Korrel8r User Guide

Version 0.6.2

Table of Contents

This documentation is part of the Korrel8r project.
It is available in HTML and PDF format.

Overview

Many observability tools and observable systems use their own unique data models, storage technologies, query languages, and related nomenclature.

The differences between these nomenclatures can make it difficult for users to recognize relationships between different Kubernetes cluster resources. As a result, troubleshooting and identifying resources affected by issues in your cluster can be time consuming and difficult.

Korrel8r is a correlation engine for observable signals that relates Kubernetes objects to different types of signal data. Given a *start* object, Korrel8r searches for a chain of rules to related *goal* objects. Korrel8r can also show the *neighbourhood* of all data related to an object of interest.

A REST API can be used by clients to make correlation queries. The REST API can be deployed as a service in a cluster, or run outside the cluster and connect to signal stores inside the cluster.

Signal data can be stored in separate signal stores, and encoded using different schema.

> To navigate between correlated data, Korrel8r needs access to the cluster API and several data stores. Currently Korrel8r can only be deployed by a user with the `cluster-admin` role.
>
> This may change in future, see: Issue 73: Authentication and Authorization for restricted access.

Architecture

About domains

A Korrel8r *domain* represents a family of related objects with a common vocabulary and related storage and query technologies. For examples see the reference documentation.

Each domain defines its own class, object, query, and store. These abstractions allow Korrel8r to treat different domains in the same way.

Class

A subset of objects in a domain with a common schema for serialization. Some domains have many classes. Domain `k8s` has a class for each resource kind. For example `k8s:Pod` and `k8s:DaemonSet`. Other domains are flat, with only a single class, for example Domain `alert`

Store

A source of stored objects from a single domain.

For example:

- Domain `log` uses Loki
- Domain `alert` uses Prometheus
- Domain `k8s` uses the Kubernetes API server

Object

The data associated with an individual signal or resource instance. For example a log record, or a serialized Kubernetes resource.

Query

A query selects a set of objects from a store. The structure of a query depends on the domain.

About objects

Korrel8r works with data *objects* which include *signals* (data about events) and *resources* (data about observable ojects).

Signal types

A Kubernetes cluster generates many types of *signals*, including the following:

Signal Type	Description
Metrics	Counts and measurements of system behavior.
Alerts	Rules that fire when metrics cross important thresholds.
Logs	Application, infrastructure and audit logs from pods and cluster nodes.
Kubernetes events	Describe significant events in a cluster.
Traces	Nested execution spans describing distributed requests.
Network events	TCP and IP level network information.

Resource types

A cluster also contains *resources* which are not generally considered to be signals, but that can be correlated with signals and other objects. Examples of resources include:

Resource Type	Description
k8s resources	Spec and status information.
Run books	Problem solving guides associated with Alerts.
k8s probes	Information about resource state.

About rules

Rules express relationships between classes, possibly in different domains.

A Rule applies to an object of a *start* class, and generates a query for a *goal* class. The start and goal can be in different domains (e.g. `k8s:Pod` → `log:application`)

A rule definition contains a *template* that uses the vocabulary of the *start domain* and generates a *query* in the vocabulary of the *goal domain*. Rules are bridge between domains different vocabularies, schema, labels and query languages.

Installing on a cluster

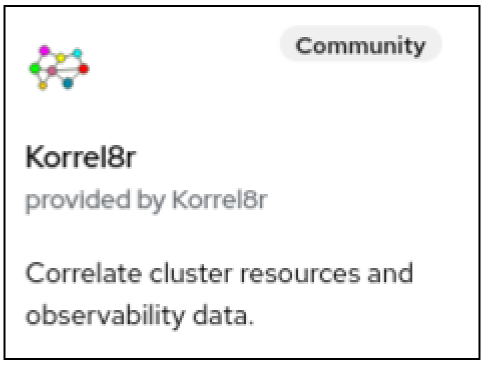
OperatorHub community operator

There is a community operator for Korrel8r on OperatorHub. It has been tested with Red Hat OpenShift Container Platform, it may or may not work with other k8s clusters.

