

Nauta User Guide

Enterprise Edition 1.0

Document Revision 1.0: April 2019

Document Revision History

Document Revision Number	Date	Comments
1.0	April 2019	Initial Release of Enterprise Version.

Terms and Conditions

Copyright © 2019 Intel Corporation. All rights reserved.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Contents

Nauta Introduction	7
Product Overview	7
Purpose of this Guide	7
Nauta Basic Concepts	8
User	8
Administrator	8
Resources	8
Data	8
Experiments	8
Predictions	8
Client Installation and Configuration	10
Supported Operating Systems	10
Required Software Packages	10
Installation	10
Setting Variables Permanently	11
Getting Started	12
Verifying Installation	12
Overview of nctl Commands	13
Adding Experiment Metrics	17
Viewing Experiment Results from the Web UI	19
Launching TensorBoard	22
Inference	24
Viewing Experiment Output Folder	27
Removing Experiments	28
Working with Datasets	29
Uploading Datasets	29
nctl mount Command	29
Mount and Access Folders	30
Uploading and Using Shared Dataset Example	31
Working with Experiments	33
Launching Jupyter Interactive Notebook	33
Submitting a Single Experiment	36
Submitting Multiple Experiments	36
Run an Experiment on Multiple Nodes	38
Mounting Storage to View Experiment Output	39
Unmounting Experiment Input from Storage	39
Canceling Experiments	39
Cancelling All Experiments with a Matching Pod-ID	40
Working with Template Packs	41

What is a Template Pack?	41
Pack Anatomy	43
Creating a New Template Pack	46
Nauta values.yaml Placeholders	47
Evaluating Experiments	49
Viewing Experiments Using the CLI	49
Viewing Experiment Logs and Results Data	51
Viewing Experiment Results at the Web UI	52
Launching TensorBoard to View Experiments	56
Evaluating Experiments with Inference Testing	59
Using the predict Command	59
Batch Inference Example	59
Streaming Inference Example	61
Example Flow	61
TensorFlow Serving Basic Example	61
Managing Users and Resources	65
Creating a User Account	65
Deleting a User Account	66
Viewing All User Activity	67
Accessing the Kubernetes Resource Dashboard	68
Access Kubernetes Resource Dashboard	68
CLI Commands	69
Viewing CLI Command Help at the Command Line	70
experiment Command	71
submit Subcommand	72
list Subcommand	75
cancel Subcommand	77
view Subcommand	78
logs Subcommand	79
interact Subcommand	81
launch Command	84
webui Subcommand	84
tensorboard Subcommand	86
mount Command	88
list Subcommand	89
predict Command	90
batch Subcommand	90
cancel Subcommand	91
launch Subcommand	92
list Subcommand	94
stream Subcommand	95

user Command	96
create Subcommand	96
delete Subcommand	98
list Subcommand	99
verify Command	100
config Command	102

Tables

Table 1: Web UI Column Information	20
Table 2: Experiment Details—1	21
Table 3: Experiment Details—2	21
Table 4: Access Permissions for Mounting Folders	30
Table 5: Template Pack Structure Additional Information	43
Table 6: Compute Configurations for Template Packs	44
Table 7: Template Pack Structure Additional Information	44
Table 8: Returned Experiment Status	49
Table 9: Web UI Column Information	52
Table 10: Experiment Details—1	53
Table 11: Experiment Details—2	54

Figures

Figure 1: Example Web UI Screen	19
Figure 2: Experiment Details—1	21
Figure 3: Experiment Details—2	22
Figure 4: TensorBoard Dashboard—Example Only	24
Figure 5: Jupyter Notebook—Example Only	34
Figure 6: Jupyter Notebook Simple Experiment Plot—Example Only	35
Figure 7: Template Pack	42
Figure 8: Viewing Experiment Results from the Web UI—Example Only	52
Figure 9: Experiment Details—1	54
Figure 10: Experiment Details—2	55
Figure 11: Launch TensorBoard from the Web UI – Example Only	57
Figure 12: Kubernetes Dashboard—Example Only	68

Nauta Introduction

Product Overview

The Nauta software provides a multi-user, distributed computing environment for running deep learning model training experiments. Results of experiments, can be viewed and monitored using a command line interface, web UI and/or TensorBoard*.

You can use existing data sets, use your own data, or downloaded data from online sources, and create public or private folders to make collaboration among teams easier. Nauta runs using the industry leading Kubernetes* and Docker* platform for scalability and ease of management.

Templates are available (and customizable) on the platform to take the complexities out of creating and running single and multi-node deep learning training experiments without all the systems overhead and scripting needed with standard container environments.

The Nauta client software runs on the following operating systems:

- Ubuntu* (16.04, 18.04)
- RedHat* 7.5
- macOS* High Sierra (10.13)
- Windows* 10

Purpose of this Guide

This guide describes how to use the Nauta and discusses the following topics main topics:

- [Nauta Basic Concepts, page 8](#)
- [Client Installation and Configuration, page 10](#)
- [Getting Started, page 12](#)
- [Overview of nctl Commands, page 13](#)
- [Working with Datasets, page 29](#)
- [Working with Experiments, page 33](#)
- [Working with Template Packs. page 41](#)
- [Evaluating Experiments, page 49](#)
- [Evaluating Experiments with Inference Testing, page 59](#)
- [Streaming Inference Example, page 61](#)
- [Managing Users and Resources, page 65](#)
- [Accessing the Kubernetes Resource Dashboard, page 68](#)
- [CLI Commands, page 69](#)

Nauta Basic Concepts

Within this user guide, the following concepts and terms are relevant to using this software: user, administrator, resources, data, experiments, and predictions, all of which are described below.

This section discusses the following main topics:

- [User, page 8](#)
- [Administrator, page 8](#)
- [Resources, page 8](#)
- [Data, page 8](#)
- [Experiments, page 8](#)
- [Predictions, page 8](#)

User

In this context, the *User* is a *Data Scientist* who wants to perform deep learning experiments to train models that will, after training and testing, be deployed in production. Using Nauta, the user can define and schedule containerized deep learning experiments using Kubernetes* on single or multiple worker nodes and check the status and results of those experiments to further adjust and run additional experiments or prepare the trained model for deployment.

Administrator

In this context, the *Administrator* or *Admin* creates and monitors users and resources. An important key concept to remember is that Admins *cannot* be users (data scientists); and, users (data scientists) cannot be Admins. Admins *are not* permitted to perform any of the user experiments or related tasks. An admin who wants to run experiments must create a separate user account for that purpose.

Resources

In this context, *Resources* are the system compute and memory resources the user will assign to a model training experiment. The user can specify the number of processing nodes and the amount of memory in the system that will be reserved for a given experiment or job. The job will not be allowed to exceed the specified memory limit. In a multi-user environment, care should be taken to not dedicate too many resources to a given job, because other applications and services may be impacted.

Data

In this context, *Data* is the set of observations used to run experiments to train, test and validate your model.

Experiments

An experiment is a unit of research that defines a single run with set data, parameters, script and results for a training or inference job. The script *must be* tailored to process the data you are using to run your experiment. Nauta allows you to run a single experiment or multiple experiments in parallel. You can perform hyper-parameter tuning by specifying parameters during submission for a specific model script (see the [experiment Command, page 71](#) for more details). Conversely, you can use different training scripts and the experiment's name to keep track of the different experiments.

Predictions

After experiments have been run and the model has been trained, you can pass in new (unlabeled) data exemplars, to obtain predicted labels and other details returned. This process is called inference. In

general, generating predictions involves pre-processing the new input data, running it through the model, and then collecting the results from the last layer of the network.

The Nauta software supports both batch and streaming inference. Batch inference involves processing a set of prepared input data to a referenced trained model and writing the inference results to a folder. Streaming inference is where the user deploys the model on the system and streaming inference instance processes singular data as it is received.

Client Installation and Configuration

The section provides instructions for installing and configuring Nauta to run on your client system. For instructions to install and configure Nauta to run on the host server, refer to the *Nauta Installation, Configuration, and Administration Guide*.

This section discusses the following main topics:

- [Supported Operating Systems](#), page 10
- [Installation](#), page 10
- [Setting Variables Permanently](#), page 11

Supported Operating Systems

This release of the Nauta client software has been validated on the following operating systems and versions.

- Ubuntu (16.04, 18.04)
- RedHat* 7.5
- macOS High Sierra (10.13)
- Windows 10

Required Software Packages

The following software *must* be installed on the client system *before* installing Nauta:

- kubectl version 1.10 or later: [Install Kubectl](#)

Installation

To install the Nauta software package, do the following:

1. Download and install *Required Software Package* above, preferably in the order given.
2. There *is no* installation utility. You can unpack this package and place the unpacked files in any location you prefer. Take note of the path.
3. Set KUBECONFIG environment variable to the Nauta configuration file for Kubernetes (Nauta configuration file in Kubernetes format) provided by your Nauta Admin. Here, <PATH> is wherever your *config* file is located.

- For **MacOS/Ubuntu**, enter:

```
export KUBECONFIG=<PATH>/<USERNAME>.config
```

For **Windows**, enter:

```
set KUBECONFIG=<PATH>\<USERNAME>.config
```

4. **Optional:** Add the package `nctl` path to your terminal PATH. `NAUTA_HOME` should be the path to the `nctl` application folder:

For **MacOS/Ubuntu**, enter:

```
export PATH=$PATH:NAUTA_HOME
```

For **Windows**, enter:

```
set PATH=%PATH%;NAUTA_HOME
```

Setting Variables Permanently

Should you want to permanently set the variables, you can add the variables to your:

- `.bashrc`
- `.bash_profile`
- or Windows system PATH

Alternatively, you may want to set the `PATH` and `KUBECONFIG` variables in the Environment Variables window. This is accessed by opening the Control Panel > System and Security > System > Advanced system settings and accessing Environment variables. This is an administrator function only.

Getting Started

This section of the guide provides brief examples for performing some of the most essential and valuable tasks supported by Nauta software.

Note: Several examples in this section require access to the internet, to download data, scripts, and so on.

The section discusses the following topics:

- [Verifying Installation, page 12](#)
- [Overview of nctl Commands, page 13](#)
- [Monitoring Training, page 16](#)
- [Adding Experiment Metrics, page 17](#)
- [Viewing Experiment Results from the Web UI, page 19](#)
- [Launching TensorBoard, page 22](#)
- [Viewing Experiment Output Folder, page 27](#)
- [Removing Experiments, page 28](#)

Verifying Installation

Check that the required software packages are available in terminal by `PATH` and verified that the correct version is used (see [Confirm Installation](#) below).

Proxy Environment Variables

If you are behind a proxy, remember to set your:

- `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` environment variables
- `http_proxy`, `https_proxy` and `no_proxy` environment variables

Confirm Installation

To verify your installation has completed, execute the following command:

```
nctl verify
```

Confirmation Message

If any installation issues are found, the command returns information about the cause: which application should be installed and in which version. This command also checks if the CLI can connect to Nauta; and, if port forwarding to Nauta is working correctly. If no issues are found, a message indicates checks were successful. The following examples are the results of this command:

```
This OS is supported.
kubectl verified successfully.
kubectl server verified successfully.
helm client verified successfully.
helm server verified successfully.
```

Overview of nctl Commands

Each `nctl` command has at least two options:

- `-v`, `--verbose` - Set verbosity level:
 - `-v` for INFO - Basic logs on INFO/EXCEPTION/ERROR levels are displayed.
 - `-vv` for DEBUG - Detailed logs on INFO/DEBUG/EXCEPTION/ERROR levels are displayed.
- `-h`, `--help` - The application displays the usage and options available for a specific command or subcommand.

Accessing Help

Access help for any command with the `--help` or `-h` parameter. The following command provides a list and brief description of all `nctl` commands.

```
nctl -help
```

The results are shown below.

```
Usage: nctl COMMAND [options] [args]...

  Nauta Client

  Displays additional help information when the -h or --help COMMAND is used.

Options:
  -h, --help Displays help messaging information.

Commands:
  config, cfg      Set limits and requested resources in templates.
  experiment, exp  Start, stop, or manage training jobs.
  launch, l        Launch the web user-interface or TensorBoard. Runs as a process
                  in the system console until the user stops the process.
                  To run in the background, add '&' at the end of the line.
  mount, m         Displays a command to mount folders on a local machine.
  predict, p       Start, stop, and manage prediction jobs and instances.
  user, u          Create, delete, or list users of the platform.
                  Can only be run by a platform administrator.
  verify, ver      Verifies whether all required external components contain
                  the proper versions installed.
  version, v       Displays the version of the installed nctl application.
```

Utility Scripts

There are two utility scripts, these are:

- `mnist_converter_pb.py`
- `mnist_checker.py`

These are used for inference process and model verification.

Note: Experiment scripts *must be* written in Python.

Submitting an Experiment

Launch training experiments with Nauta using the following:

Syntax:

```
nctl experiment submit [options] SCRIPT-LOCATION -- [script-parameters]
```

Where: The path and name of the Python script use to perform this experiment.

```
-- SCRIPT-LOCATION
```

Note: For more info about experiment submit command, refer to [submit Subcommand](#), page 72.

Example Experiments

To submit the example experiments, use the following:

Single Node Training

For *single node* training (template parameter in this case is optional), use the following:

```
nctl experiment submit -t tf-training-tfjob examples/mnist_single_node.py --name single
```

Multinode Training

For *multinode* training, use the following:

```
nctl experiment submit -t multinode-tf-training-tfjob examples/mnist_multinode.py --name multinode
```

Horovod Training

For Horovod training, use the following:

```
nctl experiment submit -t multinode-tf-training-horovod examples/mnist_horovod.py --name horovod
```

The included example scripts *do not* require an external data source. The scripts automatically download the MNIST dataset. Templates referenced here have set CPU and Memory requirements. The list of available templates can be obtained by issuing `nctl experiment template_list` command.

To change template-related requirements (if desired), refer to the template packs documentation ([Working with Template Packs](#), page 41).

Note: To run TensorBoard, TensorBoard data *must be* written to a folder in the directory `/mnt/output/experiment`. This example script satisfies this requirement; however, your scripts *must* meet the same requirement.

The following example shows how to submit a MNIST experiment and write the TensorBoard data to a folder in your Nauta output folder. Execute the following command to run this example:

```
nctl experiment submit -t tf-training-tfjob examples/mnist_single_node.py --name single
```

Result of this Command

The execution of the submit command may take a few minutes the first time. When the experiment submission is complete, the following result is displayed:

```
Submitting experiments.
| Experiment | Parameters | State | Message |
|-----+-----+-----+-----|
| single | mnist_single_node.py | QUEUED | |
```

Viewing Experiment Status

Use the following command to view the status of all your experiments:

Syntax:

```
nctl experiment list [options]
```

Example:

Execute this command:

```
nctl experiment list --brief
```

As shown below, an experiment's status displays. This is an example only. The `--brief` option returns a short version of results shown below.

```
Submitting experiments.
| Experiment | Parameters | State | Message |
|-----+-----+-----+-----|
| single | mnist_single_node.py | QUEUED | |

| Experiment | Submission date | Owner | State |
|-----+-----+-----+-----|
| mnist-single-node-tb | 2019-03-13 04:57:58 PM | user1 | QUEUED |
| mnist-tb | 2019-03-13 05:00:39 PM | user1 | COMPLETE |
| mnist-tb 2-1 | 2019-03-13 05:49:59 PM | user1 | COMPLETE |
| test-experiment | 2019-03-13 06:00:39 PM | user1 | QUEUED |
| single | 2019-03-13 01:49:59 PM | user1 | RUNNING |
```

Monitoring Training

There are four ways to monitor training in Nauta, all which are discussed in the following sections.

- [Viewing Experiment Logs, page 16](#)
- [Adding Experiment Metrics, page 17](#)
- [Viewing Experiment Results from the Web UI, page 19](#)
- [Launching TensorBoard, page 22](#)

Viewing Experiment Logs

To view the experiment log, execute the following command.

Syntax:

```
nctl experiment logs [options] EXPERIMENT-NAME
```

Example:

Execute this command:

```
nctl experiment logs single
```

As shown below, a log displays the example results.

```
2019-03-20T16:11:38+00:00 single-master-0 Step 0, Loss: 2.3015756607055664, Accuracy: 0.078125
2019-03-20T16:11:44+00:00 single-master-0 Step 100, Loss: 0.13010963797569275, Accuracy: 0.921875
2019-03-20T16:11:49+00:00 single-master-0 Step 200, Loss: 0.07017017900943756, Accuracy: 0.984375
2019-03-20T16:11:55+00:00 single-master-0 Step 300, Loss: 0.08880224078893661, Accuracy: 0.984375
2019-03-20T16:12:00+00:00 single-master-0 Step 400, Loss: 0.15115690231323242, Accuracy: 0.953125
2019-03-20T16:12:07+00:00 single-master-0 Validation accuracy: 0.980400025844574
```


Adding Experiment Metrics

Experiments launched in Nauta can output additional kinds of metrics using the *publish function* from the experiment metrics API. To see an example of metrics published with the single experiment executed in the above example, execute the following command:

```
nctl experiment list
```

A partial example result is shown below.

Experiment	Parameters	Metrics	Submission date
mnist-tb	mnist_single_node.py		2019-03-20 05:11:15 PM
single	mnist_single_node.py	accuracy: 0.96875	2019-03-20 05:03:12 PM.
		global step: 499	
		global_step: 499	
		validation accuracy: 0.9818	

Adding Experiment Metrics: Instructions

To add metrics to an experiment, you need to edit the experiment script to use the `experiment_metrics.api` and then publish the metric that you wish to display. The following steps should be performed in the script to publish a metric.

