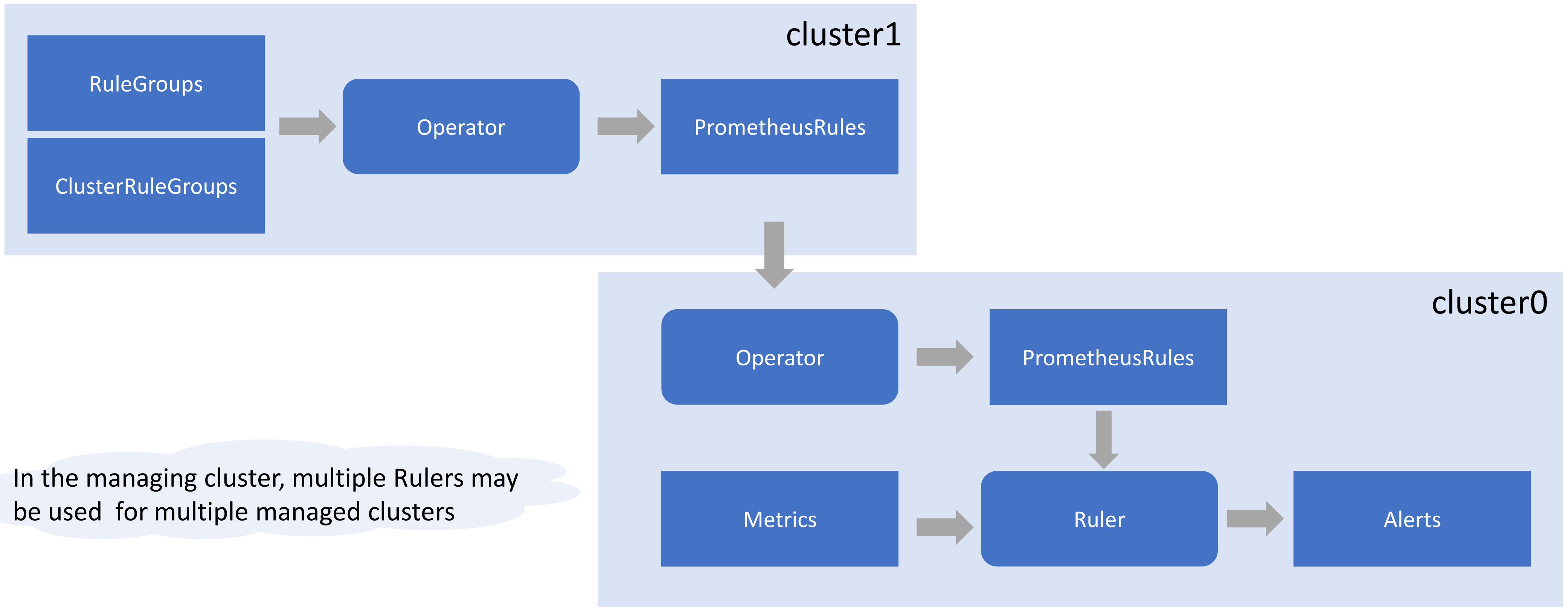
If a cluster manages its own metrics storage, the Ruler should be in the same cluster.



The cluster manages its own metrics storage

Alerting Rules - for metrics

If a cluster is managed and deposits its metrics in the managing cluster, the alerting rules will be synced to the managing cluster and also be evaluated in the managing cluster.



In the managing cluster, multiple Rulers may be used for multiple managed clusters

