Charmed MLflow

Canonical Group Ltd

Jul 24, 2024

CONTENTS

Charmed MLflow is a platform for managing the end-to-end machine learning lifecycle.

It provides tools for tracking experiments, packaging code into reproducible runs, and sharing and deploying models. It integrates with popular machine learning frameworks.

It addresses a number of common machine learning challenges: collaboration, reproducibility, maintenance, organisation and scaling.

It's ideal for data scientists, ML engineers, hobbyists and teams looking to optimise their ML workflows with charms.

CHAPTER

ONE

IN THIS DOCUMENTATION

Tutorial

Start here: a hands-on introduction to Charmed MLflow for newcomers

How-to guides

Step-by-step guides covering key operations and common tasks in Charmed MLflow

Reference

Technical information - specifications, APIs, architecture of Charmed MLflow

Explanation

Discussion and clarification of key Charmed MLflow concepts and features

CHAPTER

TWO

PROJECT AND COMMUNITY

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

- Code of conduct
- Contribute
- Join our online chat
- Upstream Project
- Discourse Forum

2.1 Tutorial

Step-by-step guides to help you get started with deploying and managing machine learning workflows using Charmed MLflow.

We provide two pathways. For users who just want to try MLflow:

2.1.1 Get Started with Charmed MLflow

Component	Version
MLflow	2

Welcome to the tutorial on Charmed MLflow! MLflow is an open-source platform, used for managing machine learning workflows. It has four primary functions that include experiment tracking, model registry, model management and code reproducibility.

So wait, what does "Charmed MLflow" mean? Is it the same thing as MLflow? Yes and no. MLflow is a complex application, consisting of many components running together and communicating with each other. Charmed MLflow is a charm bundle that allows us to deploy MLflow quickly and easily. Don't worry too much about what a "charm bundle" is right now. The key thing is that it's going to make deploying MLflow very convenient for us: we'll get MLflow up and running with just a few command line commands!

In this tutorial, we're going to explore Charmed MLflow in a practical way. Using the Juju CLI tool, we'll deploy MLflow to a local MicroK8s cloud.

Prerequisites

We are assuming that you are running this tutorial on a local machine with the following specs:

- Runs Ubuntu 22.04 or later
- Has at least 50GB free disk space

Install and prepare MicroK8s

Let's install MicroK8s. MicroK8s is installed from a snap package. The published snap maintains different `channels` for different releases of Kubernetes.

```
sudo snap install microk8s --classic --channel=1.24/stable
```

For MicroK8s to work without having to use `sudo` for every command, it creates a group called `microk8s`. To make it more convenient to run commands, you will add the current user to this group:

```
sudo usermod -a -G microk8s $USER
newgrp microk8s
```

It is also useful to make sure the user has the proper access and ownership of any `kubectl` configuration files:

```
sudo chown -f -R $USER ~/.kube
```

Enable the following MicroK8s addons to configure your Kubernetes cluster with extra services needed to run Charmed Kubeflow.

```
microk8s enable dns hostpath-storage ingress metallb:10.64.140.43-10.64.140.49
```

Here, we added a `dns` service, so the applications can find each other, storage, an ingress controller so we can access Kubeflow components and the `MetalLB` load-balancer application. You can see that we added some detail when enabling `MetalLB`, in this case the address pool to use.

> See More : MicroK8s | How to use addons

We've now installed and configured MicroK8s. It will start running automatically, but can take 5 minutes or so before it's ready for action. Run the following command to tell MicroK8s to report its status to us when it's ready:

```
microk8s status --wait-ready
```

Be patient - this command may not return straight away. The `--wait-ready` flag tells MicroK8s to wait for the Kubernetes services to initialise before returning. Once MicroK8s is ready, you will see something like the following output:

```
microk8s is running
```

Below this there will be a bunch of other information about the cluster.

Great, we have now installed and configured MicroK8s, and it's running and ready!

Install Juju

Juju is an operation Lifecycle manager (OLM) for clouds, bare metal or Kubernetes. We will be using it to deploy and manage the components which make up Kubeflow.

To install Juju from snap, run this command:

```
sudo snap install juju --classic --channel=2.9/stable
```

Now, run the following command to deploy a Juju controller to the Kubernetes we set up with MicroK8s:

```
juju bootstrap microk8s
```

Sit tight while the command completes! The controller may take a minute or two to deploy.

The controller is the agent of Juju, running on Kubernetes, which can be used to deploy and control the components of Kubeflow.

Next, we'll need to add a model for Kubeflow to the controller. Run the following command to add a model called `kubeflow`:

```
juju add-model kubeflow
```

Note: The model name here can be anything. We're just using `kubeflow` because often you may want to deploy MLflow along with Kubeflow, and in that case, the model name must be `kubeflow`. So it's not a bad habit to have.

The controller can work with different `models`, which map 1:1 to namespaces in Kubernetes. In this case, the model name must be `kubeflow`, due to an assumption made in the upstream Kubeflow Dashboard code.

Great job: Juju has now been installed and configured for Kubeflow!

Deploy MLflow bundle

Before deploying, run these commands:

```
sudo sysctl fs.inotify.max_user_instances=1280
sudo sysctl fs.inotify.max_user_watches=655360
```

We need to run the above because under the hood, MicroK8s uses `inotify` to interact with the filesystem, and in large MicroK8s deployments sometimes the default `inotify` limits are exceeded.