1. Add the metrics library API with the following entry in your experiment script.

```
from experiment_metrics.api import publish
```

2. To add a metric, publish dict key and string value. Using the validation accuracy metric as an example, the metric is published in the `mnist_single_node.py` example.

```
publish({"validation accuracv": str(validation_accuracy_val)})
```

3. Once you add a new metric, save the changes.
4. Submit the experiment again, but with a different name.
5. The published metrics can now be viewed.

```
nctl experiment list
```

Saving Metrics for Multinode Experiments

Storing at the same time two (or more) metrics with the same key from two different nodes may lead to errors (such as losing some logs) due to conflicting names. To avoid this, adding metrics for multinode experiments should be done using one of the two following methods: [Method 1](#) or [Method 2](#).

Method 1

The key of a certain metric should also contain a node identifier from which this metric comes. Creation of such identifier can be done in the following ways:

- For `horovod` multinode training jobs, result of the `rank()` function provided by the Horovod package can be used as a node's identifier.
- For `tfjob` multinode training jobs, a user can take all necessary info from the `TF_CONFIG` environment variable. An example piece of a code creating such identifier, is:

Node Identifier Example

```
tf_config = os.environ.get('TF_CONFIG')
if not tf_config:
    raise RuntimeError('TF_CONFIG not set!')

tf_config_json = json.loads(tf_config)

job_name = tf_config_json.get('task', {}).get('type')
task_index = tf_config_json.get('task', {}).get('index')
# final node identifier
node_id = '-'.join(job_name, task_index)
```

Method 2

Only one node should store metrics. Deciding which node should store metrics can be done in the following ways:

- For `horovod` multinode training jobs, the Horovod python library provides the `rank()` function that returns a number of a current worker. *Master* is marked with the number 0, so only a pod with this number should store logs.
- For `tfjob` multinode training jobs, because there is no dedicated master node, a user should choose which worker should be responsible for storing metrics. The identifier of a current worker can be obtained as described in [Method 1](#) above. Furthermore, a user should choose an identifier and store the logs, but only from a node that has this chosen ID.

Viewing Experiment Results from the Web UI

The web UI lets you explore the experiments you have submitted. To view your experiments at the web UI, enter the following command at the command prompt:

```
nctl launch webui
```

Note: If you are using CLI through remote access, you will need to setup a X server for tunneling over SSH with port forwarding or use SSH Proxy command tunneling. After establishing a tunnel from the gateway to your local machine, you can use the URL by this command provided by `nctl`. [Figure 1](#) shows an example Web UI screen.

Figure 1: Example Web UI Screen

Experiments							
<div> LAUNCH TENSORBOARD* RESET ADD/DELETE COLUMNS </div>							
TensorBoard* Eligibility	Name	Status	Submission Date	Start Date	Duration	Type	
✓	^ training-p-988-19-01-17-12-50-12	COMPLETE	01/17/2019 12:50:15 pm	01/17/2019 01:05:44 pm	0 days, 0 hrs, 1 mins, 59 s	Training	
○	^ training-p-368-19-01-17-12-41-56	FAILED	01/17/2019 12:41:59 pm	01/17/2019 12:42:14 pm	0 days, 0 hrs, 1 mins, 21 s	Training	
○	^ metrics-py-539-19-01-17-11-58-35	COMPLETE	01/17/2019 11:58:38 am	01/17/2019 12:02:34 pm	0 days, 0 hrs, 4 mins, 11 s	Training	
○	^ metrics-py-497-19-01-17-11-58-29	COMPLETE	01/17/2019 11:58:32 am	01/17/2019 12:02:28 pm	0 days, 0 hrs, 4 mins, 12 s	Training	
○	^ metrics-py-174-19-01-17-11-58-22	COMPLETE	01/17/2019 11:58:25 am	01/17/2019 12:02:18 pm	0 days, 0 hrs, 4 mins, 11 s	Training	
○	^ metrics-py-908-19-01-17-11-58-16	COMPLETE	01/17/2019 11:58:19 am	01/17/2019 12:02:10 pm	0 days, 0 hrs, 4 mins, 10 s	Training	
○	^ metrics-py-893-19-01-17-11-58-09	COMPLETE	01/17/2019 11:58:12 am	01/17/2019 12:01:57 pm	0 days, 0 hrs, 4 mins, 11 s	Training	
○	^ metrics-py-322-19-01-17-11-58-02	COMPLETE	01/17/2019 11:58:05 am	01/17/2019 12:01:48 pm	0 days, 0 hrs, 4 mins, 12 s	Training	
○	^ metrics-py-391-19-01-17-11-57-55	COMPLETE	01/17/2019 11:57:58 am	01/17/2019 11:58:13 am	0 days, 0 hrs, 4 mins, 11 s	Training	
<div> Last updated a moment ago <div> Rows per page: 25 1-14 of 14 </div> </div>							

The web UI shows the six columns listed below. Each of these column headings are clickable, so to re-sort the listing of experiments based on that column heading, ascending or descending order, click that column heading. The six columns are (shown in [Table 1](#)), Name, Status, Submission Date, Duration and Type.

Table 1: Web UI Column Information

Name	Status	Submission Date	Start Date	Duration	Type
The left-most column lists the experiments by name.	<p>This column reveals experiment's current status, one of:</p> <ul style="list-style-type: none"> • QUEUED • RUNNING • COMPLETE • CANCELED • FAILED • CREATING 	This column gives the submission date in the format: MM/DD/YYYY, hour:min:second AM/PM.	This column shows the experiment start date in the format: MM/DD/YYYY, hour:min:second AM/PM. The Start Date (or time) will always be after the Submission Date (or time).	This column shows the duration of execution for this experiment in days, hours, minutes and seconds	<p>Experiment Type can be <i>Training</i>, <i>Jupyter</i>, or <i>Inference</i>. Training indicates that the experiment was launched from the CLI. Jupyter indicates that the experiment was launched using Jupyter Notebook. Inference means that training is largely complete and you have begun running predictions (Inference) with this model. (If the model used for inference was already present, there was no training performed on Nauta.)</p>
<p>Note: You can perform the tasks discussed below at the Nauta web UI.</p>					

Expand Experiment Details

Click *listed experiment name* to see additional details for that experiment. The following details are examples only. This screen is divided into two frames: left-side and right-side frames.

Left-most Frame

The left-side frame of the experiment shows the resources and submission date and time (as shown in the Figure 2.).

Table 2: Experiment Details—1

Resources	Submission Date and Time
Resources assigned to that experiment, specifically the assigned pods and their status and container information including the CPU and memory resources assigned.	Displays the Submission Date and time.

Figure 2: Experiment Details—1

Resources:	Pods -- 1. Name: <i>mnist-tb-master-0</i> Pod Conditions: <i>Initialized: True , reason: PodCompleted, Ready: False , reason: PodCompleted, PodScheduled: True</i> Containers: ○ Name: <i>tensorflow</i> Resources: <i>cpu - 100m, , memory - 10Mi</i> Status: <i>Terminated, Completed</i> 2. Name: <i>mnist-tb-master-0</i> Pod Conditions: <i>Initialized: True , reason: PodCompleted, Ready: False , reason: PodCompleted, PodScheduled: True</i> Containers: ○ Name: <i>tensorflow</i> Resources: <i>cpu - 100m, , memory - 10Mi</i> Status: <i>Terminated, Completed</i>
Submission Date:	11/9/2018, 2:17:07 PM

Right-side Frame

The right-side frame (see Figure 3) of the experiment details window shows Start Date, End Date, Total Duration, Parameters, and Output.


Table 3: Experiment Details—2

Start Date	End date	Total Duration	Parameters	Output
The day and time this experiment was launched.	The day and time this experiment was launched.	The actual duration this experiment was instantiated.	The experiment script file name and the log directory.	Clickable links to download all logs and view the last 100 log entries.

Figure 3: Experiment Details—2

Start Date:	11/9/2018, 2:17:14 PM
End Date:	11/9/2018, 2:18:28 PM
Total Duration:	0 days, 0 hrs, 1 mins, 14 s
Parameters:	mnist_with_summaries.py, --log_dir=/mnt/output/experiment/tb
Output:	Logs, All Download
	Logs, Last 100 View

Searching on Experiments

In the **Search** field at the far right of the UI  enter a string of alphanumeric characters to match the experiment name or other parameters (such as user), and list only those matching experiments. This Search function lets the user search fields in the entire list, not just the experiment name or parameters.

Adding/Deleting Columns


Click **ADD/DELETE COLUMNS** to open a scrollable dialogue. Here, the columns currently in use are listed first with their check box checked. Scroll down to see more, available columns listed next, unchecked.

Click to check and uncheck and select the column headings you prefer. Optional column headings include parameters such as **Pods**, **End Date**, **Owner**, **Template**, **Time in Queue**, and so on.

Column headings also include metrics that have been setup using the Metrics API, for a given experiment, and you can select to show those metrics in this display as well. Column additions and deletions you make are retained between logins.

Refer to [Launching TensorBoard to View Experiments](#), page 56.

Launching Kubernetes Dashboard

Click the **Hamburger Menu**  at the far left of the UI to open a left frame. Click **Resources Dashboard** to open the Kubernetes resources dashboard. Refer to [Accessing the Kubernetes Resource Dashboard](#), page 67.

Launching TensorBoard

Generally, every file that the training script outputs to `/mnt/output/experiment` (accessed from the perspective of training script launched in Nauta) is accessible from the outside after mounting the output directory with command provided by `nctl mount`.

Use the following command to launch TensorBoard and to view graphs of this model's results. Refer to [Mounting Storage to View Experiment Output](#), page 39 for more information.

When training scripts output TensorFlow summaries to `/mnt/output/experiment`, they can be automatically picked up by a TensorBoard instance launched with this command:

Syntax:

```
nctl launch tensorboard [options] EXPERIMENT-NAME
```

Execute the following command:

```
nctl launch tensorboard single
```

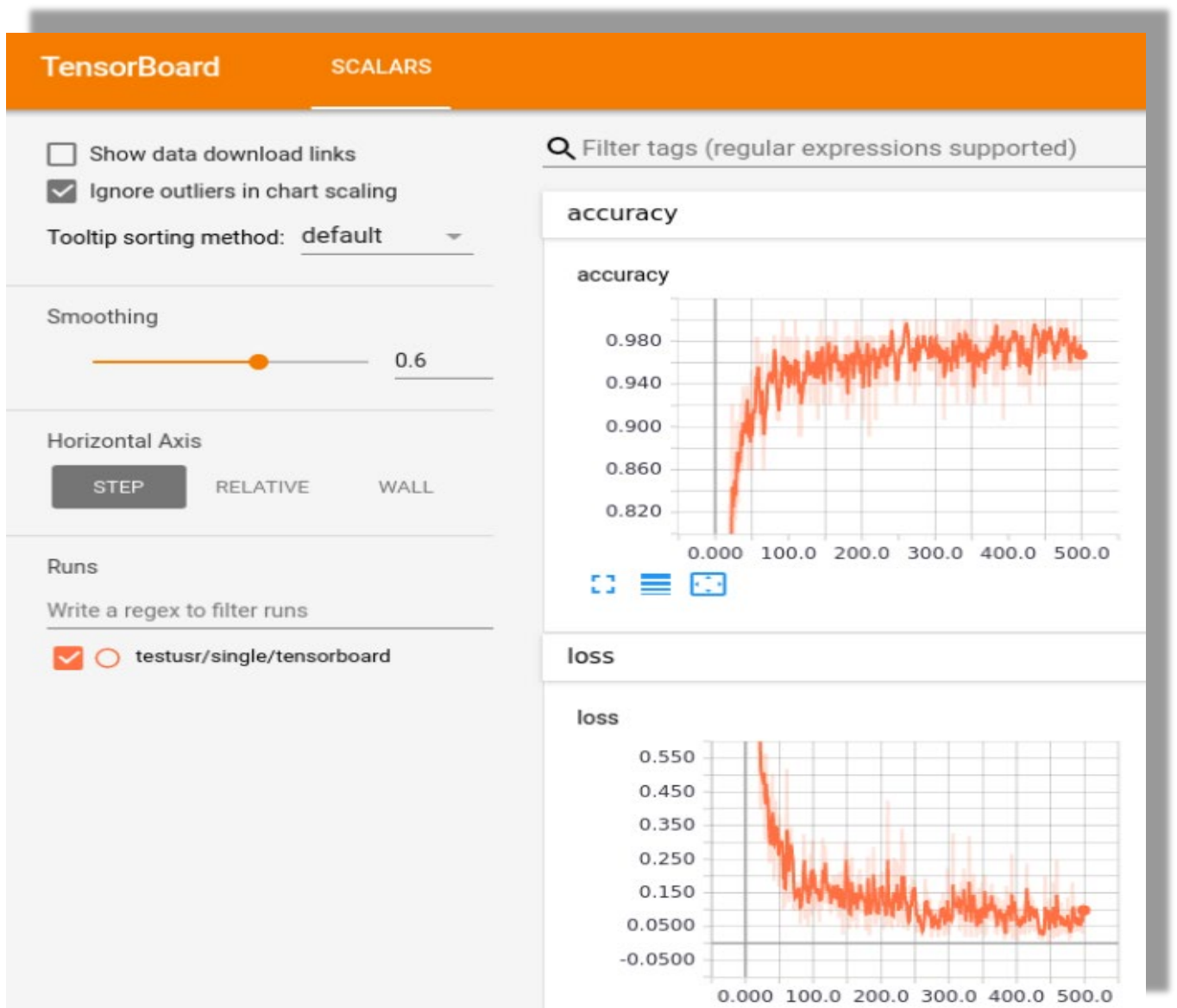
Note: If you are using CLI through remote access, you will need to setup a X server for tunneling over SSH with port forwarding or use SSH Proxy command tunneling. After establishing a tunnel from the gateway to your local machine, you can use the URL provided by `nctl`.

The following message displays the example port number.

```
Please wait for Tensorboard to run...  
Go to http://localhost: 58218  
Proxy connection created.  
Press Ctrl-C key to close a port forwarding process...
```

[Figure 4](#) shows the browser display of TensorBoard dashboard with the experiment's results.

Figure 4: TensorBoard Dashboard—Example Only



Inference

To perform inference testing (using `predict batch` command in this example) you need to:

1. Prepare data for model input.
2. Acquire a trained model.
3. Run prediction instance with trained model on this data.

Data Preparation

The example `mnist_converter_pb.py` script located in the `examples` folder can be used for data preparation. This script prepares the sample of MNIST test set and converts it to `protobuf` requests acceptable by the served model. This script is run locally and requires `tensorflow`, `numpy`, and `tensorflow_serving` modules. The `mnist_converter_pb.py` script takes two input parameters:

- `--work_dir` which defaults to `/tmp/mnist_test`. It is a path to directory used as `workdir` by this script and `mnist_checker.py`. Downloaded MNIST dataset will be stored there, as well as converted test set sample and labels cached for them.
- `--num_tests` which defaults to 100. It is a number of examples from test set which will be converted. Max value is 10000.

Running the command:

```
python examples/mnist_converter_pb.py
```

Creates `/tmp/mnist_test/conversion_out` folder, fill it with 100 `protobuf` requests, and cache labels for these requests in `/tmp/mnist_test/labels.npy` file.

Trained Model

Servable models (as with other training artifacts) can be saved by a training script. As previously mentioned, to access these you have to use the command provided by the `nctl mount` command and mount output storage locally. Example scripts all save servable models in their model's subdirectory. To use models like this for inference, you will have to mount `input` storage too, because models have to be accessible from inside of the cluster.

For the *single* experiment example, execute these commands:

```
mkdir /mnt/input
mkdir /mnt/input/single
mkdir /mnt/output
... mount command provided with nctl mount used to mount output storage to /mnt/output
... mount command provided with nctl mount used to mount input storage to /mnt/input
cp /mnt/output/single/models/* -Rf /mnt/input/single/
```

After these steps `/mnt/input/single` should contain:

```
/mnt/input/single/:
00001
/mnt/input/single/00001:
saved_model.pb  variables
/mnt/input/single/00001/variables:
variables.data-00000-of-00001  variables.index
```

Running a Prediction Instance

The following provides a brief example of running inference using the `batch` command. For more information, refer to [Evaluating Experiments with Inference Testing](#) on [page 59](#).

Before running the `batch` command, you need to copy `protobuf` requests to input storage, because they need to be accessed by the prediction instance too.

Execute these commands:

```
mkdir /mnt/input/data
cp /tmp/mnist_test/conversion_out/* /mnt/input/data
```

To create a prediction instance, execute these commands:

```
nctl predict batch -m /mnt/input/home/single -d /mnt/input/home/data --model-name mnist
--name single-predict
```

The following are the example results of this command:

Prediction Instance	Model location	State
single-predict	/mnt/input/home/single	QUEUED

Notice the additional `home` directory in path to both model and input data. This is how the path looks from the perspective of the prediction instance. The `mnist_converter_pb.py` creates requests to the MNIST model. `--model-name mnist` is where this MNIST name is given to the prediction instance.

Note: Refer to [predict Command](#), [page 90](#) for additional predict command information.

Prediction Instance Complete

After the prediction instance completes (can be checked using the `predict list` command), you can collect instance responses from output storage.

In the example, it would contain *100 protobuf* responses. These can be validated using `mnist_checker.py`.

Running the following command locally will display the error rate calculated for this model and this sample of the test set.

```
python examples/mnist_checker.py --input_dir /mnt/output/single-predict
```

Viewing Experiment Output Folder

You can use the following steps to mount the output folder and view TensorBoard data files.