Prerequisites

- You have administrator permissions on an Red Hat OpenShift Container Platform cluster version 4.12 or newer.
- You have installed the Red Hat OpenShift Logging Operator version 5.8.1 or newer provided by Red Hat.
- You have installed the Loki Operator version 5.8.1 or newer provided by Red Hat, and have created a `LokiStack` custom resource (CR).
- You have created a `ClusterLogging` CR that configures LokiStack as the log store.

Procedure

1. In the Red Hat OpenShift Container Platform web console go to `Home > Console > Operators > OperatorHub`
2. Type "korrel8r" in the filter box and select the Korrel8r Community tile

Community
Korrel8r
provided by Korrel8r
Correlate cluster resources and observability data.
3. Click Install, Install again, and you are done!

By default the operator is installed in namespace `openshift-operators`. You can change this during install.

The Korrel8r Resource

Default korrel8r-instance

These examples assume the operator is installed in the namespace `korrel8r`. Modify the example accordingly if it is installed in a different namespace.

The simplest way to run Korrel8r is to create the special *default instance* in namespace `korrel8r` with name `korrel8r-instance`.

Example of a default Korrel8r instance.

```
apiVersion: korrel8r.openshift.io/v1alpha1
kind: Korrel8r
metadata:
  name: korrel8r-instance ①
  namespace: korrel8r ②
  spec: ③
```

① Required name `korrel8r-instance` for the default instance.

② Required namespace `korrel8r` for the default instance.

③ If `spec` is absent or empty, Korrel8r uses a default configuration that connects to the default signal stores for Red Hat OpenShift Container Platform, and integrates with the Red Hat OpenShift Container Platform web console.
You can customize the behaviour of the default instance by adding `spec` fields.

A `ClusterRoleBinding` for the *default instance* is installed as part of the operator. It binds the standard kubernetes `clusterrole/view` to the default instance `ServiceAccount`. You do not need to create this binding or service account.

This allows the default instance to read most cluster resources and connect to observability stores like Prometheus and Loki. See the Kubernetes RBAC documentation for details of the `view` role.

Default instance ClusterRoleBinding installed with the operator. You do not need to create this binding.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/name: korrel8r
  name: korrel8r-instance
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
  name: korrel8r-instance
  namespace: korrel8r
```

Other Korrel8r resources

You can create other instances of Korrel8r in any namespace, with any name. You need to create your own `ClusterRoleBinding` or `RoleBinding` for non-default instances. You do not need to create `ServiceAccount`, the operator will do that automatically.

Example ClusterRoleBinding for Korrel8r my-name in my-namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding ①
metadata:
  name: my-cluster-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view ②
subjects:
- kind: ServiceAccount ③
  name: my-name
  namespace: my-namespace
```

① You can also create a `RoleBinding` to restrict Korrel8r to a single namespace. This will limit correlation results to resources in that namespace.

② You can modify the role to restrict what your Korrel8r instance can read. Korrel8r's functionality will be limited to resources and stores it can access.

③ The operator creates a `ServiceAccount` with the same namespace and name as the `Korrel8r` instance. Your `ClusterRoleBinding` or `RoleBinding` should use this as its subject. You do not need to create this `ServiceAccount`.

Using Korrel8r

On Red Hat OpenShift Container Platform

Ensure that you have installed Korrel8r

1. Refresh the web console
2. Navigate to **Observe** → **Logs**. Click the **Metrics** link to see correlated metrics of each log record.
3. Navigate to **Observe** → **Alerting**, and click an alert. Alerts that have related logs have a **see related logs** link.

Not all alerts have a **see related logs** link. Only alerts that are *related* to container workloads can have related logs. For example, `KubePodCrashLooping` is related to a pod and so can have related logs. See the There is no "see related logs" link documentation for more information.

Command Line

Installing

Install the korrel8r executable:

```
go install github.com/korrel8r/korrel8r/cmd/korrel8r@latest
```

Using

Getting help

```
korrel8r -h
korrel8r list -h
```

Log into your cluster with admin privileges to test korrel8r can read kubernetes resources:

Listing k8s resource types using default configuration URL.

```
korrel8r -c
https://raw.githubusercontent.com/korrel8r/korrel8r/main/etc/korrel8r/korrel8r.yaml
list k8s
```

Browser

The browser interface is experimental may be dropped in future. Other consoles or tools acting as korrel8r clients can provide better visualization.