Alerting Rules – for metrics

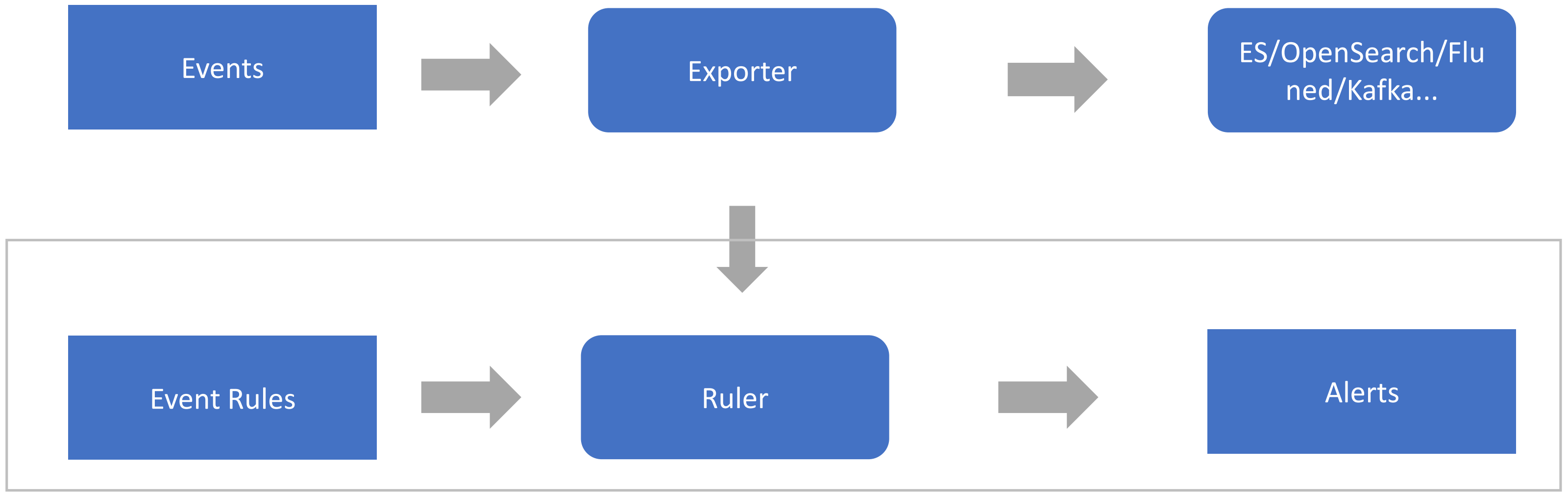
Alerts triggered by newly defined rule resources will not only contain metric labels and also be enriched by following labels for fault location.

alerttype	• To distinguish different alert sources
cluster	• Used in multi-cluster scenarios
severity	• To perform hierarchical control on alerts
rule_group	• To establish effective relationships between rule groups and alerts
rule_level	• To establish effective relationships between rule groups and alerts

Alerting Rules - for events

K8s Events generally denote some state change in the system, which have a limited retention time as K8s resource objects.

The components from kube-events project export them for long term storage, and generate alertmanager-compatible alerts by evaluation of Event Rules, which can filter out warning or critical events, or those that are of interest to the users.



https://github.com/kubesphere/kube-events

Alerting Rules - for events

The Event Rule CRD defines a configuration for alerting based on Events.

- **condition**: a sql where expression (supported by a self-implementing Event Rule Engine) to support a more flexible way to filter events.
- **labels**: extra labels to be added to alerts.
- **annotations**: detailed informations about the alert. their values can be templated to get event fields by %.

https://github.com/kubesphere/event-rule-engine

```yaml
apiVersion: events.kubesphere.io/v1alpha1
kind: Rule
metadata:
  name: demo
  namespace: demo
  labels:
    kubesphere.io/rule-scope: cluster
spec:
  rules:
    - name: PodNetworkNotReady
      condition: |
        type="Warning" and involvedObject.kind="Pod"
and reason="NetworkNotReady"
      labels:
        severity: warning
      annotations:
        message: '%message'
        summary: Pod network is not ready
        summaryCn: Pod网络异常
      enable: true
      type: alert
```

Alerting Rules - for events

A *kubesphere.io/rule-scope* label in an Event Rule instance can be used to limit the rule scope, values of which can be as follows.

