Jobs Subsystem

SciCatCon 2025

Spencer Bliven, Despina Adamopoulou, Sofya Laskina

Overview

Version 3

- Loopback (javascript)
- Hard-coded jobs/actions
- Code not following best practices

Version 4

- NestJS (typescript)
- Configure jobs without code changes
- Better testing

Jobs in backend v3

Fixed job types:

- `archive`: Indicates that data should be moved to tape storage
- `retrieve`: Indicates that data should be retrieved internally from tape storage
- `public`: Retrieve data to public storage location

Hard-coded actions for events:

- Publish message to a Message Broker
- Dispatch email following a template

Limitations:

- Customization requires rebuilding SciCat images with custom js
- Code spread among many locations, files (and site-specific monkey patches)
- All dependencies are compiled into the backend, even if unused

Jobs in scicat-backend-next v4

- Refactored code to be modular and easily expandable
- New DTO and schema
- Elevated users can create jobs for others
- Flexible authorization logic to support new job types (not necessarily dataset-based)
- New job types can be easily added to the system
- Actions easily configurable
- Validate action, to constrain `jobParams` and `jobResultObject`

Configuration – jobConfig.yaml

- Defines different types of jobs
- Not limited to `archive/retrieve/public` anymore
- Parse configuration file and validate its schema
- For each job type, register actions per job operation
- Validate actions before starting the job operation
- Perform actions after a successful job operation
- Use `JOB_CONFIGURATION_FILE` env variable

```yaml
# This represents the recommended job types
# The configuration is similar to the hard-coded jobs from v3.
configVersion: recommended
jobs:
  - jobType: archive
    create:
      auth: "#datasetOwner"
      actions:
        - actionType: validate
          datasets:
            "datasetlifecycle.archivable":
              const: true
    update:
      auth: "archivemanager"
  - jobType: retrieve
    create:
      auth: "#datasetAccess"
      actions:
      - actionType: validate
        datasets:
          "datasetlifecycle.retrievable":
            const: true
    update:
      auth: "archivemanager"
  - jobType: public
    create:
      auth: "#datasetPublic"
      actions:
        - actionType: validate
          datasets:
            isPublished:
              const: true
    update:
      auth: "archivemanager"
```

