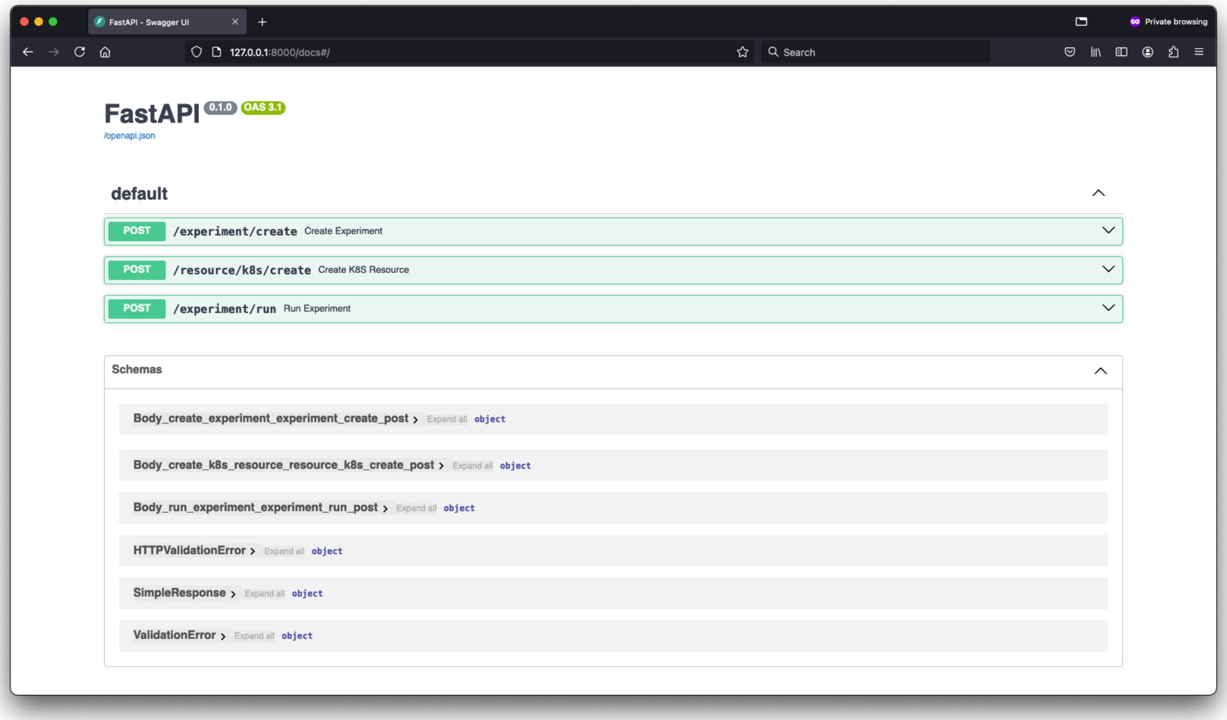
MLExp-CTRL Framework Documents

1 Sample Operation

There are two primary methods for users to interact with the system: via the Command Line Interface (CLI) for testing purposes, and via the API for production use. The functionality of each endpoint is as follows:

- ***/experiment/create***: This endpoint creates an experiment repository, then fetches the credentials for artifact stores such as Git repository and Bucket storage. It also retrieves the experiment pipeline, the DVC pipeline, and the experiment setup, including the hyperparameter file, into the experiment repository.
- ***/resource/k8s/create***: This endpoint schedules the Kubernetes resources on the cluster based on the specifications defined in the manifest file.
- ***/experiment/run***: This endpoint triggers the pipeline run of a specified experiment within the designated environment scheduled on the Kubernetes cluster.

Below is an API documentation page accessible once the framework backend is properly set up.



