Déploiement de l'application

NetApp Solutions

NetApp
May 03, 2024

Sommaire

Déploiement de l’application

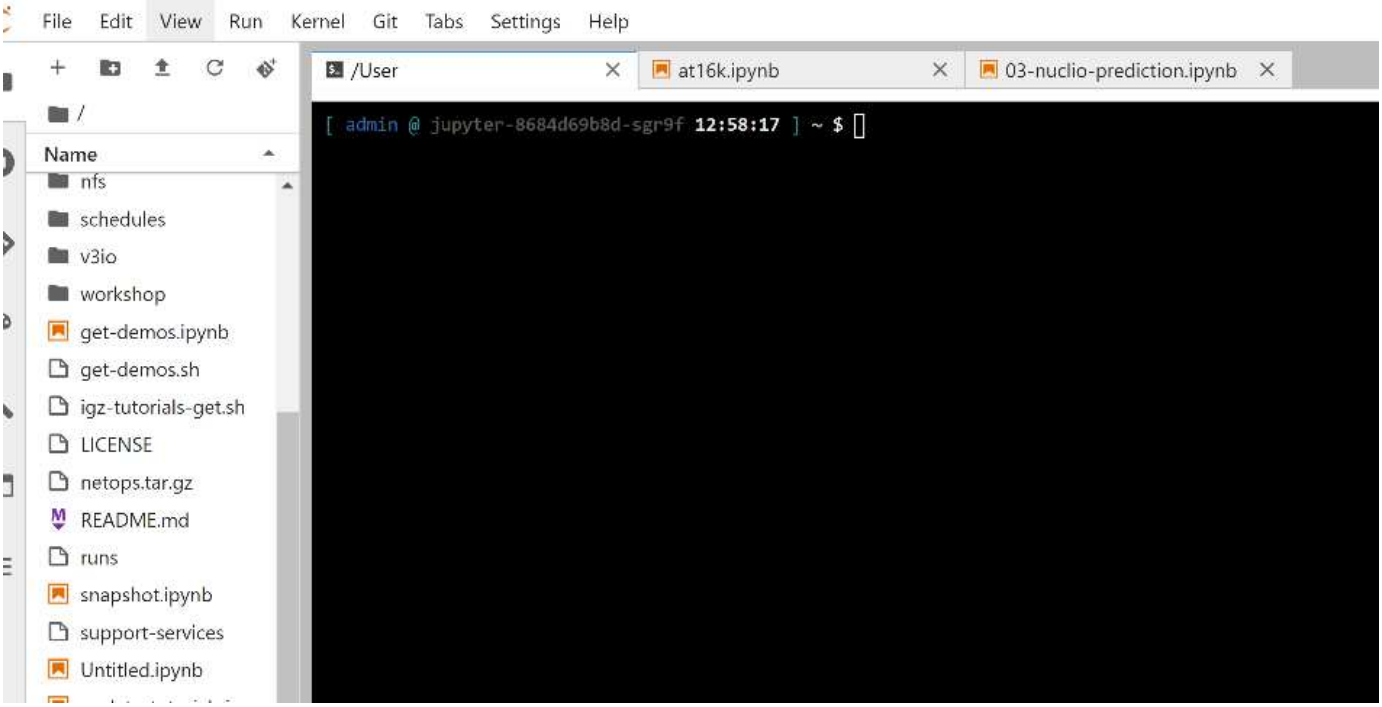
Les sections suivantes décrivent comment installer et déployer l’application.

Obtenir le code à partir de GitHub

Maintenant que le volume NetApp Cloud Volume ou NetApp Trident est disponible pour le cluster Iguazio et l’environnement de développement, vous pouvez passer en revue l’application.

Les utilisateurs ont leur propre espace de travail (répertoire). Sur chaque ordinateur portable, le chemin d’accès au répertoire de l’utilisateur est `/User`. La plateforme Iguazio gère le répertoire. Si vous suivez les instructions ci-dessus, le volume NetApp Cloud est disponible dans le `/netapp` répertoire.

Obtenir le code à partir de GitHub à l’aide d’un terminal Jupyter.



À l’invite du terminal Jupyter, clonez le projet.

```
cd /User
git clone .
```

Vous devriez maintenant voir le `netops- netapp` Dans l’arborescence des fichiers de l’espace de travail Jupyter.

Configurer l’environnement de travail

Copiez le `Notebook` `set_env-Example.ipynb` comme `set_env.ipynb`. Ouvrez et

modifiez `set_env.ipynb`. Ce bloc-notes définit des variables pour les informations d'identification, l'emplacement des fichiers et les pilotes d'exécution.

