Data Science Stack

Canonical Group Ltd

Sep 10, 2024

CONTENTS

Data science stack (DSS) is a ready-to-run environment for machine learning and data science. It's built on open-source tooling (including MicroK8s, JupyterLab and MLflow) and usable on any Ubuntu/Snap-enabled workstation.

DSS provides a Command Line Interface (CLI) for managing containerised ML environments images such as PyTorch or TensorFlow, on top of MicroK8s.

Typically, creating ML environments on a workstation involves complex and hard-to-reverse configuration. DSS solves this problem by making accessible, production-ready, isolated and reproducible ML environments, that make full use of a workstation's GPUs.

Both ML beginners and engineers who need to build complex development and runtime environments will see set-up time reduced to a minimum, allowing them to get on with useful work within minutes.

CHAPTER

ONE

IN THIS DOCUMENTATION

Tutorial

Get started - a hands-on introduction to DSS for newcomers

How-to guides

Step-by-step guides covering key operations and common tasks with DSS

Explanation

Discussion and clarification of key topics

CHAPTER

TWO

PROJECT AND COMMUNITY

Data Science Stack is an open-source project that values its community. We warmly welcome contributions, suggestions, fixes, and constructive feedback from everyone.

- Read our Code of conduct.
- Contribute and report bugs.
- Join the Discourse forum.
- Talk to us on Matrix.

2.1 Tutorial

This section of our documentation contains a step-by-step tutorial to get you started with Data Science Stack (DSS). You will learn how it works, how to customise your setup and explore what you can do with it:

2.1.1 Get started with DSS

This guide describes how you can get started with Data Science Stack (DSS). From setting up MicroK8s in your host environment, all the way to running your first notebook.

Data Science Stack is a ready-made environment that makes it seamless to run GPU-enabled containerised Machine Learning (ML) environments. It provides easy access to a solution for developing and optimising ML models, utilising your machine's GPUs and allowing users to utilise different ML environment images based on their needs.

Prerequisites

- Ubuntu 22.04 LTS.
- Snap installed.
- 50GB of disk space is recommended. This includes the requirements for MicroK8s.

Setting up MicroK8s

DSS relies on a container orchestration system, capable of exposing the host GPUs to the workloads. MicroK8s is used as the orchestration system. All the workloads and state managed by DSS are running on top of MicroK8s.

You can install MicroK8s using `snap` as follows:

```
sudo snap install microk8s --channel 1.28/stable --classic
sudo microk8s enable hostpath-storage
sudo microk8s enable dns
sudo microk8s enable rbac
```

Install the DSS CLI

DSS is a Command Line Interface (CLI)-based environment. Now, install the DSS CLI using the following command:

```
sudo snap install data-science-stack --channel latest/stable
```

Initialise DSS

Next, you need to initialise DSS on top of MicroK8s and prepare MLflow:

```
dss initialize --kubeconfig="$(sudo microk8s config)"
```

> **Note**
>
> The initialisation might take a few minutes to complete. While the process is in progress, you will see the following message: `Waiting for deployment mlflow in namespace dss to be ready...`

Once initialised, you should see an output like this:

```
Deployment mlflow in namespace dss is ready
```

Launch a Notebook

At this point, DSS is set up and you are ready to start managing containerised notebook environments. You can use the DSS CLI to launch and access them using JupyterLab.

Start your first Jupyter Notebook with the following command:

```
dss create my-tensorflow-notebook --image=kubeflownotebookswg/jupyter-tensorflow-cuda:v1.
↪8.0
```

Once the command succeeds, it returns a URL that can be used to connect to the JupyterLab User Interface (UI) of that notebook. For example, you should expect an output like this:

```
Executing create command
Waiting for deployment my-tensorflow-notebook in namespace dss to be ready...
Deployment my-tensorflow-notebook in namespace dss is ready
Success: Notebook my-tensorflow-notebook created successfully.
Access the notebook at http://10.152.183.42:80.
```

Once you know the URL, open a web browser and enter the URL into the address bar. This will direct you to the notebook UI where you can start working with your notebook.

Next Steps

- To learn more about how to interact with DSS, see *Manage DSS*.
- To learn about handling data, check out *Access your data from DSS*.
- To connect to MLflow, see *Manage MLflow*.
- To leverage your GPUs, see *Enable GPUs*.

2.2 How-to guides

The following guides cover key processes in the Data Science Stack (DSS) environment. Specifically, they demonstrate how to perform common tasks for managing DSS components.

2.2.1 DSS basics

Learn basic operations to interact with DSS, including installation and the Command Line Interface (CLI) commands:

Manage DSS

This guide describes how to manage Data Science Stack (DSS).

DSS is a Command Line Interface (CLI)-based environment and distributed as a snap.

Install DSS

> **Note**
>
> To install DSS, ensure you have previously installed Snap and MicroK8s.

You can install DSS using `snap` as follows:

```
sudo snap install data-science-stack
```

Then, you can run the DSS CLI with:

```
dss
```

Start DSS

You can initialise DSS through `dss initialize`. This command:

- Stores credentials for the MicroK8s cluster.
- Allocates storage for your DSS Jupyter Notebooks.
- Deploys an MLflow model registry.

```
dss initialize --kubeconfig "$(sudo microk8s config)"
```

The `--kubeconfig` option is used to provide your MicroK8s cluster's kubeconfig.