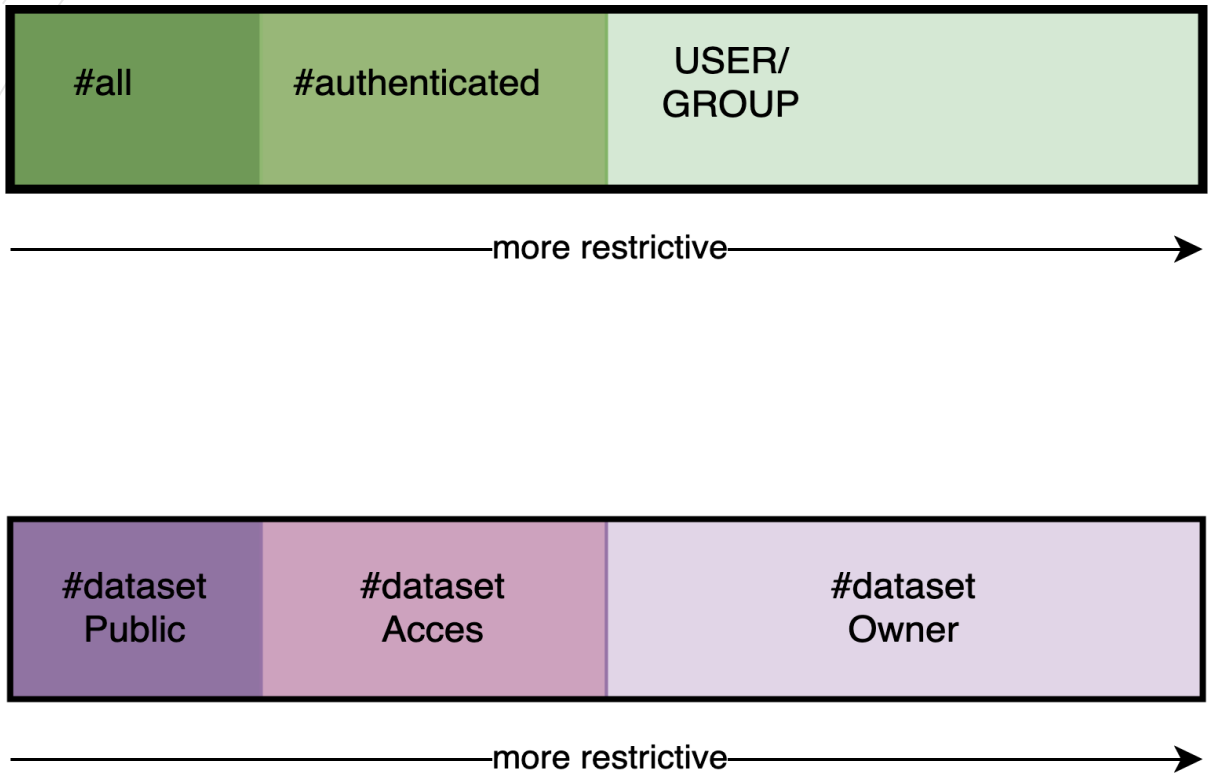
Configuration design

- `jobType`: *archive, retrieve, public*, or custom types
- jobOperations: `create,update`
- `auth`: authorization policy (eg *#all*), specific user, specific group
- `actions`: send email, call URL, post RabbitMQ message, etc

Authorization

ADMIN_GROUPS	• Can create/update/read any job for anyone. • CANNOT delete jobs!
CREATE_JOB_PRIVILEGED_GROUPS (Create Jobs admin)	• Can create jobs for anyone. • In “#datasetX” configurations the limitations on user of the job still apply. • Can read any job.
UPDATE_JOB_PRIVILEGED_GROUPS (Update Jobs admin)	• Can update jobs for anyone. • Can read any job.
DELETE_JOB_GROUPS	• Can delete any job

Permissions



Two levels of permissions are required to create a job: **endpoint** and **instance** authorization.

Creating jobs

Job Create Authorization	Endpoint Authentication Translation	Endpoint Authentication Description	Instance Authentication Translation	Instance Authentication Description
#all	*#all*	any user can access this endpoint, both anonymous and authenticated	*#all*	Any user can create this instance of the job Unsafe!
#datasetPublic	*#all*	any user can access this endpoint, both anonymous and authenticated	*#datasetPublic*	the job instance will be created only if all the datasets listed are **public**
#authenticated	*#user*	any valid users can access the endpoint, independently from their groups	*#user*	any valid users can create this instance of the job
#datasetAccess	*#all*	any user can access this endpoint, both anonymous and authenticated	*#datasetAccess*	the job instance will be created only if the specified user group or otherwise any of the user's groups has access to all the datasets listed
#datasetOwner	*#all*	any user can access this endpoint, both anonymous and authenticated	*#datasetOwner*	the job instance will be created only if the specified user group or otherwise any of the user's groups is part of all the datasets' owner group
@GROUP	*#all*	any user can access this endpoint, both anonymous and authenticated	***GROUP***	the job instance will be created only if the user belongs to the group specified
USER	*#all*	any user can access this endpoint, both anonymous and authenticated	***USER***	the job instance can be created only by the user indicated
#jobAdmin	#all	any user can access this endpoint, both anonymous and authenticated	*#jobAdmin*	the job instance can be created by users of ADMIN_GROUPS and CREATE_JOB_PRIVILEGED only

Job creation DTO

V3

```
{
  "type": "archive",
  "emailJobInitiator": "email@example.com",
  "jobParams": {},
  "datasetList": [{
    "pid": pid,
    "files": []
  }]
}
```

V4

```
{
  "type": "archive",
  "contactEmail": "email@example.com",
  "ownerUser": "owner",
  "ownerGroup": "group",
  "jobParams": {
    "datasetList": [{
      "pid": pid,
      "files": []
    }]
  }
}
```

- ownership: `ownerUser`, `ownerGroup`
- Normal users can create jobs for themselves
- `jobParams`: datasetList, action-specific configuration parameters