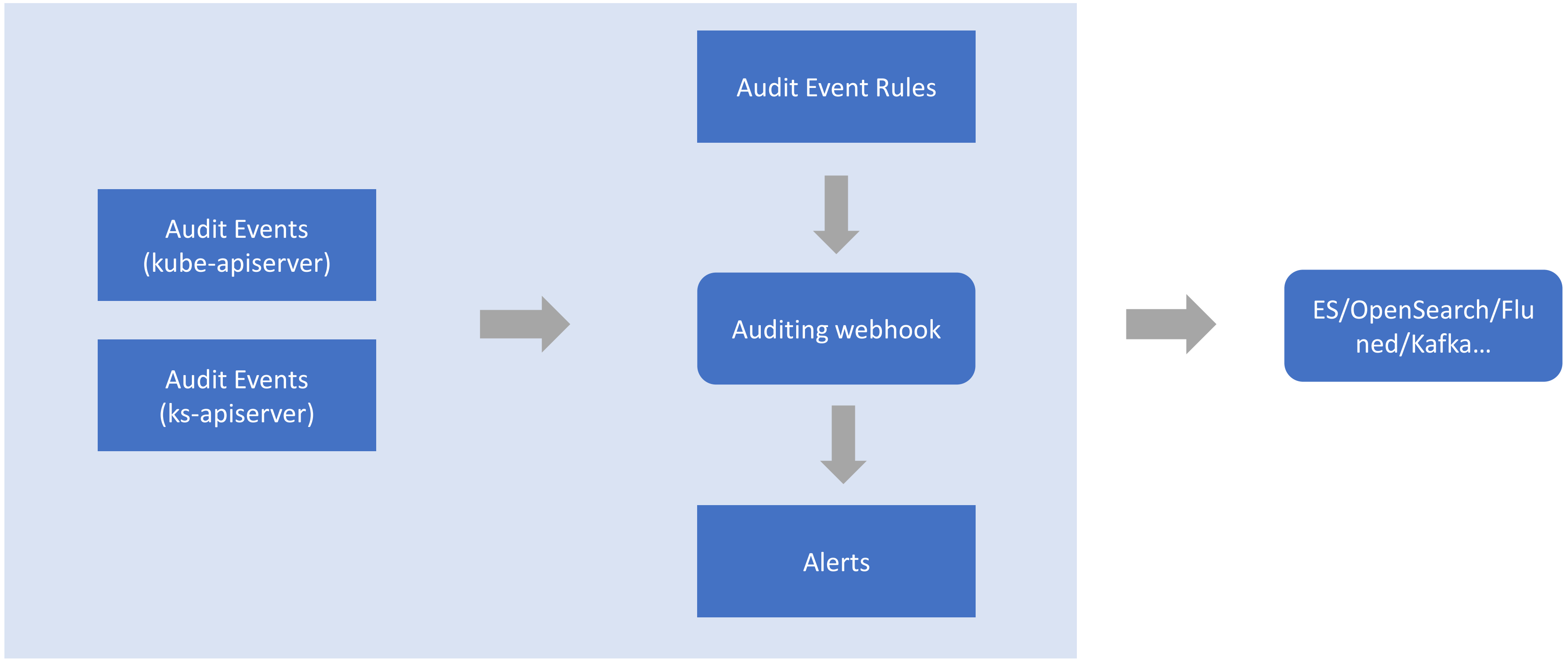
cluster	•Applied to all events in the cluster.
workspace	•Applied to events in multiple namespaces which belongs to a same workspace. A *kubesphere.io/workspace* label has to be specified in the Rule instance.
namespace	•Applied to events whose *involvedObject.namespace* equals to the namespace in which the Rule instance is.

A workspace is a logical unit to organize namespaces or something similar in KubeSphere

Alerting Rules – for audit events

KubeCon | CloudNativeCon OPEN SOURCE SUMMIT China 2023

K8s cluster can generate audit events about the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself. Based on these audit events, security-relevant alerting can be provided.


Audit Event Rules
Audit Events (kube-apiserver)
Audit Events (ks-apiserver)
Auditing webhook
Alerts
ES/OpenSearch/Fluned/Kafka…

Alerting Rules - for audit events

Audit Rule CRD defines a configuration to filter out audit events, which will be stored for long storage, or be generated to alerts to be sent to users. Every Audit Rule instance can contain multiple rules.