Let's now use Juju to deploy Charmed MLflow. Run the following command:

```
juju deploy mlflow --channel=2.1/stable --trust
```

This deploys the latest edge version of MLflow with MinIO as object storage and MySQL as metadata store.

Access MLflow

To access MLflow, visit the following URL in your web browser:

```
http://localhost:31380/
```

This will take you to the MLflow UI.

Note: by default Charmed MLflow creates a NodePort on port 31380 where you can access the MLflow UI.

That's it! Charmed MLflow has been deployed locally with MicroK8s and Juju. You can now start using MLflow.

Reference: Object storage credentials

To use MLflow you need to have credentials to the object storage. The aforementioned bundle comes with MinIO. To get the `MinIO` credentials run the following command:

```
juju run-action mlflow-server/0  get-minio-credentials --wait
```

This action will output `secret-key` and `secret-access-key`.

For users who want to try MLflow and Kubeflow together:

2.1.2 Getting Started with Charmed MLflow and Kubeflow

Component	Version
MLflow	2
Kubeflow	1.7

Welcome to this tutorial on getting started with Charmed MLflow alongside Charmed Kubeflow. If you would like to deploy Kubeflow by itself, see our tutorial on Charmed Kubeflow.

Prerequisites

This tutorial assumes you will be deploying Kubeflow and MLflow on a public cloud Virtual Machine (VM) with the following specs:

- Runs Ubuntu 20.04 (focal) or later.
- Has at least 4 cores, 32GB RAM and 100GB of disk space available.
- Is connected to the internet for downloading the required snaps and charms.

We'll also assume that you have a laptop that meets the following conditions:

- Has an SSH tunnel open to the VM with port forwarding and a SOCKS proxy. To see how to set this up, see How to setup SSH VM Access.
- Runs Ubuntu 20.04 (focal) or later.
- Has a web browser installed e.g. Chrome / Firefox / Edge.

In the remainder of this tutorial, unless otherwise stated, it is assumed you will be running all command line operations on the VM, through the open SSH tunnel. It's also assumed you'll be using the web browser on your local machine to access the Kubeflow and MLflow dashboards.

Deploy MLflow

Follow the steps in this tutorial to deploy MLflow on your VM: *Get Started with Charmed MLflow*. Before moving on with this tutorial, confirm that you can now access the MLflow UI on `http://localhost:31380`.

Deploy Kubeflow bundle

Let's deploy Charmed Kubeflow alongside MLflow. Run the following command to initiate the deployment:

```
juju deploy kubeflow --trust  --channel=1.7/stable
```

Configure Dashboard Access

Run the following commands:

```
juju config dex-auth public-url=http://10.64.140.43.nip.io
juju config oidc-gatekeeper public-url=http://10.64.140.43.nip.io
```

This tells the authentication and authorisation components of the bundle that users who access the bundle will be doing so via the URL `http://10.64.140.43.nip.io`. In turn, this allows those components to construct appropriate responses to incoming traffic.

Now set the dashboard username and password:

```
juju config dex-auth static-username=user123@email.com
juju config dex-auth static-password=user123
```

Deploy Resource Dispatcher

Next, let's deploy the resource dispatcher. The resource dispatcher is an optional component which will distribute Kubernetes objects related to MLflow credentials to all user namespaces in Kubeflow. This means that all your Kubeflow users can access the MLflow model registry from their namespaces. To deploy the dispatcher, run the following command:

```
juju deploy resource-dispatcher --channel 1.0/stable --trust
```

This will deploy the latest edge version of the dispatcher. See Resource Dispatcher on GitHub for more info. Now we must relate the dispatcher to MLflow:

```
juju relate mlflow-server:secrets resource-dispatcher:secrets
juju relate mlflow-server:pod-defaults resource-dispatcher:pod-defaults
```

Monitor The Deployment

Now, at this point, we've deployed MLflow and Kubeflow and we've related them via the resource dispatcher. But that doesn't mean our system is ready yet: Juju will need to download charm data from CharmHub and the charms themselves will take some time to initialise.

So, how do you know when all the charms are ready, then? You can do this using the `juju status` command. First, let's run a basic status command and review the output. Run the following command to print out the status of all the components of Juju:

```
juju status
```

Review the output for yourself. You should see some summary information, a list of Apps and associated information, and another list of Units and their associated information. Don't worry too much about what this all means for now. If you're interested in learning more about this command and its output, see the Juju Status command.

The main thing we're interested in at this stage is the statuses of all the applications and units running through Juju. We want all the statuses to eventually become `active`, indicating that the bundle is ready. Run the following command to keep a watch on the components which are not active yet:

```
watch -c 'juju status --color | grep -E "blocked|error|maintenance|waiting|App|Unit"'
```

This will periodically run a `juju status` command and filter to components which are in a state of `blocked`, `error`, `maintenance` or `waiting` i.e. not `active`. When this output becomes empty except for the "App" and "Unit" headings, then we know all statuses are active and our system is ready.

Don't be surprised if some of the components' statuses change to `blocked` or `error` every now and then. This is expected behaviour, and these statuses should resolve by themselves as the bundle configures itself. However, if components remain stuck in the same error states, consult the troubleshooting steps below.

Expand to troubleshoot: Waiting for gateway relation

An issue you might have is the `tensorboard-controller` component might be stuck with a status of `waiting` and a message "Waiting for gateway relation". To fix this, run:

```
juju run --unit istio-pilot/0 -- "export JUJU_DISPATCH_PATH=hooks/config-changed; ./
↪dispatch"
```

This is a known issue, see TensorBoard controller GitHub issue for more info.