Mount a folder to your Nauta namespace output directory:

1. **macOS/Ubuntu:** Mount your Nauta output directory to a local folder. Create a folder for mounting: `my_output_folder`.

```
mkdir my_output_folder
```

2. To see the correct mount options and command, execute:

```
nctl mount
```

3. Use the mounting command that was displayed to mount Nauta storage to your local machine. The following are examples of mounting the local folder to the Nauta output folder for each OS:

- o **macOS:**

```
mount_mbfsc // 'USERNAME:PASSWORD'@CLUSTER-URL/output my_output_folder
```

- o **Ubuntu:**

```
sudo mount.cifs -o username=USERNAME,password=PASSWORD,rw,uid=1000 \ //CLUSTER-URL/output my_output_folder
```

- o **Windows:** Use Y: drive as mount point.

```
net use Y: \\CLUSTER-URL\output /user:USERNAME PASSWORD
```

4. Navigate to the mounted location.
 - o **MacOS/Ubuntu only:** Navigate to the `my_output_folder`.
 - o **Windows only:** Open Explorer Window and navigate to Y: drive.
5. See the saved event file by navigating to `mnist-single-node/tensorboard`. An example file is shown below.

```
events.out.tfevents.1542752172.mnist-single-node-master-0
```

6. Unmount Nauta storage using one of the following commands:

- o **MacOS:** `umount my_output_folder`
- o **Ubuntu:** `sudo umount my_output_folder`
- o **Windows:** Eject or `net use Y: /delete`

Additional Mounting and Unmounting Information

For more information on mounting, refer to [Working with Datasets](#), page 29. To unmount previously mounted Nauta input storage from a local folder/machine, refer to [Unmounting Experiment Input from Storage](#), page 39.

Removing Experiments

An experiment that has been completed and is no longer needed can be removed from the experiment list using the `cancel` command and its `--purge` option.

- If the `--purge` option *is not* set in the `cancel` command, the experiment will only change status to CANCELLED.
- If the `--purge` option is set in the `cancel` command, experiment objects and logs will be irreversibly removed (the experiment's artifacts will remain in the Nauta storage output folder).

Syntax:

```
nctl experiment cancel [options] EXPERIMENT-NAME
```

Enter this command, substituting your experiment name:

```
nctl experiment cancel --purge <your-experiment>
```

Working with Datasets

The section covers the following main topics:

- [Uploading Datasets, page 29](#)
- [nctl mount Command, page 29](#)
- [Mount and Access Folders, page 30](#)
- [Uploading and Using Shared Dataset Example, page 31](#)

Uploading Datasets

Nauta uses NFS to connect to a storage location where each user has folders that have been setup to store experiment input and output data. This option allows the user to upload files and datasets for private use and for sharing. Once uploaded, the files are referenced by the path.

All data in the folders are retained until the user manually removes it from the NFS storage. Refer to the following sections in this chapter for information on how to access and use Nauta storage.

nctl mount Command

The `mount` command displays another command that can be used to mount Nauta folders to a user's local machine. When a user executes the command, information similar the following is displayed (this example is for macOS).

Use the following command to mount those folders (all of the following is displayed, although this is an example only).

Use the following command to mount those folders:

- replace <MOUNTPOINT> with a proper location on your local machine)
- replace <NAUTA_FOLDER> with one of the following:
 - input - User's private input folder (read/write)
(can be accessed as /mnt/input/home from training script).
 - output - User's private output folder (read/write)
(can be accessed as /mnt/output/home from training script).
 - input-shared - Shared input folder (read/write)
(can be accessed as /mnt/input/root/public from training script).
 - output-shared - Shared output folder (read/write)
(can be accessed as /mnt/output/root/public from training script).
 - input-output-ro - Full input and output directories, read only.

Additionally, each experiment has a special folder that can be accessed as /mnt/output/experiment from training script. This folder is shared by Samba as output/<EXPERIMENT_NAME>.

```
-----
mount_smbfs
// 'jane:UX9ucv2uphw3iFgmDgivAo5p7PYwP34'@10.91.120.167.lab.nervana.sclab.intel.com/
<NAUTA_FOLDER> <MOUNTPOINT>
```

Use following command to unmount previously mounted folder:

```
umount <MOUNTPOINT> [-f]
```

In case of problems with unmounting (disconnected disk etc.) try out -f (force) option. For more info about these options refer to man umount.

Other nctl mount and mount Information

The `nctl mount` command also returns a command to unmount a folder. Nauta uses the mount command that is native to each operating system, so the command printed out *may not* appear as in this example. In addition, *all variables* are shown in upper-case.

Mount and Access Folders

Table 4 displays the access permissions for each mounting folder.

Table 4: Access Permissions for Mounting Folders

Nauta Folder	Reference Path	User Access	Shared Access
input	/mnt/input/home	read/write	-
output	/mnt/output/home	read/write	-
input-shared	/mnt/input/root/public	read/write	read/write
output-shared	/mnt/output/root/public	read/write	read/write

Nauta Folder	Reference Path	User Access	Shared Access
input-output-ro		read	read

Uploading and Using Shared Dataset Example

The default configuration is to mount local folders to Nauta user private input and output storage folders. Perform these steps below to mount a local folder, `my-input`, to Nauta storage so that input data can be referenced from the storage when performing training.

1. **Linux/macOS only:** Create a folder for mounting named `my-input` folder:

```
mkdir my-input
```

2. Use `nctl mount` to display the mounting command for your operating system.

```
nctl mount
```

3. Enter the mount command that is provided by `nctl mount` using the input-shared folder as the `NAUTA-FOLDER` and `my-shared-input` folder or `Y:` as the `MOUNTPOINT`. Examples of mounting the local folder to the Nauta input folder, are:

- o **MacOS only:**

```
mount_mbfsc // 'USERNAME:PASSWORD'@CLUSTER-URL/input my-input
```

- o **Ubuntu only:**

```
sudo mount.cifs -o username=USERNAME,password=PASSWORD,rw,uid=1000 //CLUSTER-URL/input my-input
```

- o **Windows only:** Use `Y:` drive as mount point net case `Y:`:

```
net case Y: \\CLUSTER-URL\input /user:USERNAME PASSWORD
```

4. Navigate to the mounted location:
 - o **MacOS/Ubuntu only:** Navigate to `my-input` folder.
 - o **Windows:** Open Explorer Window and navigate to `Y:` drive
5. Copy a dataset or files to the folder for use in experiments. The files will be located in the Nauta storage until deleted.
6. Using the MNIST example from [Submitting an Experiment on page 14](#), you can download the MNIST dataset from this link: [MNIST Dataset](#).
7. Create a `MNIST` folder in the Nauta input folder.
8. Copy the downloaded files to the folder.
9. Submit an experiment referencing the new shared dataset. From the `nctl home` directory, run this command:

```
nctl experiment submit --name mnist-input examples/mnist_single_node.py -- --data_dir=/mnt/input/home/MNIST
```

Uploading and Using a Shared Dataset

If you want to copy your data to a shared folder, use `input-shared` instead of `input` in step 3. Using the shared Nauta storage will allow all Nauta users to use the same `MNIST` dataset by referencing the shared path:

```
/mnt/input/root/public/MNIST
```

Uploading During Experiment Submission

Uploading additional datasets or files is an option available for the `submit` command, using the following option:

```
-sfl, --script_folder_location
```

Where `script_folder_location` is the name of a folder with additional files used by a script, for example: other `.py` files, datasets, and so on. If the option *is not* included, the files *will not* be included in the experiment.

Syntax:

```
nctl experiment submit --script_folder_location DATASET-PATH SCRIPT-LOCATION
```

This option may be used only for small datasets for development purposes (datasets larger than several MB should be uploaded using standard mechanism described above).

WARNING: Submitting large amount of data using this option will prolong experiments' submission time.

Working with Experiments

This section provides instructions about the following topics:

- [Launching Jupyter Interactive Notebook](#), page 33
- [Submitting a Single Experiment](#), page 36
- [Submitting Multiple Experiments](#), page 36
- [Running an Experiment on Multiple Nodes](#), page 38
- [Mounting Storage to View Experiment Output](#), page 39
- [Unmounting Experiment Input from Storage](#), page 39
- [Cancelling Experiments](#), page 39
- [Cancelling All Experiments with a Matching Pod-ID](#), 40

Launching Jupyter Interactive Notebook

You can use Jupyter Notebook to run and display the results of your experiments.

Launching Jupyter Interactive Notebook Instructions

This release of Nauta supports Python 3 and 2.7 for scripts. Launch Jupyter Notebook using the following command:

Syntax:

```
nctl experiment interact [options]
```

Options, include:

- `name` - The name of this Jupyter Notebook session.
- `filename` - File with a notebook or Python script that should be opened in Jupyter Notebook.

For detailed command syntax information, refer to: [experiment interact Subcommand](#), page 81.

Execute this command to launch Jupyter:

```
nctl experiment interact
```

Storage and Session Data

Files located in the input storage are accessible through Jupyter Notebooks. Only files that are written to `/output/home/` are persistently stored. Therefore, changes made to other files, including model scripts, during the session *will not* be saved after the session is closed. It is recommended that you save session data to the `output/<experiment>` folder for future use.

Note: Files that are accessible through the Jupyter Notebook are the same folders accessible to the user for experiments.

Tunneling

If you are using CLI through remote access, you will need to setup a X server for tunneling over SSH with port forwarding or use SSH Proxy command tunneling. After establishing a tunnel from the gateway to your local machine, you can use the URL provided by `nctl`.

The following result displays.

```
Submitting experiments.
| Experiment                | Parameters | State   | Message |
|-----+-----+-----+-----|
| jup-936-18-09-17-20-14-58 |           | QUEUED |         |

Browser will start in a few seconds. Please wait...
Go to http://localhost:28113
Proxy connection created.
Press Ctrl-C key to close a port forwarding process...
```

Figure 5 shows the Jupyter Notebook launched in a default web browser.

Figure 5: Jupyter Notebook—Example Only

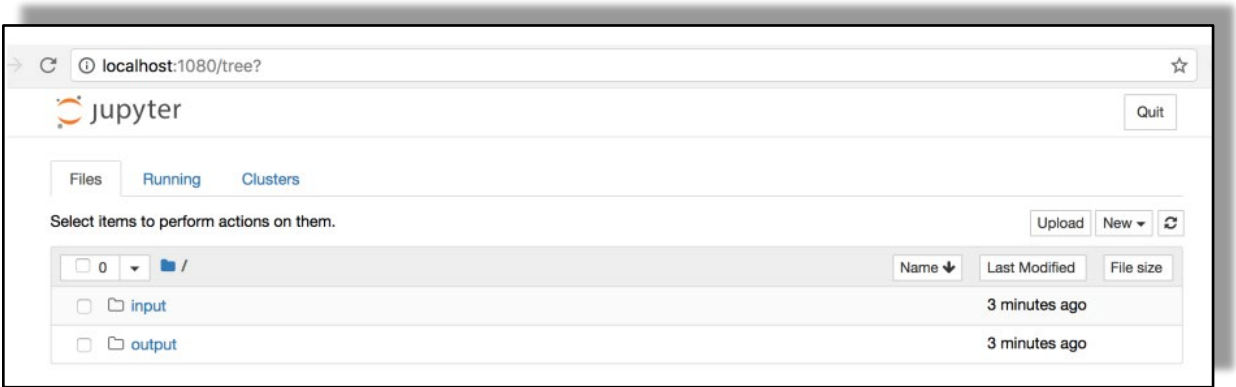
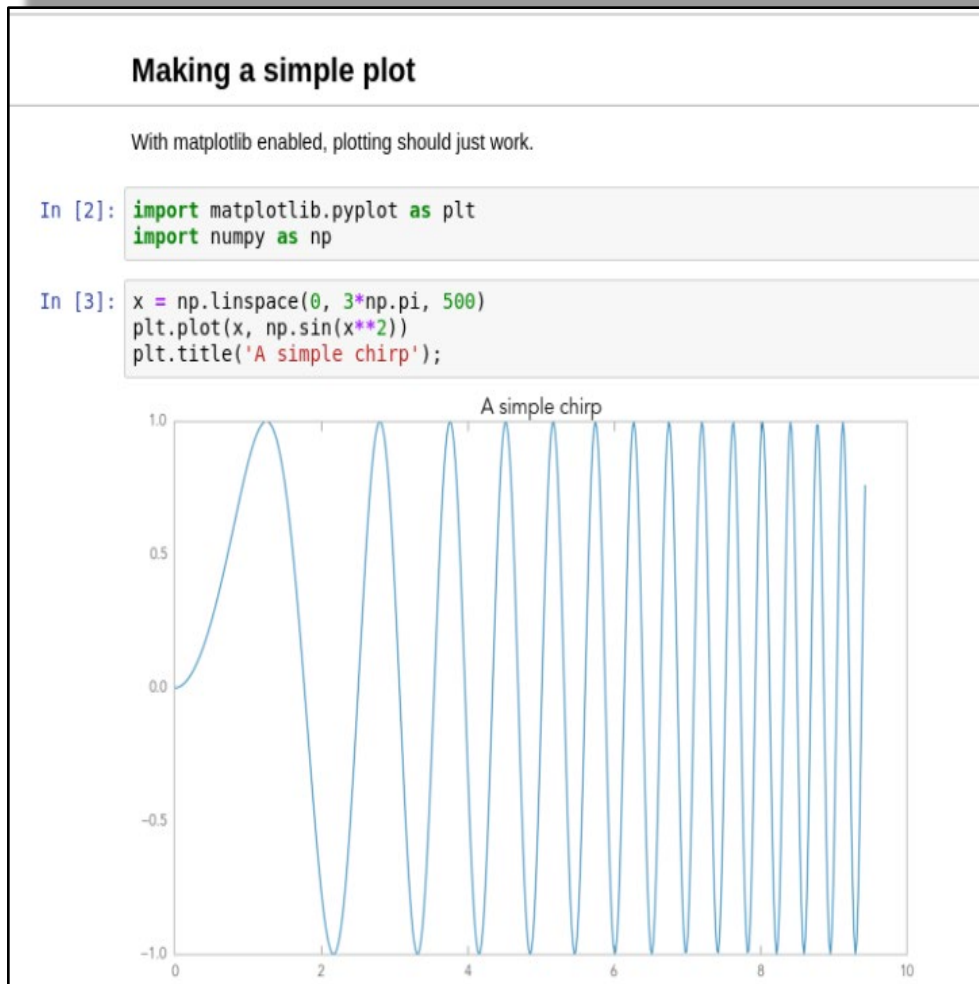


Figure 6 shows an example of an active Jupyter Notebook: a simple experiment plot.

Figure 6: Jupyter Notebook Simple Experiment Plot—Example Only



Cancelling a Jupyter Notebook

In Nauta, running a Jupyter notebook is done through an interact session. The session will remain open such that the Jupyter notebook that is running will continue when the browser is closed. Therefore, *you must* manually cancel the interact session, or it will continue to allocate resources.

Steps to Manage and Cancel Interacts

1. To see all running jobs, enter the following command:

```
nctl experiment list --status RUNNING
```

2. To cancel a running interact job, execute:

```
nctl experiment list --status RUNNING
```

- o EXPERIMENT-NAME is the interact session name.
- o The `--purge` option should be used if the user wishes to remove the session from the experiment list. Refer to [Removing Experiments, page 28](#).

3. To verify that cancellation has completed, enter this command:

```
nctl experiment list --status RUNNING
```

Submitting a Single Experiment

Your script *must be* written to process your input data as it is presented, or conversely, your data *must be* formatted to be processed by your script. No specific data requirements are made by the Nauta software.

Note: Refer to [Submitting an Experiment, page 14](#) for more information.

Submitting Multiple Experiments

This section describes how to launch multiple experiments using the same script.

Storage locations for your input and output folders are determined by the mount command. Refer to [Working with Datasets, page 29](#).

To submit multiple individual experiments that use the same script, use the following syntax for this command.

Syntax:

```
nctl exp submit -parameter-range SCRIPT_LOCATION [-- script-parameters]
```

Note: SCRIPT_NAME above refers to values (set or of a range) of a single parameter.

Example:

An example command is shown below:

```
nctl experiment submit -parameter-range lr "{0.1, 0.2, 0.3}"  
examples/mnist_single_node.py -- --data_dir=/mnt/input/root/public/MNIST
```

Refer to [Working with Datasets](#) for instructions on uploading the dataset to the `input_shared` folder.

Parameter Ranges and Parameter Sets

Parameters can include either:

- `parameter-range` argument that defines the name of a parameter together with its values expressed as either a range or an explicit set of values

-Or-

- `parameter-set` argument that specifies a number of distinct combinations of parameter values.

Example:

An *example* of this command using `parameter-range` is shown below.

```
nctl experiment submit --name parameter-range -parameter-range lr "{0.1, 0.2, 0.3}"
mnist_single_node.py -- --data_dir=/mnt/input/root/public/MNIST
```

The following result displays.

```
Please confirm that the following experiments should be submitted.
```

Experiment	Parameters
parameter-range-1	mnist_single_node.py
	lr=0.1
parameter-range-2	mnist_single_node.py
	lr=0.2
parameter-range-3	mnist_single_node.py
	lr=0.3

Do you want to continue? [Y/n]: y

Experiment	Parameters	State	Message
parameter-range-1	mnist_single_node.py lr=0.1	QUEUED	
parameter-range-2	mnist_single_node.py lr=0.2	QUEUED	
parameter-range-3	mnist_single_node.py lr=0.3	QUEUED	

Note: Your script *must be* written to process your input data as it is presented, or conversely, your data *must be* formatted to be processed by your script. No specific data requirements are made by the Nauta software

Run an Experiment on Multiple Nodes

This section describes how to submit an experiment to run on multiple processing nodes, to accelerate the job. Storage locations for your input and output folders are determined by the mount command. Refer to the [section Working with Datasets](#), page 29.