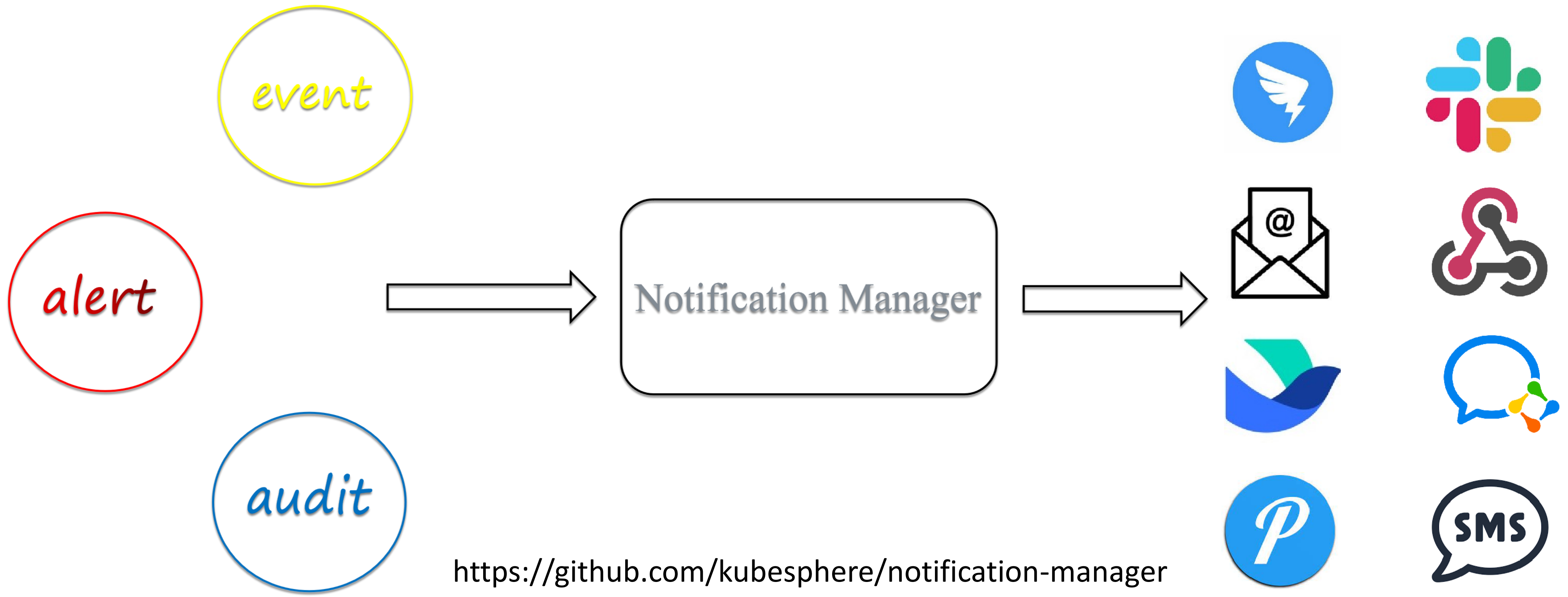
Four **type**s of rule are defined:

- **rule**: a real complete rule with a **condition** field, which is also a previous sql-like expression.
- **macro**: a lite rule which can be called by other **macro** or **rule** rules.
- **list**: a list which can be called by **macro** or **rule** rules.
- **alias**: just a alias name of one varible.

```yaml
apiVersion: auditing.kubesphere.io/vlalphal
kind: Rule
metadata:
  generation: 1
  labels:
    type: archiving
  name: archiving-rule
  namespace: kubesphere-logging-system
spec:
  rules:
  - desc: all action not need to be audit
    list:
    - get
    - list
    - watch
    name: ignore-action
    type: list
  - condition: Verb not in ${ignore-action}
    desc: All audit event except get, list, watch event
    enable: true
    name: archiving
    priority: DEBUG
    type: rule
```