Be patient, it can take up to an hour for all those charms to download and initialise. In the meantime, why not try our Juju tutorial?

Integrate MLflow with Notebook

In this section, we're going to create a notebook server in Kubeflow and connect it to MLflow. This will allow our notebook logic to talk to MLflow in the background. Let's get started.

First, to be able to use MLflow credentials in your Kubeflow notebook, visit the dashboard at `http://10.64.140.43.nip.io/` and fill the username and password which you configured in the previous section e.g. `user123@email.com` and `user123`.

Click on start setup to setup the Kubeflow user for the first time.

Select `Finish` to finish the process.

Now a Kubernetes namespace was created for your user. To use MLflow with this user, label the namespace with the following command:

```
microk8s kubectl label ns user123 user.kubeflow.org/enabled="true"
```

You will get the following output: `namespace/user123 labeled`.

For more info on the label command, check Kubernetes labels. For more info on Kubernetes namespaces for users, see the upstream docs on Multi-user isolation.

Now go back to the Dashboard. From the left panel, choose notebooks. Select +New Notebook.

At this point, we can name the notebook as we want, and choose the desired image and resource limits. For now, let's just keep things simple:

1. For `Name`, enter `test-notebook`.
2. Expand the *Custom Notebook* section and for `image`, select `kubeflownotebookswg/jupyter-tensorflow-full:v1.7.0`.

Now, in order to allow our notebook server access to MLflow, we need to enable some special configuration options. Scroll down to `Data Volumes -> Advanced options` and from the `Configurations` dropdown, choose the following options:

1. Allow access to Kubeflow pipelines.
2. Allow access to MinIO.
3. Allow access to MLflow.

Note: Remember we related Kubeflow to MLflow earlier using the resource dispatcher? This is why we're seeing the MinIO and MLflow options in the dropdown!

Great, that's all the configuration for the notebook server done. Hit the Launch button to launch the notebook server. Be patient, the notebook server will take a little while to initialise.

When the notebook server is ready, you'll see it listed in the Notebooks table with a success status. At this point, select `Connect` to connect to the notebook server.

When you connect to the notebook server, you'll be taken to the notebook environment in a new tab. Because of our earlier configurations, this environment is now connected to MLflow in the background. This means the notebooks we create here can access MLflow. Cool!

To test this, create a new notebook and paste the following command into it, in a cell:

```
!printenv | grep MLFLOW
```

Run the cell. This will print out two environment variables `MLFLOW_S3_ENDPOINT_URL` and `MLFLOW_TRACKING_URI`, confirming MLflow is indeed connected.

Great, we've launched a notebook server that's connected to MLflow! Now let's upload some example notebooks to this server to see MLflow in practice.

Run MLflow examples

To run MLflow examples on your newly created notebook server, click on the source control icon in the leftmost navigation bar.

From the menu, choose the `Clone a Repository` option.

Now insert this repository address `https://github.com/canonical/kubeflow-examples.git`.

This will clone a whole `kubeflow-examples` repository onto the notebook server. The cloned repository will be a folder on the server, with the same name as the remote repository. Go inside the folder and after that, choose the `mlflow-v2-examples` sub-folder.

There you will find two notebooks:

- `notebook-example.ipynb`: demonstrates how to talk to MLflow from inside a notebook. The example uses a simple classifier which is stored in the MLflow registry.
- `pipeline-example.ipynb`: demonstrates how to talk to MLflow from a Kubeflow pipeline. The example creates and executes a three-step Kubeflow pipeline with the last step writing a model object to the MLflow registry.

Go ahead, try those notebooks out for yourself! You can run them cell by cell using the run button, or all at once using the double chevron `>>`.

Note: If you get an error in the Notebooks related to `sklearn`, try replacing `sklearn` with `scikit-learn`. See here for more details.

2.2 How-to guides

These guides provide practical instructions for specific tasks related to deploying, managing and using MLflow.

2.2.1 Preparation

Create an MLOps-ready Charmed Kubernetes cluster

This how-to guide will show you how to create a Charmed Kubernetes (CK8s) cluster with an appropriate configuration for deploying an MLOps platforms such as Kubeflow or MLflow.

Prerequisites

- A local machine with Ubuntu 22.04 or later.
- An AWS account (How to create an AWS account).

Install and set up AWS CLI

First, install the AWS CLI on your local machine, and then set it up. You can use any of the authentication methods available for the AWS CLI. For example, you can use IAM user credentials.

Install other tools

To install some helpful tools, run this command:

```
sudo snap install juju --classic --channel=2.9/stable
for snap in juju-wait kubectl jq; \
  do sudo snap install $snap --classic; \
done
```

This installs the following:

- `juju`: Needed to deploy and manage the CK8s cluster.
- `juju-wait`: CLI tool used for waiting during Juju deployments.
- `kubectl`: Kubernetes client used to communicate with a Kubernetes cluster.
- `jq`: A lightweight and versatile command-line tool for parsing and manipulating JSON data.

Setup Juju with AWS

Set up Juju to communicate with AWS.

```
juju add-credential aws
```

You will be prompted for information related to your AWS account that you provided while setting up the AWS CLI (e.g., access key, secret access key).

