Results

Introduction to API Usage

Results List

Results Det

Result records List

Result logs List

Introduction to API Usage

TOC

- Authentication/Authorization
  - Impersonation
    - How to Use Impersonation?
  - Troubleshooting
- Filtering
  - Results
  - Records and Logs
    - The `data` Field in Records
    - The `data` Field in Logs
  - How to Create CEL Filtering Expressions
    - Accessing Fields
    - Using Operators
    - Using Functions
  - Using CEL Filtering Expressions with gRPC
  - Using CEL Filtering Expressions with REST
  - Using CEL Filtering Expressions with `tkn-results`
  - Common Filtering Examples
- Sorting
- Pagination
- Cross-parent Reading Results

Cross-Results Reading Records

Metrics

Health

Check Status

Authentication/Authorization

The reference implementation of the Results API expects to obtain cluster-generated authentication tokens↗ from the cluster in which it runs. In most cases, using a service account will be the simplest way to interact with the API.

RBAC authorization↗ is used to control access to API resources.

The following attributes are recognized:

Attribute	Value
apiGroups	results.tekton.dev
resources	results, records
verbs	create, get, list, update, delete

For example, a read-only role may look as follows:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: tekton-results-readonly
rules:
  - apiGroups: ["results.tekton.dev"]
    resources: ["results", "records"]
    verbs: ["get", "list"]
```

In the reference implementation, all permissions are namespace-scoped (this is what serves as the parent resource for the API).

For convenience, the following [ClusterRoles] define common access patterns:

ClusterRole	Description
tekton-results-readonly	Read-only access to all Results API resources
tekton-results-readwrite	Includes `tekton-results-readonly` + create or update all Results API resources
tekton-results-admin	Includes `tekton-results-readwrite` + permission to delete Results API resources

Impersonation

Kubernetes impersonation ↗ is used to refine access to the API.

How to Use Impersonation?

- Set the feature flag `AUTH_IMPERSONATE` to `true` in the API server configuration.
- Create two `ServiceAccount`s, one for managing impersonation permissions and the other for the permissions of the impersonated user.

```
kubectl create serviceaccount impersonate-admin -n tekton-pipelines
```

```
kubectl create serviceaccount impersonate-user -n user-namespace
```

- Create the following `ClusterRole` and `ClusterRoleBinding` for the `impersonate-admin` service account.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tekton-results-impersonate
rules:
  - apiGroups: [""]
    resources: ["users", "groups", "serviceaccounts"]
    verbs: ["impersonate"]
  - apiGroups: ["authentication.k8s.io"]
    resources: ["uids"]
    verbs: ["impersonate"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tekton-results-impersonate
subjects:
  - kind: ServiceAccount
    name: impersonate-admin
    namespace: tekton-pipelines
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-results-impersonate
```

- Finally, create a `RoleBinding` for the `impersonate-user` service account.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: tekton-results-user
  namespace: user-namespace
subjects:
  - kind: ServiceAccount
    name: impersonate-user
    namespace: user-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tekton-results-readonly
```

- Now retrieve the token for the `impersonate-admin` service account and store it in a variable.

```
token=$(kubectl create token impersonate-admin -n tekton-pipelines)
```

- You can then call the API in the following format:

```
curl -s --cacert /var/tmp/tekton/ssl/tls.crt  \
  -H 'authorization: Bearer '${token} \
  -H 'Impersonate-User: system:serviceaccount:user-namespace:impersonat
e-user' \
  https://localhost:8080/apis/results.tekton.dev/v1alpha2/parents/user-
namespace/results
```

If the API server uses TLS, you will need to provide the TLS certificate.

Troubleshooting

You can run the following command to query the permissions of the cluster. This is very useful for debugging permission denied errors:

```
kubectl create --as=system:serviceaccount:tekton-pipelines:tekton-results
-watcher -n tekton-pipelines -f - -o yaml << EOF
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
spec:
  resourceAttributes:
    group: results.tekton.dev
    resource: results
    verb: get
EOF
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
metadata:
  creationTimestamp: null
  managedFields:
  - apiVersion: authorization.k8s.io/v1
    fieldsType: FieldsV1
    fieldsV1:
      f:spec:
        f:resourceAttributes:
          .: {}
          f:group: {}
          f:resource: {}
          f:verb: {}
    manager: kubectl
    operation: Update
    time: "2021-02-02T22:37:32Z"
spec:
  resourceAttributes:
    group: results.tekton.dev
    resource: results
    verb: get
status:
  allowed: true
  reason: 'RBAC: allowed by ClusterRoleBinding "tekton-results-watcher" o
f ClusterRole
    "tekton-results-watcher" to ServiceAccount "tekton-results-watcher/te
kton-pipelines"'
```

Filtering

The reference implementation of the Results API uses CEL ↗ as its filtering specification. The filtering specification expects a boolean Results value. This document covers a small portion of CEL that is useful for filtering Results and records.

Results

The following is a mapping between Results JSON/protobuf fields and CEL references:

Field	CEL Reference Field	Description
-	`parent`	The name of the parent (workspace/namespace) of Results.
`uid`	`uid`	The unique identifier of Results.
`annotations`	`annotations`	Annotations added to Results.
`summary`	`summary`	The summary of Results.
`createTime`	`create_time`	The creation time of Results.
`updateTime`	`update_time`	The last update time of Results.

The `summary.status` field is an enumeration and must be used without quotes in filtering expressions (`'` or `"`). Possible values are:

- UNKNOWN
- SUCCESS
- FAILURE
- TIMEOUT
- CANCELLED

Records and Logs

The following is a mapping between record JSON/protobuf fields and CEL references:

Field	CEL Reference Field	Description
`name`	`name`	The name of the record
`data.type`	`data_type`	The type identifier of record data. Please see the values below.
`data.value`	`data`	The data contained in the record. In JSON and protobuf responses, it appears as a base 64 encoded string.

Possible values for `data_type` and `summary.type` (for Results) are:

- `tekton.dev/v1beta1.TaskRun` or `TASK_RUN`
- `tekton.dev/v1beta1.PipelineRun` or `PIPELINE_RUN`
- `results.tekton.dev/v1alpha2.Log`

The `data` Field in Records

The `data` field is a base64 encoded string of an object manifest. If you directly request this data using CLI, REST, or gRPC, you will receive a base64 encoded string. You can decode it using the `base64 -d` command. While it's not human-readable, you can directly filter the response using filters without decoding it.

Here is an example of a JSON object contained in the `data` field of records. This can be directly mapped to the YAML representation we typically use.

```
{
  "kind": "PipelineRun",
  "spec": {
    "timeout": "1h0m0s",
    "pipelineSpec": {
      "tasks": [
        {
          "name": "hello",
          "taskSpec": {
            "spec": null,
            "steps": [
              {
                "name": "hello",
                "image": "ubuntu",
                "script": "echo hello world!",
                "resources": {}
              }
            ],
            "metadata": {}
          }
        }
      ]
    },
    "serviceAccountName": "default"
  },
  "status": {
    "startTime": "2023-08-22T09:08:59Z",
    "conditions": [
      {
        "type": "Succeeded",
        "reason": "Succeeded",
        "status": "True",
        "message": "Tasks Completed: 1 (Failed: 0, Cancelled 0), Skipped:
0",
        "lastTransitionTime": "2023-08-22T09:09:31Z"
      }
    ],
    "pipelineSpec": {
      "tasks": [
        {
          "name": "hello",
          "taskSpec": {
            "spec": null,
```

```
            "steps": [
              {
                "name": "hello",
                "image": "ubuntu",
                "script": "echo hello world!",
                "resources": {}
              }
            ],
            "metadata": {}
          }
        }
      ]
    },
    "completionTime": "2023-08-22T09:09:31Z",
    "childReferences": [
      {
        "kind": "TaskRun",
        "name": "hello-hello",
        "apiVersion": "tekton.dev/v1beta1",
        "pipelineTaskName": "hello"
      }
    ]
  },
  "metadata": {
    "uid": "1638b693-844d-4f13-b767-d7d84ac4ab3d",
    "name": "hello",
    "labels": {
      "tekton.dev/pipeline": "hello"
    },
    "namespace": "default",
    "generation": 1,
    "annotations": {
      "results.tekton.dev/record": "default/results/1638b693-844d-4f13-b7
67-d7d84ac4ab3d/records/1638b693-844d-4f13-b767-d7d84ac4ab3d",
      "results.tekton.dev/result": "default/results/1638b693-844d-4f13-b7
67-d7d84ac4ab3d",
      "results.tekton.dev/resultAnnotations": "{\"repo\": \"tektoncd/resu
lts\", \"commit\": \"1a6b908\"}",
      "results.tekton.dev/recordSummaryAnnotations": "{\"foo\": \"bar
\"}",
      "kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion
\":\"tekton.dev/v1beta1\",\"kind\":\"PipelineRun\",\"metadata\":{\"annota
tions\":{\"results.tekton.dev/recordSummaryAnnotations\":\"{\\\"foo\\\":
\\\"bar\\\"}\",\"results.tekton.dev/resultAnnotations\":\"{\\\"repo\\\":
```