The `korrel8r web` command provides browser access to:

- Interactive queries and correlation graphs `http://localhost:8080/correlate`
- Interactive REST API documentation on `http://localhost:8080/api`

REST Service

 Under Construction

Configuration

Korrel8r loads configuration from the file specified by the `--config` flag:

```
$ korrel8r --config <path_to_config_file>
```

The default configuration can be loaded from the internet:

```
korrel8r -c
https://raw.githubusercontent.com/korrel8r/korrel8r/main/etc/korrel8r/korrel8r.yaml
get
```

You can examine the default configuration files at the Korrel8r project provides some example configuration files

The configuration file is a YAML file with the following sections:

Include paths to other configuratoin files to include in this one.

```
include:
  - "path_to_config_file"
```

Stores defines how to connect to data stores. See Stores.

```
stores:
  - domain: "domain_name" ①
    # Domain-specific fields ②
```

① Domain name of the store (required).

② Domain-specific fields for connection parameters. See Domain Reference.

Rules defines rules to relate different classes of data. See Rules.

```
rules:
  - name: "rule_name" ①
    start: ②
      domain: "domain_name"
      classes:
        - "class_name"
    goal: ③
      domain: "domain_name"
      classes:
        - "class_name"
    result:
      query: "query_template" ④
```

① Name identifies the rule in graphs and for debugging.

② Start objects for this rule must belong to one of the `classes` in the `domain`.

③ Goal queries generated by this rule may must retrieve one of the `classes` in the `domain`.

④ Result queries are generated by executing the `query` template with the start object as context.

Aliases is a list of short-hand alias names for groups of rules.

```
aliases:
  - name: "alias_name" ①
    domain: "domain_name" ②
    classes: ③
      - "class_name"
```

① Alias name can be used as a class name in rule definitions.

② Domain for classes in this alias.

③ Classes belonging to this alias.

Stores

Every entry in the `stores` section has a `domain` field to identify the domain. Other fields depend on the domain, see Domain Reference. Store fields can contain URL strings or templates.

Example: configuring a store URL from an Openshift Route resource.

```
stores:
  - domain: log
    lokiStack: <-
      {{$r := get "k8s:Route:{namespace: openshift-logging, name: logging-loki}" -}}
①
      https://{{ (first $r).Spec.Host -}} ②
```

① Get a list of routes in "openshift-logging" named "logging-loki".

② Use the .Spec.Host field of the first route as the host for the store URL.

Rules

A rule has the following key elements:

- A set of *start* classes. The rule can apply to objects belonging to one of these classes.
- A set of *goal* classes. The rule can generate queries for any of these classes.
- A result query template that generates a goal query from a start object.

The query template should generate a string of the form:

```
<domain-name>:<class-name>:<query-details>
```

The *query-details* part depends on the domain, see Domain Reference

Example of rules and aliases.

```
FIXME NEED EXAMPLE.
```

Templates

Some fields in the `stores` and `rules` section are Go templates. This is the same template syntax used by the Kubernetes `kubectl` tool with the `--output=template` option. Korrel8r provides some additional *template functions* that can be used in configuration templates.

Additional template functions

- General purpose functions from the slim-sprig library are always available.
- Some domains provide additional functions, see Domain Reference.
- The following functions are always available:
  FIXME DOCUMENT TEMPLATE FUNCTIONS

Troubleshooting

Installation errors on Red Hat OpenShift Container Platform 4.15

Red Hat OpenShift Container Platform 4.15 enforces additional security restrictions that did were not included in Red Hat OpenShift Container Platform 4.14 and earlier versions. Installing the Operator bundle on Red Hat OpenShift Container Platform 4.15 causes security policy errors.