Create Juju controller

Bootstrap a Juju controller that will be responsible for deploying cluster applications.

```
juju bootstrap aws kf-controller
```

Deploy Charmed Kubernetes 1.24

Clone the Charmed Kubernetes bundle repository, and update CPU, disk, and memory constraints to meet Kubeflow requirements.

```
git clone https://github.com/charmed-kubernetes/bundle
sed -i '/^ *charm: kubernetes-worker/,/^ *[^:]*:/s/constraints: cores=2 mem=8G root-
↪disk=16G/constraints: cores=8 mem=32G root-disk=200G/' ./bundle/releases/1.24/bundle.
↪yaml
```

Deploy the Charmed Kubernetes bundle on AWS with the storage overlay. This overlay enables you to create Kubernetes volumes backed by AWS EBS.

```
juju deploy ./bundle/releases/1.24/bundle.yaml \
  --overlay ./bundle/overlays/aws-storage-overlay.yaml --trust
```

Wait until all components are ready.

```
juju-wait -m default -t 3600
```

Retrieve the Kubernetes configuration from the control plane leader unit.

```
mkdir ~/.kube
juju ssh kubernetes-control-plane/leader -- cat config > ~/.kube/config
```

Now you can use `kubectl` to talk to your newly created Charmed Kubernetes cluster.

2.2.2 Deployment

Deploy Charmed MLflow to Charmed Kubernetes on AWS

Component	Version
MLflow	2

This guide shows how to connect Juju to an existing Charmed Kubernetes (CK8s) cluster and deploy the MLflow bundle on top of it.

Prerequisites

We assume that you have access to a CK8s cluster using `kubectl`. If you don't have a cluster set up, you can follow this guide: *Create CK8s on AWS*.

Install Juju

Install Juju:

```
sudo snap install juju --classic --channel=2.9/stable
```

Connect Juju to Charmed Kubernetes cluster

Configure Juju to communicate with the CK8s cluster by creating a controller:

```
juju add-k8s charmed-k8s-aws --controller $(juju switch | cut -d: -f1) \
 --storage=cdk-ebs
```

Create a model. The model name is up to you. However, if you plan to connect MLflow with Kubeflow you must use `kubeflow` as the model name.

```
juju add-model kubeflow charmed-k8s-aws
```

Deploy MLflow bundle

Deploy the MLflow bundle:

```
juju deploy mlflow --channel=2.1/stable --trust
```

Wait until the deployments are active:

```
juju-wait -m kubeflow -t 2700
```

Connect to MLflow dashboard

By default, the MLflow UI is exposed as a NodePort Kubernetes service, accessible at each node's IP address. MLflow runs on port 31380 by default. AWS nodes are EC2 instances. To connect to an instance, it must be configured to allow traffic to this port.

You can connect to any EC2 instance in the cluster. List all available nodes in your Kubernetes cluster and choose any `EXTERNAL-IP` that you will use to access the MLflow UI:

```
kubectl get nodes -o wide
```

In your AWS account find the EC2 instance with that particular `EXTERNAL-IP` and enable access to the port 31380 in the inbound rules of the security group. To see how, consult AWS docs.

Open a web browser and visit `<nodes-ip-address>:31380` to access the MLflow UI.

Deploy Charmed MLflow to EKS

Component	Version
MLflow	2

This guide shows how to deploy Charmed MLflow on AWS Elastic Kubernetes Service (EKS). In this guide, we will create an AWS EKS cluster, connect Juju to it, and deploy the MLflow bundle.

Prerequisites:

We assume the following:

- Your machine runs Ubuntu 22.04 or later
- You have an AWS account (How to create an AWS account)

Create EKS cluster

See the EKS creation guide for how to do that.

Setup Juju

Set up your local `juju` to talk to the remote Kubernetes (K8s) cloud. First, install `juju`:

```
sudo snap install juju --classic
```

Connect Juju to Kubernetes:

```
juju add-k8s kubeflow
```

Note: You must choose the name `kubeflow` if you plan to connect MLflow to Kubeflow. Otherwise you can choose any name.

Create a controller:

```
juju bootstrap --no-gui kubeflow kubeflow-controller
```

Note: You can use whatever controller name you like here, we chose `kubeflow-controller`.

Add a Juju model:

```
juju add-model kubeflow
```

Note: You must choose the name `kubeflow` if you plan to connect MLflow to Kubeflow. Otherwise you can choose any name.

Deploy MLflow bundle

Deploy the MLflow bundle with the following command:

```
juju deploy mlflow --channel=2.1/stable --trust
```

Wait until all charms are in the active state. You can check the state of the charms with the command:

```
juju status --watch 5s --relations
```

Deploy Charmed MLflow and Kubeflow to EKS

Component	Version
MLflow	2

This guide shows how to deploy Charmed MLflow alongside Kubeflow on AWS Elastic Kubernetes Service (EKS). In this guide, we will create an AWS EKS cluster, connect Juju to it, deploy the MLflow and Kubeflow bundles, and relate them to each other.

Prerequisites

We assume the following:

- Your machine runs Ubuntu 22.04 or later
- You have an AWS account (How to create an AWS account)

Deploy EKS cluster

See our EKS creation guide for a complete guide on how to do this. **Do not forget** to edit the `instanceType` field under `managedNodeGroups[0].instanceType` from `t2.2xlarge` to `t3.2xlarge`, as instructed in the guide, since worker nodes of type `t3.2xlarge` are required for deploying both MLflow and Kubeflow.