Backwards Compatibility

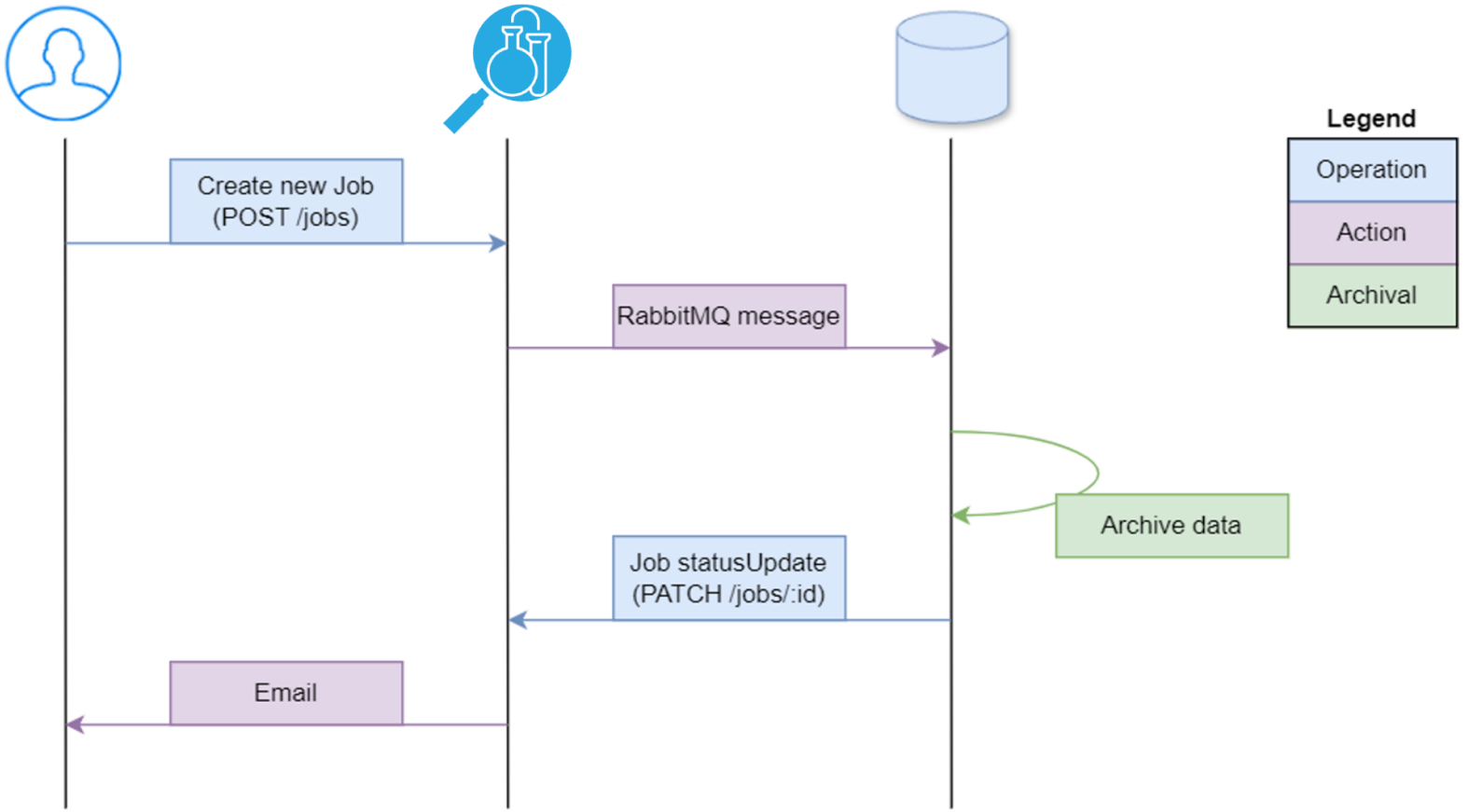
V3

```
{
  "type": "archive",
  "emailJobInitiator": "email@example.com",
  "jobParams": {
    "username": "owner",
    "ownerGroup": "group",
    "param": param
  },
  "datasetList": [{
    "pid": pid,
    "files": []
  }]
}
```

V4

```
{
  "type": "archive",
  "contactEmail": "email@example.com",
  "ownerUser": "owner",
  "ownerGroup": "group",
  "jobParams": {
    "param": param,
    "datasetList": [{
      "pid": pid,
      "files": []
    }]
  }
}
```