This experiment uses a template. For more information, refer to [Working with Template Packs](#), page 41.

To run a multi-node experiment, the script must support it. Following is the generic syntax (the line wrap is *not* intended).

Syntax:

```
nctl experiment submit [options] SCRIPT_LOCATION --template [MULTINODE_TEMPLATE_NAME]
SCRIPT_LOCATION [-- script-parameters]
```

Example:

The template `multinode-tf-training-tfjob` is included with Nauta software. The following is an example command using this template (line wrap is *not* intended):

```
nctl experiment submit --name multinodes --template multinode-tf-training-tfjob
/examples/mnist_multinode.py -- -- data_dir=/mnt/input/root/public/MNIST
```

Result:

The following result displays (a partial) queued job.

```
Submitting experiments.
| Run          | Parameters used          | Status  | Message |
|-----+-----+-----+-----|
| multinodes   |                          | QUEUED  |         |
```

In the above command, to optionally set the number of workers and servers, set these as parameters below. The default values are 3 worker nodes and 1 parameter server. The following parameters are set to 2 worker nodes and 1 parameter server.

```
-p workers_Count 2
-p pServersCount 1
```

Mounting Storage to View Experiment Output

Refer to the [section Working with Datasets](#), page 29.

Unmounting Experiment Input from Storage

Perform these steps to unmount previously mounted Nauta storage from a local folder/machine.

1. Use the `mount` command to display the command that should be used to unmount your local folder/machine from your Nauta input folder.

```
nctl mount
```

2. Perform the unmount using `umount` or `net use` as appropriate. The command is dependent on the operating system.

Canceling Experiments

The `nctl experiment cancel` command stops and cancels any experiment queued or in progress. Furthermore, the command also cancels any experiment based on the name of an *experiment/pod/status* of a pod. If any such an object is found, the command queries if these objects (one or more) should be cancelled.

To cancel one or more experiments, execute the following command:

```
nctl experiment cancel[Options] EXPERIMENT-NAME
```

The value of this argument should be created using rules described here. Use this command to cancel one or more experiments with matching or partially-matching names, a matching pod ID, matching pod status, or combinations of these criteria. For example, the following command will cancel all experiments with a matching or partially matching name:

Syntax:

```
nctl experiment cancel --match EXPERIMENT-NAME
```

Cancelling All Experiments with a Matching Pod-ID

The following command will cancel all experiments with a matching pod-ID, using one or more comma-separated IDs:

Syntax:

```
nctl experiment cancel --pod-ids [pod_ID] EXPERIMENT-NAME
```

The following command will cancel all experiments with a matching pod-status, using one of the following statuses: [PENDING, RUNNING, SUCCEEDED, FAILED, UNKNOWN]:

```
nctl experiment cancel --pod-status [PENDING, RUNNING, SUCCEEDED, FAILED, UNKNOWN]  
EXPERIMENT-NAME
```

Any of the above criteria can be combined. You can also purge all information concerning any experiment using the `-p` or `--purge` option.

Working with Template Packs

This section discusses the following topics:

- [What is a Template Pack?, page 41](#)
- [Pack Anatomy, page 43](#)
- [Creating a New Template Pack, page 46](#)
- [Nauta values.yaml Placeholders, page 47](#)

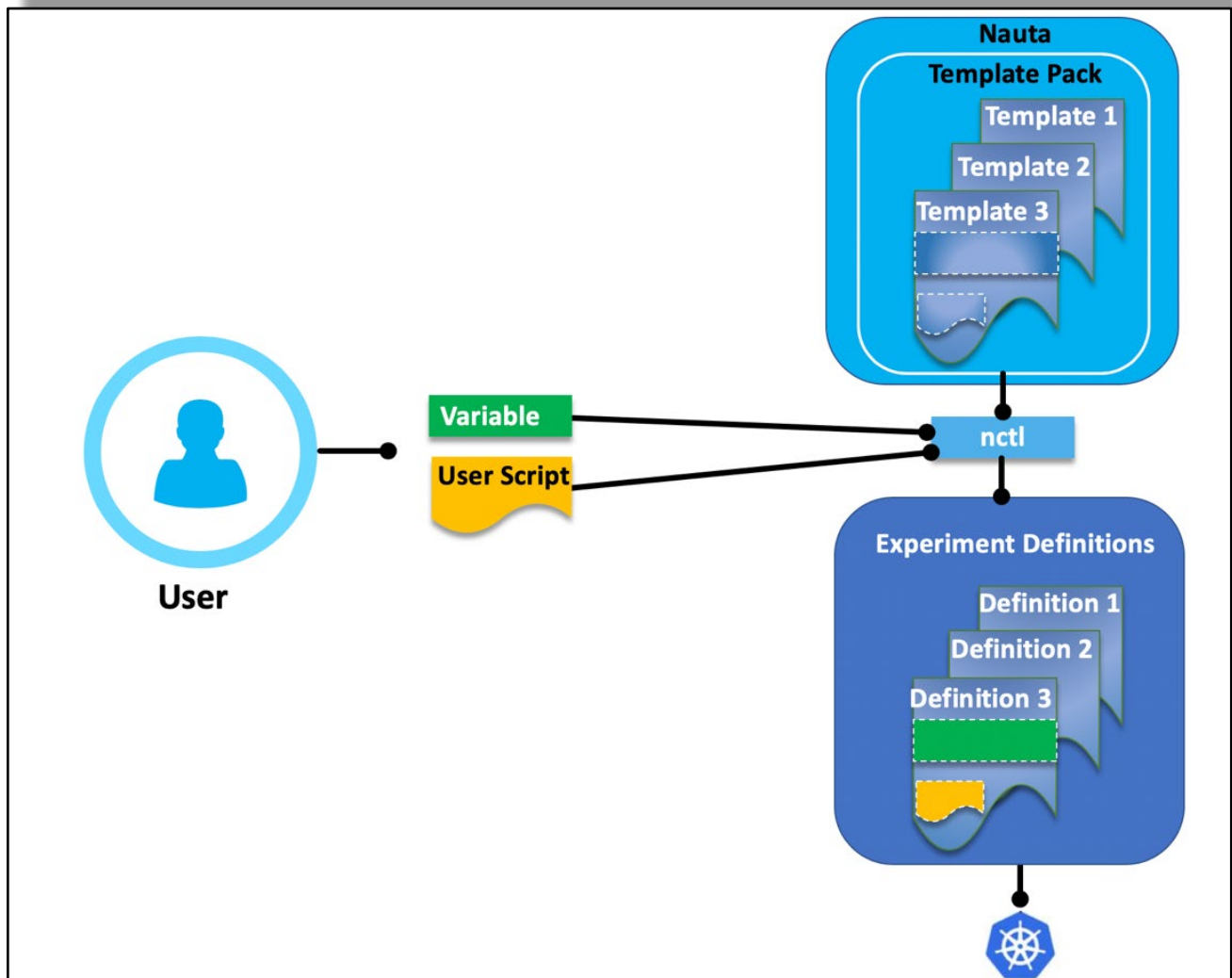
What is a Template Pack?

Every experiment run on the Nauta application utilizes a template pack (as shown in [Figure 7](#)). For each experiment, a template pack defines the experiment's complete runtime environment and any supporting infrastructure required to run that experiment.

Each template pack includes a number of elements or templates that together define the Kubernetes (K8s) application, which executes a user-provided experiment script based on specific supporting technology.

Each template pack includes templates that define a Dockerfile, the Kubernetes service definition, deployments, jobs, a configuration map and any other standard Kubernetes elements needed to create a runtime environment for an experiment instance.

Figure 7: Template Pack



Individual elements within a pack are referred to as templates because they contain a number of placeholders that are substituted with appropriate values by Nauta software during experiment submission. Some placeholders are required, while others are optional. These placeholders define items such as: experiment name, user namespace, the address of the local Nauta Docker registry, and other variables that may change between different experiment runs.

The core Kubernetes definitions within each pack are grouped into [Helm](#) packages referred to as *Charts*. Helm is the de-facto standard for Kubernetes application packaging and reusing this package format allows leveraging of the large resource of community-developed Helm charts when creating new Nauta template packs.

Note 1: While Nauta is able to re-use Helm charts mostly verbatim, there are a number of required placeholders that need to be added to these charts for Nauta to track and manage the resulting experiments. Refer to [Creating a New Template Pack](#), page 46 for details.

Note 2: All supported Nauta template packs are distributed with the nctl package.

Pack Anatomy

Location

When the `nctl` package is installed on the client machine, the template packs that come with the official package are deposited in the folder:

```
NAUTA_HOME/config/packs
```

Each pack resides in a dedicated sub-folder, named after the pack.

The Pack Folder Structure

The individual items that form a single pack are laid out in its folder as follows:

```
<PACK_NAME>/
  Dockerfile
  charts/
    Chart.yaml
    values.yaml
  templates/
```

Table 5: Template Pack Structure Additional Information

Dockerfile	Charts	Chart.yaml	values.yaml	templates
<i>Dockerfile</i> is the Docker file that defines the Docker image which serves as the runtime for the experiment's script supplied by the user. Any dependencies needed to build the Docker image must be placed in this directory, next to the <i>Dockerfile</i> .	<i>charts</i> is a directory that hosts the Helm chart that specifies the definitions of all Kubernetes entities used to deploy and support the experiment's Docker image in the cluster.	<i>Chart.yaml</i> provides the key metadata for the chart, such as name and version, and about the chart.	<i>values.yaml</i> serves a key role as it provides definitions for various Helm template placeholders (refer to Helm's Chart Template Guide for details) used throughout the chart (mostly in the individual Kubernetes definitions contained within the templates sub-folder). This file is also parsed and analyzed by <code>nctl</code> to perform substitution on <i>placeholders</i> in <i>values.yaml</i> on page 47	<i>templates</i> folder groups all the YAML files that provide definitions for various Kubernetes (K8s) entities, which define the packs deployment and runtime environment. These definitions are referred to as templates as they may include Helm template placeholders substituted for actual values in the process of deploying the chart on the cluster.

Provided Template Packs

The Nauta software is shipped with a number of built-in template packs that represent the types of experiments officially supported and validated.

For each of the packs there are two versions provided: one that supports Python 2.7.x user scripts (packs with `-py2` suffix in the name) and one that supports Python 3.5.x user scripts.

Packs with multi-prefix in the name support multi-node experiments, while those with a single-prefix are designed for single node experiments only.

All packs are optimized for non-trivial deep learning tasks executed on Intel's two socket Xeon systems, and therefore the default compute configuration is the following in [Table 6](#).

Table 6: Compute Configurations for Template Packs

Type	CPU	Memory	Total Experiment per Node
Single-node packs	1 CPU per node	~0.4 available memory	2
Multi-node packs	2 CPUs per node	~0.9 available memory	1

In general, the single-node packs are configured to take roughly half of the available resources on a single node (so that the user can *fit* two experiments on a single node), while multi-node packs utilize the entire resources on each node that participates in the multi-node configuration.

While these defaults are intended to guarantee the best possible experience when training on Nauta, it is possible to adjust the compute resource requirements either on per-experiment basis or permanently (refer to [Customizing the Provided Packs](#), page 44 for more information). [Table 7](#) describes the template packs provided with Nauta.

Table 7: Template Pack Structure Additional Information

multi-tf-training-horovod	multi-tf-training-horovod-py2	multi-tf-training-tfjob	multi-tf-training-tfjob-py2	single-tf-training-tfjob	single-tf-training-tfjob-py2
- A TensorFlow multi-node training job based on Horovod using Python 3.	A TensorFlow multi-node training job based on Horovod using Python 2	- A TensorFlow multi-node training job based on TF-operator using Python 3	A TensorFlow multi-node training job based on TF-operator using Python 2	A TensorFlow single-node training job based on TF-operator using Python 3	A TensorFlow single-node training job based on TF-operator using Python 2

Customizing the Provided Packs

Any customizations to template packs revolve mostly around the `values.yaml` file included in the pack's underlying Helm chart. As mentioned in [The Pack Folder Structure](#) on page 43, this file provides key definitions that are referenced throughout the rest of the Helm chart, and therefore it plays a crucial role in the process of converting the chart's templates into actual Kubernetes definitions deployed on the cluster.

By convention, the definitions contained in the `values.yaml` file typically reference parameters that are intended to be customized by end-users, so in most cases it is safe to manipulate those without corrupting the pack.

Note: This is in contrast to parameters *not* intended for customization. In addition, these parameters typically live within the templates themselves.

Altering Parameters Listed in values.yaml

When altering parameters listed in the `values.yaml` file, there are two approaches:

1. Users may manually modify the pack's `values.yaml` file using a text editor. Any modifications done using this approach will be permanent and apply to all subsequent experiments based on this pack.
2. Users may alter some of the parameters listed in `values.yaml` file temporarily and *only* for a single experiment. To do so, the user may specify alternative values for any of the parameters listed in `values.yaml` using the `--pack_param` option when submitting an experiment (refer to [CLI Commands](#), page 69 for more details).

Advanced users who want full control over how their experiments are deployed and executed on the Kubernetes cluster may also directly modify the templates residing in the `<PACK-NAME>/charts/templates/` folder. Doing this, however requires a good grasp of Kubernetes concepts, notation, and debugging techniques, and is therefore *not recommended*.

Creating a New Template Pack

Prerequisites

Creating a new pack, while not overly complex, requires some familiarity with the technologies that packs are built on. Therefore, it is recommended to have at least some working experience in the following areas do this:

- Creating/modifying Helm charts and specifically using the [Helm templates](#).
- Defining and managing Kubernetes entities such as `Pods`, `Jobs`, `Deployments`, `Services`, and so on.

Where to Start

Creating new template packs for Nauta is greatly simplified by leveraging the relatively ubiquitous Helm chart format as the foundation.

Thus, the starting point for a new template pack is typically an existing Helm chart that packages the technology of choice for execution on a K8s cluster. Consider creating a chart from scratch only if an existing chart *is not* available. The process of creating a new Helm chart from scratch is exhaustively described in the [Official Helm documentation](#).

A Template Pack in Five Simple Steps

Once a working Helm chart is available, the process of adapting it for use as a Nauta template is as follows:

1. Pick the pack's name.

The name should be unique and not conflict with any other packs available in the local `packs` folder. After naming the pack, create a corresponding directory in the `packs` folder and populate its `charts` subfolder with the contents of the chart. Do not forget to set this pack name also in the `Chart.yaml` file. Otherwise the new template *will not* work.

2. Create a Dockerfile.

This Dockerfile will be used to build the image that will host the experiment's scripts. As such, it should include all libraries and other dependencies that experiments based on this pack will use at runtime.

3. Update `values.yaml` (or create it if it *does not* exist).

The following items that must be placed in the chart's `values.yaml` file in order to enable proper experiment tracking:

- The `podCount` element must be defined and initialized with the expected number of experiment pods that must enter the Running state in order for Nauta to consider the experiment as started.
- If the experiment script to be used with the pack accepts any command-line arguments, then a `commandline` parameter must be specified and assigned the value of `NAUTA.CommandLine`. (Refer to [NAUTA.CommandLine](#), page 47). This will allow the command line parameters specified in the `nctl experiment submit` command to be propagated to the relevant Helm chart elements (by referencing the `commandline` parameter specified in `values.yaml`)
- An image parameter must be specified and assigned the value of `NAUTA.ExperimentImage`. (Refer to [NAUTA.ExperimentImage](#), page 47).

- The actual name of this parameter *does not* matter as long as it is properly referenced wherever a container image for the experiment is specified within the chart templates.
4. Add tracking labels.
- The `podCount` element specified above indicates how many pods to expect within a normally functioning experiment based on this pack. The way Nauta identifies the pods that belong to particular experiment is based on specific labels that need to be assigned to each pod that *should* be included in the `podCount` number. The label in question is `runName` and it needs to be assigned the value corresponding to the name of the current Helm release (by assigning the `Helm {{ .Release.name }}` template placeholder).
- Note:** Not all pods within an experiment need to be accounted for in `podCount` and assigned the aforementioned label. Nauta only needs to track the pods in which the runtime state is representative of the overall experiment status. If, for instance, an experiment is composed of a *master* pod which in turn manages its fleet of worker pods, then its sufficient to set `podCount` to 1 and only track the *master* as long as it's state (*PENDING*, *RUNNING*, *FAILED*, and so on) is representative for the entire group.
5. Update container image references.
- All container image definitions with the chart's templates that need to point to the image running the experiment script (as defined in the `Dockerfile` in step #1) need to refer to the corresponding `image` Helm template placeholder as previously defined in `values.yaml` (step #3 above).

Nauta values.yaml Placeholders

NAUTA.CommandLine

The `NAUTA.CommandLine` placeholder, when placed within the `values.yaml` file, will be substituted for the list of command line parameters specified when submitting an experiment via `nctl experiment submit` command.

To pass this list as the command line into one of the containers defined in the pack's templates, it needs to be first assigned to a parameter within `values.yaml`. This parameter then needs to be referenced within the chart's templates just like any other Helm template parameter.

The following example snippet shows the placeholder being used to initialize a parameter named: `commandline`:

```
commandline:
  args:
    {% for arg in NAUTA.CommandLine %}
    - {{ arg }}
    {% endfor %}
```

NAUTA.ExperimentImage

The `NAUTA.ExperimentImage` placeholder carries the full reference to the Docker image resulting from building the `Dockerfile` specified within the pack.