FastAPI 0.1.0 OAS 3.1

/openapi.json

default

POST `/experiment/create` Create Experiment

POST `/resource/k8s/create` Create K8S Resource

POST `/experiment/run` Run Experiment

Schemas

Body_create_experiment_experiment_create_post > Expand all `object`

Body_create_k8s_resource_resource_k8s_create_post > Expand all `object`

Body_run_experiment_experiment_run_post > Expand all `object`

HTTPValidationError > Expand all `object`

SimpleResponse > Expand all `object`

ValidationError > Expand all `object`

1.1 Experiment Creation

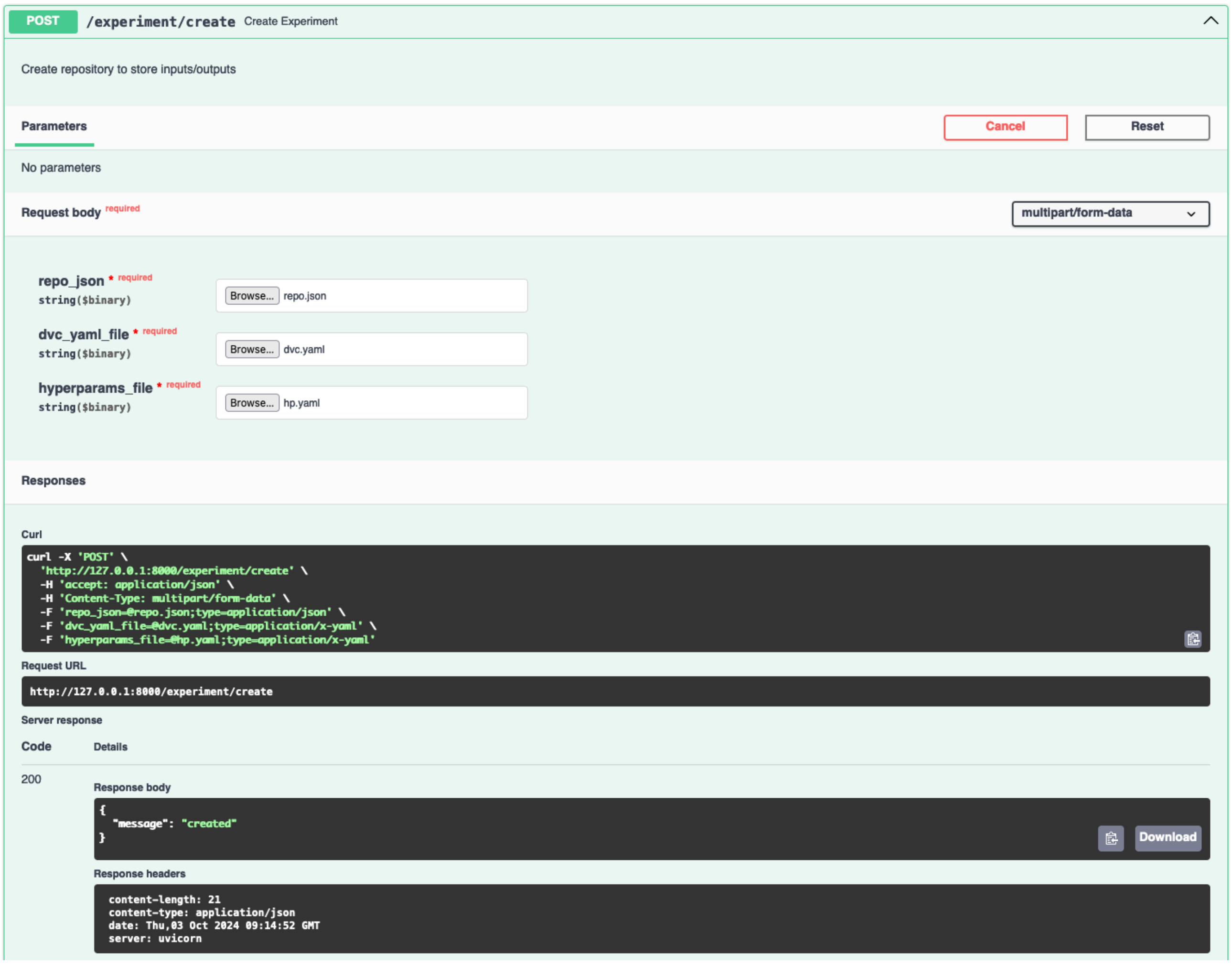
The first endpoint requires three dependencies:

1. **Repositories Resource List** (repo_json): This file supports four types of repositories:

- Git Repository: For storing ML workload repositories.
- DVC-backed Git Repository: Requires endpoints for both Git remote and Bucket Storage remote. This repository type is suitable for storing large artifacts, such as datasets and ML models. It supports step-by-step execution using the DVC pipeline to control execution flow and track artifacts required or produced at each stage.

- Object Storage Bucket: Endpoint for storing large artifacts remotely.
- Container Registry: For reference purposes, allowing traceability to the exact execution environment used to run workloads.

2. **Workflow Pipeline** (dvc_yaml_file): This pipeline file lists the execution stages to be performed by the workload, the dependencies of each stage, and the outputs.
3. **Experiment Setting** (Hyperparameter_file): This artifact has a loosely defined format and stores the hyperparameter settings for the ML process. One experiment corresponds to one hyperparameter file. To create further runs of the experiment with different hyperparameter settings, a new commit with a separate settings file will be made.


POST /experiment/create Create Experiment

Create repository to store inputs/outputs

Parameters

No parameters

Request body required — multipart/form-data

repo_json * required string($binary)	repo.json
dvc_yaml_file * required string($binary)	dvc.yaml
hyperparams_file * required string($binary)	hp.yaml

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/experiment/create' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'repo_json=@repo.json;type=application/json' \
  -F 'dvc_yaml_file=@dvc.yaml;type=application/x-yaml' \
  -F 'hyperparams_file=@hp.yaml;type=application/x-yaml'
```

Request URL