You can use the following workarounds:

- Deploy Korrel8r in the `default` namespace instead of creating a new `korrel8r` namespace, by running the following command:

  ```
  $ operator-sdk -n default run bundle quay.io/korrel8r/operator-bundle:latest
  ```

- Apply labels to the `korrel8r` namespace before installing, by running the following commands:

  ```
  $ kubectl label ns/korrel8r pod-security.kubernetes.io/enforce=privileged
  --overwrite
  ```

  ```
  $ kubectl label ns/korrel8r pod-security.kubernetes.io/warn=privileged --overwrite
  ```

See also https://issues.redhat.com/browse/OU-304.

There is no "see related logs" link

The **see related logs** link does not appear unless the following criteria is met:

1. The alert is *related* to a container workload.
2. The workload has generated logs.
3. The logs have been collected by the logging subsystem for Red Hat OpenShift.

For example, the `UpdateAvailable` alert indicates an update is available for the entire cluster, it is not related to any specific workload.

You can force the creation of an alert with **see related logs** by using the following procedure.

Procedure

1. Run the following command to create a broken deployment in a system namespace:

   ```
   kubectl apply -f - << EOF
   apiVersion: apps/v1
   ```

```
kind: Deployment
metadata:
  name: bad-deployment
  namespace: default ①
spec:
  selector:
    matchLabels:
      app: bad-deployment
  template:
    metadata:
      labels:
        app: bad-deployment
    spec:
      containers: ②
      - name: bad-deployment
        image: quay.io/openshift-logging/vector:5.8
```

① The deployment must be in a system namespace (such as `default`) to cause the desired alerts.

② This container deliberately tries to start a `vector` server with no configuration file. The server will log a few messages, and then exit with an error. Any container could be used for this.

2. View the alerts:

   a. Go to **Observe** → **Alerting** and click **clear all filters**. View the `Pending` alerts.

      Alerts first appear in the `Pending` state. They do not start `Firing` until the container has been crashing for some time. By showing `Pending` alerts you can see them much more quickly.

   b. Look for `KubeContainerWaiting` alerts, `KubePodCrashLooping` alerts, or `KubePodNotReady` alerts. These alerts have a **show related logs** link.

There are no logs for the logging subsystem for Red Hat OpenShift

The log collector does not collect its own logs. Doing so might create a circular condition where collecting its own logs causes the collector to log something, which it then collects, which causes it to log something, and so on in an endless cycle.

To avoid this risk, the log collector does not collect any logs from pods that are part of the logging system.

You can still see view logging subsystem for Red Hat OpenShift logs by using the `kubectl logs` command directly.

Reference

Domains

Reference details for the for the classes, objects, queries and stores of each available domain.

Domain `alert`

Domain `alert` provides Prometheus alerts, queries and access to Thanos and AlertManager stores.

Class

There is a single class `alert:alert`.

Object

An alert object is represented by this Go type. Rules starting from an alert should use the capitalized Go field names rather than the lowercase JSON names.

```
type Object struct {
    // Common fields.
    Labels      map[string]string   `json:"labels"`
    Annotations map[string]string   `json:"annotations"`
    Status      string          `json:"status"` // inactive|pending|firing|suppressed
    StartsAt    time.Time       `json:"startsAt"`

    // Prometheus fields.
    Value       string  `json:"value"`
    Expression  string  `json:"expression"`
    Fingerprint string  `json:"fingerprint"`

    // Alertmanager fields.
    EndsAt      time.Time   `json:"endsAt"`
    UpdatedAt   time.Time   `json:"updatedAt"`
    Receivers   []Receiver  `json:"receivers"`
    InhibitedBy []string    `json:"inhibitedBy"`
    SilencedBy  []string    `json:"silencedBy"`
    GeneratorURL    string      `json:"generatorURL"`
}
```

Query

A JSON map of string names to string values, matched against alert labels, for example:

```
alert:alert:{"alertname":"KubeStatefulSetReplicasMismatch","container":"kube-rbac-
proxy-main","namespace":"openshift-logging"}
```

Store

A client of Prometheus and/or AlertManager. Store configuration:

```
domain: alert
metrics: PROMETHEUS_URL
alertmanager: ALERTMANAGER_URL
```

Either or both of `metrics` or `alertmanager` may be present.