Si vous suivez les instructions ci-dessus, les seules modifications à apporter sont les suivantes :

1. Pour bénéficier de cette valeur, nous vous leguazio : `docker_registry`

   Exemple : `docker-registry.default-tenant.app.clusterq.iguaziodev.com:80`

2. Changer `admin` Pour votre nom d'utilisateur Iguazio :

   `IGZ_CONTAINER_PATH = '/users/admin'`

   Détails de connexion du système ONTAP : Incluez le nom du volume généré lors de l'installation de Trident. Le paramètre suivant est défini pour un cluster ONTAP sur site :

   ```
   ontapClusterMgmtHostname = '0.0.0.0'
   ontapClusterAdminUsername = 'USER'
   ontapClusterAdminPassword = 'PASSWORD'
   sourceVolumeName = 'SOURCE VOLUME'
   ```

   Le paramètre suivant est pour Cloud Volumes ONTAP :

```
MANAGER=ontapClusterMgmtHostname
svm='svm'
email='email'
password=ontapClusterAdminPassword
weid="weid"
volume=sourceVolumeName
```

Créer des images Docker de base

Tout ce dont vous avez besoin pour créer un pipeline DE ML est inclus dans la plateforme Iguazio. Le développeur peut définir les spécifications des images Docker nécessaires à l'exécution du pipeline et à l'exécution de la création d'images à partir de Jupyter Notebook. Ouvrez le bloc-notes `create-images.ipynb` Et exécuter toutes les cellules.

Ce bloc-notes crée deux images que nous utilisons dans le pipeline.

- `iguazio/netapp`. Utilisé pour traiter les tâches DE ML.

Create image for training pipeline

```
[4]: fn.build_config(image=docker_registry+'/iguazio/netapp', commands=['pip install \
     v3io_frames fsspec>=0.3.3 PyYAML==5.1.2 pyarrow==0.15.1 pandas==0.25.3 matplotlib seaborn yellowb
     fn.deploy()
```

- `netapp/pipeline`. Contient des utilitaires pour la gestion des copies NetApp Snapshot.

Create image for Ontap utilitites

```
[9]: fn.build_config(image=docker_registry + '/netapp/pipeline:latest',commands=['apt -y update','pip install v3io_frames netapp_ontap'
     fn.deploy()
```

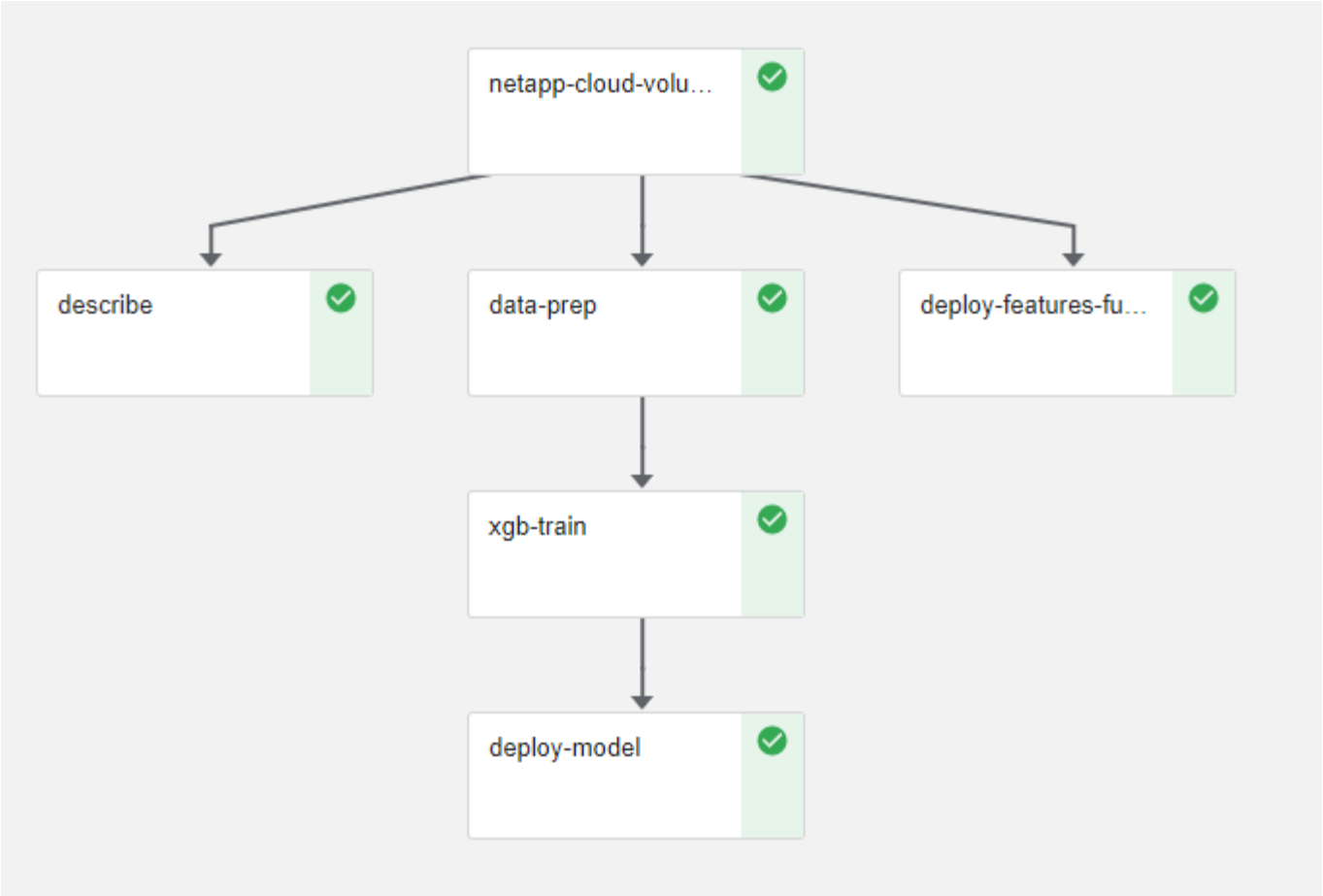
Examinez les ordinateurs portables Jupyter individuels