Setup Juju

Set up your local `juju` to talk to the remote Kubernetes cloud. First, install Juju with:

```
sudo snap install juju --classic
```

Connect it to Kubernetes:

```
juju add-k8s kubeflow
```

Create the controller:

```
juju bootstrap --no-gui kubeflow kubeflow-controller
```

Note: we chose the name `kubeflow-controller`, but you can choose any other name.

Add a Juju model:

```
juju add-model kubeflow
```

Deploy MLflow bundle

Deploy the MLflow bundle with the following command:

```
juju deploy mlflow --channel=2.1/stable --trust
```

Wait until all charms are in the active state. You can check the state of the charms with the command:

```
juju status --watch 5s --relations
```

Deploy Kubeflow bundle

Deploy the Kubeflow bundle with the following command:

```
juju deploy kubeflow --channel=1.7/stable --trust
```

Wait until all charms are in the active state. You can check the state of the charms with the command:

```
juju status --watch 5s --relations
```

Relate MLflow to Kubeflow

The resource dispatcher is used to connect MLflow with Kubeflow. In particular, it is responsible for configuring MLflow related Kubernetes objects for Kubeflow user namespaces. Deploy the resource dispatcher to the cluster with the command:

```
juju deploy resource-dispatcher --channel 1.0/stable --trust
```

Relate the resource dispatcher to MLflow with the following commands:

```
juju relate mlflow-server:secrets resource-dispatcher:secrets
juju relate mlflow-server:pod-defaults resource-dispatcher:pod-defaults
```

Wait until all charms are in the active state. You can check the state of the charms with the command:

```
juju status --watch 5s --relations
```

Configure Kubeflow dashboard

Get the hostname from the `istio-ingressgateway-workload` Kubernetes load balancer service:

```
export INGRESS_HOST=$(kubectl get svc -n kubeflow istio-ingressgateway-workload -o␣
↪jsonpath='{.status.loadBalancer.ingress[0].hostname}')
```

Then, configure OIDC and DEX with the `INGRESS_HOST` we just retrieved, and also a username and password of your choosing:

```
juju config dex-auth public-url="http://${INGRESS_HOST}"
juju config oidc-gatekeeper public-url="http://${INGRESS_HOST}"
juju config dex-auth static-password=user123
juju config dex-auth static-username=user123@email.com
```

Wait until all charms are in the active state. You can check the state of the charms with the command:

```
juju status --watch 5s --relations
```

Now you can access the Kubeflow dashboard at the value from `INGRESS_HOST` in your browser.

2.2.3 Integration

Integrate MLflow with the Canonical Observability Stack (COS)

This guide shows how to integrate MLflow with the Canonical Observability Stack (COS).

Prerequisites

This guide assumes:

1. You have deployed the COS stack in the `cos` model. For steps on how to do this, see the MicroK8s tutorial.
2. You have deployed the MLflow bundle in the `kubeflow` model. For steps on how to do this, see *Get Started with Charmed MLflow*.

Deploy Grafana Agent

Deploy the Grafana Agent to your `kubeflow` model alongside the MLflow bundle. Run the following command:

```
juju deploy grafana-agent-k8s --channel=edge --trust
```

Relate MLflow Server Prometheus Metrics to Grafana Agent

Establish the relationship between the MLflow Server Prometheus metrics and the Grafana Agent. Use the following command:

```
juju add-relation mlflow-server:metrics-endpoint grafana-agent-k8s:metrics-endpoint
```

Relate Grafana Agent to Prometheus in the COS Model

Next, relate the Grafana Agent to Prometheus in the `cos` model. Execute the following command:

```
juju add-relation grafana-agent-k8s admin/cos.prometheus-receive-remote-write
```

Relate MLflow Server in the Kubeflow Model to Grafana Charm in the COS Model

Establish the relationship between the MLflow Server in the `kubeflow` model and the Grafana charm in the `cos` model. Run the following command:

```
juju add-relation mlflow-server admin/cos.grafana-dashboards
```

Obtain the Grafana Dashboard Admin Password

Switch the model to `cos` and retrieve the Grafana dashboard admin password. Execute the following commands:

```
juju switch cos
juju run-action grafana/0 get-admin-password --wait
```

Obtain the Grafana Dashboard URL

To access the Grafana dashboard, you need the URL. Run the following command to get the URLs for the COS endpoints:

```
juju show-unit catalogue/0 | grep url
```

You will see a list of endpoints similar to the following:

```
url: http://10.43.8.34:80/cos-catalogue
url: http://10.43.8.34/cos-grafana
url: http://10.43.8.34:80/cos-prometheus-0
url: http://10.43.8.34:80/cos-alertmanager
```

Choose the `cos-grafana` URL and access it in your browser.

Login to Grafana

Login to Grafana with the password obtained from the previous section. The username is `admin`.

Access the dashboard in the UI

Go to the left sidebar and choose the MLflow Dashboards from the list. From the General dashboards folder choose the `MLflow metrics Dashboard`. When accessing the dashboard for the first time, choose some reasonable time range from the top right dropdown.

Integrate Charmed MLflow with Charmed Kubeflow on Charmed Kubernetes

Component	Version
MLflow	2

In this guide, we will guide you through the process of integrating Charmed MLflow with Charmed Kubeflow on Charmed Kubernetes.

Prerequisites

We assume that:

- You have access to a Charmed Kubernetes cluster using `kubectl`. If you don't have a cluster set up, you can follow the *creation guide* to deploy one on AWS.
- You have deployed the Charmed Kubeflow bundle. If you don't have it, here is a guide on how to do it.
- You have deployed the Charmed MLflow bundle. To see how, follow our *deployment guide*.