> **Note**
>
> Note the use of quotes for the `--kubeconfig` option. Without them, the content may be interpreted by your shell.

You should expect an output like this:

```
Executing initialize command
Storing provided kubeconfig to /home/user/.dss/config
Waiting for deployment mlflow in namespace dss to be ready...
Deployment mlflow in namespace dss is ready
DSS initialized. To create your first notebook run the command:

dss create

Examples:
  dss create my-notebook --image=pytorch
  dss create my-notebook --image=kubeflownotebookswg/jupyter-scipy:v1.8.0
```

Remove DSS

You can remove DSS from your MicroK8s cluster through `dss purge`. This command purges all the DSS components, including:

- All Jupyter Notebooks.
- The MLflow server.
- Any data stored within the DSS environment.

> **Note**
>
> This action removes the components of the DSS environment, but it does not remove the DSS CLI or your MicroK8s cluster. To remove those, delete their snaps.

```
dss purge
```

> **Caution**
>
> This action is irreversible. All data stored within the DSS environment will be lost.

You should expect an output like this:

```
Waiting for namespace dss to be deleted...
Success: All DSS components and notebooks purged successfully from the Kubernetes␣
↪cluster.
```

Get DSS status

You can check the DSS status through `dss status`. This command provides a quick way to check the status of your DSS environment, including the MLflow status and whether a GPU is detected in your environment.

```
dss status
```

If you already have a DSS environment running and no GPU available, the expected output is:

```
MLflow deployment: Ready
MLflow URL: http://10.152.183.68:5000
GPU acceleration: Disabled
```

List DSS commands

You can get the list of available commands for DSS through the `dss` command with the `--help` option:

```
dss --help
```

You should expect an output like this:

```
Usage: dss [OPTIONS] COMMAND [ARGS]...

Command line interface for managing the DSS application.

Options:
--help  Show this message and exit.

Commands:
create      Create a Jupyter notebook in DSS and connect it to MLflow.
initialize  Initialize DSS on the given Kubernetes cluster.
list        Lists all created notebooks in the DSS environment.
logs        Prints the logs for the specified notebook or DSS component.
purge       Removes all notebooks and DSS components.
remove      Remove a Jupter Notebook in DSS with the name NAME.
start       Starts a stopped notebook in the DSS environment.
status      Checks the status of key components within the DSS...
stop        Stops a running notebook in the DSS environment.
```

Get details about a specific command:

To see the usage and options of a DSS command, run `dss <command>` with the `--help` option. For example:

```
dss logs --help
```

You should expect an output like this:

```
Usage: dss logs [OPTIONS] [NOTEBOOK_NAME]

Prints the logs for the specified notebook or DSS component.

Examples:
    dss logs my-notebook
    dss logs --mlflow
    dss logs --all

Options:
--kubeconfig TEXT  Path to a Kubernetes config file. Defaults to the value
                    of the KUBECONFIG environment variable, else to
                    './kubeconfig'.
--all              Print the logs for all notebooks and MLflow.
--mlflow           Print the logs for the MLflow deployment.
--help             Show this message and exit.
```

See also

- To learn how to manage your Jupyter Notebooks, check *Manage Jupyter Notebooks*.
- If you are interested in managing MLflow within your DSS environment, see *Manage MLflow*.

2.2.2 Jupyter Notebooks within DSS

Learn how to manage Jupyter Notebooks within your DSS environment:

Manage Jupyter Notebooks

This guide describes how to manage Jupyter Notebooks within your Data Science Stack (DSS) environment.

All actions can be performed using the DSS Command Line Interface (CLI).

Create a notebook

You can create a Jupyter Notebook using the DSS CLI. This notebook includes different packages and toolkits depending on the image used to create it.

1. **Select an image**:

Before creating a notebook, you need to select an image that includes the packages and toolkits you need. To see a list of recommended images and their aliases, do:

```
dss create --help
```

The output includes a list of recommended images and their aliases. For example, this guide uses the image *kubeflownotebookswg/jupyter-scipy:v1.8.0*

2. **Create the notebook**:

Create a new notebook as follows:

```
dss create my-notebook --image kubeflownotebookswg/jupyter-scipy:v1.8.0
```

This command starts a notebook server with the selected image. You should expect an output like this:

```
Executing create command
Waiting for deployment test-notebook in namespace dss to be ready...
Deployment test-notebook in namespace dss is ready
Success: Notebook test-notebook created successfully.
Access the notebook at http://10.152.183.42:80.
```

Create an NVIDIA GPU-enabled notebook

You can create an NVIDIA GPU-enabled Jupyter Notebook containing CUDA runtimes and Machine Learning (ML) frameworks, and access its JupyterLab server.

> **Note**
>
> To launch an NVIDIA GPU-enabled notebook, you must first *install* the NVIDIA Operator and *verify* DSS can detect the GPU. See *Enable NVIDIA GPUs* for more details.