Le tableau suivant répertorie les bibliothèques et les structures que nous avons utilisées pour créer cette tâche. Tous ces composants ont été pleinement intégrés aux contrôles d'accès et de sécurité basés sur les rôles d'Iguazio.

Bibliothèques/Framework	Description
MLRun	Géré par Iguazio pour l'assemblage, l'exécution et le contrôle d'un pipeline DE ML/IA.
Nucio	Un cadre de fonctions sans serveur intégré à Iguazio. Également disponible en tant que projet open source géré par Iguazio.
Kubeflow	Framework basé sur Kubernetes pour déployer le pipeline. Il s'agit également d'un projet open-source auquel Iguazio contribue. Il est intégré à Iguazio pour renforcer la sécurité et l'intégration avec le reste de l'infrastructure.
Docker	Un registre Docker fonctionne à la demande sur la plateforme Iguazio. Vous pouvez également le modifier pour vous connecter à votre registre.
NetApp Cloud volumes	Les systèmes Cloud volumes exécutés sur AWS nous permettent d'accéder à d'importants volumes de données et de créer des copies Snapshot pour les datasets utilisés à des fins d'entraînement.
Trident	Trident est un projet open source géré par NetApp. Il facilite l'intégration aux ressources de stockage et de calcul dans Kubernetes.

Nous avons utilisé plusieurs ordinateurs portables pour construire le pipeline DE ML. Chaque ordinateur portable peut être testé individuellement avant d'être rassemblé dans le pipeline. Nous abordons chaque ordinateur portable individuellement après le processus de déploiement de cette application de démonstration.

Le résultat souhaité est un pipeline qui forme un modèle basé sur une copie Snapshot des données et déploie le modèle pour l'inférence. Un schéma fonctionnel d'un pipeline MLRun terminé est présenté dans l'image suivante.



Déployer la fonction Data Generation

Cette section décrit comment nous avons utilisé les fonctions sans serveur Nucio pour générer des données de périphériques réseau. L'utilisation est adaptée à partir d'un client Iguazio qui a déployé le pipeline et utilisé les services Iguazio pour surveiller et prévoir les défaillances des périphériques réseau.

Nous avons simulé des données provenant de périphériques réseau. Exécution du bloc-notes Jupyter `data-generator.ipynb` Crée une fonction sans serveur qui s'exécute toutes les 10 minutes et génère un fichier parquet avec de nouvelles données. Pour déployer la fonction, exécutez toutes les cellules de ce bloc-notes. Voir la "Site Web de Nucio" pour passer en revue les composants inconnus de cet ordinateur portable.