```
http://127.0.0.1:8000/experiment/create
```

Server response

Code 200

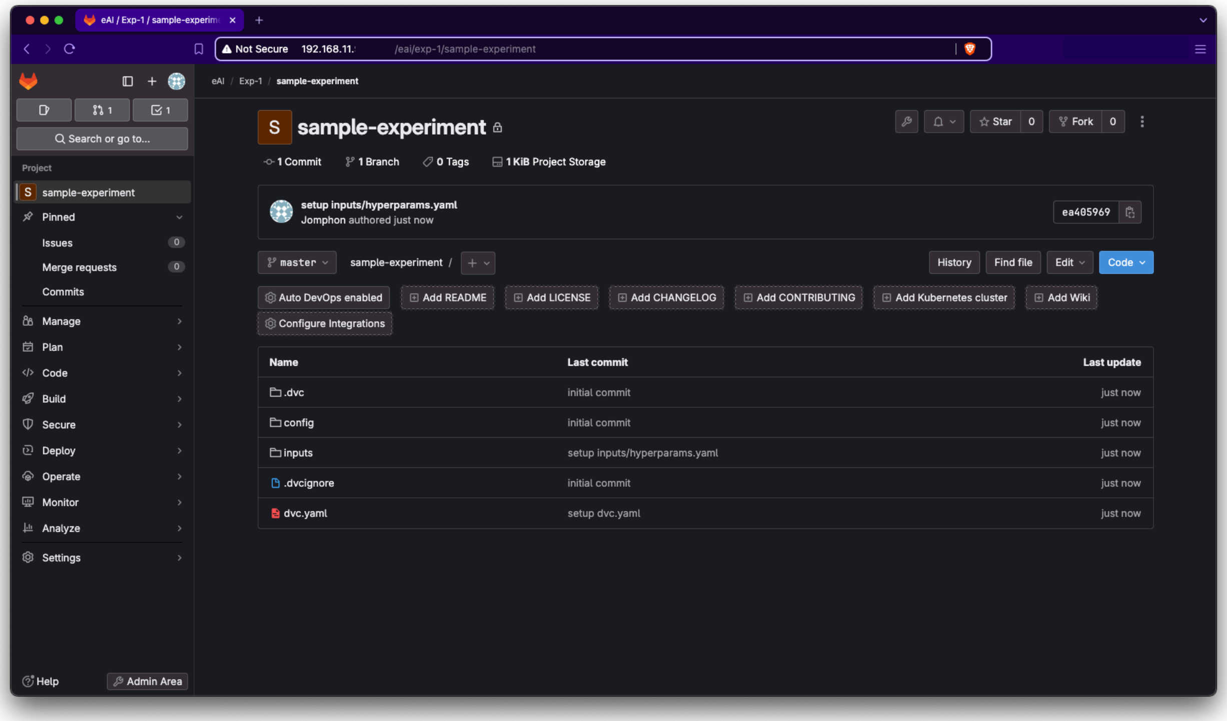
Response body

```
{
  "message": "created"
}
```

Response headers

```
content-length: 21
content-type: application/json
date: Thu,03 Oct 2024 09:14:52 GMT
server: uvicorn
```

Once the endpoint is triggered, if the operation is successful, the API will return the message “created.” We can then verify that the DVC-backed Git repository intended for storing ML experiment artifacts and outcomes has been successfully created, in this case, named sample-experiment.



sample-experiment

1 Commit · 1 Branch · 0 Tags · 1 KiB Project Storage

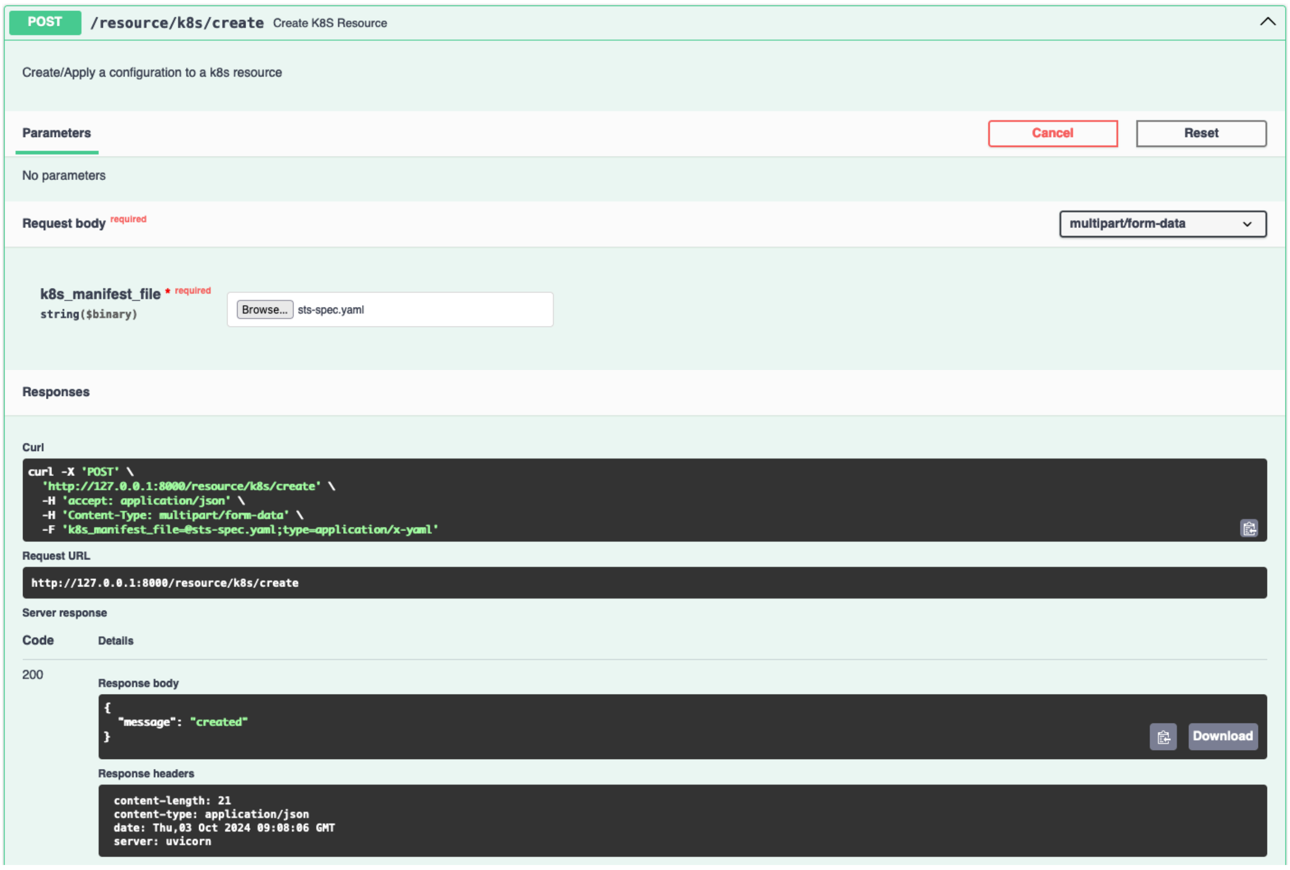
setup inputs/hyperparams.yaml
Jomphon authored just now — ea405969

master · sample-experiment /

Name	Last commit	Last update
.dvc	initial commit	just now
config	initial commit	just now
inputs	setup inputs/hyperparams.yaml	just now
.dvcignore	initial commit	just now
dvc.yaml	setup dvc.yaml	just now

1.2 Scheduling the Environment on Kubernetes Cluster

The second endpoint assists in scheduling resources on the Kubernetes cluster. The resource specifications are defined in a standard configuration file called a Kubernetes manifest file. This configuration allows for flexible determination of various aspects supporting the containerized experiment environment, such as computational resources, accelerator requirements (e.g., GPU), storage, based container image, and more.



POST /resource/k8s/create Create K8S Resource

Create/Apply a configuration to a k8s resource

Parameters

No parameters

Request body required — multipart/form-data

k8s_manifest_file * required
string($binary) — sts-spec.yaml

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/resource/k8s/create' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'k8s_manifest_file=@sts-spec.yaml;type=application/x-yaml'
```