Deploy resource dispatcher

Deploy the resource dispatcher:

```
juju deploy resource-dispatcher --channel 1.0/stable --trust
```

Relate Resource dispatcher to MLflow

Relate the Resource dispatcher to MLflow:

```
juju relate mlflow-server:secrets resource-dispatcher:secrets
juju relate mlflow-server:pod-defaults resource-dispatcher:pod-defaults
```

Integrate MLflow with Kubeflow notebook

Please refer to this doc: *Getting Started with Charmed MLflow and Kubeflow*.

Integrate MLflow with Jupyter Notebooks

To run Jupyter Notebooks in Charmed MLflow, JupyterLab must be deployed and a number of configurations made.

Prerequisites

- You are deploying Jupyter Notebook and MLflow on a workstation running Ubuntu 20.04 (focal) or later.
- Your workstation has at least 4 cores, 32GB RAM, and 32GB of disk space available.
- Your workstation is connected to the internet for downloading the required snaps and charms.

Deploy MLflow

Follow the steps in this tutorial to deploy MLflow on your VM: *Get Started with Charmed MLflow*. Confirm that you can now access the MLflow UI on `http://localhost:31380.`

Deploy JupyterLab

Install JupyterLab:

```
pip install jupyterlab
```

Run JupyterLab:

```
jupyter lab
```

Access MLflow UI

Access the MLflow UI:

```
mlflow ui
```

Configure MinIO and MLflow

Before you can run your first experiment, there are a couple of things to adjust — the MLflow URI and the MinIO URI. To do this:

1. Open a new terminal window connected to the instance you have been using.
2. Enter the following command to check the status:

   ```
   juju status
   ```

3. Now, go back to the Notebook and update the MLflow URL and MinIO URL as needed.
4. Once those are updated, there is one last step you need to do. Return to the terminal and run:

   ```
   juju run-action mlflow-server/0 get-minio-credentials - wait
   ```

   This will display the secret-key and secret-access-key. Be sure to update them in the Notebook as well.

Now, you are ready to run your first experiment. After finalising the run, you can go to the MLflow UI and view the experiment results.

2.2.4 Upgrading

Migrate Charmed MLflow Version 1 to Version 2

This guide shows how to migrate Charmed MLflow version 1 to version 2. This guide assumes you are running the old Charmed MLflow stack version 1, which runs with MariaDB. With MLflow version 2, we only support the MySQL integration. This guide outlines how to move data from MariaDB to MySQL and how to migrate data from version 1 to version 2.1.1. Data from the object store doesn't need to be migrated.

Prerequisites

This guide assumes the following:

1. You have deployed MLflow version 1 with MariaDB, MLflow server version 1.x, and MinIO.
2. You have CLI access to the machine where the Juju controller is deployed (all commands will be executed from there).

MariaDB Backup

Install the `mysqldump` command:

```
sudo apt update
sudo apt install mysql-client
```

Backup the MariaDB database with the following command:

```
mysqldump --host=<mariadb-charm-ip-address> --user=root --password=root --column-
↪statistics=0 --databases database > mlflow-db.sql
```

Deploy MySQL Charm

Deploy the MySQL charm, which is needed for MLflow v2:

```
juju deploy mysql-k8s --channel 8.0/beta --series jammy --trust
```

Note: For MLflow version `v.2.1`, we deploy the 8.0/beta version of the charm. You may deploy a more up to date version in your case.

Please wait until the charm goes to active in `juju status`. Then run the following command to get the password for MySQL:

```
juju run-action mysql-k8s/0 get-password --wait
```

Adjust the Database Backup

Rename the database from `database` (used in MariaDB) to `mlflow` (used in MySQL):

```
sed 's/`database`/`mlflow`/g' mlflow-db.sql > mlflow-db-updated.sql
```

Rename any duplicate constraints as MySQL does not allow that. In practice, the only duplicate constraint we've encountered is `CONSTRAINT_1`. It has two occurrences. The first occurrence can be renamed to `CONSTRAINT-1`, for example:

```
sed -i '0,/`CONSTRAINT_1`/s//`CONSTRAINT-1`/' mlflow-db-updated.sql
```

You can do all the above modifications in the text editor of your choice if you prefer.

Move Database to MySQL

Install the MySQL CLI tool:

```
sudo apt update
sudo apt-get install mysql-shell
```

Connect to the MySQL charm:

```
mysql --user=root --host=<mysql-unit-ip> -p
# you will be prompted for password
```

Create the MySQL database called `mlflow`:

```
CREATE DATABASE mlflow;
```

Leave the client with `ctrl + D`.

Move the updated database dump file to MySQL:

```
mysql -u root -p <mysql_password> mlflow <mlflow-db-updated.sql
```

Migrate MySQL Database

Install the MLflow Python client version 2.1.1:

```
pip install mlflow==2.1.1
```

Run the migration script against the MySQL `mlflow` database:

```
mlflow db upgrade mysql+pymysql://root:<mysql-password>@<mysql-ip>/mlflow
```

Update MLflow Server

Remove relations from the old MLflow server:

```
juju remove-relation mlflow-db:mysql mlflow-server:db
juju remove-relation minio mlflow-server
```

Update the MLflow server:

```
juju refresh mlflow-server --channel 2.1/edge
```

Create relations with MinIO and MySQL:

```
juju relate mysql-k8s mlflow-server
juju relate minio mlflow-server
```

2.2.5 Managing

Backup MLflow data

This how-to guide will show you how to make a backup of all of MLflow's data, that live in MySQL and S3.