Une cellule avec le commentaire suivant est ignorée lors de la génération de la fonction. Chaque cellule de l'ordinateur portable est supposée faire partie de la fonction. Importez le module Nucio pour l'activer `%nuclio magic`.

```
# nuclio: ignore
import nuclio
```

Dans la spécification de la fonction, nous avons défini l'environnement dans lequel la fonction s'exécute, la façon dont elle est déclenchée et les ressources qu'elle consomme.

```
spec = nuclio.ConfigSpec(config={"spec.triggers.inference.kind":"cron",

"spec.triggers.inference.attributes.interval" :"10m",
                                "spec.readinessTimeoutSeconds" : 60,
                                "spec.minReplicas" : 1},……
```

Le `init_context` La fonction est appelée par Nucio framework lors de l'initialisation de la fonction.

```
def init_context(context):
    ….
```

Tout code qui ne figure pas dans une fonction est appelé lors de l'initialisation de la fonction. Lorsque vous l'appelez, une fonction de gestionnaire est exécutée. Vous pouvez modifier le nom du gestionnaire et le spécifier dans la spécification de fonction.

```
def handler(context, event):
            …
```

Vous pouvez tester la fonction à partir de l'ordinateur portable avant le déploiement.

```
%%time
# nuclio: ignore
init_context(context)
event = nuclio.Event(body='')
output = handler(context, event)
output
```

La fonction peut être déployée à partir de l'ordinateur portable ou déployée à partir d'un pipeline ci/CD (adaptation de ce code).

```
addr = nuclio.deploy_file(name='generator',project='netops',spec=spec,
tag='v1.1')
```

Ordinateurs portables Pipeline

Ces ordinateurs portables ne sont pas conçus pour être exécutés individuellement pour cette configuration. Il s'agit simplement d'un examen de chaque ordinateur portable. Nous les avons appelés dans le cadre du pipeline. Pour les exécuter individuellement, consultez la documentation MLRun pour les exécuter en tant que travaux Kubernetes.

snap_cv.ipynb

Cet ordinateur portable gère les copies Snapshot Cloud Volume au début du pipeline. Elle transmet le nom du

volume au contexte du pipeline. Cet ordinateur portable appelle un script shell pour gérer la copie Snapshot. Lors de l'exécution dans le pipeline, le contexte d'exécution contient des variables qui aident à localiser tous les fichiers nécessaires à l'exécution. Lors de l'écriture de ce code, le développeur n'a pas à s'inquiéter de l'emplacement du fichier dans le conteneur qui l'exécute. Comme décrit plus loin, cette application est déployée avec toutes ses dépendances, et c'est la définition des paramètres de pipeline qui fournit le contexte d'exécution.

```
command = os.path.join(context.get_param('APP_DIR'),"snap_cv.sh")
```

L'emplacement de copie Snapshot créé est placé dans le contexte MLRun pour être utilisé par étapes dans le pipeline.

```
context.log_result('snapVolumeDetails',snap_path)
```

Les trois ordinateurs portables suivants sont exécutés en parallèle.

data-prep.ipynb

Les metrics brutes doivent être intégrés à des fonctionnalités pour permettre l'entraînement des modèles. Cet ordinateur portable lit les mesures brutes du répertoire Snapshot et écrit les fonctionnalités d'entraînement des modèles dans le volume NetApp.

Lors de l'exécution dans le contexte du pipeline, l'entrée `DATA_DIR` Contient l'emplacement de la copie Snapshot.

```
metrics_table = os.path.join(str(mlruncontext.get_input('DATA_DIR',
os.getenv('DATA_DIR','/netpp'))),
                             mlruncontext.get_param('metrics_table',
os.getenv('metrics_table','netops_metrics_parquet')))
```

description.ipynb

Pour visualiser les mesures entrantes, nous déployons une étape de pipeline qui fournit des tracés et des graphiques disponibles via les UI Kubeflow et MLRun. Chaque exécution a sa propre version de cet outil de visualisation.

```
ax.set_title("features correlation")
plt.savefig(os.path.join(base_path, "plots/corr.png"))
context.log_artifact(PlotArtifact("correlation",  body=plt.gcf()),
local_path="plots/corr.html")
```

deploy-feature-function.ipynb

Nous surveillons en permanence les indicateurs des anomalies pour y détecter des anomalies. Cet ordinateur portable crée une fonction sans serveur qui génère les fonctionnalités qui doivent exécuter la prédiction des mesures entrantes. Ce bloc-notes appelle la création de la fonction. Le code de fonction se trouve dans le

bloc-notes `data- prep.ipynb`. Notez que nous utilisons le même bloc-notes qu'une étape dans le pipeline à cette fin.

formation.ipynb

Une fois les fonctions créées, nous déclenchements l'entraînement du modèle. Cette étape permet d'utiliser le modèle à utiliser pour l'inférence. Nous recueillons également des statistiques pour garder le suivi de chaque exécution (expérience).