Request URL

```
http://127.0.0.1:8000/resource/k8s/create
```

Server response

Code Details

200

Response body

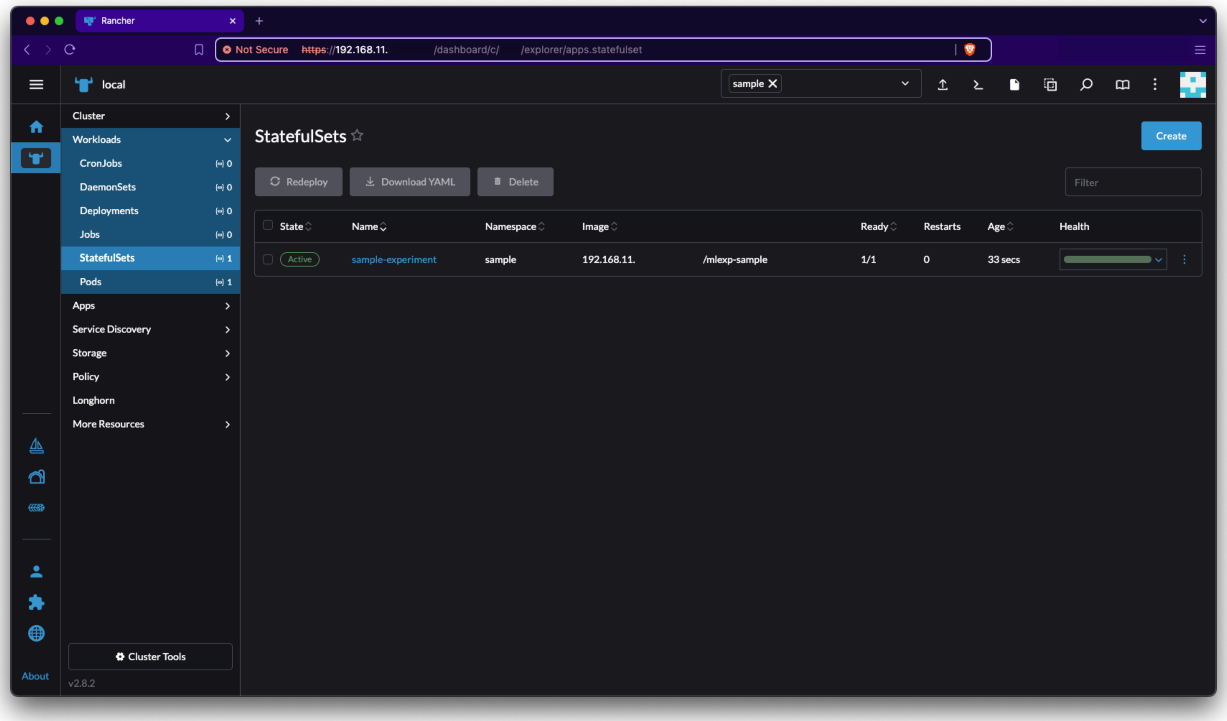
```
{
  "message": "created"
}
```

Response headers

```
 content-length: 21
 content-type: application/json
 date: Thu,03 Oct 2024 09:08:06 GMT
 server: uvicorn
```

The module passes these specifications to the backend Kubernetes cluster. When the resource is properly scheduled, the API returns the message “created” as a response.