Pre-requisites

1. Access to a S3 storage - only AWS S3 and S3 RadosGW are supported
2. Admin access to the Kubernetes cluster where Charmed MLflow is deployed
3. Juju admin access to the *mlflow* model
4. rclone installed and configured to connect to the S3 storage from 1
5. *s3-integrator* deployed and configured
    1. https://charmhub.io/mysql-k8s/docs/h-configure-s3-aws
    2. https://charmhub.io/mysql-k8s/docs/h-configure-s3-radosgw
6. yq binary

Note: This S3 storage will be used for storing all backup data from MLflow.

Throughout the following guide we'll use the following ENV vars in the commands

```
S3_BUCKET=backup-bucket-2024
RCLONE_S3_REMOTE=remote-s3
RCLONE_MINIO_MLFLOW_REMOTE=minio-mlflow
RCLONE_BWIDTH_LIMIT=20M
```

Through the guide we'll be using rclone to both get files from MinIO and push the backup to an S3 endpoint. An example configuration looks like this:

```
[minio-mlflow]
type = s3
provider = Minio
access_key_id = minio
secret_access_key = ...
endpoint = http://localhost:9000
acl = private
```

Note: You can check where this configuration file is located with *rclone config file*

Backup MLflow DBs

1. Scale up *mlflow-mysql*

Warning: In a single node setup, the *Primary* database will become unavailable during the backup. It is recommended to have a multinode setup before backing up the data.

```
juju scale-application mlflow-mysql 2
```

2. Create a backup of DB

To see how to make a backup of MLflow's MySQL database, follow this guide on how to Create a backup.

Note: Please replace *mysql-k8s* with the name of the database you intend to create a backup for in the commands form that guide. E.g. *mlflow-mysql* instead of *mysql-k8s*.

Backup *mlflow* MinIO bucket

Note: The name of the MLflow MinIO bucket defaults to *mlflow*, the bucket name can be verified with *juju config mlflow default_artifact_root*.

1. Configure *rclone* for MinIO

You can use this sample *rclone* configuration as a reference:

```
[minio-mlflow]
type = s3
provider = Minio
access_key_id = minio
secret_access_key = ...
```

(continues on next page)

(continued from previous page)

```
endpoint = http://localhost:9000
acl = private
```

Note that the machine will need to use a URL to access MinIO. In this case we'll use kubectl to do a port forward:

```
kubectl port-forward -n kubeflow svc/mlflow-minio 9000:9000
```

Note: In order to find the *secret-access-key* for MinIO you'll need to run the following command:

```
juju show-unit mlflow-server/0 \
    | yq '.mlflow-server/0.relation-info.[] | select (.related-endpoint == "object-
↪storage") | .application-data.data' \
    | yq '.secret-key'
```

In the future the MinIO Charm will be extended so that it can send it's data directly to the S3 endpoint.

2. Sync buckets from MinIO to S3

```
rclone --size-only sync \
  --bwlimit $RCLONE_BWIDTH_LIMIT \
  $RCLONE_MINIO_MLFLOW_REMOTE:mlflow \
  $RCLONE_S3_REMOTE:$S3_BUCKET/mlflow
```

Next Steps

- Want to restore your Charmed MLflow from a backup? See *Restore MLflow data*

Restore MLflow data

The following instructions will allow you to restore the Charmed MLflow control plane data from a compatible S3 storage.

Pre-requisites

1. Access to a S3 storage - only AWS S3 and S3 RadosGW are supported
2. Admin access to the Kubernetes cluster where Charmed MLflow is deployed
3. Juju admin access to the *mlflow* model
4. rclone installed and configured to connect to the S3 storage from 1
5. *s3-integrator* deployed and configured
    1. https://charmhub.io/mysql-k8s/docs/h-configure-s3-aws
    2. https://charmhub.io/mysql-k8s/docs/h-configure-s3-radosgw
6. yq binary

Note: This S3 storage will be used for storing all backup data from MLflow.

Throughout the following guide we'll use the following ENV vars in the commands

```
S3_BUCKET=backup-bucket-2024
RCLONE_S3_REMOTE=remote-s3
RCLONE_MINIO_MLFLOW_REMOTE=minio-mlflow
RCLONE_BWIDTH_LIMIT=20M
```

Through the guide we'll be using rclone to both get files from MinIO and push the backup to an S3 endpoint. An example configuration looks like this:

```
[minio-mlflow]
type = s3
provider = Minio
access_key_id = minio
secret_access_key = ...
endpoint = http://localhost:9000
acl = private
```

Note: You can check where this configuration file is located with *rclone config file*

Restore DB from S3

1. Scale up *mlflow-mysql*:

Warning: In a single node setup, the *Primary* database will become unavailable during the backup. It is recommended to have a multinode setup before backing up the data.

```
juju scale-application mlflow-mysql 2
```

2. Restore MySQL

Note: Please replace *mysql-k8s* with the name of the database you intend to create a backup for in the commands form that guide. E.g. *mlflow-mysql* instead of *mysql-k8s*.

Restore *mlflow* MinIO bucket

Note: The name of the MLflow MinIO bucket defaults to *mlflow*, the bucket name can be verified with *juju config mlflow default_artifact_root*.