To see the list of available CUDA images, run:

```
dss create --help | grep cuda
```

You should see an output similar to this:

```
- pytorch-cuda = kubeflownotebookswg/jupyter-pytorch-cuda-full:v1.8.0
- tensorflow-cuda = kubeflownotebookswg/jupyter-tensorflow-cuda-full:v1.8.0
```

Select one of them and create a notebook as follows:

```
dss create my-notebook --image=tensorflow-cuda
```

You can confirm your GPU is detected and usable by running the following within your notebook:

```
import tensorflow as tf

tf.config.list_physical_devices('GPU')
```

Create an Intel GPU-enabled notebook

You can create an Intel GPU-enabled Jupyter Notebook with Intel Extension for PyTorch (IPEX) or Intel Extension for TensorFlow (ITEX).

> **Note**
>
> To launch an Intel GPU-enabled notebook, you must first *Enable Intel GPUs*.

To see the list of available Intel images, run:

```
dss create --help | grep intel
```

You should see an output similar to this:

```
- pytorch-intel= intel/intel-extension-for-pytorch:2.1.20-xpu-idp-jupyter
- tensorflow-intel = intel/intel-extension-for-tensorflow:2.15.0-xpu-idp-jupyter
```

Select one of them and create a notebook as follows:

```
dss create my-itex-notebook --image=tensorflow-intel
```

> **Note**
>
> Once created, you can *access it* and run Intel-based ML workloads within your DSS environment. See IPEX example and ITEX example for detailed examples using the Intel extensions for Pytorch and Tensorflow respectively.

You can confirm your Intel GPU is detected and usable by running the following within your notebook:

```
import tensorflow as tf

tf.config.experimental.list_physical_devices()
```

For example, you should expect an output like the following for a host system containing an Intel CPU and a single Intel GPU:

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'), PhysicalDevice(name='/
↪physical_device:XPU:0', device_type='XPU')]
```

> **Note**
>
> Intel denotes XPU the combination of an Intel CPU with GPU.

List created notebooks

You can check the current state of all notebooks within your DSS environment. To view the full list, run:

```
dss list
```

This command displays each notebook name along with its associated image, state and URL if applicable. For example:

```
Name          Image                                                URL
my-notebook   kubeflownotebookswg/jupyter-tensorflow-full:v1.8.0  http://10.152.183.
↪164:80  (Active)
data-prep     kubeflownotebookswg/jupyter-minimal:v1.5.0           (Downloading)
test-env      kubeflownotebookswg/jupyter-scipy-notebook:v1.9.0    (Stopping)
```

Notebook states

Each notebook can be in one of the following states:

- **Active**: The notebook is running and accessible. You can use the URL under the *URL* column to access it.
- **Stopped**: The notebook is not running.
- **Stopping**: The notebook is in the process of stopping. It is advisable to wait until the process completes, transitioning to *Stopped*.
- **Starting**: The notebook is initialising and will soon be *Active*.
- **Downloading**: The notebook is downloading the specified OCI Image. This is a transient state before it becomes *Active*.
- **Removing**: The notebook is in the process of being removed. This is a transient state before it is fully removed.

Remove a notebook

You can remove a Jupyter Notebook using the DSS CLI. It is a non-blocking process, meaning you can continue other work while the deletion completes.

> **Note**
>
> When you remove a notebook, any data stored under *~/shared* within the notebook will be preserved and remain accessible to other notebooks. This shared storage is designed to ensure that valuable data is not lost even when individual notebooks are removed from the environment.

1. **Remove the notebook**:

To delete the notebook, use the `dss remove` command followed by the name of the notebook, `my-notebook` in this example:

```
dss remove my-notebook
```

You should expect an output like this:

```
Removing the notebook my-notebook. Check `dss list` for the status of the notebook.
```

2. **Verify the notebook has been removed**:

To confirm the notebook has been removed, you can check the list of notebooks again:

```
dss list
```

If the notebook has been successfully removed, it will no longer appear in the list. If it's still showing as *Removing*, you may need to wait a bit longer or investigate if there are any issues preventing its deletion.

Start a notebook

You can start a notebook using the DSS CLI. This enables you to resume your work without needing to configure a new notebook.

1. **Start the notebook**:

To start the notebook, use the `dss start` command followed by the name of the notebook, `my-notebook` in this example:

```
dss start my-notebook
```

You should expect an output like this:

```
Executing start command
Starting the notebook my-notebook. Check `dss list` for the status of the notebook.
```

2. **Verify the notebook is running**:

After starting it, the notebook may go through *different states*. To check its state, run:

```
dss list
```

Once ready, you should expect an output like this:

```
Name          Image                                                URL
my-notebook   kubeflownotebookswg/jupyter-tensorflow-full:v1.8.0  http://10.152.183.
↪164:80
```

You can use this URL to *access the notebook*.

Stop a notebook

You can stop a notebook using the DSS CLI. Stopping a notebook frees up resources and ensures data safety when not actively working on it.

1. **Stop the notebook**:

To stop a running notebook, use the `dss stop` command followed by the name of the notebook, `my-notebook` in this example:

```
dss stop my-notebook
```

You should see an output like this:

```
Stopping the notebook my-notebook. Check `dss list` for the status of the notebook.
```

2. **Verify the notebook has stopped**:

After stopping it, the notebook may go through *different states*. To confirm it has stopped, check its state:

```
dss list
```

You should expect an output like this:

```
Name          Image                                                  URL
my-notebook   kubeflownotebookswg/jupyter-tensorflow-full:v1.8.0  (Stopped)
```

Access a notebook

You can access a notebook User Interface (UI) using the DSS CLI. Accessing the UI enables you to interact directly with your notebook, run code, and visualise data. This is done through a web browser by navigating to the URL associated with your active notebook.

> **Note**
>
> Ensure your notebook is in *Active state* to be able to access it. Otherwise, you may need to *start* it or check for any issues that are preventing it from being accessible.

1. **Find the notebook URL**:

To find the URL of your notebook, first list all the notebooks:

```
dss list
```

Look for your notebook in the output, and specifically check the URL column. An active notebook has associated a URL, which indicates it is ready for accessing.

You should expect an output like this:

```
Name          Image                                                  URL
my-notebook   kubeflownotebookswg/jupyter-tensorflow-full:v1.8.0  http://10.152.183.
↪164:80
```

2. **Access the Notebook UI**:

Once you know the URL, open a web browser and enter the URL into the address bar. This will direct you to the notebook UI where you can start working with your notebook.

Get notebook logs

You can retrieve logs for a Jupyter Notebook using the DSS CLI. Retrieving logs can help you troubleshoot issues, monitor notebook activities, or verify actions taken in the notebook.

To get the logs for a certain notebook, use the `dss logs` command followed by the name of the notebook, `my-notebook` in this example:

```
dss logs my-notebook
```

You should expect an output like this:

```
Logs for my-notebook-8cf4d9bc-jm9zm:
s6-rc: info: service s6rc-oneshot-runner: starting
s6-rc: info: service s6rc-oneshot-runner successfully started
s6-rc: info: service fix-attrs: starting
s6-rc: info: service fix-attrs successfully started
s6-rc: info: service legacy-cont-init: starting
cont-init: info: running /etc/cont-init.d/01-copy-tmp-home
cont-init: info: /etc/cont-init.d/01-copy-tmp-home exited 0
s6-rc: info: service legacy-cont-init successfully started
s6-rc: info: service legacy-services: starting
services-up: info: copying legacy longrun jupyterlab (no readiness notification)
s6-rc: info: service legacy-services successfully started
[W 2024-04-30 13:44:20.991 ServerApp] ServerApp.token config is deprecated in 2.0. Use␣
↪IdentityProvider.token.
[I 2024-04-30 13:44:20.996 ServerApp] Package jupyterlab took 0.0000s to import
[I 2024-04-30 13:44:20.997 ServerApp] Package jupyter_server_fileid took 0.0013s to␣
↪import
[I 2024-04-30 13:44:20.998 ServerApp] Package jupyter_server_mathjax took 0.0007s to␣
↪import
[I 2024-04-30 13:44:21.001 ServerApp] Package jupyter_server_terminals took 0.0024s to␣
↪import
[I 2024-04-30 13:44:21.012 ServerApp] Package jupyter_server_ydoc took 0.0105s to import
[I 2024-04-30 13:44:21.022 ServerApp] Package jupyterlab_git took 0.0104s to import
[I 2024-04-30 13:44:21.022 ServerApp] Package nbclassic took 0.0000s to import
```

Connect from notebook to MLflow

You can integrate MLflow with your Jupyter Notebook for tracking experiments using DSS.

MLflow is a platform for managing the end-to-end machine learning life cycle. It includes tracking experiments, packaging code into reproducible runs, and sharing and deploying models.

DSS environments are pre-configured to interact with an MLflow server through the *MLFLOW_TRACKING_URI* environment variable set in each notebook.

Installing MLflow

To interact with MLflow, the MLflow Python library needs to be installed within your notebook environment. There are two ways to install the MLflow library:

1. **Within a notebook cell** (Recommended):

It's recommended to install MLflow directly within a notebook cell to ensure the library is available for all subsequent cells during your session:

```
%%bash
pip install mlflow
```

2. **Using the notebook terminal**:

Alternatively, you can install MLflow from the notebook terminal with the same command. This method also installs MLflow for the current session:

```
pip install mlflow
```

Note that any installations via the notebook or terminal will not persist after the notebook is restarted. Therefore, the first method is preferred to ensure consistency across sessions.

Connecting to MLflow library

After installing MLflow, you can directly interact with the MLflow server configured for your DSS environment:

```
import mlflow

c = mlflow.MlflowClient()

print(c.tracking_uri)

c.create_experiment("test-experiment")
```

This example shows how to initialise the MLflow client, check the tracking URI, and create a new experiment. The *MLFLOW_TRACKING_URI* should already be set in your environment, allowing you to focus on your experiments without manual configuration.

For more detailed information on using MLflow, including advanced configurations and features, refer to the official MLflow Docs.

Access your data from DSS

You can access the stored data from your notebooks using the DSS CLI. Accessing your data is useful when you want to browse or modify the files stored from your notebooks.

> **Note**
>
> By default, your notebooks data are stored in a directory under */var/snap/microk8s/common/default-storage*. See Microk8s hostpath docs for more information.

This directory is shared by all your DSS notebooks.

1. **Find the directory of your stored data**

To find the directory containing your notebooks data, list the directories under */var/snap/microk8s/common/default-storage*:

```
ls /var/snap/microk8s/common/default-storage/
```

You should see an output like this:

```
dss-notebooks-pvc-00037e23-e2e2-4ab4-9088-45099154da30
```

The storage directory is the one prefixed with *dss-notebooks-pvc* as shown in the output.

> **Note**
>
> The characters that follow *dss-notebooks-pvc-* may not be the same for all DSS environments.

2. **Access your notebooks data**

From your local file browser, navigate to the folder */var/snap/microk8s/common/default-storage/[directory name]*. Use the directory name you got from the previous step.

Now, you can view and manage all your stored notebooks data.

See also

- To learn how to manage your DSS environment, check *Manage DSS*.
- If you are interested in managing MLflow within your DSS environment, see *Manage MLflow*.

2.2.3 MLflow within DSS

Learn how to manage MLflow within your DSS environment:

Manage MLflow

This guide describes how to manage MLflow within your Data Science Stack (DSS) environment.

MLflow is a platform for managing the end-to-end machine learning life cycle. It includes tracking experiments, packaging code into reproducible runs, and sharing and deploying models.

Access MLflow

You can access the MLflow User Interface (UI) within your Data Science Stack (DSS) environment through a web browser, by navigating to the URL associated with MLflow. This UI allows you to interact directly with your MLflow experiments and models.

1. **Get the MLflow URL**:

To find the URL of MLflow, run:

```
dss status
```

Look for the MLflow URL in the output. For example:

```
MLflow deployment: Ready
MLflow URL: http://10.152.183.205:5000
```

> **Note**
>
> To access the UI, your MLflow deployment should be *Ready*.

2. **Access the MLflow UI**:

Once you know the URL, open a web browser and enter the URL into the address bar. This will direct you to the MLflow interface.

Get MLflow logs

You can retrieve logs for MLflow within your Data Science Stack (DSS) environment. Retrieving logs is a critical task for maintaining and troubleshooting MLflow.

To get MLflow logs, use the `dss logs` command with the `--mlflow` option:

```
dss logs --mlflow
```

You should expect an output like this:

```
Logs for mlflow-6bbfc5db5-xlfvj:
[2024-04-30 07:57:54 +0000] [22] [INFO] Starting gunicorn 20.1.0
[2024-04-30 07:57:54 +0000] [22] [INFO] Listening at: http://0.0.0.0:5000 (22)
[2024-04-30 07:57:54 +0000] [22] [INFO] Using worker: sync
[2024-04-30 07:57:54 +0000] [23] [INFO] Booting worker with pid: 23
[2024-04-30 07:57:54 +0000] [24] [INFO] Booting worker with pid: 24
[2024-04-30 07:57:54 +0000] [25] [INFO] Booting worker with pid: 25
[2024-04-30 07:57:54 +0000] [26] [INFO] Booting worker with pid: 26
```

Get MLflow artefacts

MLflow artefacts, including models, experiments and runs are stored within your DSS environment. They can be accessed and downloaded from the *MLflow UI*.

See also

- To learn how to manage your DSS environment, check *Manage DSS*.
- If you are interested in managing Jupyter Notebooks within your DSS environment, see *Manage Jupyter Notebooks*.
- See Charmed MLflow for more details on MLflow.

2.2.4 Enable GPUs

Learn how to configure DSS to leverage your GPUs:

Enable GPUs

The following guides cover configuration aspects to leverage your GPUs within the Data Science Stack (DSS) environment.

Enable Intel GPUs

This guide describes how to configure Data Science Stack (DSS) to utilise the Intel GPUs on your machine.

You can do so done by enabling the Intel device plugin on your MicroK8s cluster.

Prerequisites

- Ubuntu 22.04.
- DSS is *installed* and *initialised*.
- The kubectl snap package is installed.
- Your machine includes an Intel GPU.

Note

After installing *kubectl*, configure MicroK8s to run *kubectl* commands as follows:

```
mkdir -p ~/.kube
microk8s config > ~/.kube/config
```

Verify the Intel GPU drivers

To confirm that your machine has the Intel GPU drivers set up, first install the *intel-gpu-tools* package:

```
sudo apt install intel-gpu-tools
```

Now list the Intel GPU devices on your machine as follows:

```
intel_gpu_top -L
```

If the drivers are correctly installed, you should see information about your GPU device such as the following:

```
card0                     8086:56a0
pci:vendor=8086,device=56A0,card=0
└─renderD128
```

Note

For Intel discrete GPUs on Ubuntu versions older than 24.04, you may need to perform additional steps such as installing a HWE kernel.

Enable the Intel GPU plugin

To ensure DSS can utilise Intel GPUs, you have to enable the Intel GPU plugin in your MicroK8s cluster.

1. Use *kubectl kustomize* to build the plugin YAML configuration files:

```
VERSION=v0.30.0
kubectl kustomize https://github.com/intel/intel-device-plugins-for-kubernetes/
↪deployments/nfd?ref=${VERSION} > node_feature_discovery.yaml
kubectl kustomize https://github.com/intel/intel-device-plugins-for-kubernetes/
↪deployments/nfd/overlays/node-feature-rules?ref=${VERSION} > node_feature_rules.yaml
kubectl kustomize https://github.com/intel/intel-device-plugins-for-kubernetes/
↪deployments/gpu_plugin/overlays/nfd_labeled_nodes?ref=${VERSION} > gpu_plugin.yaml
```

To allow multiple containers to utilise the same GPU, run:

```
sed -i 's/enable-monitoring/enable-monitoring\n        - -shared-dev-num=10/' gpu_plugin.
↪yaml
```

2. Apply the built YAML files to your MicroK8s cluster:

```
kubectl apply -f node_feature_discovery.yaml
kubectl apply -f node_feature_rules.yaml
kubectl apply -f gpu_plugin.yaml
```

The MicroK8s cluster is now configured to recognise and utilise your Intel GPU.

> **Note**
>
> After the YAML configuration files have been applied, they can be safely deleted.

Verify the Intel GPU plugin

To verify the Intel GPU plugin is installed and the MicroK8s cluster recognises your GPU, run:

```
kubectl get nodes --show-labels | grep intel
```

You should see an output with the cluster name such as the following:

```
kubectl get nodes --show-labels | grep intel
fluent-greenshank   Ready    <none>   18s   v1.30.3   beta.kubernetes.io/arch=amd64,beta.
↪kubernetes.io/os=linux,intel.feature.node.kubernetes.io/gpu=true
```

Verify DSS detects the GPU

Verify DSS has detected the GPU by checking the DSS status. To do so, run the following command using the DSS CLI:

```
dss status
```

You should expect an output like this:

```
Output:
MLflow deployment: Ready
MLflow URL: http://10.152.183.68:5000
NVIDIA GPU acceleration: Disabled
Intel GPU acceleration: Enabled
```

See also

- To learn how to manage your DSS environment, check *Manage DSS*.
- If you are interested in managing Jupyter Notebooks within your DSS environment, see *Manage Jupyter Notebooks*.

Enable NVIDIA GPUs

This guide describes how to configure Data Science Stack (DSS) to utilise your NVIDIA GPUs.

You can do so by configuring the underlying MicroK8s, on which DSS relies on for running the containerised workloads.

Prerequisites

- DSS is *installed* and *initialised*.
- Your machine includes an NVIDIA GPU.

Install the NVIDIA Operator

To ensure DSS can utilise NVIDIA GPUs:

1. The NVIDIA drivers must be installed.
2. MicroK8s must be set up to utilise NVIDIA drivers.

MicroK8s is leveraging the NVIDIA Operator for setting up and configuring the NVIDIA runtime. The NVIDIA Operator also installs the NVIDIA drivers, if they are not present already on your machine.

To enable the NVIDIA runtimes on MicroK8s, run the following command:

```
sudo microk8s enable gpu
```

> **Note**
>
> The NVIDIA Operator detects the installed NVIDIA drivers in your machine. If they are not installed, it will do so automatically.

Verify the NVIDIA Operator is up

Before spinning up workloads, the GPU Operator has to be successfully initialised. To do so, you need to assure DaemonSet is ready and the Validator Pod has succeeded.

> **Note**
>
> This process can take approximately 5 minutes to complete.

Ensure DaemonSet is ready

First, ensure that the DaemonSet for the Operator Validator is created:

```
while ! sudo microk8s.kubectl get ds \
    -n gpu-operator-resources \
    nvidia-operator-validator
do
  sleep 5
done
```

> **Note**
>
> It takes some seconds for the DaemonSet to get created. The above command returns a message like the following at the beginning of the process: `Error from server (NotFound): daemonsets.apps "nvidia-operator-validator" not found`

Once completed, you should expect an output like this:

```
Error from server (NotFound): daemonsets.apps "nvidia-operator-validator" not found
...
Error from server (NotFound): daemonsets.apps "nvidia-operator-validator" not found
NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE␣
↪SELECTOR                                     AGE
nvidia-operator-validator   1         1         0       1            0           nvidia.
↪com/gpu.deploy.operator-validator=true   17s
```

Ensure the Validator Pod succeeded

Next, you need to wait for the Validator Pod to succeed:

```
echo "Waiting for the NVIDIA Operator validations to complete..."

while ! sudo microk8s.kubectl logs \
    -n gpu-operator-resources \
    -l app=nvidia-operator-validator \
    -c nvidia-operator-validator | grep "all validations are successful"
do
    sleep 5
done
```

Note

It takes some seconds for the Validator Pod to get initialised. The above command returns a message like the following at the beginning of the process: `Error from server (BadRequest): container "nvidia-operator-validator" in pod "nvidia-operator-validator-4rq5n" is waiting to start: PodInitializing`

Once completed, you should expect an output like this:

```
Error from server (BadRequest): container "nvidia-operator-validator" in pod "nvidia-
↪operator-validator-4rq5n" is waiting to start: PodInitializing
...
Error from server (BadRequest): container "nvidia-operator-validator" in pod "nvidia-
↪operator-validator-4rq5n" is waiting to start: PodInitializing
all validations are successful
```

Verify DSS detects the GPU

At this point, the underlying MicroK8s cluster has been configured for handling the NVIDIA GPU. Verify the DSS CLI has detected the GPU by checking the DSS status as follows:

```
dss status
```

You should expect an output like this:

```
MLflow deployment: Ready
MLflow URL: http://10.152.183.74:5000
GPU acceleration: Enabled (NVIDIA-GeForce-RTX-3070-Ti)
```

Note

The GPU model *NVIDIA-GeForce-RTX-3070-Ti* might differ from your setup.

See also

- To learn how to manage your DSS environment, check *Manage DSS*.
- If you are interested in managing Jupyter Notebooks within your DSS environment, see *Manage Jupyter Notebooks*.

2.3 Explanation

The following guides cover key concepts and features of Data Science Stack (DSS).

2.3.1 Architecture overview

Understand the underlying architecture of DSS, its components and interactions:

DSS architecture

This guide provides an overview of the Data science stack (DSS) architecture, its main components and their interactions.

DSS is a ready-to-run environment for Machine Learning (ML) and Data Science (DS). It's built on open-source tooling, including MicroK8s, JupyterLab and MLflow.

DSS is distributed as a snap and usable on any Ubuntu workstation. This provides robust security management and user-friendly version control, enabling seamless updates and auto-rollback in case of failure.

Using DSS, you can perform the following tasks:

- Installing and managing the DSS Python library.
- Deploying and managing Jupyter Notebooks.
- Deploying and managing MLflow.
- Running GPU workloads.

Architecture overview

The DSS architecture includes these layers:

- *Application*.
- *ML tools*.
- *Orchestration*.
- *Operating system*.

The following diagram showcases it:

More details on each layer are discussed in the following sections.

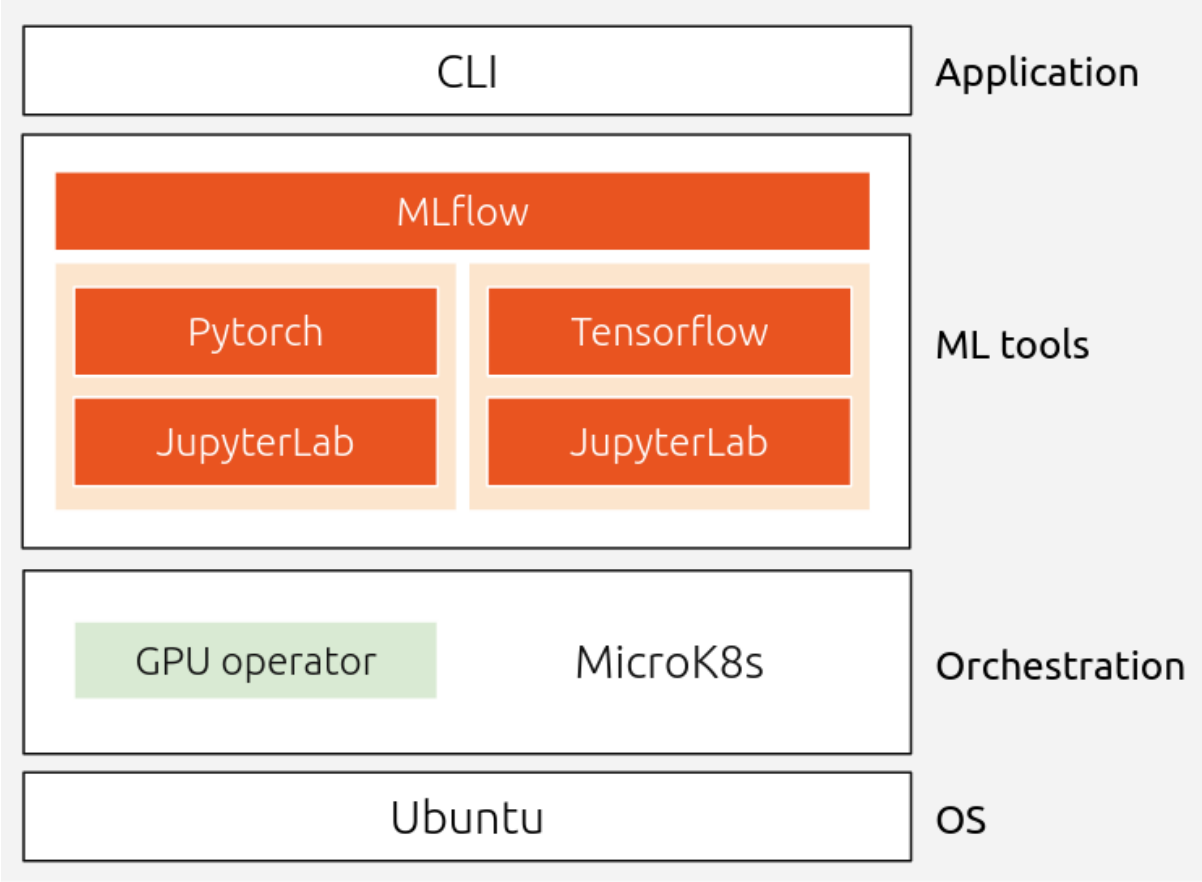

Fig. 1: Architecture overview

Application

DSS is a Command Line Interface (CLI)-based tool, accessible from the Ubuntu terminal. See *Manage DSS* to learn about how to manage your DSS environment and the available CLI commands.

ML tools

DSS includes:

- Jupyter Notebooks: Open source environment that provides a flexible interface to organise DS projects and ML workloads.
- MLflow: Open source platform for managing the ML life cycle, including experiment tracking and model registry.
- ML frameworks: DSS comes by default with PyTorch and Tensorflow. Users can manually add other frameworks, depending on their needs and use cases.

Jupyter Notebooks

A Jupyter Notebook is essentially a Kubernetes deployment, also known as *Pod*, running a Docker image with Jupyter Lab and a dedicated ML framework, such as Pytorch or Tensorflow. For each Jupyter Notebook, DSS mounts a Hostpath directory-backed persistent volume to the data directory. All Jupyter Notebooks share the same persistent volume, allowing them to exchange data seamlessly. The full path to that persistent volume is */home/jovyan/shared*.

MLflow