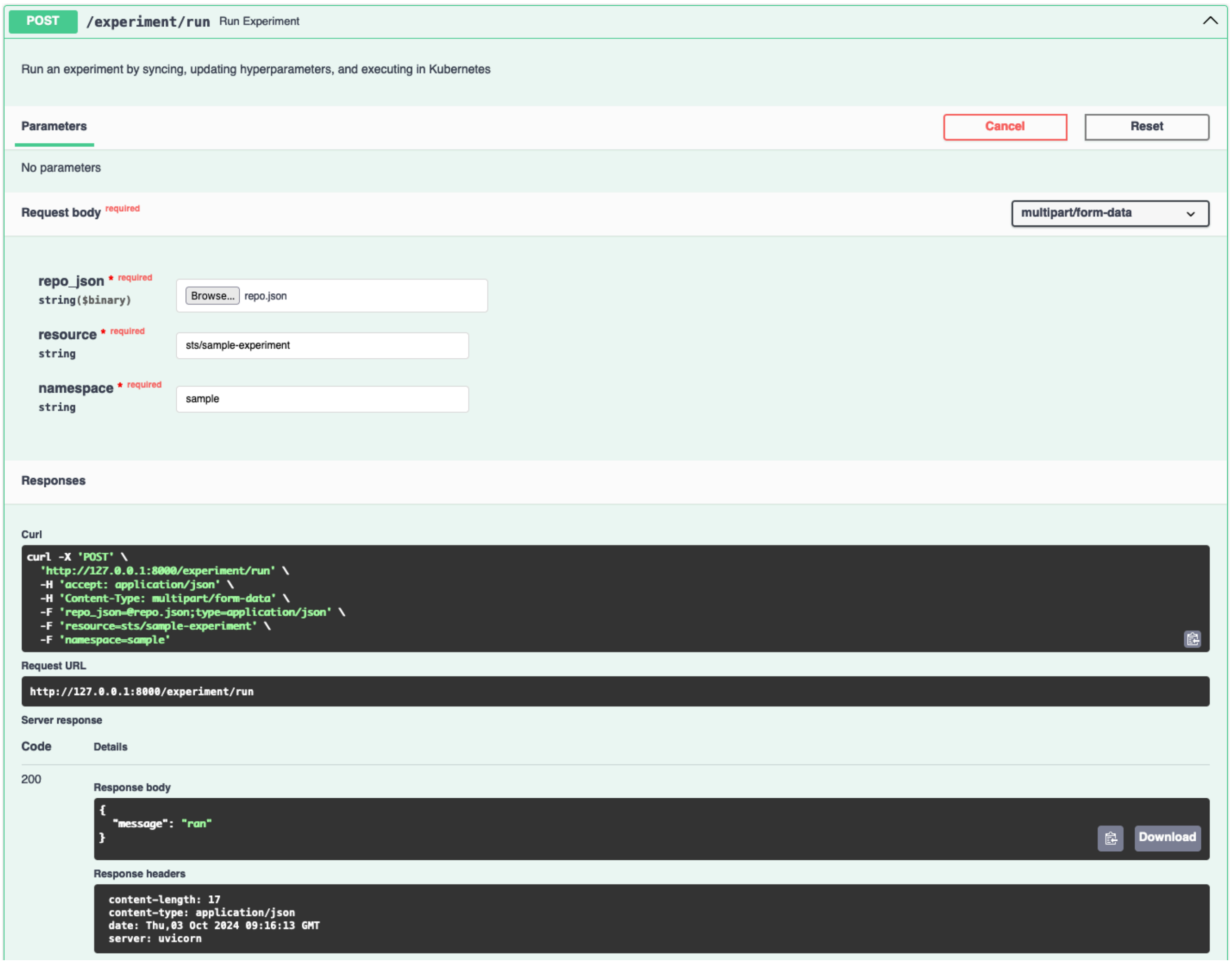
To verify that the resource is properly created, we can visualize it using a tool called Rancher. Here, we can see that our training environment, of type StatefulSet, is successfully scheduled and ready for the workload to run on it.



State	Name	Namespace	Image	Ready	Restarts	Age	Health
Active	sample-experiment	sample	192.168.11. /mlexp-sample	1/1	0	33 secs	

1.3 Executing workload pipeline

The third endpoint assists in triggering the operation of the workload within the container environment by starting the DCV pipeline. This API refers to the content of the repository JSON for authentication, then requires the user to pass the Kubernetes resource name along with the namespace where the environment is scheduled, subsequently triggering the pipeline entry point within the container to start the process. Once the pipeline is completed, it automatically tag all artifacts from each execution stages and pushes these artifacts and results up to the experiment repository for later retrieval by the user. The status message ran indicated that the pipeline has been trigger and versioned properly to the remote storage.



`/experiment/run` Run Experiment

Run an experiment by syncing, updating hyperparameters, and executing in Kubernetes

Parameters

No parameters

Request body required — multipart/form-data

repo_json * required `string($binary)`	repo.json
resource * required `string`	sts/sample-experiment
namespace * required `string`	sample

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/experiment/run' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'repo_json=@repo.json;type=application/json' \
  -F 'resource=sts/sample-experiment' \
  -F 'namespace=sample'
```

Request URL