Domain `k8s`

Domain `k8s` implements Kubernetes resources stored in a Kube API server.

Class

A k8s class corresponds to a kind of Kubernetes resource, the class name is `KIND.VERSION.GROUP` VERSION and/or GROUP can be omitted if there is no ambiguity. Example class names: `k8s:Pod.v1`, `ks8:Pod`, `k8s:Deployment.v1.apps`, `k8s:Deployment.apps`, `k8s:Deployment`

Object

Objects are represented by the standard Go types used by `k8s.io/client-go/api`, and by Kube-generated CRD struct types. Rules starting from the k8s domain should use the capitalized Go field names rather than the lowercase JSON field names.

Query

Queries are the JSON-serialized form of this struct:

```
type Query struct {
    // Namespace restricts the search to a namespace.
    Namespace   string  `json:"namespace,omitempty"`
    Name        string  `json:"name,omitempty"`
    // Labels restricts the search to objects with matching label values (optional)
    Labels  client.MatchingLabels   `json:"labels,omitempty"`
    // Fields restricts the search to objects with matching field values (optional)
    Fields  client.MatchingFields   `json:"fields,omitempty"`
    // contains filtered or unexported fields
}
```

For example:

```
k8s:Pod.v1.:{"namespace":"openshift-cluster-version","name":"cluster-version-operator-
8d86bcb65-btlgn"}
```

Store

k8s stores connects to the current logged-in Kubernetes cluster, no other configuration is needed than:

```
domain: k8s
```

Domain `log`

Domain `log` is a domain for openshift-logging ViaQ logs stored in Loki or LokiStack.

Class

There are 3 classes corresponding to the 3 openshift logging log types:

```
log:application
log:infrastructure
log:audit
```

Object

A log object is a JSON map\[string]any in ViaQ format.

Query

A query is a LogQL query string, prefixed by the logging class, for example:

```
log:infrastructure:{ kubernetes_namespace_name="openshift-cluster-version",
kubernetes_pod_name=~".*-operator-.*" }
```

Store

To connect to a lokiStack store use this configuration:

```
domain: log
lokistack: URL_OF_LOKISTACK_PROXY
```

To connect to plain loki store use:

```
domain: log
loki: URL_OF_LOKI
```

Domain `metric`

Domain `metric` represents Prometheus metric samples as objects.

Class

There is only one class: `metric:metric`

Object

A metric sample, which includes a metric time series (name and labels), a timestamp and a value.

Query

Queries are PromQL time series selector strings, prefixed by `metric:metric:` for example:

```
metric:metric:http_requests_total{environment=~"staging|testing|development",method!="
GET"}
```

Store

Prometheus is the store, store configuration:

```
domain: metric
metric: URL_OF_PROMETHEUS
```

Domain `netflow`

Domain `netflow` is a domain for network observability flow events stored in Loki or LokiStack.

Class

There is a single class `netflow:network`

Object

A log object is a JSON `map\[string]any` in NetFlow format.

Query

A query is a LogQL query string, prefixed by `netflow:network:`, for example:

```
netflow:network:{SrcK8S_Type="Pod", SrcK8S_Namespace="myNamespace"}
```

Store

To connect to a netflow lokiStack store use this configuration:

```
domain: netflow
lokistack: URL_OF_LOKISTACK_PROXY
```

To connect to plain loki store use:

```
domain: netflow
loki: URL_OF_LOKI
```

REST API

REST API for the Korrel8r correlation engine.

Version

v1alpha1

License

Apache 2.0

Contact

Project Korrel8r https://github.com/korrel8r/korrel8r

Content negotiation

URI Schemes

- http
- https

Consumes

- application/json

Produces

- application/json

Endpoints by group

configuration

Method	URI	Name	Summary
GET	/api/v1alpha1/configuration	get configuration	Dump configuration files and their contents.
GET	/api/v1alpha1/domains	get domains	List all configured domains and stores.
GET	/api/v1alpha1/domains/{domain}/classes	get domains domain classes	Get class names and descriptions for the domain.

search

Method	URI	Name	Summary
GET	/api/v1alpha1/objects	get objects	Execute a query, returns a list of JSON objects.
POST	/api/v1alpha1/graphs/goals	post graphs goals	Create a correlation graph from start objects to goal queries.
POST	/api/v1alpha1/graphs/neighbours	post graphs neighbours	Create a correlation graph of neighbours of a start object to a given depth.
POST	/api/v1alpha1/lists/goals	post lists goals	Generate a list of goal nodes related to a starting point.