During experiment submission the image will be built by Docker and deposited in the Nauta Docker Registry under the locator represented by this placeholder.

Hence, the placeholder shall be used to initialize a template parameter within the `values.yaml` file, that will later be referenced within the chart's templates to specify the experiment image.

Below is a sample definition of a parameter within `values.yaml`, followed by a sample reference to the image in pod template.

```
<values.yaml>
image: {{ NAUTA.ExperimentImage }}

<pod.yaml>
containers:
- name: tensorflow
  image: "{{ .Values.image }}"
```


Evaluating Experiments

This section discusses the following topics:

- Viewing Experiments Using the CLI, page 49
- Viewing Experiment Logs and Results Data, page 51
- Viewing Experiment Results at the Web UI, page 52
- Launching TensorBoard to View Experiments, page

Viewing Experiments Using the CLI

Viewing all Experiments

To list *all* experiments you have submitted, run the next command. The possible returned statuses are:

Table 8: Returned Experiment Status

QUEUED	RUNNING	COMPLETE	CANCELLED	FAILED	CREATING
Experiment has been scheduled but <i>is not</i> yet running	Experiment is running	Experiment completed successfully	Experiment has been cancelled by a user	Experiment has failed	Experiment is being created

Syntax:

```
nctl experiment list [options]
```

Example:

An example is shown below:

```
nctl experiment list
```

Result:

The following (partial output) example results are shown below.

Experiment	Parameters	Metrics	Submission date
single	mnist_single_node.py	accuracy: 0.96875 global_step: 499 loss: 0.08342029 validation accuracy: 0.9818	2019-03-20 05:03:12 PM
single2	mnist_single_node.py	accuracy: 0.953125 global_step: 499 loss: 0.078533165 validation accuracy: 0.9838	2019-03-20 05:06:19 PM

Viewing a Single Experiment's Details

The primary purpose of the next command is to provide Kubernetes pod-level information and container information for this experiment. This includes the pod ID, the POD status, information about input and

output volumes used in this experiment, and CPU and memory resources requested to perform this experiment.

Use the following command to view a single experiment's details (this is an example only):

Syntax:

```
nctl experiment view [options] EXPERIMENT-NAME
```

Example:

An example experiment view (the example name used is mnist-tb) is shown below.

```
nctl experiment view mnist-tb
```

Result:

The following (partial output) example results are shown below.

```
| Experiment | Parameters | Metrics | Submission date |
|-----|-----|-----|-----|
| mnist-tb | mnist_single_node.py | accuracy: 1.0 | 2019-03-20 05:11:15 PM |
| | | global step: 499 | |
| | | loss: 0.035771053 | |
| | | validation_accuracy: 0.9804 | |
Pods participating in the execution:
| Name | Uid | Pod Conditions | Container Details |
|-----|-----|-----|-----|
| mnist-tb-master-0 | ca2ca2c4-4b2a-1 | Initialized: True | - Name: tensorflow
| | 1e9-9c55-525816 | reason: PodCompleted | - Status: Terminated, Completed
| | 060100 | Ready: False | - Volumes:
| | | reason: PodCompleted | input-home @ /mnt/input/home
| | | PodScheduled: True | input-public @ /mnt/input/root
| | | | output-home @ /mnt/output/home
Resources used by pods:
| Resource type | Total usage |
|-----|-----|
| CPU requests: | 4750m |
| Memory requests: | 4GiB 523MiB 562KiB |
| CPU limits: | 4750m |
| Memory limits: | 4GiB 523MiB 562KiB |
```

Volumes list include mount mode for each volume (in <> brackets), which can be either ro (read-only) or rw (read-write)

Viewing Experiment Logs and Results Data

Each experiment generates logs. This is the information generated during the run of the experiment and saved. If an experiment *did not* print out data during execution, the logs will be blank.

Separate from logs, the results or output of an experiment can be found by mounting the your output folder or output-shared folder. A training script should write to the Nauta output folder to save any output files.

Execute following command to view logs from a given experiment.

Syntax:

```
nctl experiment logs [options] EXPERIMENT-NAME
```

Example:

```
nctl experiment logs mnist-tb
```

Result:

The following result displays an example.

```
2019-03-20T16:11:38+00:00 mnist-tb-master-0 Step 0, Loss: 2.3015756607055664, Accuracy: 0.078125
2019-03-20T16:11:44+00:00 mnist-tb-master-0 Step 100, Loss: 0.13010963797569275, Accuracy: 0.921875
2019-03-20T16:11:49+00:00 mnist-tb-master-0 Step 200, Loss: 0.07017017900943756, Accuracy: 0.984375
2019-03-20T16:11:55+00:00 mnist-tb-master-0 Step 300, Loss: 0.08880224078893661, Accuracy: 0.984375
2019-03-20T16:12:00+00:00 mnist-tb-master-0 Step 400, Loss: 0.15115690231323242, Accuracy: 0.953125
2019-03-20T16:12:07+00:00 mnist-tb-master-0 Validation accuracy: 0.980400025844574
```

Note: Logs generated with sub-millisecond frequency may appear out of order when displayed. This is caused by the 1ms resolution of the underlying logging solution.

Viewing Experiment Results at the Web UI

The web UI lets you explore the experiments you have submitted. To view your experiments at the web UI, execute the following command:

```
nctl launch webui
```

The following screen displays (this is an example only).

Figure 8: Viewing Experiment Results from the Web UI—Example Only

TensorBoard* Eligibility	Name	Status	Submission Date	Start Date	Duration	Type
✓	training-p-988-19-01-17-12-50-12	COMPLETE	01/17/2019 12:50:15 pm	01/17/2019 01:05:44 pm	0 days, 0 hrs, 1 mins, 59 s	Training
○	training-p-368-19-01-17-12-41-56	FAILED	01/17/2019 12:41:59 pm	01/17/2019 12:42:14 pm	0 days, 0 hrs, 1 mins, 21 s	Training
○	metrics-py-519-19-01-17-11-58-35	COMPLETE	01/17/2019 11:58:38 am	01/17/2019 12:02:34 pm	0 days, 0 hrs, 4 mins, 11 s	Training
○	metrics-py-497-19-01-17-11-58-29	COMPLETE	01/17/2019 11:58:32 am	01/17/2019 12:02:28 pm	0 days, 0 hrs, 4 mins, 12 s	Training
○	metrics-py-174-19-01-17-11-58-22	COMPLETE	01/17/2019 11:58:25 am	01/17/2019 12:02:18 pm	0 days, 0 hrs, 4 mins, 11 s	Training
○	metrics-py-908-19-01-17-11-58-16	COMPLETE	01/17/2019 11:58:19 am	01/17/2019 12:02:10 pm	0 days, 0 hrs, 4 mins, 10 s	Training
○	metrics-py-893-19-01-17-11-58-09	COMPLETE	01/17/2019 11:58:12 am	01/17/2019 12:01:57 pm	0 days, 0 hrs, 4 mins, 11 s	Training
○	metrics-py-322-19-01-17-11-58-02	COMPLETE	01/17/2019 11:58:05 am	01/17/2019 12:01:48 pm	0 days, 0 hrs, 4 mins, 12 s	Training
○	metrics-py-391-19-01-17-11-57-55	COMPLETE	01/17/2019 11:57:58 am	01/17/2019 11:58:13 am	0 days, 0 hrs, 4 mins, 11 s	Training

The web UI shows the six columns listed below. Each of these column headings are clickable, so to re-sort the listing of experiments based on that column heading, ascending or descending order, click that column heading. The six columns are (shown in Table 9):

Table 9: Web UI Column Information

Name	Status	Submission Date	Start Date	Duration	Type
The left-most column lists the experiments by name	This column reveals experiment's current status, one of: <ul style="list-style-type: none"> • QUEUED • RUNNING • COMPLETE • CANCELED • FAILED • CREATING 	This column gives the submission date in the format: MM/DD/YYYY, hour:min:second AM/PM	This column shows the experiment start date in the format: MM/DD/YYYY, hour:min:second AM/PM. The Start Date (or time) will always be after the Submission Date (or time).	This column shows the duration of execution for this experiment in days, hours, minutes and seconds	Experiment Type can be <i>Training</i> , <i>Jupyter</i> , or <i>Inference</i> . Training indicates that the experiment was launched from the CLI. Jupyter indicates that the experiment was launched using Jupyter Notebook. Inference means that

Name	Status	Submission Date	Start Date	Duration	Type
					training is largely complete and you have begun running predictions (Inference) with this model. (If the model used for inference was already present, there was no training performed on Nauta.)
Note: You can perform the tasks discussed below at the Nauta web UI.					

Expand Experiment Details

Click **listed experiment name** to see additional details for that experiment. The following details are examples only. This screen is divided into two frames. left and right-side frames:

Left-side Frame

The left-side frame (see [Figure 9](#)) of the experiment details window shows Resources and Submission Date.

Table 10: Experiment Details—1

Resources	Submission Date and Time
Resources assigned to that experiment, specifically the assigned pods and their status and container information including the CPU and memory resources assigned	Displays the Submission Date and time.

Figure 9: Experiment Details—1

Resources:	Pods -- 1. Name: <i>mnist-tb-master-0</i> Pod Conditions: <i>Initialized: True , reason: PodCompleted, Ready: False , reason: PodCompleted, PodScheduled: True</i> Containers: ○ Name: <i>tensorflow</i> Resources: <i>cpu - 100m, , memory - 10Mi</i> Status: <i>Terminated, Completed</i> 2. Name: <i>mnist-tb-master-0</i> Pod Conditions: <i>Initialized: True , reason: PodCompleted, Ready: False , reason: PodCompleted, PodScheduled: True</i> Containers: ○ Name: <i>tensorflow</i> Resources: <i>cpu - 100m, , memory - 10Mi</i> Status: <i>Terminated, Completed</i>
Submission Date:	11/9/2018, 2:17:07 PM

Right-side Frame

The right-side frame (see [Figure 10](#)) of the experiment details window shows Start Date, End Date, Total Duration, Parameters, and Output.


Table 11: Experiment Details—2

Start Date	End date	Total Duration	Parameters	Output
The day and time this experiment was launched	The day and time this experiment was launched	The actual duration this experiment was instantiated	The experiment script file name and the log directory	Clickable links to download all logs and view the last 100 log entries

Figure 10: Experiment Details—2

Start Date:	11/9/2018, 2:17:14 PM
End Date:	11/9/2018, 2:18:28 PM
Total Duration:	0 days, 0 hrs, 1 mins, 14 s
Parameters:	mnist_with_summaries.py, --log_dir=/mnt/output/experiment/tb
Output:	Logs, All Download Logs, Last 100 View

Searching on Experiments

In the **Search** field at the far right of the UI , enter a string of alphanumeric characters to match the experiment name or other parameters (such as user), and list only those matching experiments. This Search function lets the user search fields in the entire list, not just the experiment name or parameters.

Adding/Deleting Columns

ADD/DELETE COLUMNS Button

Click **ADD/DELETE COLUMNS** to open a scrollable dialogue. The columns currently in use are listed first with their check box checked. Scroll down to see more, available columns listed next, unchecked.

Check/Uncheck Column Headings


Click to check and uncheck and select the column headings you prefer. Optional column headings include parameters such as Pods, End Date, Owner, Template, Time in Queue, and so on.

Column Heading Metrics

Column headings also include metrics that have been setup using the Metrics API, for a given experiment, and you can select to show those metrics in this display as well.

Refer to [Launching TensorBoard to View Experiments](#), page 56 for more information.

Launching Kubernetes Dashboard

1. Click the **Hamburger Menu**  at the far left of the UI to open a left frame.
2. Click **Resources Dashboard** to open the Kubernetes resources dashboard.

Refer to [Accessing the Kubernetes Resource Dashboard](#), page 67.

Launching TensorBoard to View Experiments

You can launch TensorBoard* from the Nauta web UI or the CLI; both methods are described.

Launching TensorBoard from the Web UI

To view the experiment's results in TensorBoard, TensorBoard data must be written to a folder in the directory `/mnt/output/experiment`.

To launch TensorBoard from the web UI and view results for individual experiments, execute these steps:

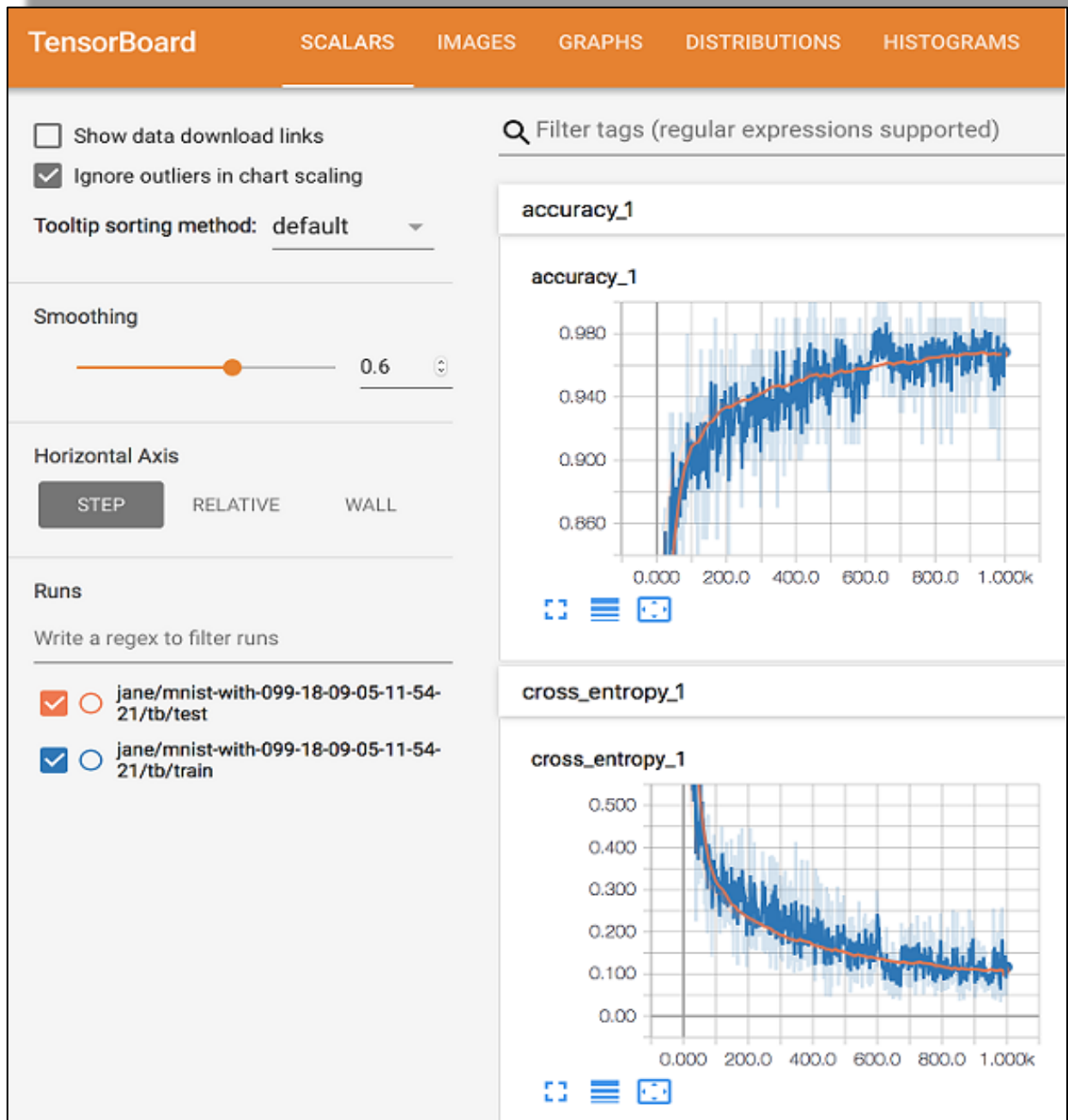
1. Open the web ui by executing this command:

```
nctl launch webui
```

2. At the web UI, identify the experiment that you want to see displayed in TensorBoard. Click on the check box to the left of the experiment name.
3. With an experiment selected (checked), the **LAUNCH TENSORBOARD** button becomes active. Click **LAUNCH TENSORBOARD**. TensorBoard is launched on a separate browser tab/window with graphs showing the experiment's results.

The following screen displays (this is an example only).

Figure 11: Launch TensorBoard from the Web UI – Example Only



Launching TensorBoard from the CLI

1. To launch TensorBoard from the CLI, enter the following command:

```
nctl launch tb <experiment_name>
```

2. The following result displays.

```
Please wait for Tensorboard to run...  
Go to http://localhost: 58218  
Proxy connection created.  
Press Ctrl-C key to close a port forwarding process...>
```

This command will launch a local browser. If the command was run with the `--no-launch` option, then you need to copy the returned URL into a web browser. TensorBoard is launched with graphs showing the experiment's results (as shown above).

You can also launch TensorBoard and with the `nctl experiment view` command:

```
nctl experiment view -tensorboard <experiment name>
```

Note: This command exposes a TensorBoard instance with data from the named experiment.

Evaluating Experiments with Inference Testing

For guidance on using inference testing to evaluate an experiment, refer to the topics shown below.

- [Using the predict Command](#), page 59
- [Batch Inference](#), page 59
- [Streaming Inference Example](#), page 61

Using the predict Command

Use the `predict` command to start, stop, and manage prediction jobs. Refer to the [predict Command](#), page 90 for a detailed description of this command.