Notification Manager

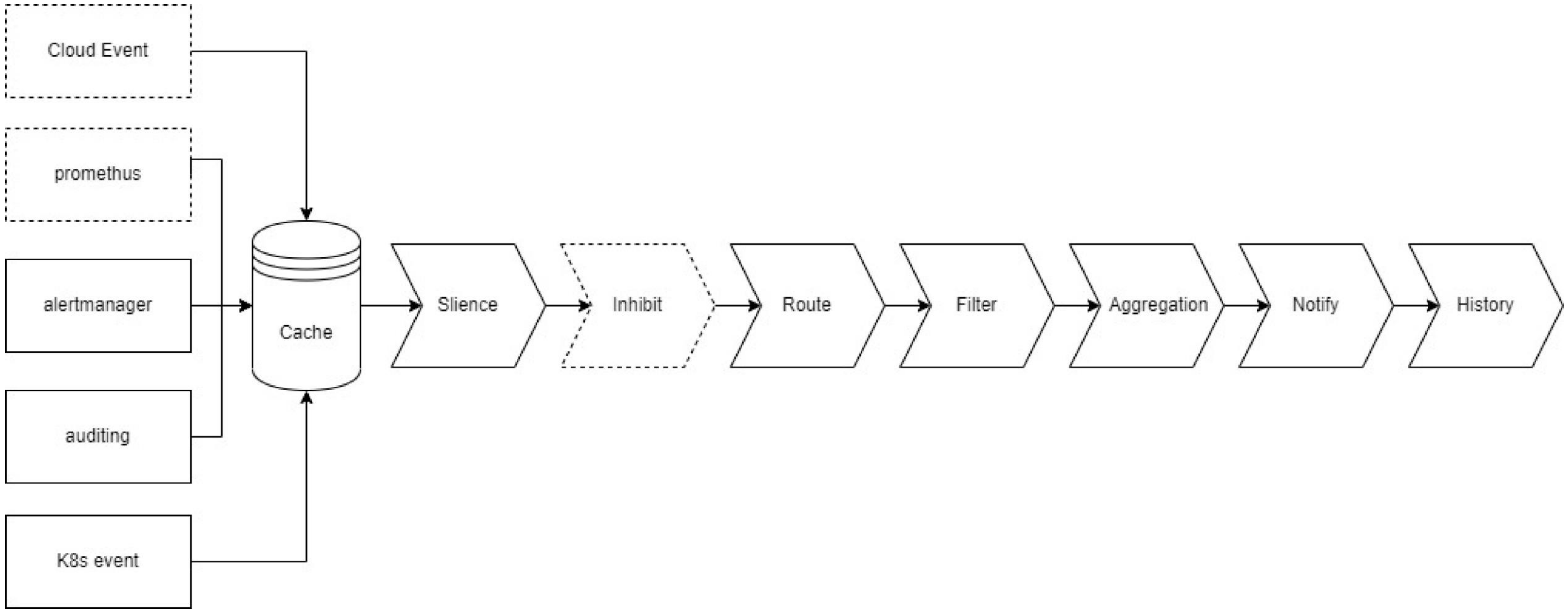
All previous alerts will be sent to notification-manager which is a K8s native notification management with multi-tenancy and various channels support. Some CRDs are introduced to implement that.



https://github.com/kubesphere/notification-manager

Receiver is used to define the notification format and destinations to which notifications will be sent.

Notification Manager - Architecture



Notification Manager - Silence

Slience is used to define policies to mute notifications for a given time. A silence is configured based on a label selector. If the incoming alert matches the label selector of an active silence, no notifications will be sent out for that alert.

- Supports setting global silence rules and tenant silence rules.
- Support periodic silence.
- Support silence for a specific period of time.

```yaml
apiVersion: notification.kubesphere.io/v2beta2
kind: Silence
metadata:
  name: silence
  labels:
    type: tenant
    user: admin
spec:
  matcher:
    matchExpressions:
    - key: namespace
      operator: In
      values:
      - test
  # startsAt: "2022-12-01T00:00:00Z"
  schedule: "0 22 * * * *"
  duration: 8h
```

Notification Manager - Inhibit

Inhibit allows specific alerts to be silenced based on other alerts. For example, when a cluster node goes down, a large number of alerts will be generated, but most of these alerts are invalid.

Sending these invalid alerts to users will not only cause a waste of resources,

And it will cause interference to users in locating faults. At this time, you can use inhibit rules to silence other alerts and only retain the alerts about node downtime.

Notification Manager - Route

Route will choose the correct **Receivers** according to the rules set by the user for alerts.

Here we introduce a new concept, **Receiver**. so what is **Receiver?**

Receiver is used to define the notification format and destinations to which notifications will be sent. A Receiver contains the following information:

- Information about the receiving channel, such as email address, slack channel, etc.
- Template to generate notification message.
- Label selector to filter alerts.

Notification Manager – Receiver

Receiver can be categorized into 2 types global and tenant.