Paths

Dump configuration files and their contents.

```
GET /api/v1alpha1/configuration
```

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

ConfigConfigs

List all configured domains and stores.

```
GET /api/v1alpha1/domains
```

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

[]RestDomain

Get class names and descriptions for the domain.

```
GET /api/v1alpha1/domains/{domain}/classes
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
domain	`path`	string	`string`		required		Domain to get classes from.

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

RestClasses

Execute a query, returns a list of JSON objects.

```
GET /api/v1alpha1/objects
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
query	`query`	string	`string`		required		query string

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

[]interface{}

Create a correlation graph from start objects to goal queries.

```
POST /api/v1alpha1/graphs/goals
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
withRules	`query`	boolean	`bool`		optional		include rules in graph edges
start	`body`	RestGoalsRequest	`models.RestGoalsRequest`		□		search from start to goal classes

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

RestGraph

Create a correlation graph of neighbours of a start object to a given depth.

```
POST /api/v1alpha1/graphs/neighbours
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
withRules	`query`	boolean	`bool`		optional		include rules in graph edges
start	`body`	RestNeighboursRequest	`models.RestNeighboursRequest`		□		search from neighbours

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

RestGraph

Generate a list of goal nodes related to a starting point.

```
POST /api/v1alpha1/lists/goals
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
start	`body`	RestGoalsRequest	`models.RestGoalsRequest`		□		search from start to goal classes

All responses

Code	Status	Description	Has headers	Schema
200	OK	OK		schema

Responses

200 - OK

Status: OK

Schema

[]RestNode

Models

config.Class

Properties

Name	Type	Go type	Required	Default	Description	Example
classes	[]string	`[]string`			Classes are the names of classes in this group.	
domain	string	`string`			Domain of the classes, all must be in the same domain.	
name	string	`string`			Name is the short name for a group of classes.	

config.ClassSpec

Properties

Name	Type	Go type	Required	Default	Description	Example
classes	[]string	`[]string`			Classes is a list of class names to be selected from the domain. If absent, all classes in the domain are selected.	
domain	string	`string`			Domain is the domain for selected classes.	

config.Config

Properties

Name	Type	Go type	Required	Default	Description	Example
aliases	[]ConfigClass	`[]*ConfigClass`			Aliases defines short names for groups of related classes.	
include	[]string	`[]string`			Include lists additional configuration files or URLs to include.	
rules	[]ConfigRule	`[]*ConfigRule`			Rules define the relationships that korrel8r will follow.	
stores	[]ConfigStore	`[]ConfigStore`			Stores is a list of store configurations.	

config.Configs

ConfigConfigs

config.ResultSpec

Properties

Name	Type	Go type	Required	Default	Description	Example
query	string	`string`			Query template generates a query object suitable for the goal store.	

config.Rule

Properties

Name	Type	Go type	Required	Default	Description	Example
goal	ConfigRule	`ConfigRule`			Goal specifies the set of classes that this rule can produce.	
name	string	`string`			Name is a short, descriptive name. If omitted, a name is generated from Start and Goal.	
result	ConfigRule	`ConfigRule`			TemplateResult contains templates to generate the result of applying this rule. Each template is applied to an object from one of the `start` classes. If any template yields a blank string or an error, the rule does not apply.	
start	ConfigRule	`ConfigRule`			Start specifies the set of classes that this rule can apply to.	

config.Store

ConfigStore

korrel8r.Constraint

Properties

Name	Type	Go type	Required	Default	Description	Example
end	string	`string`			Exclude results after End.	
limit	integer	`int64`			Max number of entries to return.	