Batch Inference Example

Example Flow

An example of the general flow is shown below. Ensure that you have:

1. Acquired the dataset and the trained model.
2. Converted dataset into *Serialized Protocol Buffers* (PBs). Refer to [Protocol Buffers](#) for additional PB information.
3. Invoked `nctl mount`.
4. Mounted the Samba shared folder by invoking the command displayed (in the step 3).
5. Copied the serialized PBs and the trained model to the *just-mounted* shared folder.
6. Run `nctl predict batch` command.

If the general flow requirements *are not* met, the user *will not* be able to complete the example.

MNIST Example

You need to have preprocessed MNIST data for feeding the batch inference. You can generate example data by performing the following steps:

MNIST Data Preprocessing

1. Create a virtual environment, `venv`, by executing the following command:

```
python3 -m venv .venv
```

2. Install required dependency in `venv`:

```
source .venv/bin/activate  
pip install tensorflow-serving-api
```

3. Run the `mnist_converter_pb.py` script using just generated `venv`:

```
python mnist_converter_pb.py
```

Parameters of mnist_converter_pb.py

- `work_dir` - Location where files related with conversion will be stored. The default is: `/tmp/mnist_tests`.
- `num_tests` - Number of examples to convert. Default: 100.

Note: Results of conversion are stored in `conversion_out` directory under `work_dir` parameter. The default is: `/tmp/mnist_tests/conversion_out`.

Start Prediction

1. Mount Samba input shared folder to your directory, assumed `/mnt/input`.
2. Then, use the command printed by `nctl mount`.
3. Copy model generated by: `trained_mnist_model` to: `/mnt/input`.
4. Enter the following command:

```
nctl predict batch --model-location /mnt/input/home/trained_mnist_model --data /mnt/input/home/parsed --model-name mnist
```

Other Important Information

Paths

Paths provided in locations such as, `--model-location` and `--data` need to point (for files/directory) from the container's context, *not* from a user's filesystem or mounts. These paths can be mapped using instructions from `nctl mount`.

For example, if you have mounted Samba `/input` and copied the files there, you should pass:

```
/mnt/input/home/<file>
```

Model Name

The `--model-name` is optional, but it *must* match the model name provided during data preprocessing, since generated requests *must* define which servable they target.

In the `mnist_converter_pb.py` script, you can find `request.model_spec.name = 'mnist'`. This saves the model name in requests, and that name *must* match a value passed as: `--model-name`.

5. If *not* provided, it assumes that the model name is equal to last directory in model location:

```
/mnt/input/home/trained_mnist_model --> trained_mnist_model
```

Useful References

- [Serving a TensorFlow Model](#)
- [Protocol Buffer Basics: Python](#)
- [mnist_client.py Script](#)
- [TensorFlow Serving with Docker](#)

Streaming Inference Example

This section discusses the following main topics:

- [Example Flow](#), page 61
- [TensorFlow Serving Basic Example](#), page 61
- [Streaming Inference with TensorFlow Serving REST API](#), page 63
- [Accessing the REST API with curl](#), page 63
- [Using Port Forwarding](#), page 63
- [Streaming Inference with TensorFlow Serving gRPC API](#), page 64
- [Useful External References](#), page 64

Example Flow

A basic task flow is used in this example.

1. The user has saved a trained TensorFlow Serving compatible model.
2. The user will be sending data for inference in JSON format, or in binary format using gRPC API.
3. The user runs `nctl predict launch` command.
4. The user sends inference data using the `nctl predict stream` command, Tensorflow Serving REST API, or TensorFlow Serving gRPC API.

TensorFlow Serving Basic Example

Launching a Streaming Inference Instance

Basic models for testing TensorFlow Serving are included in the following GitHub [TensorFlow Serving Repository](#). This example will use the `saved_model_half_plus_two_cpu` model for showing streaming prediction capabilities.

To use that model for streaming inference, perform following steps:

1. Clone the [TensorFlow Serving Repository](#) executing the following command:

```
git clone https://github.com/tensorflow/serving
```

2. Perform step 3 or step 4 below, based on preference.
3. Run the following command:

```
nctl predict launch --local_model_location <directory where you  
have cloned Tensorflow Serving>/serving/tensorflow_serving/ \  
servables/tensorflow/testdata \ /saved_model_half_plus_two_cpu
```

4. Alternatively, for step 3, you may want to save a trained model on input shared folder, so it can be reused by other experiments/prediction instances. To run these commands:

- a. Use the `mount` command to mount Nauta input folder to local machine:

```
nctl mount
```

- b. Use the resulting command printed by `nctl mount`. After executing command printed by `nctl mount` command, you will be able to access input shared folder on your local file system.
- c. Copy the `saved_model_half_plus_two_cpu` model to input shared folder:

```
cp -r <directory where you have cloned Tensorflow  
Serving>/serving/tensorflow_serving/servables/tensorflow/testdata/saved_model  
_half_plus_two_cpu <directory where you have mounted /mnt/input share>
```

- d. Run the following command:

```
nctl predict launch --model-location \  
/mnt/input/saved_model_half_plus_two_cpu
```

Note: `--model-name` can be passed optionally to `nctl predict launch` command. If not provided, it assumes that model name is equal to the last directory in model location:

```
/mnt/input/home/trained_mnist_model -> trained_mnist_model
```

Using a Streaming Inference Instance

After running the `predict launch` command, `nctl` will create a streaming inference instance that can be used in multiple ways, as described below.

Streaming Inference with `nctl predict stream` Command

The `nctl predict stream` command allows performing inference on input data stored in JSON format. This method is convenient for manually testing a trained model and provides a simple way to get inference results. For `saved_model_half_plus_two_cpu`, write the following input data and save it in `inference-data.json` file:

```
{"instances": [1.0, 2.0, 5.0]}
```

The model `saved_model_half_plus_two_cpu` is a quite simple model: for given x input value it predicts result of $x/2 + 2$ operations. Having passed following inputs to the model: 1.0, 2.0, and 5.0, and so expected predictions results are 2.5, 3.0, and 4.5. In order to use that data for prediction, check the name of the running prediction instance with `saved_model_half_plus_two_cpu` model (the name will be displayed after `nctl predict launch` command executes; you can also use `nctl predict list` command for listing running prediction instances). Then run following command:

```
nctl predict stream --name <prediction instance name> --data inference-data.json
```

The following results will be produced:

```
{ "predictions": [2.5, 3.0, 4.5] }
```

TensorFlow Serving exposes three different method verbs for getting inference results. Selecting the proper method verb depends on the used model and the expected results. Refer to [RESTful API](#) for more detailed information. These method verbs are:

- `classify`
- `regress`
- `predict`

By default, `nctl predict stream stream` will use the `predict` method verb. You can change it by passing the `--method-verb` parameter to the `nctl predict stream` command, for example:

```
nctl predict stream --name <prediction instance name> --data inference-data.json --method-verb classify
```

Streaming Inference with TensorFlow Serving REST API

Another way to interact with a running prediction instance is to use TensorFlow Serving REST API. This approach could be useful for more sophisticated use cases, like integrating data-collecting scripts and applications with prediction instances.

The URL and authorization header for accessing TensorFlow Serving REST API will be shown after prediction instance is submitted, as in the example below.

```
Prediction instance URL (append method verb manually, e.g. :predict):  
https://192.168.0.1:8443/api/v1/namespaces/jdoe/services/saved-mode-621-18-11-07-15-00-34:rest-port/proxy/v1/models/saved_model_half_plus_two_cpu  
  
Authorize with following header:  
Authorization: Bearer  
1234567890abcdefghijklmnopqrstuvwxyz
```

Accessing the REST API with curl

Here is an example of accessing REST API Using curl, with the following command:

```
curl -k -X POST -d @inference-data.json -H 'Authorization: Bearer <authorization token data>' localhost:8501/v1/models/<model_name, e.g. saved_model_half_plus_two_cpu>:predict
```

Using Port Forwarding

Alternatively, the Kubernetes port forwarding mechanism may be used. You can create a port forwarding tunnel to the prediction instance with the following command:

```
kubect1 port-forward service/<prediction instance name> :8501
```

Or if you want to start a port forwarding tunnel in background:

```
kubect1 port-forward service/<prediction instance name> <some local port number>:8501 &
```

Note: Local port number of the tunnel you entered above will be produced by `kubect1 port-forward` if you *do not* explicitly specify it.

Now you can access REST API on the following URL:

```
localhost:<local tunnel port number>/v1/models/<model_name, e.g. saved_model_half_plus_two_cpu>:<method verb>
```

Example of Accessing REST API Using curl

To access REST API using curl, execute the following command:

```
curl -X POST -d @inference-data.json localhost:8501/v1/models/<model_name, e.g. saved_model_half_plus_two_cpu>:predict
```

Streaming Inference with TensorFlow Serving gRPC API

Another way to interact with running prediction instance is to use TensorFlow Serving gRPC. This approach could be useful for more sophisticated use cases, like integrating data collecting *scripts/applications* with prediction instances. It should provide better performance than REST API.

To access TensorFlow Serving gRPC API of running prediction instance, the Kubernetes port forwarding mechanism must be used. Create a port forwarding tunnel to a prediction instance with following command:

```
kubectl port-forward service/<prediction instance name> :8500
```

Or if you want to start port forwarding tunnel in background:

```
kubectl port-forward service/<prediction instance name> <some local port number>:8500 &
```

Note: The local port number of the tunnel you entered above; it will be produced by `kubectl port-forward` if you do not explicitly specify it.

You can access the gRPC API by using a dedicated client gRPC client (such as the following GitHub Python script: [mnist_client.py](#)). Alternatively, use gRPC CLI client of your choice (such as: [Polyglot](#) and/or [gRPC](#)) and connect to: `localhost:<local tunnel port number>`.

Useful External References

- [Serving a TensorFlow Model](#)
- [TensorFlow Serving with Docker](#)
- [RESTful API](#)

Managing Users and Resources

This section discusses the following topics:

- [Creating a User Account](#), page 65
- [Deleting a User Account](#), page 66
- [Viewing All User Activity](#), page 67

Creating a User Account

The user is the data scientist who wants to perform deep learning experiments to train models that will, after training and testing, be deployed in the field. Creating a new user account creates a user account configuration file compliant in format with `kubectl` configuration files.

Experiments and User Access

The user has full control (`list/read/create/terminate`) over their own experiments and has read access (`list/read`) of experiments belonging to other users on this cluster.

Note: Only an Administrator *can create* a user account

User Name Limitations

Users with the same name *cannot* be created directly after being removed. This is due to a user's related Kubernetes objects that are deleted asynchronously by Kubernetes and this can take some time. Consider waiting 10 minutes before creating a user with the same name; the same name as the user just deleted.

In addition, user names are limited to a 32-character maximum and there are no special characters except for hyphens. However, all names *must start* with a letter, *not a number*. You can use a hyphen to join user names, for example: john-doe.

Create the User

To create a user, execute the following steps:

1. The `nctl user create <username>` command sets up a namespace and associated roles for the named user on the cluster. It sets up home directories, named after the username, on the `input` and `output` network shares with file-system level access privileges. Create the user:

```
nctl user create <username>
```

2. The above command also creates a configuration file named `<username>.config` that the Admin provides to the user. The user then copies that file into a local folder.
3. Use the `export` command to set this variable for the user:

```
export KUBECONFIG=~/<local_user_folder>/<username>.config
```

4. Verify that the new user has been created with the following command:

```
nctl user list
```

5. The above command lists all users, including the new user just added. A partial example is shown below.

Name	Creation date	Date of last submitted job
user1	2019-03-12 08:30:45 PM	2019-02-27 07:55:13 PM
user2	2019-03-12 09:50:50 PM	
user3	2019-03-12 09:51:31 PM	

Deleting a User Account

Only an Administrator can delete user accounts. Deleting a user removes that user's account from the Nauta software and removes log in access to the system. The command halts and removes all experiments and pods; however, all artifacts related to that user's account, such as, the user's input and output folders and all data related to past experiments will remain.

Removing a User

To remove a user, execute the following command:

```
nctl user delete <username>
```

This command asks for confirmation.

```
Do you want to continue? [y/N]: Press y to confirm deletion.
```

Limitations

The command may take up to 30 seconds to delete the user and you may receive the message: `User is still being deleted`. Check the status of the user in a while. Recheck, as desired.

Using the purge Command

To permanently remove (purge) all artifacts associated with the user, including all data related to past experiments submitted by that user (but excluding the contents of the user's input and output folders):

Purging Process

To purge a user, execute the following command:

```
nctl user delete <username> --purge
```

This command asks for confirmation.

```
Do you want to continue? [y/N]: Press y to confirm deletion.
```

Limitations

The Nauta `user delete` command may take up to 30 seconds to delete the user. A new user with the same username *cannot* be created until after the delete command confirms that the first user with the same name has been deleted.

Viewing All User Activity

The command `nctl user list` displays all current users and all of their experiments (with status). Furthermore, the command shows the following information:

- **Name:** user name
- **Creation date:** the date this user account was created
- **Date of last submitted job:** experiment
- **Number of running jobs:** experiments
- **Number of queued jobs:** experiments submitted but not yet running

Administrators are not listed. Previously deleted users are not shown. To *create* a user account, refer to [Creating a User Account, page 65](#) and to *delete* a user account, refer to [Deleting a User Account, page 66](#).

Enter the following command.

```
nctl user list
```

A partial example of the results is shown below.

Name	Creation date	Date of last submitted job
user1	2019-03-12 08:30:45 PM	2019-03-02 05:25:14 PM
user2	2019-03-12 09:50:50 PM	
user3	2019-03-12 09:51:31 PM	

Accessing the Kubernetes Resource Dashboard

Kubernetes provides a way to manage containerized workloads and services, to manage resources given to a particular experiment and monitor workload statuses and resource consumption. Refer to [Kubernetes Web UI \(Dashboard\)](#) for detailed Kubernetes information.

Access Kubernetes Resource Dashboard

To access Kubernetes:


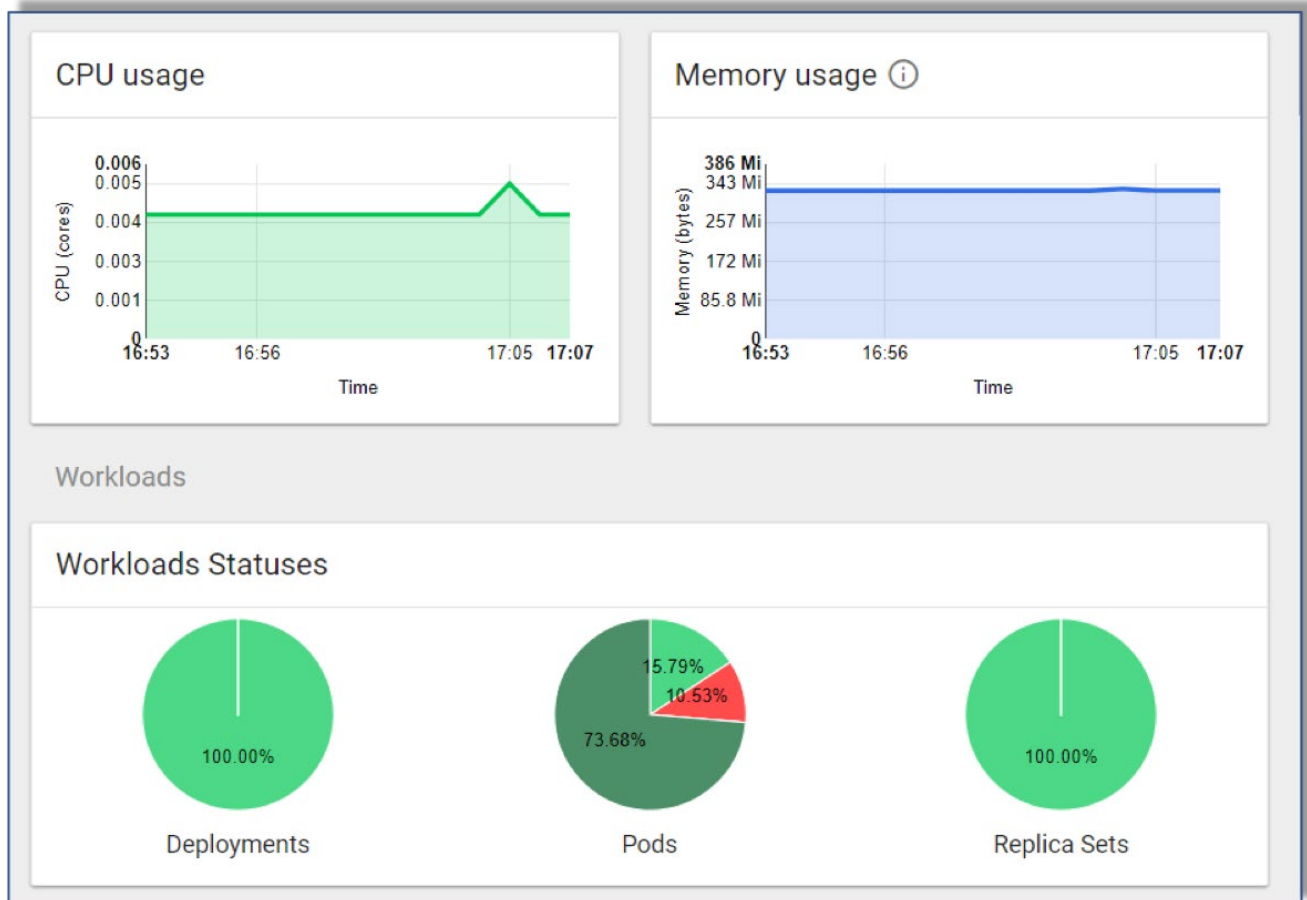
1. Click the **Hamburger Menu**  at the far left of the UI to open a left frame.
2. Click **Resources Dashboard** to open the Kubernetes resources dashboard in a new browser window/tab.