- A global receiver receives all notifications and then send notifications regardless tenant info(user or namespace).
- A tenant receiver only receives notifications from the namespaces that the tenant has access to.

Receiver is used to define the notification format and destinations to which notifications will be sent.

Notification Manager – How to select Receiver

How to determine which Receiver the alert needs to be sent to? There are two methods:

- Specified via **Router.**
- Match by namespace. Alerts without namespace label will be automatically sent to the global level Receiver. Other alerts are sent based on the namespace to the Receivers of the tenant that has access to this namespace.

Router CRD is used to send the specified notifications to the specified receivers.

Notification Manager - Router

The **Router** defines a configuration to route the specified alerts to the specified receivers.

The receivers can be selected as follows:

- **name**: a Receiver instance name.
- **regexName**: A regular expression to match the Receiver instance name.
- **selector**: A label selector used to select Receiver instances.
- **type**: The integration type in the Receiver instance.

```
apiversion: notification.kubesphere.io/v2beta2
kind: Router
metadata:
  name: router
spec:
  alertSelector:
    matchExpressions:
    - key: alertname
      operator: In
      values:
      - CPUThrottlingHigh
  receivers:
    name:
    - user1
    regexName: user2.*?
```

Router CRD is used to send the specified notifications to the specified receivers.

Notification Manager – Route

Users can set two matching methods at the same time and set the priorities of the two matching methods to achieve a more flexible routing strategy.

- All：Use both matching methods at the same time.
- RouterFirst：Routing rule matching will be used first, and automatic matching will be used if matching is not successful.
- RouterOnly：If the Receiver is not matched using routing rules, the notification will not be sent.

Notification Manager – Filter

Now we need to filter out invalid or

uninteresting alerts for every selected receivers.

There are two ways to filter alerts:

- Filter using a tenant **Silence.**
- Set a label selector in Receiver.

```yaml
apiVersion: notification.kubesphere.io/v2beta2
kind: Receiver
metadata:
  name: receiver
  labels:
    type: tenant
    user: admin
spec:
  email:
    alertSelector:
      matchExpressions:
        - key: severity
          operator: In
          values:
            - error
            - critical
```

Notification Manager – Template

Now all alerts are usefull for the user, it should send a notification to the user. First we need to generate a message from the alerts.

The notifications are rendered by notification templates first before they're sent to receivers. The Notification Manager comes with a set of default templates and also supports customizing templates.

Users can define global templates as well as templates for each receiver.

```yaml
apiVersion: notification.kubesphere.io/v2beta2
kind: Receiver
metadata:
  name: receiver
  labels:
    type: tenant
    user: admin
spec:
  email:
    to:
    - amdin@kubesphere.io
    template: admin.default.html
    subjectTemplate: nm.default.subject
    tmplType: html
```

Receiver is used to define the notification format and destinations to which notifications will be sent.

Notification Manager – Config

Now everything is ready, it's time to send the notifications to the users, but we are missing some key information, about how to send the notification. maybe it is：

- SMTP server and a email address to send email.
- A slack APP token to send the notification to slack channel.
- AppID and AppSecret for Feishu

So, we need something to define these infromation.

Receiver is used to define the notification format and destinations to which notifications will be sent.

Notification Manager – Config

The **Config** CRD defines the information needed to send notifications, such as SMTP server, DingTalk setting, slack token, etc.

Two type of Configs is categorized by a **type** label:

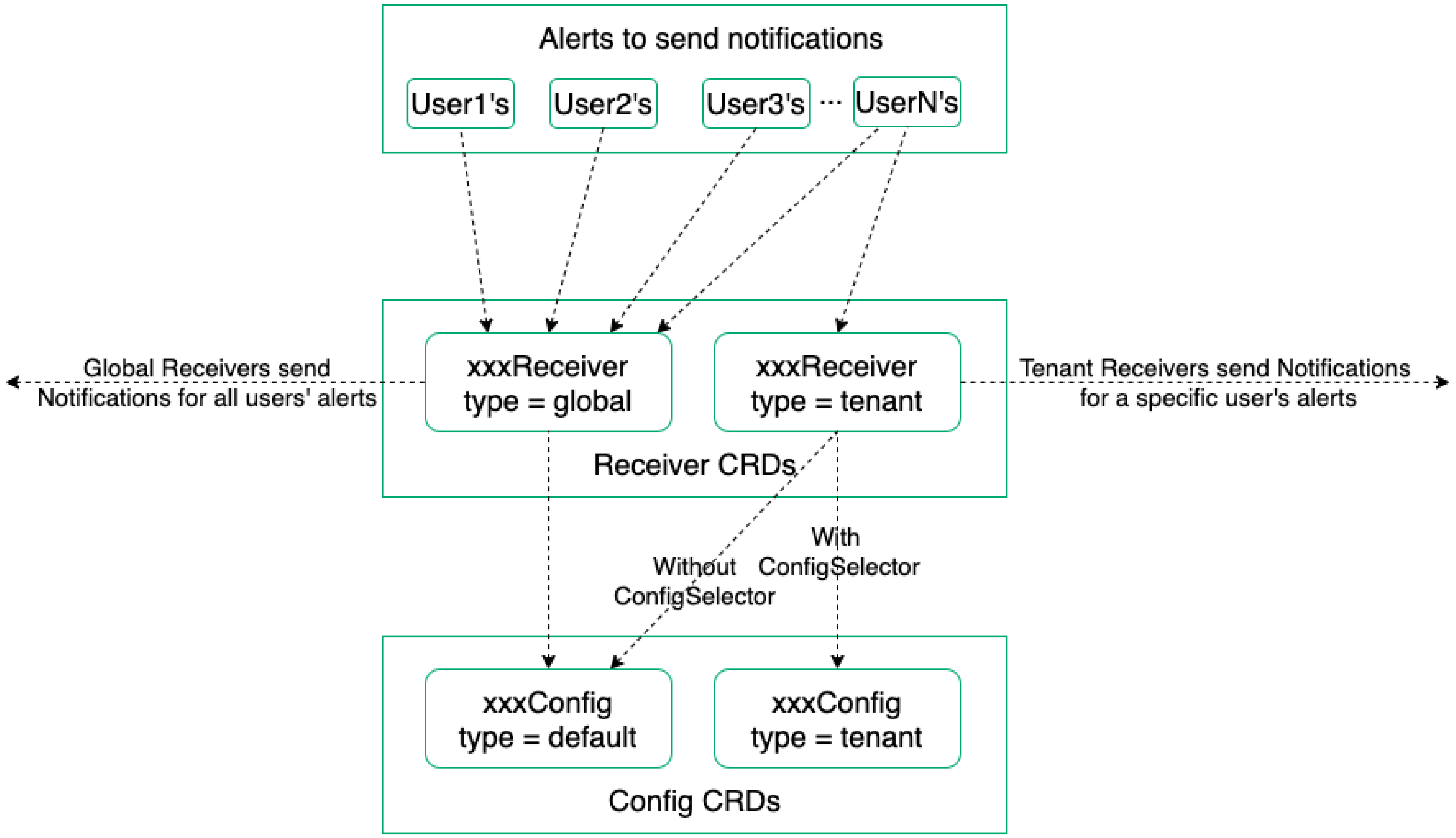
- The tenant config can only be selected by tenant receivers with the same tenant label .
- The default config can be selected by all receivers. Usually admin will set a global default config.

The **spec** defines different fields to point to the corresponding notification integration.

```yaml
apiVersion: notification.kubesphere.io/v2beta2
kind: Config
metadata:
  name: default-config
  labels:
    type: default
spec:
  email:
    authPassword:
      valueFrom:
        secretKeyRef:
          key: password
          name: default-config-secret
    authUsername: test
    from: test@kubesphere.io
    smartHost:
      host: imap.kubesphere.io
      port: 25
```

Receiver is used to define the notification format and destinations to which notifications will be sent.

Notification Manager - Relation


Alerts to send notifications
User1's
User2's
User3's
···
UserN's
Global Receivers send Notifications for all users' alerts
xxxReceiver type = global
xxxReceiver type = tenant
Tenant Receivers send Notifications for a specific user's alerts
Receiver CRDs
With ConfigSelector
Without ConfigSelector
xxxConfig type = default
xxxConfig type = tenant
Config CRDs

KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2023

THANK YOU!