start	string	`string`			Exclude results before Start.	

rest.Classes

> Classes maps class names to a short description.

RestClasses

rest.Domain

> Domain configuration information.

Properties

Name	Type	Go type	Required	Default	Description	Example
errors	[]string	`[]string`				
name	string	`string`				
stores	[]ConfigStore	`[]ConfigStore`				

rest.Edge

Properties

Name	Type	Go type	Required	Default	Description	Example
goal	string	`string`			Goal is the class name of the goal node.	`domain:class`
rules	[]RestRule	`[]*RestRule`			Rules is the set of rules followed along this edge (optional).	
start	string	`string`			Start is the class name of the start node.	

rest.GoalsRequest

> Starting point for a goals search.

Properties

Name	Type	Go type	Required	Default	Description	Example
goals	[]string	`[]string`			Goal classes for correlation.	`["domain:class"]`
start	RestGoalsRequest	`RestGoalsRequest`			Start of correlation search.	

rest.Graph

> Graph resulting from a correlation search.

Properties

Name	Type	Go type	Required	Default	Description	Example
edges	[]RestEdge	`[]*RestEdge`				
nodes	[]RestNode	`[]*RestNode`				

rest.NeighboursRequest

> Starting point for a neighbours search.

Properties

Name	Type	Go type	Required	Default	Description	Example
depth	integer	`int64`			Max depth of neighbours graph.	
start	RestNeighboursRequest	`RestNeighboursRequest`			Start of correlation search.	

rest.Node

Properties

Name	Type	Go type	Required	Default	Description	Example
class	string	`string`			Class is the full class name in "DOMAIN:CLASS" form.	`domain:class`
count	integer	`int64`			Count of results found for this class, after de-duplication.	
queries	[]RestQueryCount	`[]*RestQueryCount`			Queries yielding results for this class.	

rest.QueryCount

> Query run during a correlation with a count of results found.

Properties

Name	Type	Go type	Required	Default	Description	Example
count	integer	`int64`			Count of results or -1 if the query was not executed.	
query	string	`string`			Query for correlation data.	

rest.Rule

Properties

Name	Type	Go type	Required	Default	Description	Example
name	string	`string`			Name is an optional descriptive name.	
queries	[]RestQueryCount	`[]*RestQueryCount`			Queries generated while following this rule.	

rest.Start

> Starting point for correlation.

Properties

Name	Type	Go type	Required	Default	Description	Example
class	string	`string`			Class of starting objects	
constraint	RestStart	`RestStart`			Constraint (optional) to limit the results.	
objects	interface{}	`interface{}`			Objects serialized as JSON to, must be of start class.	
queries	[]string	`[]string`			Queries for starting objects, must return the start class.	

Kubernetes API

Packages

- korrel8r.openshift.io/v1alpha1

korrel8r.openshift.io/v1alpha1

Package v1alpha1 contains API Schema definitions for the korrel8r v1alpha1 API group.

Resource Types

- Korrel8r

Config

Config wraps the korrel8r Config struct for API code generation.

Appears In:

- Korrel8rSpec

Field	Description
`rules` *Rule array*	Rules define the relationships that korrel8r will follow.
`groups` *Group array*	Groups defines short names for groups of related classes.
`stores` *StoreConfig array*	Stores is a list of store configurations.
`include` *string array*	Include lists additional configuration files or URLs to include.

Korrel8r

Korrel8r is a service that correlates observabililty signals in the cluster.

Field	Description
`apiVersion` *string*	`korrel8r.openshift.io/v1alpha1`
`kind` *string*	`Korrel8r`
`metadata` *ObjectMeta*	Refer to Kubernetes API documentation for fields of `metadata`.
`spec` *Korrel8rSpec*	

Korrel8rSpec

Korrel8rSpec defines the desired state of Korrel8r

Appears In:

- Korrel8r

Field	Description
`config` *Config*	Config is the configuration for a korrel8r deployment. If not provided there is a default configuration suitable for use in an openshift cluster. The "include" section can load additional configuration files provided at /etc/korrel8r
`verbose` *integer*	Verbose sets the numeric logging verbosity for the KORREL8R_VERBOSE environment variable.