Figure 12 shows an example Kubernetes Dashboard.

Figure 12: Kubernetes Dashboard—Example Only



CLI Commands

The `--help` command provides man-page style help for each `nctl` command. You can view help for any command and subcommand, and all related parameters.

This section discusses the following main topics:

- [Viewing CLI Command Help at the Command Line, page 70](#)
- [experiment Command, page 71](#)
 - [submit Subcommand, page 72](#)
 - [list Subcommand, page 75](#)
 - [cancel Subcommand, page 77](#)
 - [view Subcommand, page 78](#)
 - [logs Subcommand, page 79](#)
 - [interact Subcommand, page 81](#)
 - [template_list Subcommand, page 83](#)
- [launch Command, page 84](#)
 - [webui Subcommand, page 84](#)
 - [tensorboard Subcommand, page 86](#)
- [mount Command, page 88](#)
 - [list Subcommand, page 89](#)
- [predict Command, page 90](#)
 - [batch Subcommand, page 90](#)
 - [cancel Subcommand, page 91](#)
 - [launch Subcommand, page 92](#)
 - [list Subcommand, page 94](#)
 - [stream Subcommand, page 95](#)
- [user Command, page 96](#)
 - [create Subcommand, page 96](#)
 - [delete Subcommand, page 98](#)
 - [list Subcommand, page 99](#)
- [verify Command, page 100](#)
- [version Command, page 101](#)
- [config Command, page 102](#)

Viewing CLI Command Help at the Command Line

The `--help` option provides man-page style help for each `nctl` command. You can view help for any command and subcommand, and all related parameters.

Entering `nctl --help` provides a listing of all `nctl` commands (without subcommands), as shown next.

```
Usage: nctl COMMAND [options] [args]...

Nauta Client

Displays additional help information when the -h or --help COMMAND is used.

Options:
  -h, --help  Displays help messaging information.

Commands:
  config, cfg      Set limits and requested resources in templates.
  experiment, exp  Start, stop, or manage training jobs.
  launch, l        Launch the web user-interface or TensorBoard. Runs as a process
                  in the system console until the user stops the process.
                  To run in the background, add '&' at the end of the line.
  mount, m         Displays a command to mount folders on a local machine.
  predict, p       Start, stop, and manage prediction jobs and instances.
  user, u          Create, delete, or list users of the platform.
                  Can only be run by a platform administrator.
  verify, ver      Verifies whether all required external components contain
                  the proper versions installed.
  version, v       Displays the version of the installed nctl application.
  workflow, wf     Start, stop, and manage workflows.
```

You can view help for any command and available subcommand(s). The following example shows generic syntax; brackets are optional parameters, but `[subcommand]` requires `[command]`.

```
nctl [command_name] [subcommand] --help
```

experiment Command

The `experiment` command's overall purpose of this command and subcommands is to submit and manage experiments. The following are the subcommands for the `nctl` `experiment` command.

- [submit Subcommand, page 72](#)
- [list Subcommand, page 75](#)
- [cancel Subcommand, page 77](#)
- [view Subcommand, page 78](#)
- [logs Subcommand, page 79](#)
- [interact Subcommand, page 81](#)
- [template_list Subcommand, page 83](#)

submit Subcommand

Synopsis

The `submit` subcommand submits training jobs. Use this command to submit single and multi-node training jobs (by passing `-t` parameter with a name of a multi-node pack), and many jobs at once (by passing `-pr/-ps` parameters).

Syntax

```
nctl experiment submit [options] SCRIPT-LOCATION [-- script-parameters]
```

Arguments

Name	Required	Description
SCRIPT-LOCATION	Yes	Location and name of a Python script with a description of training.
script-parameters	No	String with a list of parameters that are passed to a training script. All such parameters should be added at the end of command after "--" string.

Options

Name	Required	Description
<code>-sfl, --script_folder_location <folder_name> PATH</code>	No	Location and name of a folder with additional files used by a script, for example: other .py files, data, and so on. If not given, then its content <i>will not</i> be copied into the Docker image created by the <code>nctl submit</code> command. <code>nctl</code> copies all content, preserving its structure, including subfolder(s).
<code>-t, --template <template_name> TEXT</code>	No	Name of a template that will be used by <code>nctl</code> to create a description of a job to be submitted. If not given, a default template for single node TensorFlow training is used (<code>tf-training</code>). List of available templates can be obtained by issuing <code>nctl experiment template_list</code> command.
<code>-n, --name TEXT</code>	No	Name assigned to an experiment.
<code>-p, --pack-param <TEXT TEXT>...</code>	No	Additional pack param in format: 'key value' or 'key.subkey.subkey2 value'. For lists use: 'key '['val1', 'val2']"' For maps use: 'key '{"a': 'b'}"'

Name	Required	Description
<code>-pr, --parameter-range</code> <code>TEXT...</code> <code>[definition]</code> <code><TEXT TEXT>...</code>	No	<p>If the parameter is given, <code>nctl</code> starts as many experiments as there is a combination of parameters passed in <code>-pr</code> options. Optional <code>[param-name]</code> is a name of a parameter that is passed to a training script. <code>[definition]</code></p> <p>Contains values of this parameter that are passed to different instance of experiments. <code>[definition]</code> can have two forms:</p> <ul style="list-style-type: none"> range: <code>{x...y:step}</code> This form says that <code>nctl</code> will launch a number of experiments equal to a number of values between <code>x</code> and <code>y</code> (including both values) with <code>step</code>. set of values: <code>{x, y, z}</code> This form says that <code>nctl</code> will launch number of experiments equal to a number of values given in this definition.
<code>-ps, --parameter-set</code> <code>[definition]</code> <code>TEXT</code>	No	<p>When used, <code>nctl</code> launches an experiment with a set of parameters defined in <code>[definition]</code> argument. Optional. Format of the <code>[definition]</code> argument is as follows:</p> <pre>{[param1_name]: [parameter1_value], [parameter2_name]: [parameter2_value], ..., [paramn_name]: [paramn_value]}.</pre> <p>All parameters given in the <code>[definition]</code> argument will be passed to a training script under their names stated in this argument. If the <code>-ps</code> parameter is given more than once, then <code>nctl</code> will start as many experiments as there is occurrences of this parameter in a call.</p>
<code>-e, --env TEXT</code>	No	<p>A set of values of one or several parameters; the environment variables passed to training.</p> <p>You can pass as many environmental variables, as desired. Each variable should be passed as a separate <code>-e</code> parameter.</p>
<code>-r, --requirements</code> <code>PATH</code>	No	<p>Path to file with experiment's pip requirements. Dependencies listed in this file will be automatically installed using pip.</p>
<code>-v, --verbose</code>	No	<p>Set verbosity level:</p> <ul style="list-style-type: none"> <code>-v</code> for INFO <code>-vv</code> for DEBUG
<code>-h, --help</code>	No	<p>Displays help messaging information.</p>

Additional Remarks

For both types of parameters: `-ps` and `-pr`; if, the parameter stated in their definitions is also given in a `[script_parameters]` argument of the `nctl` command, then values taken from `-ps` and `-pr` are passed to a script.

If a combination of both parameters is given, then `nctl` launches a number of experiments equal to combination of values passed in those parameters. For example, if the following combination of parameters is passed to `nctl` command:

```
-pr param1 "{0.1, 0.2, 0.3}" -ps "{param2: 3, param4: 5}" -ps "{param6: 7}"
```

Then the following experiments will be launched:

```
param1 = 0.1, param2 = 3, param4 = 5, param6 - not set  
param1 = 0.2, param2 = 3, param4 = 5, param6 - not set  
param1 = 0.3, param2 = 3, param4 = 5, param6 - not set  
param1 = 0.1, param2 = not set, param4 = not set, param6 - 7  
param1 = 0.2, param2 = not set, param4 = not set, param6 - 7  
param1 = 0.3, param2 = not set, param4 = not set, param6 - 7
```

Returns

This command returns a list of submitted experiments with their names and statuses. In case of problems during submission, the command displays message/messages describing the causes. Errors may cause some experiments *to not be* created and will be empty. If any error appears, then messages describing it are displayed with experiment's names/statuses.

If one or more of experiment has not been submitted successfully, then the command returns an exit code: `> 0`. The exact value of the code depends on the cause of error(s) that prevented submitting the experiment(s).

Example

The following arguments: `-data_dir` and `--num_gpus` are passed to a script.

```
nctl experiment submit mnist_single_node.py -sfl /data -- --data_dir=/app/data --  
num_gpus=0
```

Starts a single node training job using `mnist_single_node.py` script located in a folder from which `nctl` command was issued. The content of the `/data` folder is copied into the Docker image (into `/app` folder, which is a work directory of Docker images created using `tf-training pack`).

list Subcommand

Synopsis

The `list` subcommand Displays a list of all experiments with some basic information for each, regardless of the owner. Results are sorted using the *date-of-creation* of the experiment, starting with the most recent experiment.

Syntax

```
nctl experiment list [options]
```

Options

Name	Required	Description
-a, --all_users	No	List contains experiments submitted by of all users.
-n, --name TEXT	No	A regular expression to filter list to experiments that match this expression.
-s, --status	No	QUEUED, RUNNING, COMPLETE, CANCELLED, FAILED, CREATING - Lists experiments based on indicated status.
-u, --uninitialized	No	List uninitialized experiments, that is, experiments without resources submitted for creation.
-c, --count INTEGER RANGE	No	An integer, command displays c last rows.
-b, --brief	No	Print short version of the result table. Only 'name', 'submission date', 'owner' and 'state' columns will be printed.
-v, --verbose	No	Set verbosity level: -v for INFO -vv for DEBUG
-h, --help	No	Displays help messaging information.

Returns

List of experiments matching criteria given in command's options. Each row contains the experiment name and additional data of each experiment, such parameters used for this certain training, time and date when it was submitted, name of a user which submitted this training and current status of an experiment. Below is an example returned by this command (the brief option is shown).

Experiment	Submission date	Owner	State
mnist-single-node-tb	2019-03-13 04:57:58 PM	user1	QUEUED
mnist-tb	2019-03-13 05:00:39 PM	user1	COMPLETE
mnist-tb 2-1	2019-03-13 05:49:59 PM	user1	COMPLETE
test-experiment	2019-03-13 06:00:39 PM	user1	QUEUED
single-experiment	2019-03-13 01:49:59 PM	user1	QUEUED

Examples

The following command displays all experiments submitted by a current user.

```
nctl experiment list
```

The following command displays all experiments submitted by a current user and with name starting with train word.

```
nctl experiment list -n train
```

cancel Subcommand

Synopsis

The `cancel` subcommand cancels training chosen based on provided parameters.

Syntax

```
nctl experiment cancel [options] NAME
```

Arguments

Name	Required	Description
NAME	Yes	Name of an experiment/pod/status of a pod to be cancelled. If any such an object is found, the command displays question whether this object should be cancelled.

options

Name	Required	Description
-m, --match TEXT	No	If given, the command searches for experiments matching the value of this option. This option cannot be used along with the NAME argument.
-p, --purge	No	When used, all information concerning experiments is removed from the system.
-i, --pod-ids TEXT	No	Comma-separated pods IDs. When used, the command matches pods by their IDs and deletes them.
-s, --pod-status TEXT	No	One of: PENDING, RUNNING, SUCCEEDED, FAILED, or UNKNOWN. If given, the command searches pods by their status and deletes them.
-v, --verbose	No	Set verbosity level: -v for INFO -vv for DEBUG
-h, --help	No	Displays help messaging information.

Returns

The description of a problem; if, any problem occurs. Otherwise, the displays information that training job/jobs was/were cancelled successfully.

Example

```
nctl experiment cancel t20180423121021851
```

Outcome

This cancels the experiment with `t20180423121021851` name, as shown in the example.

view Subcommand

Synopsis

The `view` subcommand displays basic details of an experiment, such as the name of an experiment, parameters, submission date, and so on.

Syntax

```
nctl experiment view [options] EXPERIMENT-NAME
```

Arguments

Name	Required	Description
EXPERIMENT-NAME	Yes	Name of an experiment to be displayed.

Options

Name	Required	Description
-tb, --tensorboard	No	If given, the command exposes a TensorBoard instance with an experiment's data.
-u, --username TEXT	No	Name of the user who submitted this experiment. If not given, then only experiments of a current user are shown.
-v, --verbose	No	Set verbosity level: -v for INFO, -vv for DEBUG
-h, --help	No	Displays help messaging information.

Returns

Displays details of an experiment. If `-tb, --tensorboard` option is given, then the command also returns a link to TensorBoard's instance with data from the experiment or tries to open TensorBoard in a browser.

Example

```
nctl experiment view experiment_name_2 -tb
```

Displays details of an `experiment_name_2` experiment and exposes TensorBoard instance with experiment's data to a user.

logs Subcommand

Synopsis

The `logs` subcommand displays logs from experiments. Logs to be displayed are chosen based on parameters given in command's call.

Syntax

```
nctl experiment logs [options] EXPERIMENT-NAME
```

Arguments

Name	Required	Description
EXPERIMENT-NAME	Yes	Displays the name of experiment logs.

Options

Name	Required	Description
<code>-s, --min-severity</code>	No	Minimal severity of logs. Available choices are: <ul style="list-style-type: none"> <code>CRITICAL</code> - Displays only CRITICAL logs <code>ERROR</code> - Displays ERROR and CRITICAL logs <code>WARNING</code> - Displays ERROR, CRITICAL and WARNING logs <code>INFO</code> - Displays ERROR, CRITICAL, WARNING and INFO <code>DEBUG</code> - Displays ERROR, CRITICAL, WARNING, INFO and DEBUG
<code>-sd, --start-date</code>	No	Retrieve logs produced from this date (format ISO-8061 - yyyy-mm-ddThh:mm:ss)
<code>-ed, --end-date</code>	No	Retrieve logs produced until this date (format ISO-8061 - yyyy-mm-ddThh:mm:ss)
<code>-i, --pod-ids TEXT</code>	No	Comma-separated pods IDs. If given, then matches pods by their IDs and only logs from these pods from an experiment with <code>EXPERIMENT_NAME</code> name will be returned.
<code>-p, --pod-status TEXT</code>	No	One of: 'PENDING', 'RUNNING', 'SUCCEEDED', 'FAILED', or 'UNKNOWN' commands returns logs with matching status from an experiment and matching <code>EXPERIMENT-NAME</code> .
<code>-m, --match TEXT</code>	No	If given, this command searches for logs from experiments matching the value of this option. This option <i>cannot</i> be used along with the <code>NAME</code> argument.
<code>-o, --output</code>	No	If given, the logs are stored in a file with a name derived from a name of an experiment.
<code>-pa, --pager</code>	No	Display logs in interactive pager. Press q to exit the pager.
<code>-f, --follow</code>	No	Specify if logs should be streamed. Only logs from a single experiment can be streamed.

Name	Required	Description
-v, --verbose	No	Set verbosity level: -v for INFO -vv for DEBUG
-h, --help	No	Displays help messaging information.

Returns

Should issues arise, a message (or messages) with a description of their cause (or causes) displays. Otherwise, this command returns experiment logs that can be filtered using command options.

Example

```
nctl experiment logs experiment_name_2 --min-severity DEBUG
```

Displays logs from `experiment_name_2` experiment with severity `DEBUG` and higher (`INFO`, `WARNING`, and so on).

interact Subcommand

Synopsis

The `interact` subcommand launches a local browser with Jupyter notebook. If a script's name is given as a parameter of a command, then this script is displayed in a notebook.

Syntax

```
nctl experiment interact [options]
```

Options

Name	Required	Description
<code>-n, --name TEXT</code>	No	Name of a Jupyter notebook session. If session with a given name already exists, then a user is connected to this session.
<code>-f, --filename TEXT</code>	No	Name of a script used in the Jupyter notebook's session.
<code>-p, --pack-param <TEXT TEXT>...</code>	No	Additional pack parameter in format: 'key value' or 'key.subkey.subkey2 value'. <ul style="list-style-type: none"> For lists use: 'key '['val1', 'val2']' For maps use: 'key '{"a": "b"}'
<code>--no-launch</code>	No	Run command without a web browser starting, only proxy tunnel is created.
<code>-pn, --port-number INTEGER RANGE</code>	No	Port on which service will be exposed locally.
<code>-e, --env TEXT</code>	No	Environment variables passed to Jupyter instance and you can pass as many environmental variables as is desired. Each variable should be passed as a separate <code>-e</code> parameter.
<code>t, --template [jupyter,jupyter-py2]</code>	No	Name of a Jupyter notebook template used to create a deployment. Supported templates for interact command are: jupyter (python3) and jupyter-py2 (python2).
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

Should issues arise, a message provides a description of possible causes. Otherwise, it launches a default web browser with Jupyter notebook and displays the address under which this session is provided.

Example

Launches in a default browser a Jupyter notebook with `training_script.py` script.

```
nctl experiment interact --filename training_script.py
```

template_list Subcommand

Synopsis

The `template_list` subcommand returns a list of templates installed on a client machine. Template contains all details needed to properly deploy training job on a cluster.

Syntax

```
nctl experiment template_list [options]
```

Options

Name	Required	Description
-v, --verbose	No	Set verbosity level: -v for INFO -vv for DEBUG
-h, --help	No	Displays help messaging information.

Returns

The command returns a list of existing templates, or a *Lack of installed packs* message, if there are no templates installed.