Actions overview



Implemented actions

URL

```
- actionType: url
  url: http://localhost:3000/api/v3/health?jobid={{{request.id}}}
  method: GET
  headers:
    accept: application/json
    Authorization: "Bearer {{{env.ARCHIVER_AUTH_TOKEN}}}",
  body: "{{{jsonify job}}}
```

Validate

Enforce custom constraints on `jobParams` or `jobResultObject` for each job type.

```
- actionType: validate
  request:
    <path>: <typecheck>
  datasets:
    <path>: <typecheck>
```

Checks the incoming request body (the DTO).

Requires that a list of datasets be included in `jobParams.datasetList`. Checks are applied to each dataset.

Implemented actions

Email

```
- actionType: email
  to: "{{{job.contactEmail}}}"
  from: "sender@example.com",
  subject: "[SciCat] Your {{{job.type}}} job was submitted successfully."
  bodyTemplateFile: "path/to/job-template-file.html"
```

env variables:

```
EMAIL_TYPE=<"smtp"|"ms365">
EMAIL_FROM=<MESSAGE_FROM>
SMTP_HOST=<SMTP_HOST>
SMTP_PORT=<SMTP_PORT>
SMTP_SECURE=<"yes"|"no">
MS365_TENANT_ID=<tenantId>
MS365_CLIENT_ID=<clientId>
MS365_CLIENT_SECRET=<clientSecret>
```

RabbitMQ

```
- actionType: rabbitmq
  exchange: jobs.write
  key: jobqueue
  queue: client.jobs.write
```

env variables:

```
RABBITMQ_ENABLED=yes
RABBITMQ_HOSTNAME=rabbitmq
RABBITMQ_PORT=5672
RABBITMQ_USERNAME=rabbitmq
RABBITMQ_PASSWORD=SECRET_PASSWORD
```

Implemented actions

Defines which actions are performed based on a condition. The action itself contains a list of sub-actions that will be performed conditionally.

Switch

```
- actionType: switch
  phase: validate | perform | all
  property: some.property
  cases:
  - match: exact value match
    actions: [...]
  - regex: /^regex$/i
    actions: [...]
  - schema: {JSON schema}
    actions: [...]
  - # default
    actions: [...]
```

Determines which phases the switch statement runs: `validate | perform | all`

Name of a variable to test against

One or more conditions to match the property against. Conditions are tested in order with no "fall through" behavior: `match | regex | schema`

```
- actionType: switch
  phase: perform
  property: job.statusCode
  cases:
    - match: finishedSuccessful
      actions:
      - actionType: email
        to: {{{job.contactEmail}}}
        subject: "[SciCat] Your {{{job.type}}} job was successful!"
        bodyTemplateFile: retrieve-success.html
    - actions:
      - actionType: email
        to: {{{job.contactEmail}}}
        subject: "[SciCat] Your {{{job.type}}} job has state {{{job.statusCode}}}"
        bodyTemplateFile: retrieve-failure.html
```

Templating

- Most actions accept **handlebars** templates
- They get filled in when the action runs

Note: Most variables should use handlebars "triple-stash" `{{{ var }}}` to disable html escaping

Fun fact:
In YAML, the ampersand (&) and asterisk (*) are used for creating and referencing **anchors** and **aliases**, respectively.

Top-level variable	Type	Examples
`request`	`CreateJobDto` or `UpdateJobDto`	`{{{request.type}}}` `{{{request.jobParams}}}`
`job`	`JobClass`	`{{{job.id}}}`
`datasets`	`DatasetClass[]`	`{{#each datasets}}{{{pid}}}{{/each}}`
`env`	`object`	`{{{env.SECRET_TOKEN}}}`

Testing

- Integration tests are written for every:
  - user type,
  - job configuration,
  - (optional) dataset permissions.
- Tables with tests outcomes available for each of the endpoints
- ≈ 400 tests to test authorization and CRUD operations
- Unit tests for all actions

Documentation

Developer guide:
https://www.scicatproject.org/documentation/Development/v4.x/backend/configuration/jobconfig.html

Thank you!