```
\\\"tektoncd/results\\\", \\\"commit\\\": \\\"1a6b908\\\"}\"},\"name
\":\"hello\",\"namespace\":\"default\"},\"spec\":{\"pipelineSpec\":{\"tas
ks\":[{\"name\":\"hello\",\"taskSpec\":{\"steps\":[{\"image\":\"ubuntu
\",\"name\":\"hello\",\"script\":\"echo hello world!\"}]}}]}}}\n"
    },
    "managedFields": [
      {
        "time": "2023-08-22T09:08:59Z",
        "manager": "controller",
        "fieldsV1": {
          "f:metadata": {
            "f:labels": {
              ".": {},
              "f:tekton.dev/pipeline": {}
            }
          }
        },
        "operation": "Update",
        "apiVersion": "tekton.dev/v1beta1",
        "fieldsType": "FieldsV1"
      },
      {
        "time": "2023-08-22T09:08:59Z",
        "manager": "kubectl-client-side-apply",
        "fieldsV1": {
          "f:spec": {
            ".": {},
            "f:pipelineSpec": {
              ".": {},
              "f:tasks": {}
            }
          },
          "f:metadata": {
            "f:annotations": {
              ".": {},
              "f:results.tekton.dev/resultAnnotations": {},
              "f:results.tekton.dev/recordSummaryAnnotations": {},
              "f:kubectl.kubernetes.io/last-applied-configuration": {}
            }
          }
        },
        "operation": "Update",
        "apiVersion": "tekton.dev/v1beta1",
        "fieldsType": "FieldsV1"
```

```
      },
      {
        "time": "2023-08-22T09:08:59Z",
        "manager": "watcher",
        "fieldsV1": {
          "f:metadata": {
            "f:annotations": {
              "f:results.tekton.dev/record": {},
              "f:results.tekton.dev/result": {}
            }
          }
        },
        "operation": "Update",
        "apiVersion": "tekton.dev/v1beta1",
        "fieldsType": "FieldsV1"
      },
      {
        "time": "2023-08-22T09:09:31Z",
        "manager": "controller",
        "fieldsV1": {
          "f:status": {
            ".": {},
            "f:startTime": {},
            "f:conditions": {},
            "f:pipelineSpec": {
              ".": {},
              "f:tasks": {}
            },
            "f:completionTime": {},
            "f:childReferences": {}
          }
        },
        "operation": "Update",
        "apiVersion": "tekton.dev/v1beta1",
        "fieldsType": "FieldsV1",
        "subresource": "status"
      }
    ],
    "resourceVersion": "1567",
    "creationTimestamp": "2023-08-22T09:08:59Z"
  },
  "apiVersion": "tekton.dev/v1beta1"
}
```

You can now access the required fields using dot notation and treat `data` as the parent object. For example:

Purpose	Filtering Expression
The name of the PipelineRun	`data.metadata.name`
The name of the ServiceAccount used in the PipelineRun	`data.spec.serviceAccountName`
The name of the first task in the PipelineRun	`data.spec.pipelineSpec.tasks[0].name`
The start time of the PipelineRun	`data.status.startTime`
The labels of the PipelineRun	`data.metadata.labels`
The annotations of the PipelineRun	`data.metadata.annotations`
A specific annotation of the PipelineRun	`data.metadata.annotations['foo']`

The `data` Field in Logs

The `data` field is a base64 encoded custom object of type `Log`. Accessing fields is similar to records. Here is an example of a JSON object contained in the `data` field of logs.