Example

```
nctl experiment template_list
```

launch Command

Synopsis

The `launch` command launches a browser for Web UI or TensorBoard. The following main topics are discussed in this section:

webui Subcommand

Synopsis

The `webui` subcommand launches the Nauta web user interface with credentials.

Note: If you are using CLI through remote access, you will need to setup an X server for tunneling over SSH with port forwarding or use SSH Proxy command tunneling. After establishing a tunnel from the gateway to your local machine, you can use the URL provided by this command.

Syntax

```
nctl launch webui [options]
```

Arguments

None.

Options

Name	Required	Description
<code>--no-launch</code>	No	Run this command without a web browser starting; only proxy tunnel is created.
<code>-p, --port <port></code> INTEGER RANGE	No	If given, the application will be exposed on a local machine under [port] port.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

Link to an exposed application.

Example

```
nctl launch webui
```

This command returns a Go to URL. The following is an example only:

```
Go to
http://localhost:32950?token=eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3N1cnZpY2VhY2NvdW50Iiia3ViZXJ1cy5pb3VudC9uYWw5L3BhY2UiOiJrdWJlLXN5c3RlbnSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWZjY291bnQvc2VjcmV0Lm5hbWUiOiJuYXV0YS1rOHMtcGxhdGZvc0tYWRTaW4tdG9rZW4taH3dG0iLCJrdWJlcm5ldGVzLmlvL3N1cnZpY2VhY2NvdW50L3N1cnZpY2U0YWNjb3VudC5uYWw1IjoibmFldGEtazhzLXBsYXRmb3JtLWFKbWluIiwia3ViZXJ1cy5pb3VudC9uYWw5L3BhY2UiOiJrdWJlLWZjY291bnQudWlkIjoizjQ4OWFiYTETNTVkyY0xMWU5LWI4OWUtNTI1ODE2MDcwNDAwIiwic3ViIjoic3lzdGVtOnN1cnZpY2VhY2NvdW50Omt1YmUtc3lzdGVtOm5hdXRhLW54cy1wbGF0Zm9ybS1hZG1pbiJ9.LduRAyaM1cYUUTXnudhtwYds
wLycBmwdc8rb5k0BWQ71eSm5L0WEmnhn1TznC7D4MnONoF9OxnbgrnduTfQWhf2BQObeMfO5STCHXaF-
iTEM2Lrscafg1kpiE1Optyb0KLnuSxSDAw5FLfWRAMcWHrxh7NrbO6asC6d-0N9x4dQuecaVTok9aLS402KuV_NXN-
PFaD7n3cbhl0IwDp9aqHqjAW93_DKcK60yTry7YNRwhldYK_7gqKySRZaEypFleWFWLrnnDc8g7mLRn0omEFi2GFisxD
14-JNj_lnrkn-KICRexiI_rA_mjp2k8Dka64ecKgI4uBl-s1lkgy8ewi_G2SwU-
E0TpTG_alyfzat2fo1DDcQQltkFYpgCJH5mvIiSXXTO9oVHuw3hwg-CrFAtJu4Eglhb30bDSJGTP4QA60XjA7urv5-
tP3RzIH0DqeL2dfgTb9OwAuZQhLMzbBiwiG-OHCSPUMSqzZPA9D1-
appTCfapnUAx62pq9nZ3Xcxka4glWA4aWw6OijCavDKIj0Lnd5jI-
W3UjWm05o1idUoI9TwYohZTEottKXLZFHIVi9dscEwGxXpXqMhLB7is-x8Lwu5j0EKOgJonuckuTCqu-k-K--
psZBjMduojAdNMST-Rl9dW60gH6-oF3Uc5Qslm0J-DPAfIA6dY
Proxy connection created.
Press Ctrl-C key to close a port forwarding process...
```

tensorboard Subcommand

Synopsis

The `tensorboard` subcommand launches the TensorBoard web user interface front-end with credentials, with the indicated experiment loaded.

Note: If you are using CLI through remote access, you will need to setup an X server for tunneling over SSH with port forwarding or use SSH Proxy command tunneling. After establishing a tunnel from the gateway to your local machine, you can use the URL provided by this command.

Syntax

```
nctl launch tb [options] EXPERIMENT-NAME
```

Arguments

Name	Required	Description
EXPERIMENT-NAME	Yes	Experiment name

A user can pass one or more names of experiments separated with spaces. If experiment that should be displayed in TensorBoard belongs to a current user; the user has to give only the name. If this experiment is owned by another user; then, the name of an experiment should be preceded with a name of this second user in the following format: `username/experiment-name`.

Options

Name	Required	Description
<code>--no-launch</code>	No	To create tunnel without launching web browser.
<code>-tscp,</code> <code>--tensorboard-</code> <code>service-client-</code> <code>port</code>	No	Local port on which TensorBoard service client will be started.
<code>--p, --port</code> <code>INTEGER RANGE</code>	No	Port on which service will be exposed locally.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

Link to an exposed application.

Examples

```
nctl launch tensorboard experiment75
```

An example might appear as:

```
http://127.0.0.1/tensorboard/token=AB123CA27F
```

mount Command

Synopsis

The `mount` command displays another command that can be used to mount/unmount a client's folders on/from a user's local machine. Also see, [list Subcommand](#), [page 89](#).

Syntax

```
nctl mount [options]
```

Options

Name	Required	Description
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

This command returns another command that can be used to mount a client's folders on a user's local machine. It also shows what command should be used to unmount client's folder after it is no longer needed.

list Subcommand

Synopsis

The `list` Displays a list of Nauta related folders mounted on a user's machine. If run using admin credentials, displays mounts of all users.

Syntax

```
nctl mount list
```

Returns

List of mounted folders. Each row contains additional information (for example: remote and local location) concerning those mounts. Set of data displayed by this command depends on operating system.

Additional Remarks

This command displays only those mounts that exposing Nauta shares. Other mounted folders *are not* taken into account.

predict Command

Synopsis

Use this command to start, stop, and manage prediction jobs.

batch Subcommand

Synopsis

The `batch` command starts a new batch instance that performs prediction on provided data and uses a specified dataset to perform inference. The results are stored in an output file.

Syntax

```
nctl predict batch [options]
```

Options

Name	Required	Description
<code>-n, --name</code>	No	Name of predict session.
<code>-m, --model-location TEXT</code>	Yes	Path to saved model that will be used for inference. Model must be located on one of the input or output system shares (e.g. <code>/mnt/input/saved_model</code>). Model content will be copied into an image.
<code>-l, --local_model_location PATH</code>	Yes	Local path to saved model that will be used for inference. Model content will be copied into an image.
<code>-d, --data TEXT</code>	Yes	Location of a folder with data that will be used to perform the batch inference. Value should point out the location from one of the system's shared folder.
<code>-o, --output TEXT</code>	No	Location of a folder where outputs from inferences will be stored. Value should point out the location from one of the system's shared folder.
<code>-mn, --model-name</code>	No	Name of a model passed as a servable name. By default, it is the name of directory in model's location.
<code>-tr, --tf-record</code>	No	If given, the batch prediction accepts files in <code>TFRecord</code> formats. Otherwise, files should be delivered in <code>protobuf</code> format.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

The description of a problem; if, any problem occurs. Otherwise information that training job/jobs was/were cancelled successfully.

cancel Subcommand

Synopsis

The `cancel` subcommand cancels prediction instance(s) chosen based on criteria given as a parameter.

Syntax

```
nctl predict cancel [options] [name]
```

Arguments

Name	Required	Description
NAME	No	Name of predict instance to be cancelled. [name] argument value can be empty when <code>match</code> option is used.

Options

Name	Required	Description
<code>-m, --match</code>	No	If given, the command searches for prediction instances matching the value of this option.
<code>-p, --purge</code>	No	If given, , then all information concerning all prediction instances, completed and currently running, is removed from the system.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

The description of a problem; if, any problem occurs. Otherwise information that training job/jobs was/were cancelled successfully.

launch Subcommand

Synopsis

The `launch` subcommand starts a new prediction instance that can be used for performing prediction, classification and regression tasks on trained model. The created prediction instance is for streaming prediction only.

Syntax

```
nctl predict launch [options]
```

Options

Name	Required	Description
<code>-n, --name TEXT</code>	No	The name of this prediction instance.
<code>-m, --model-location TEXT</code>	Yes	Path to saved model that will be used for inference. Model must be located on one of the input or output system shared folder (e.g. <code>/mnt/input/home/saved_model</code>).
<code>-l, --local_model_location PATH</code>	No	Local path to saved model that will be used for inference. Model content will be copied into an image.
<code>-mn, --model-name TEXT</code>	No	Name of a model passed as a servable name. By default, it is the name of directory in model's location.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

Prediction instance URL and authorization token, as well as information about the experiment (name, model location, state).

```
nctl predict launch -n stream-inference --model-location /mnt/input/home/stream-inference
```

Submitting prediction instance.

Prediction instance	Model Location	Status
stream-inference	/mnt/input/home/stream-inference	QUEUED

Prediction instance URL (append method verb manually, e.g. :predict):

<https://192.168.0.1:8443/api/v1/namespaces/User1/services/stream-inference:rest-port/proxy/v1/models/stream-inference>

Authorize with following header:

Authorization: Bearer

abHLdALDSALDVsfkI4AdfklaADvsd9AZTJJyF8xr3tAWBhUQdoIVahPcms1muBbXCvwRaPzzHHO6IaIsUwj
TXEGtSmW68Koyx0dB26csZRQgxF-BxwwLtrXn_UWLLet9YGzuovjI9K0LPP6kKavJqn5HtLGzEKdG3F6e6L9Qk-
3XzGI9t36uzfsv86SPODS5TaOKU2 jJ-
X9WUmMXgEkT2BApU90JmJrh2jC7ZRH3xaUwfWu00g4IoUkE83bKxpRvvQYA10aCX_J4BCsVQRp-
bUdacq7UVahXI9NkbGao83XCqQuLNgV0hAxD6VC..

list Subcommand

Synopsis

The `list` subcommand displays a list of inference instances with some basic information regarding each of them. Results are sorted using a date of creation starting with the most recent and filtered by optional criteria.

Syntax

```
nctl predict list [options]
```

Options

Name	Required	Description
<code>-a, --all_users</code>	No	Show all prediction instances, regardless of the owner.
<code>-n, --name TEXT</code>	No	A regular expression to narrow down list to prediction instances that match this expression.
<code>-s, --status [QUEUED, RUNNING, COMPLETE, CANCELLED, FAILED, CREATING]</code>	No	A regular expression to filter list to prediction instances with matching status.
<code>-u, --uninitialized</code>	No	List uninitialized prediction instances: for example, prediction instances without resources submitted for creation.
<code>-c/--count INTEGER RANGE</code>	No	If given, command displays c most-recent rows.
<code>-b, --brief</code>	No	Print short version of the result table. Only 'name', 'submission date', 'owner' and 'state' columns will be printed.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

List of inference instances.

stream Subcommand

Synopsis

The `stream` subcommand performs stream inference task on launched prediction instance.

Syntax

```
nctl predict stream [options]
```

Options

Name	Required	Description
<code>-n, --name TEXT</code>	Yes	Name of prediction session.
<code>-d, --data PATH</code>	Yes	Path to JSON data file that will be streamed to prediction instance. Data must be formatted such that it is compatible with the <code>SignatureDef</code> specified within the model deployed in selected prediction instance.
<code>-m, --method-verb [classify, regress, predict]</code>	No	Method verb that will be used when performing inference. Predict verb is used by default.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

user Command

Use this command to create, delete, and manage users.

- [create Subcommand](#), page 96
- [delete Subcommand](#), page 98
- [list Subcommand](#), page 99

create Subcommand

Synopsis

The `create` subcommand creates and initializes a new Nauta user. This command *must be* executed when `kubect1` is used by a `nctl` command entered by a k8s administrator. If this command is executed by someone other than a k8s administrator, it fails. By default, this command saves a configuration of a newly created user to a file. The format of this file is compliant with a format of `kubect1` configuration files.

Syntax

```
nctl user create [options] USERNAME
```

Arguments

Name	Required	Description
USERNAME	Yes	Name of a user that will be created. This value must a valid OS-level user.

Options

Name	Required	Description
<code>-l, --list-only</code>	No	If given, the content of the generated user's config file is displayed on the screen only. If not given, the file with configuration is saved on disk.
<code>-f, --filename TEXT</code>	No	The name of file where user's configuration will be stored. If not given, the configuration is stored in the <code>config.<username></code> file.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Additional Remarks

In case of any errors during saving of a file with a configuration, the command displays a content of the configuration file on the screen, even if `-l` option was *not* used.

If an administrator tries to create a user with a name that was used previously by a deleted user, it may happen that the `create` command displays information that the previous user is still being deleted, even if the previous user *is not* listed on a list of existing users. In this case the operation of a creation of a new user should be postponed for 10 minutes, until all user's objects are removed.

Returns

If issues occur, a message describing their cause/causes displays. Otherwise, the message is returned indicating success. If the `list-only` option was given, the command also displays the content of a configuration file.

User Name Requirements

The *User Name* *must* meet the following requirements:

3. Cannot be longer than 32-characters.
4. Cannot be an empty string.
5. Must conform to Kubernetes naming convention: can contain only lower case alphanumeric characters and "-" and "."

Example

```
nctl user create jdoe
```

Outcome

This creates the user `jdoe`, as shown in the example.

delete Subcommand

Synopsis

The `delete` subcommand deletes a user with a given name. If option `-p`, `--purge` was added, it also removes all artifacts related to a removed user, such as: content of user's folders, experiment's data, and runs.

Syntax

```
nctl user delete USERNAME
```

Arguments

Name	Required	Description
USERNAME	Yes	The name of a user who should be removed from the system.

Additional Remarks

Before removing a user, the command asks for a final confirmation. If a you choose Yes, the chosen user is deleted.

Deletion of a user may take a while to be fully completed. Command waits for up to 30 seconds for a complete removal of user. If after this time user *has not* been deleted completely, the command displays information that a user is still being deleted. In this case the user *will not* be listed on a list of existing users, but there *is no possibility* to create a user with the same name until the command completes and the user is deleted.

Options

Name	Required	Description
<code>-p</code> , <code>--purge</code>	No	If set, the system also removes all logs generated by the user's experiments.
<code>-v</code> , <code>--verbose</code>	No	Set verbosity level: <code>-v</code> for INFO <code>-vv</code> for DEBUG
<code>-h</code> , <code>--help</code>	No	Displays help messaging information.

Returns

A message regarding the command's completion. If issues occur, a short description of their causes displays.

Example

```
nctl user delete jdoe -p
```

This removes the created `jdoe` user along with all their artifacts.

list Subcommand

Synopsis

The `list` subcommand lists all currently configured users.

Syntax

```
nctl user list [options]
```

Options

Name	Required	Description
<code>-c, --count INTEGER RANGE</code>	No	If given, the command displays c last rows.
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

verify Command

Synopsis

The `verify` command checks whether all prerequisites required by `nctl` are installed and have proper versions.

Syntax

```
nctl verify
```

Options

Name	Required	Description
<code>-v, --verbose</code>	No	Set verbosity level: -v for INFO, -vv for DEBUG
<code>-h, --help</code>	No	Displays help messaging information.

Returns

In the case of any installation issues, the command returns information about their cause (which application should be installed and in which version). If no issues are found, a message indicates checks were successful.

Example

```
This OS is supported.  
kubectl verified successfully.  
kubectl server verified successfully.  
helm client verified successfully.  
helm server verified successfully.
```

version Command

Synopsis

The `version` command returns the version of Nauta.

Syntax

```
nctl version
```

Options

Name	Required	Description
-v, --verbose	No	Set verbosity level: -v for INFO, -vv for DEBUG
-h, --help	No	Show help message and exit.

Returns

The version command returns the currently installed `nctl` application version of both client platform and server.

Example

```
| Component      | Version |
|-----+-----|
| nctl application | 1.0.0-ent-20190403020148 |
| nauta platform  | 1.0.0-ent-20190403020148 |
```

config Command

The `config` command is used to adjust a packs' settings to resources available on a cluster.

Synopsis

The `config` command allows a user to change the current system's settings concerning maximum and requested resources used by training jobs initiated by Nauta. The command takes CPU number and memory amount provided by a user and calculates new values preserving the same coefficient between available resources and resources defined in every template, as it was before execution of this command.

Syntax

```
nctl config [options]
```

Options

Name	Required	Description
-c, --cpu	Yes	This is the number of CPUs available on a cluster's node with the lowest number of CPU. Value should be given in format accepted by k8s. This can be a plain number or a number followed by 'm' suffix.
-m, --memory	Yes	This is the amount of a memory available on a cluster's node with the lowest amount of memory. Value should be given in format accepted by k8s. This can be a plain number or a number followed by a one of the following suffixes: E, P, T, G, M, K, Ei, Pi, Ti, Gi, Mi, Ki.
-v, --verbose	No	Set verbosity level: -v for INFO -vv for DEBUG
-h, --help	No	Displays help messaging information.

Note: Number of CPUs shown should be interpreted according to the following article: [Meaning of CPU](#) and the amount of memory given here should be interpreted according to the following article: [Meaning of Memory](#).

Returns

In case of any problems, a message describing the cause/causes of the issue displays. Otherwise message is returned indicating success.

Example

```
nctl config --cpu 10 --memory 1Gi
```

Outcome

This Calculates resources' settings for all packs installed together with `nctl` application. It assumes, that the maximal available number of CPU on a node is 10 and that this node provides 1Gb of RAM. Furthermore, limited and requested resources are calculated using those maximal values.