```
http://127.0.0.1:8000/experiment/run
```

Server response

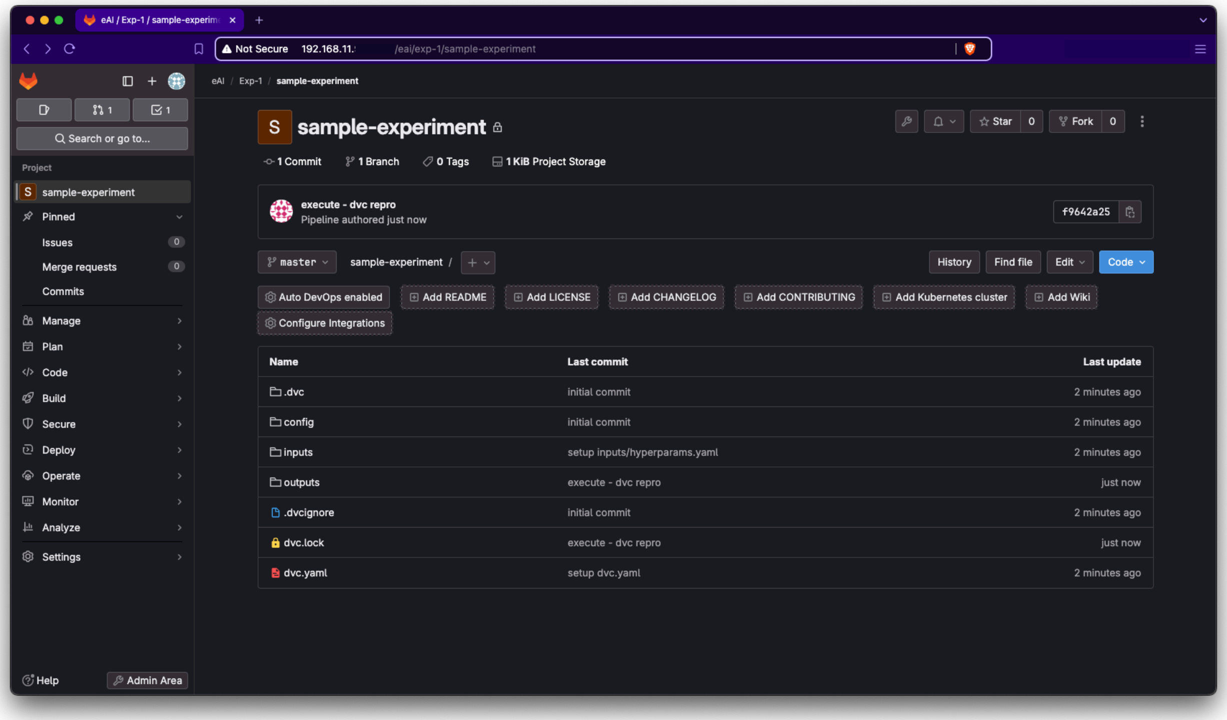
Code 200

Response body

```
{
  "message": "ran"
}
```

Response headers

```
content-length: 17
content-type: application/json
date: Thu,03 Oct 2024 09:16:13 GMT
server: uvicorn
```



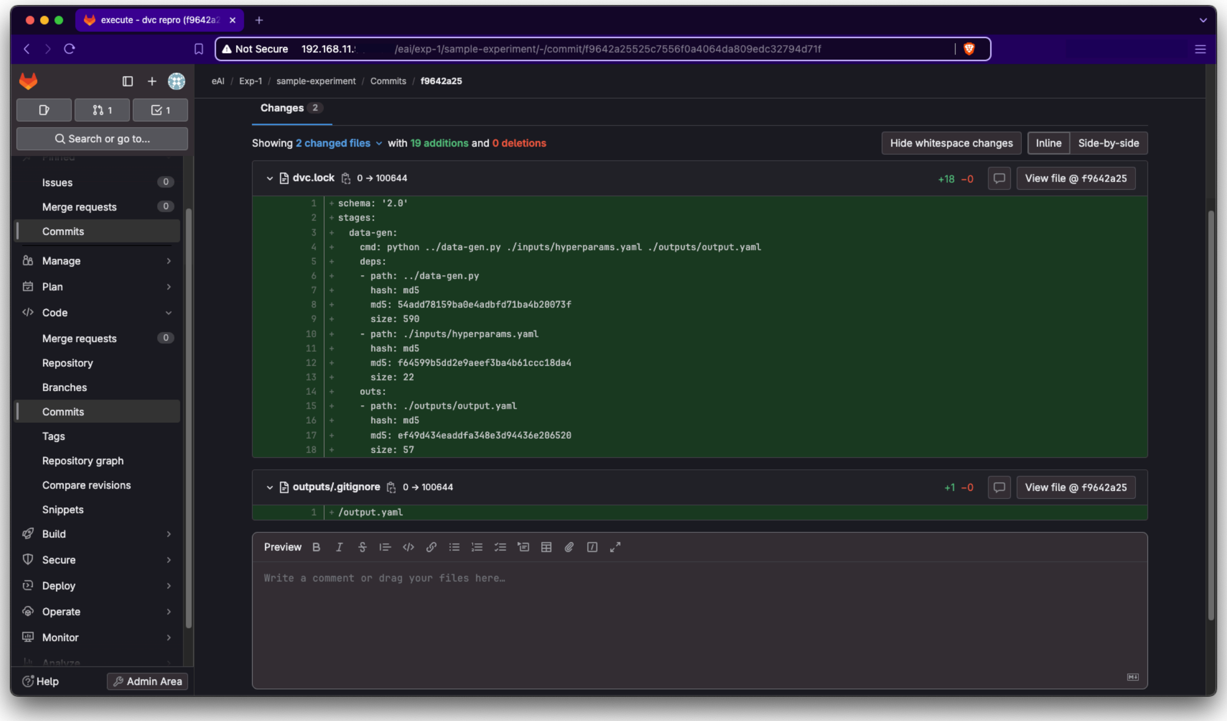
sample-experiment

1 Commit · 1 Branch · 0 Tags · 1 KiB Project Storage

execute - dvc repro
Pipeline authored just now — f9642a25

Name	Last commit	Last update
.dvc	initial commit	2 minutes ago
config	initial commit	2 minutes ago
inputs	setup inputs/hyperparams.yaml	2 minutes ago
outputs	execute - dvc repro	just now
.dvcignore	initial commit	2 minutes ago
dvc.lock	execute - dvc repro	just now
dvc.yaml	setup dvc.yaml	2 minutes ago

As a proof of concept, instead of a full ML workload, this example creates a one-stage pipeline that simply retrieves basic information from within the container and pushes the result up to the experiment repository.



Showing 2 changed files with 19 additions and 0 deletions

dvc.lock 0 → 100644 +18 −0

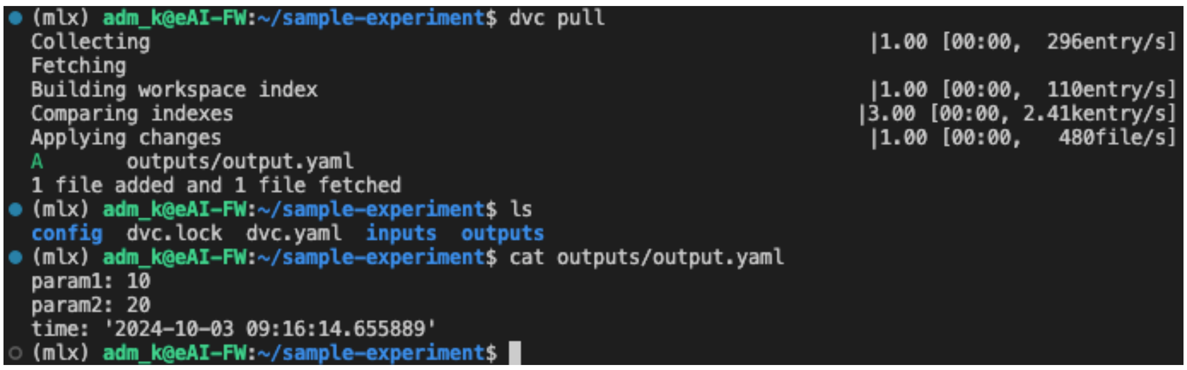
```
+ schema: '2.0'
+ stages:
+   data-gen:
+     cmd: python ../data-gen.py ./inputs/hyperparams.yaml ./outputs/output.yaml
+     deps:
+     - path: ../data-gen.py
+       hash: md5
+       md5: 54add78159ba0e4adbfd71ba4b20073f
+       size: 590
+     - path: ./inputs/hyperparams.yaml
+       hash: md5
+       md5: f64599b5dd2e9aeef3ba4b61ccc18da4
+       size: 22
+     outs:
+     - path: ./outputs/output.yaml
+       hash: md5
+       md5: ef49d434eaddfa348e3d94436e206520
+       size: 57
```

outputs/.gitignore 0 → 100644 +1 −0