```
{
  "kind": "Log",
  "spec": {
    "type": "File",
    "resource": {
      "uid": "dbe14a60-1fc8-458e-a49b-264771557c3e",
      "kind": "TaskRun",
      "name": "hello",
      "namespace": "default"
    }
  },
  "status": {
    "path": "default/2d27dde5-e201-35f9-b658-2456f2955903/hello-log",
    "size": 83
  },
  "metadata": {
    "uid": "2d27dde5-e201-35f9-b658-2456f2955903",
    "name": "hello-log",
    "namespace": "default",
    "creationTimestamp": null
  },
  "apiVersion": "results.tekton.dev/v1alpha2"
}
```


Here are some examples of accessing log fields using dot notation:

Purpose	Filtering Expression
The type of running that created this log	`data.spec.resource.kind`
The size of the log	`data.status.size`
The name of the running that created this log	`data.spec.resource.name`

How to Create CEL Filtering Expressions

CEL expressions consist of identifiers, literals, operators, and functions. Here, we will learn how to create CEL filtering expressions using the fields mentioned above. This is not an

exhaustive list of CEL expressions. For more information, please refer to the CEL specification ↗.

Accessing Fields

CEL expressions typically correspond one-to-one with JSON/protobuf fields. In Tekton Results, we created some additional aliases for convenient access. You can see all of these in the table above. Additionally, you can use dot notation to access any field in the JSON/protobuf objects. See the examples in the table below.

Purpose	Filtering Expression	Description
The status of Results	`summary.status`	The `status` of Results is a sub-object of `summary`.
The data type of the record	`data_type`	`data_type` is an alias for the `data.type` of the record.
The data type of Results	`summary.type`	The `type` of Results is a sub-object of `summary`.
The name of the first step of the running task retrieved from status	`data.status.taskSpec.steps[0].name`	The JSON path is `data -> status -> taskSpec -> steps -> 0 -> name`.

Using Operators

Now that we can access fields, you can create filters using operators. Here's a list of operators that can be used in CEL expressions:

Operator	Description	Example
`==`	Equality	`data_type == "tekton.dev/v1beta1.TaskRun"`
`!=`	Inequality	`summary.status != SUCCESS`

Operator	Description	Example
`IN`	In list	`data.metadata.name in ['hello', 'foo', 'bar']`
`!`	Negation	`!(data.status.name in ['hello', 'foo', 'bar'])`
`&&`	Logical and	`data_type == "tekton.dev/v1beta1.TaskRun" && name.startsWith("foo/results/bar")`
`||`	Logical or	`data_type == "tekton.dev/v1beta1.TaskRun" || data_type == "tekton.dev/v1beta1.PipelineRun"`
`+`, `-`, `*`, `/`, `%`	Arithmetic operators	`data.status.completionTime - data.status.startTime > duration('5m')`
`>`, `>=`, `<`, `<=`	Comparison operators	`data.status.completionTime > data.status.startTime`

Using Functions

Many functions can be used in CEL expressions. Here's a list of functions that can be used in CEL expressions. The strings in function parameters represent the expected type of the arguments:

Function	Description	Example
`startsWith('string')`	Check if a string starts with a certain prefix	`data.metadata.name.startsWith("foo")`
`endsWith('string')`	Check if a string ends with a certain suffix	`data.metadata.name.endsWith("bar")`

Function	Description	Example
`contains('string')`	Check if a field exists or an object contains a key or value	`data.metadata.annotations.contains('bar')`
`timestamp('RFC3339-timestamp')`	For comparing timestamps	`data.status.startTime > timestamp("2021-02-02T22:37:32Z")`
`getDate()`	Returns the date from a timestamp	`data.status.completionTime.getDate() == 7`
`getDayOfWeek()`, `getDayOfMonth()`, `getDayOfYear()`	Returns the day of the week, day of the month, or day of the year from a timestamp	`data.status.completionTime.getDayOfMonth() == 7`
`getFullYear()`	Returns the year from a timestamp	`data.status.startTime.getFullYear() == 2023`
`getHours()`, `getMinutes()`, `getSeconds()`	Returns hours, minutes, or seconds from a timestamp	`data.status.completionTime.getHours() >= 9`
`string('input')`	Converts valid input to a string	`string(data.status.completionTime) == "2021-02-02T22:37:32Z"`
`matches('regex')`	Checks if a string matches a regular expression	`name.matches("^foo.*$")`

You can also nest function calls and mix operators to create complex filtering expressions. Make sure to use the correct types for function arguments. You can see a more in-depth

function reference in the CEL specification. The functions mentioned above are the most commonly used and effective.

Using CEL Filtering Expressions with gRPC

You can pass a filter to gRPC requests by specifying `filter=<cel-expression>`. Be sure to use the correct quoting in your queries or escape if necessary. Here's an example:

```
grpc_cli call --channel_creds_type=ssl \
  --ssl_target=tekton-results-api-service.tekton-pipelines.svc.cluster.lo
cal \
  --call_creds=access_token=$ACCESS_TOKEN \
  localhost:8080 tekton.results.v1alpha2.Results.ListResults \
  'parent:"default",filter:"data_type==TASK_RUN"'
```

Using CEL Filtering Expressions with REST

You can pass a filter to REST requests by specifying `filter=<cel-expression>` in your query. Here's an example:

```
curl --insecure \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "Accept: application/json" \
  https://localhost:8080/apis/results.tekton.dev/v1alpha2/parents/-/resul
ts/-?filter=data.status.completionTime.getDate()==7
```

Using CEL Filtering Expressions with `tkn-results`

If you have installed `tkn-results` CLI independently or as a plugin to `tkn`, you can filter Results using the `--filter=<cel-expression>` flag. Here's an example:

```
tkn results records list default/results/- --filter="data.metadata.annota
tions.contains('bar')"
```

Common Filtering Examples

These examples showcase the most common filtering expressions that are very useful for everyday use. Remember that not all of these filters apply to Results, records, and logs. You must provide the correct filter for the correct resource.

Purpose	Filtering Expression
Get all records where TaskRun/PipelineRun name is `hello`	`data.metadata.name == 'hello'`
Get all records of TaskRun belonging to PipelineRun 'foo'	`data.metadata.labels['tekton.dev/pipelineRun'] == 'foo'`
Get all records of TaskRun/PipelineRun belonging to Pipeline 'bar'	`data.metadata.labels['tekton.dev/pipeline'] == 'bar'`
The same as the above query, but I only want PipelineRun	`data.metadata.labels['tekton.dev/pipeline'] == 'bar' && data_type == 'PIPELINE_RUN'`
Get records of TaskRun whose names start with `hello`	`data.metadata.name.startsWith('hello')&&dat_type==TASK_RUN`
Get Results of all successful TaskRuns	`summary.status == SUCCESS && summary.type == 'TASK_RUN'`
Get records of PipelineRun whose completion time exceeds 5 minutes	`data.status.completionTime - data.status.startTime > duration('5m') && data_type == 'PIPELINE_RUN'`
Get records of runs completed today	`data.status.completionTime.getDate() == 7`

Purpose	Filtering Expression
(assuming today is the 7th)	
Get records of PipelineRun that have annotation containing `bar`	`data.metadata.annotations.contains('bar') && data_type == 'PIPELINE_RUN'`
Get records of PipelineRun with an annotation containing `bar` and whose names start with `foo`	`data.metadata.annotations.contains('bar') && data.metadata.name.startsWith('foo') && data_type == 'PIPELINE_RUN'`
Get Results containing annotations `foo` and `bar`	`summary.annotations.contains('foo') && summary.annotations.contains('bar')`
Get Results of all failed runs	`!(summary.status == SUCCESS)`
Get records of all failed runs	`!(data.status.conditions[0].status == 'True')`
Get all records of PipelineRun with 3 or more tasks	`size(data.status.pipelineSpec.tasks) >= 3 && data_type == 'PIPELINE_RUN'`

Sorting

The reference implementation of the Results API supports sorting Results and records responses using optional direction qualifiers (`asc` or `desc`).

To request a list of objects in a specific order, include the `order_by` query parameter in the request. Pass the name of the field to be sorted. Multiple fields can be specified using a comma-separated list. Examples:

- `create_time`

- `update_time asc`
- `create_time desc, update_time asc`

Fields supported in `order_by`:

Field Name
`create_time`
`update_time`

Pagination

The reference implementation of the Results API supports pagination for Results, records, and logs. The default number of objects in a single page is 50, with a maximum of 10,000.

To paginate responses, include the `page_size` query parameter in the request. It must be an integer value between 0 and 10,000. If `page_size` is less than the total number of objects available for a specific query, the response will contain a `NextPageToken`. You can pass this value to the `page_token` query parameter to retrieve the next page. These two queries are independent and can be used separately or together.

Name	Description
`page_size`	The number of objects to retrieve in the response.
`page_token`	The token for the page to be retrieved.

Cross-parent Reading Results

You can read Results across parents by specifying `-` as the parent name. This is useful for listing all Results stored in the system without prior knowledge of the available parents.

Cross-Results Reading Records

You can read records across Results by specifying `-` as the Result name portion, or `-` as the parent name. (For example, `default/results/-` or `-/results/-`). This can be used to read and filter matching records when the exact Result name is unknown.

Metrics

The API server includes an HTTP server to expose gRPC server Prometheus metrics. By default, metrics are exposed on port `:9090`. For more detailed information about the metrics structure, see https://github.com/grpc-ecosystem/go-grpc-prometheus#metrics↗.

Health

The API server includes gRPC and REST endpoints to monitor the service status of the API server and the individual services.

Check Status