1. Configure *rclone* for MinIO

You can use this sample *rclone* configuration as a reference:

```
[minio-mlflow]
type = s3
provider = Minio
access_key_id = minio
secret_access_key = ...
endpoint = http://localhost:9000
acl = private
```

Note that the machine will need to use a URL to access MinIO. In this case we'll use kubectl to do a port forward:

```
kubectl port-forward -n kubeflow svc/mlflow-minio 9000:9000
```

Note: In order to find the *secret-access-key* for MinIO you'll need to run the following command:

```
juju show-unit mlflow-server/0 \
    | yq '.mlflow-server/0.relation-info.[] | select (.related-endpoint == "object-
↪storage") | .application-data.data' \
    | yq '.secret-key'
```

In the future the MinIO Charm will be extended so that it can send it's data directly to the S3 endpoint.

2. Sync buckets from S3 to MinIO

```
rclone --size-only sync \
  --bwlimit $RCLONE_BWIDTH_LIMIT \
  $RCLONE_S3_REMOTE:$S3_BUCKET/mlflow \
  $RCLONE_MINIO_MLFLOW_REMOTE:mlflow
```

Next Steps

- Want to create a backup of MLflow's data? See *Backup MLflow data*

2.3 Reference

Coming soon.

2.4 Explanation

2.4.1 Why choose Charmed MLflow?

Are you considering using Charmed MLflow? Wondering what the advantages are of charmed MLflow vs. upstream MLflow?

Knowing the answer to this will help any prospective MLflow users decide whether they want the charmed version.

Simplified deployment

Charmed MLflow offers simplified deployment. Like any charmed product, Charmed MLflow is deployed as a charm bundle using Juju. Deploying an application with Juju is arguably simpler than deploying to a raw Kubernetes cluster.

Security, stability, and maintenance

Charmed MLflow benefits from the following:

- Upgrade guides.
- Automated security scanning: The bundle is scanned periodically.
- Security patching: Charmed MLflow follows Canonical's process and procedure for security patching. Vulnerabilities are prioritised based on severity, the presence of patches in the upstream project, and the risk of exploitation.
- Maintained images: All Charmed MLflow images are actively maintained.
- Comprehensive testing: Charmed MLflow is thoroughly tested on multiple platforms, including public cloud, local workstations, on-premises deployments, and various CNCF-compliant Kubernetes distributions.

Integration

Charmed MLflow provides integration capabilities, including:

- Customised Prometheus exporter metrics
- Customised MLflow dashboard for Grafana
- Canonical Observability Stack
- Charmed Kubeflow: including the ability use the MLflow registry directly from Kubeflow pipelines and notebooks

Enterprise Offering

Charmed MLflow offers an enterprise offering from Canonical, which includes:

- 24/7 support for deployment, up-time monitoring, and security patching with Charmed MLflow.
- Hardening features and compliance with standards like Federal Risk and Authorisation Management Program, Health Insurance Portability and Accountability Act, and Payment Card Industry Digital Signature Standard, making it suitable for enterprises running AI/ML workloads in highly regulated environments.
- Timely patches for common vulnerabilities and exposures (CVEs).
- A ten-year security maintenance commitment.
- Hybrid cloud and multi-cloud support.
- Bug fixing.
- Optionally managed services, allowing your team to focus on development rather than operations.
- Consultancy services to assess the best tools and architecture for your specific use cases.
- A simple per-node subscription model.

2.5 Contribute to MLflow

2.5.1 Overview

This document outlines the processes and practices recommended for contributing enhancements to this operator.

2.5.2 Talk to us First

Before developing enhancements to this charm, you should open an issue explaining your use case. If you would like to chat with us about your use-cases or proposed implementation, you can reach us at MLOps Mattermost public channel or on Discourse.

2.5.3 Pull Requests

Please help us out in ensuring easy to review branches by rebasing your pull request branch onto the `main` branch. This also avoids merge commits and creates a linear Git commit history.

All pull requests require review before being merged. Code review typically examines:

- code quality
- test coverage
- user experience for Juju administrators of this charm.

2.5.4 Recommended Knowledge

Familiarising yourself with the Charmed Operator Framework library will help you a lot when working on new features or bug fixes.

2.5.5 Developing

You can use the environments created by `tox` for development:

```
tox --notest -e unit
source .tox/unit/bin/activate
```

Testing

```
tox -e lint          # code style
tox -e unit          # unit tests
tox -e integration   # integration tests
tox                  # runs 'lint' and 'unit' environments
```

2.5.6 Build Charm

Build the charm in this git repository using:

```
charmcraft pack
```

Deploy

```
# Create a model
juju add-model dev
# Enable DEBUG logging
juju model-config logging-config="<root>=INFO;unit=DEBUG"
# Deploy the charm
juju deploy ./mlflow-server_ubuntu-20.04-amd64.charm \
    --resource oci-image=$(yq '.resources."oci-image"."upstream-source"' metadata.yaml)
```

2.5.7 Updating the charm for new versions of the workload

To upgrade the source and resources of this charm, you must:

1. Bump the `oci-image` in `metadata.yaml`
2. Update the charm source for any changes, such as:
   - YAML manifests in `src/` and/or any Kubernetes resource in `pod_spec`
   - New or changed configurations passed to pebble workloads or through `pod.set_spec`
3. Ensure integration and unit tests are passing; fix/adapt them otherwise

2.5.8 Canonical Contributor Agreement

Canonical welcomes contributions to this charm. Please check out our contributor agreement if you're interested in contributing.