```
+ /output.yaml
```

2 Retrieving Experiment Results

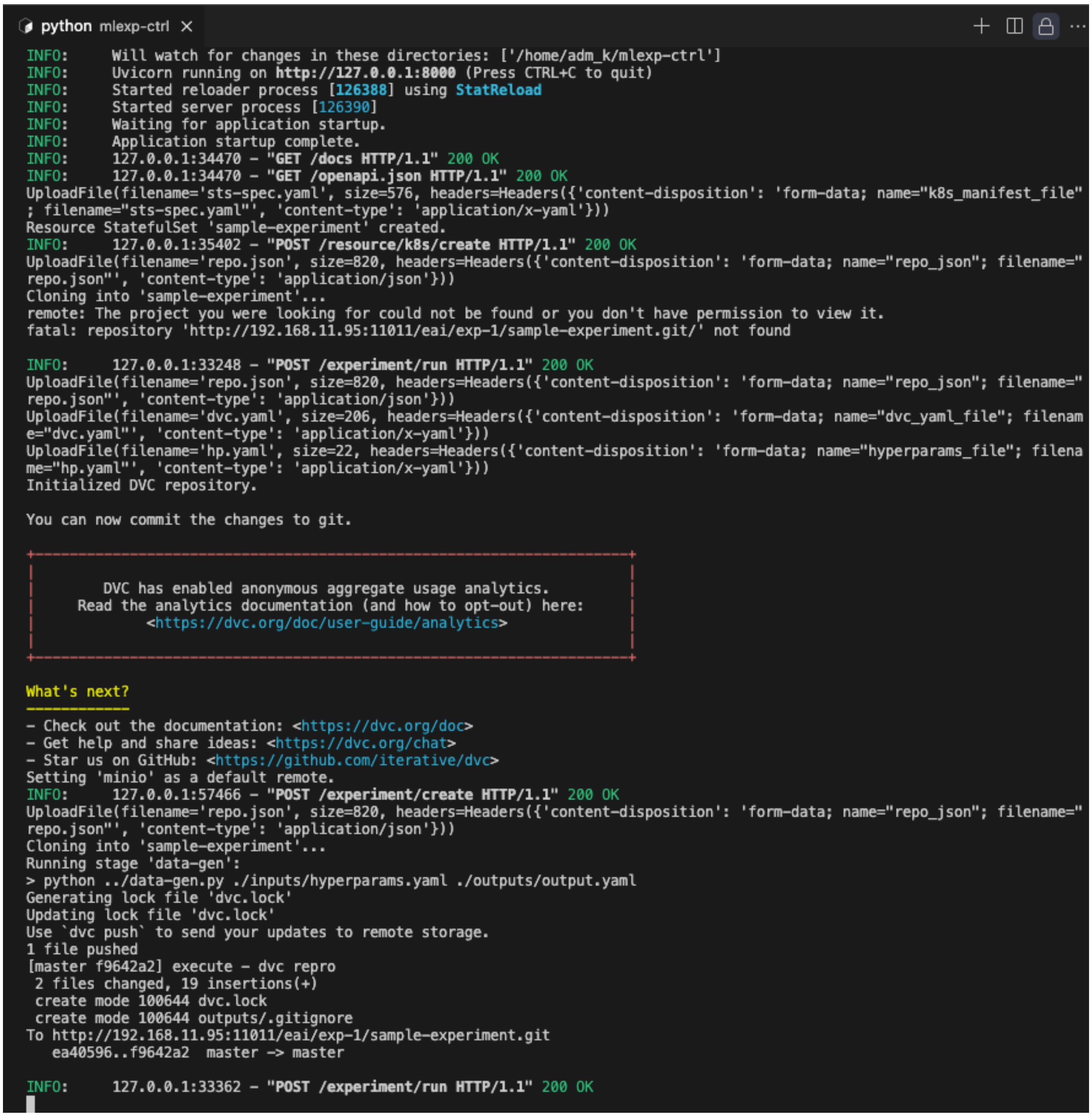
Once the experiment pipeline is completed and the results have been pushed to the experiment repository, clients can retrieve specific results, such as model analysis results, while leaving heavier artifacts like ML models or other checkpoints on remote storage. Here is an example of a client-side pull of the sample workload output, demonstrating its proper retrieval. This capability opens opportunities for further adoption by various client applications that may need parts of the ML experiment at different times and would operate on a pull basis, where resources are only served once called by the frontend client.