```
# Check the status of the API server using gRPC
grpcurl --insecure localhost:8080 grpc.health.v1.Health/Check

# Check the status of an individual service using gRPC
grpcurl --insecure -d '{"service": "tekton.results.v1alpha2.Results"}' lo
calhost:8080 grpc.health.v1.Health/Check

# Check the status of the API server using REST
curl -k https://localhost:8080/healthz

# Check the status of an individual service using REST
curl -k https://localhost:8080/healthz?service=tekton.results.v1alpha2.Re
sults
```

Results List

/v1alpha2/parents/{parent}/results

Common Parameters

- `parent` (*in path*): `string` required
  Parent name refers to the namespace name or workspace name.
- `filter` (*in query*): `string`
  This query can be used to pass CEL Expressions to filter the response. See more details here↗.
- `page_token` (*in query*): `string`
  It can be used to fetch the next set of responses when the response is paginated. It corresponds to `NextPageToken` in the response.
- `page_size` (*in query*): `integer`
  Number of response to fetch in one request. This query can be used for pagination.
- `order_by` (*in query*): `string`
  This query can be used to order the response based on parameters. More details can be found here↗. Note: Add `%20` instead of space when adding an optional direction qualifier, e.g `created_by%20asc`.

`get` Get the list of the Results

Results can be read across parents by specifying `-` as the `parent`. This is useful for listing all results stored in the system without prior knowledge about the available parents.

Response

- `200` `object` : List of Results with nextPageToken
  Properties:
  - `results` : `[]Result`
  - `nextPageToken` : `string`

ResultsList

List of Results with nextPageToken

Result

Results are aggregators of Records, allowing users to refer to groups of Records as a single entity.

- `annotations` : `object`
  Arbitrary user provided labels for the result.
- `uid` : `string`
  server assigned identified of the Results
- `name` : `string`
  User assigned identifier of the Result. Encodes Parent information.
- `summary` : RecordSummary
  RecordSummary is a high level overview of a Record, typically representing a "root" record for a result. It includes type agonstic information so that UI's and other tools do not need to be aware of underlying types.
- `createdTime` : `string`
  DEPRECATED: Use createTime instead
- `createTime` : `string`
  Server assigned timestamp for when the result was created.
- `updatedTime` : `string`
  DEPRECATED: Use updateTime instead

- `updateTime` : `string`
  Server assigned timestamp for when the result was updated.
- `id` : `string`
  DEPRECATED: use uid instead
- `etag` : `string`
  This is etag for Results. If this is provided on update, it must match the server's etag.

RecordSummary

RecordSummary is a high level overview of a Record, typically representing a "root" record for a result. It includes type agonstic information so that UI's and other tools do not need to be aware of underlying types.

- `record` : `string`
  The name of the Record this summary represents.
- `type` : `string`
  Identifier of underlying data.
- `status` : `string`
  Completion status of the Record.
- `startTime` : `string`
- `endTime` : `string`
- `annotations` : `object`
  Key-value pairs representing abitrary underlying record data that clients want to include that aren't covered by the above fields.

Status

Completion status of the Record.

Results Details

/v1alpha2/parents/{parent}/results/{result_uid}

Common Parameters

- `parent` (*in path*): `string` required
  Parent name refers to the namespace name or workspace name.
- `result_uid` (*in path*): `string` required
  Result UID is the server assigned identifier of the result.

`get` Get a single result given the UID

Response

- `200` Result: Default response

`post` Create a Result given data and UID

Request Body

Result required

Response

- `200` Result: Default response

`delete` Delete a particular result using UID

Response

- `200` Result: Default response

`patch` Update result given the UID

Request Body

Result required

Response

- `200` Result: Default response

Result

Results are aggregators of Records, allowing users to refer to groups of Records as a single entity.