MLflow operates in local mode, meaning that metadata and artefacts are, by default, stored in a local directory.

This local directory is backed by a persistent volume, mounted to a Hostpath directory of the MLflow Pod. The persistent volume can be found in the directory */mlruns*.

Orchestration

DSS requires a container orchestration solution. DSS relies on MicroK8s, a lightweight Kubernetes distribution.

Therefore, MicroK8s needs to be deployed before installing DSS on the host machine. It must be configured with the storage add-on. This is required to use Hostpath storage in the cluster. See *Setting up MicroK8s* to learn how to install MicroK8s.

GPU support

DSS can run with or without the use of GPUs. If needed, MicroK8s can be configured with the desired GPU add-on.

DSS is designed to support the deployment of containerised GPU workloads on NVIDIA GPUs. MicroK8s simplifies the GPU access and usage through the NVIDIA GPU Operator.

DSS does not automatically install the tools and libraries required for running GPU workloads. To do so, it relies on MicroK8s for the required operating-system drivers. It also relies on the chosen image, for example, CUDA when working with NVIDIA GPUs.

Caution

GPUs from other silicon vendors rather than NVIDIA can be configured. However, its functionality is not guaranteed.

Storage

DSS expects a default storage class in the Kubernetes deployment, which is used to persist Jupyter Notebooks and MLflow artefacts. In MicroK8s, the Hostpath storage add-on is chosen, used to provision Kubernetes' *PersistentVolumeClaims* (PVCs).

A shared PVC is used across all Jupyter Notebooks to share and persist data. MLflow also uses its dedicated PVC to store the logged artefacts. This is the DSS default storage configuration and cannot be altered.

This choice ensures that all storage is backed up on the host machine in the event of MicroK8s restarts.

Note

By default, you can access the DSS storage anytime under your local directory */var/snap/microk8s/common/default-storage*.

The following diagram summarises the DSS storage:

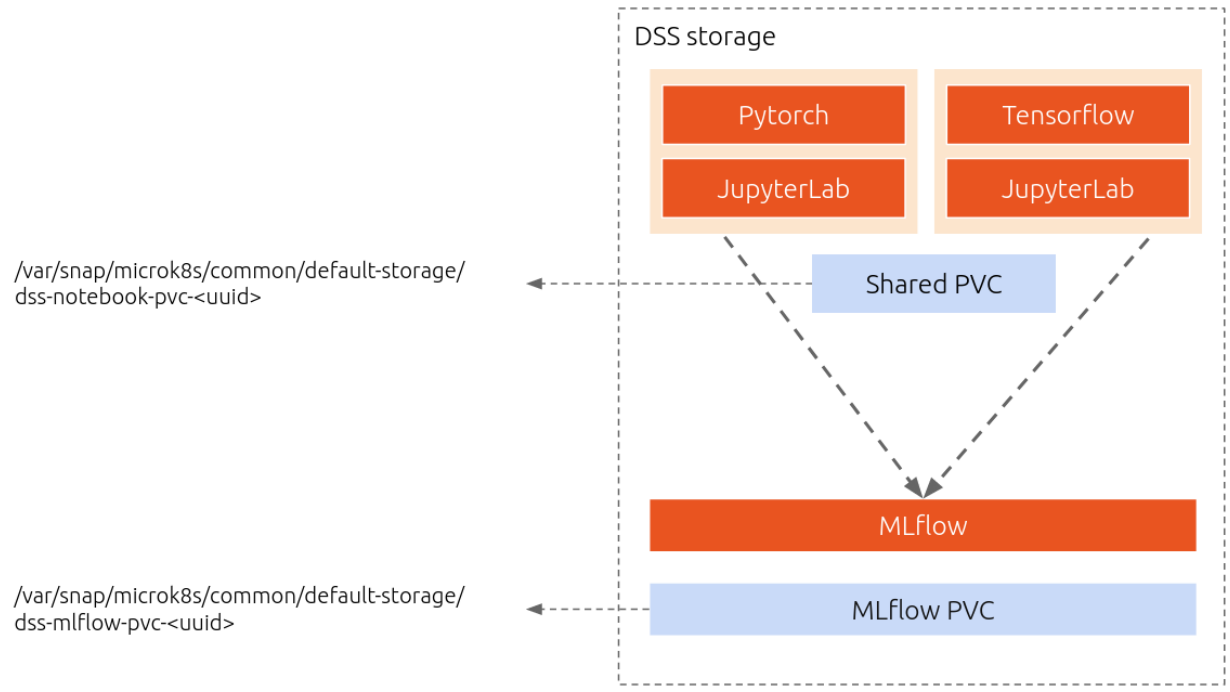

Fig. 2: Storage overview

Operating system

DSS is native on Ubuntu, being developed, tested and validated on it. Moreover, the solution can be used on any Linux distribution.

Namespace configuration

DSS runs on a dedicated Kubernetes namespace. By default, it contains two Kubernetes Pods.

The NVIDIA GPU support runs on another dedicated namespace. This includes the GPU Operator for managing access and usage.

Accessibility

Jupyter Notebooks and MLflow can be accessed from a web browser through the Pod IP that is given access through MicroK8s. See *Access a notebook* and *Access MLflow* for more details.