```
(mlx) adm_k@eAI-FW:~/sample-experiment$ dvc pull
Collecting                                                  |1.00 [00:00,  296entry/s]
Fetching
Building workspace index                                    |1.00 [00:00,  110entry/s]
Comparing indexes                                          |3.00 [00:00, 2.41kentry/s]
Applying changes                                            |1.00 [00:00,   480file/s]
A       outputs/output.yaml
1 file added and 1 file fetched
(mlx) adm_k@eAI-FW:~/sample-experiment$ ls
config  dvc.lock  dvc.yaml  inputs  outputs
(mlx) adm_k@eAI-FW:~/sample-experiment$ cat outputs/output.yaml
param1: 10
param2: 20
time: '2024-10-03 09:16:14.655889'
(mlx) adm_k@eAI-FW:~/sample-experiment$
```

3 Reference - API Server Log

The entire interaction from the client side to the API server is shown in the log output below:



```
INFO:     Will watch for changes in these directories: ['/home/adm_k/mlexp-ctrl']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [126388] using StatReload
INFO:     Started server process [126390]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:34470 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:34470 - "GET /openapi.json HTTP/1.1" 200 OK
UploadFile(filename='sts-spec.yaml', size=576, headers=Headers({'content-disposition': 'form-data; name="k8s_manifest_file"
; filename="sts-spec.yaml"', 'content-type': 'application/x-yaml'}))
Resource StatefulSet 'sample-experiment' created.
INFO:     127.0.0.1:35402 - "POST /resource/k8s/create HTTP/1.1" 200 OK
UploadFile(filename='repo.json', size=820, headers=Headers({'content-disposition': 'form-data; name="repo_json"; filename="
repo.json"', 'content-type': 'application/json'}))
Cloning into 'sample-experiment'...
remote: The project you were looking for could not be found or you don't have permission to view it.
fatal: repository 'http://192.168.11.95:11011/eai/exp-1/sample-experiment.git/' not found

INFO:     127.0.0.1:33248 - "POST /experiment/run HTTP/1.1" 200 OK
UploadFile(filename='repo.json', size=820, headers=Headers({'content-disposition': 'form-data; name="repo_json"; filename="
repo.json"', 'content-type': 'application/json'}))
UploadFile(filename='dvc.yaml', size=206, headers=Headers({'content-disposition': 'form-data; name="dvc_yaml_file"; filenam
e="dvc.yaml"', 'content-type': 'application/x-yaml'}))
UploadFile(filename='hp.yaml', size=22, headers=Headers({'content-disposition': 'form-data; name="hyperparams_file"; filena
me="hp.yaml"', 'content-type': 'application/x-yaml'}))
Initialized DVC repository.

You can now commit the changes to git.

+---------------------------------------------------------------------+
|                                                                     |
|        DVC has enabled anonymous aggregate usage analytics.        |
|     Read the analytics documentation (and how to opt-out) here:    |
|             <https://dvc.org/doc/user-guide/analytics>             |
|                                                                     |
+---------------------------------------------------------------------+

What's next?
------------
- Check out the documentation: <https://dvc.org/doc>
- Get help and share ideas: <https://dvc.org/chat>
- Star us on GitHub: <https://github.com/iterative/dvc>
Setting 'minio' as a default remote.
INFO:     127.0.0.1:57466 - "POST /experiment/create HTTP/1.1" 200 OK
UploadFile(filename='repo.json', size=820, headers=Headers({'content-disposition': 'form-data; name="repo_json"; filename="
repo.json"', 'content-type': 'application/json'}))
Cloning into 'sample-experiment'...
Running stage 'data-gen':
> python ../data-gen.py ./inputs/hyperparams.yaml ./outputs/output.yaml
Generating lock file 'dvc.lock'
Updating lock file 'dvc.lock'
Use `dvc push` to send your updates to remote storage.
1 file pushed
[master f9642a2] execute - dvc repro
 2 files changed, 19 insertions(+)
 create mode 100644 dvc.lock
 create mode 100644 outputs/.gitignore
To http://192.168.11.95:11011/eai/exp-1/sample-experiment.git
   ea40596..f9642a2  master -> master

INFO:     127.0.0.1:33362 - "POST /experiment/run HTTP/1.1" 200 OK
```