- `annotations` : `object`
  Arbitrary user provided labels for the result.
- `uid` : `string`
  server assigned identified of the Results
- `name` : `string`
  User assigned identifier of the Result. Encodes Parent information.
- `summary` : RecordSummary
  RecordSummary is a high level overview of a Record, typically representing a "root" record for a result. It includes type agonstic information so that UI's and other tools do not need to be aware of underlying types.
- `createdTime` : `string`
  DEPRECATED: Use createTime instead
- `createTime` : `string`

Server assigned timestamp for when the result was created.

- `updatedTime` : `string`
  DEPRECATED: Use updateTime instead
- `updateTime` : `string`
  Server assigned timestamp for when the result was updated.
- `id` : `string`
  DEPRECATED: use uid instead
- `etag` : `string`
  This is etag for Results. If this is provided on update, it must match the server's etag.

RecordSummary

RecordSummary is a high level overview of a Record, typically representing a "root" record for a result. It includes type agonstic information so that UI's and other tools do not need to be aware of underlying types.

- `record` : `string`
  The name of the Record this summary represents.
- `type` : `string`
  Identifier of underlying data.
- `status` : `string`
  Completion status of the Record.
- `startTime` : `string`
- `endTime` : `string`
- `annotations` : `object`
  Key-value pairs representing abitrary underlying record data that clients want to include that aren't covered by the above fields.

Status

Completion status of the Record.

Result records List

/v1alpha2/parents/{parent}/results/{result_uid}/records

Common Parameters

- `parent` (*in path*): `string` required
  Parent name refers to the namespace name or workspace name.
- `result_uid` (*in path*): `string` required
  Result UID is the server assigned identifier of the result.
- `filter` (*in query*): `string`
  This query can be used to pass CEL Expressions to filter the response. See more details here↗.
- `order_by` (*in query*): `string`
  This query can be used to order the response based on parameters. More details can be found here↗. Note: Add `%20` instead of space when adding an optional direction qualifier, e.g `created_by%20asc`.
- `page_size` (*in query*): `integer`
  Number of response to fetch in one request. This query can be used for pagination.
- `page_token` (*in query*): `string`
  It can be used to fetch the next set of responses when the response is paginated. It corresponds to `NextPageToken` in the response.

`get` Get list of records

Records can be read across Results by specifying `-` as the `result_uid` or across parents by specifying `-` as the `parent` . (e.g., **default/results/-** or **-/results/-**). This can be used to read and filter matching Records without knowing the exact Result name.

Response

- `200` `object` : List of Records with nextPageToken.
  Properties:
  - `records` : `[]Record`
  - `nextPageToken` : `string`

RecordsList

List of Records with nextPageToken.

Record

Record belonging to a Result. Typically will be Tekton TaskRun/PipelineRun, but may also include other execution information e.g alternative configs, DSLs, input payloads, post-execution actions etc.

- `id` : `string`
  DEPRECATED: use uid instead.
- `uid` : `string`
  Server assigned identifier of the Record.
- `etag` : `string`
  The etag for this record. If this is provided on update, it must match the server's etag.
- `name` : `string`
  Resource name, must be rooted in parent result
- `createdTime` : `string`
  DEPRECATED: use createTime instead.

- `createTime` : `string`
  Server assigned timestamp for when the record was created.
- `updatedTime` : `string`
  DEPRECATED: use updateTime instead.
- `updateTime` : `string`
  Server assigned timestamp for when the record was updated.
- `data` : Any
  Any represents lossely typed data to be stored within a Record.

Any

Any represents lossely typed data to be stored within a Record.

- `value` : `string`
  JSON encoded data.
- `type` : `string`
  It is unique identifier if the data type stored in the value. This is used as a type to hint to determine how to unmarshal values. Limited to 128 characters. Currently RecordType can be one of the following:
  - PipelineRun Record: `tekton.dev/v1beta1.PipelineRun` (older records) or `tekton.dev/v1.PipelineRun`
  - TaskRun Record: `tekton.dev/v1beta1.TaskRun` (older records) or `tekton.dev/v1.TaskRun`
  - Log Record: `results.tekton.dev/v1alpha2.Log` (older records) or `results.tekton.dev/v1alpha3.Log`

RecordType

It is unique identifier if the data type stored in the value. This is used as a type to hint to determine how to unmarshal values. Limited to 128 characters. Currently RecordType can be one of the following:

- PipelineRun Record: `tekton.dev/v1beta1.PipelineRun` (older records) or `tekton.dev/v1.PipelineRun`
- TaskRun Record: `tekton.dev/v1beta1.TaskRun` (older records) or `tekton.dev/v1.TaskRun`
- Log Record: `results.tekton.dev/v1alpha2.Log` (older records) or `results.tekton.dev/v1alpha3.Log`

Result logs List

/v1alpha2/parents/{parent}/results/{result_uid}/logs

Common Parameters

- `parent` (*in path*): `string` required
  Parent name refers to the namespace name or workspace name.
- `result_uid` (*in path*): `string` required
  Result UID is the server assigned identifier of the result.
- `filter` (*in query*): `string`
  This query can be used to pass CEL Expressions to filter the response. See more details here↗.
- `page_size` (*in query*): `integer`
  Number of response to fetch in one request. This query can be used for pagination.
- `page_token` (*in query*): `string`
  It can be used to fetch the next set of responses when the response is paginated. It corresponds to `NextPageToken` in the response.
- `order_by` (*in query*): `string`
  This query can be used to order the response based on parameters. More details can be found here↗. Note: Add `%20` instead of space when adding an optional direction qualifier, e.g `created_by%20asc`.

`get` List Logs given the Result UID

Logs can be read across Results by specifying `-` as the `result_uid` or across parents by specifying `-` as the `parent`. (e.g., **default/results/-** or **-/results/-**). This can be used to read and filter matching Logs without knowing the exact Result name.

Response

- `200` `object` : List of Records with nextPageToken.
  Properties:
  - `records` : `[]Record`
  - `nextPageToken` : `string`

RecordsList

List of Records with nextPageToken.

Record

Record belonging to a Result. Typically will be Tekton TaskRun/PipelineRun, but may also include other execution information e.g alternative configs, DSLs, input payloads, post-execution actions etc.

- `id` : `string`
  DEPRECATED: use uid instead.
- `uid` : `string`
  Server assigned identifier of the Record.
- `etag` : `string`
  The etag for this record. If this is provided on update, it must match the server's etag.
- `name` : `string`
  Resource name, must be rooted in parent result
- `createdTime` : `string`
  DEPRECATED: use createTime instead.

- `createTime` : `string`
  Server assigned timestamp for when the record was created.
- `updatedTime` : `string`
  DEPRECATED: use updateTime instead.
- `updateTime` : `string`
  Server assigned timestamp for when the record was updated.
- `data` : Any
  Any represents lossely typed data to be stored within a Record.

Any

Any represents lossely typed data to be stored within a Record.

- `value` : `string`
  JSON encoded data.
- `type` : `string`
  It is unique identifier if the data type stored in the value. This is used as a type to hint to determine how to unmarshal values. Limited to 128 characters. Currently RecordType can be one of the following:
  - PipelineRun Record: `tekton.dev/v1beta1.PipelineRun` (older records) or `tekton.dev/v1.PipelineRun`
  - TaskRun Record: `tekton.dev/v1beta1.TaskRun` (older records) or `tekton.dev/v1.TaskRun`
  - Log Record: `results.tekton.dev/v1alpha2.Log` (older records) or `results.tekton.dev/v1alpha3.Log`

RecordType

It is unique identifier if the data type stored in the value. This is used as a type to hint to determine how to unmarshal values. Limited to 128 characters. Currently RecordType can be one of the following:

- PipelineRun Record: `tekton.dev/v1beta1.PipelineRun` (older records) or `tekton.dev/v1.PipelineRun`
- TaskRun Record: `tekton.dev/v1beta1.TaskRun` (older records) or `tekton.dev/v1.TaskRun`
- Log Record: `results.tekton.dev/v1alpha2.Log` (older records) or `results.tekton.dev/v1alpha3.Log`