Par exemple, la commande suivante saisit le score de précision dans le contexte de cette expérience. Cette valeur est visible dans Kubeflow et MLRun.

```
context.log_result('accuracy',score)
```

déploiement-inférence-fonction.ipynb

La dernière étape du pipeline consiste à déployer le modèle comme une fonction sans serveur pour l'inférence continue. Ce bloc-notes appelle la création de la fonction sans serveur définie dans `nuclio-inference-function.ipynb`.

Examiner et créer le pipeline

La combinaison de l'exécution de tous les ordinateurs portables dans un pipeline permet à la série d'expériences continue de réévaluer la précision du modèle par rapport aux nouvelles mesures. Tout d'abord, ouvrez le `pipeline.ipynb` bloc-notes. Nous vous montrerons comment NetApp et Iguazio simplifient le déploiement de ce pipeline DE ML.

Nous utilisons MLRun pour fournir un contexte et gérer l'allocation des ressources à chaque étape du pipeline. Le service MLRun API s'exécute dans la plateforme Iguazio et constitue le point d'interaction avec les ressources Kubernetes. Chaque développeur ne peut pas demander des ressources directement ; l'API traite les demandes et active les contrôles d'accès.

```
# MLRun API connection definition
mlconf.dbpath = 'http://mlrun-api:8080'
```

Le pipeline peut fonctionner avec NetApp Cloud volumes et les volumes sur site. Cette démonstration a été conçue pour utiliser Cloud volumes, mais vous pouvez voir dans le code l'option d'exécution sur site.

```
# Initialize the NetApp snap fucntion once for all functions in a notebook
if [ NETAPP_CLOUD_VOLUME ]:
    snapfn =
code_to_function('snap',project='NetApp',kind='job',filename="snap_cv.ipyn
b").apply(mount_v3io())
    snap_params = {
    "metrics_table" : metrics_table,
    "NETAPP_MOUNT_PATH" : NETAPP_MOUNT_PATH,
    'MANAGER' : MANAGER,
    'svm' : svm,
    'email': email,
    'password': password ,
    'weid': weid,
    'volume': volume,
    "APP_DIR" : APP_DIR
       }
else:
    snapfn =
code_to_function('snap',project='NetApp',kind='job',filename="snapshot.ipy
nb").apply(mount_v3io())
….
snapfn.spec.image = docker_registry + '/netapp/pipeline:latest'
snapfn.spec.volume_mounts =
[snapfn.spec.volume_mounts[0],netapp_volume_mounts]
      snapfn.spec.volumes = [ snapfn.spec.volumes[0],netapp_volumes]
```

La première action nécessaire pour transformer un bloc-notes Jupyter en étape Kubeflow consiste à transformer le code en une fonction. Une fonction présente toutes les caractéristiques requises pour exécuter cet ordinateur portable. Lorsque vous faites défiler le bloc-notes, vous pouvez voir que nous définissons une fonction pour chaque étape du pipeline.

Partie du bloc-notes	Description
<code_to_function> (partie du module MLRun)	Nom de la fonction : nom du projet. permet d'organiser tous les artefacts du projet. Ceci est visible dans l'interface utilisateur MLRun. Nature. Dans ce cas, une tâche Kubernetes. Il peut s'agir de DASK, mpi, sparkk8s, et plus encore. Voir la documentation MLRun pour plus de détails. Fichier. Nom du bloc-notes. Ce peut également être un emplacement dans Git (HTTP).
image	Nom de l'image Docker utilisée à cette étape. Nous avons créé ceci précédemment avec le bloc-notes create-image.ipynb.
volume_montages et volumes	Informations détaillées sur le montage de NetApp Cloud Volume au moment de l'exécution.

Nous définissons également des paramètres pour les étapes.

```
params={   "FEATURES_TABLE":FEATURES_TABLE,
           "SAVE_TO" : SAVE_TO,
           "metrics_table" : metrics_table,
           'FROM_TSDB': 0,
           'PREDICTIONS_TABLE': PREDICTIONS_TABLE,
           'TRAIN_ON_LAST': '1d',
           'TRAIN_SIZE':0.7,
           'NUMBER_OF_SHARDS' : 4,
           'MODEL_FILENAME' : 'netops.v3.model.pickle',
           'APP_DIR' : APP_DIR,
           'FUNCTION_NAME' : 'netops-inference',
           'PROJECT_NAME' : 'netops',
           'NETAPP_SIM' : NETAPP_SIM,
           'NETAPP_MOUNT_PATH': NETAPP_MOUNT_PATH,
           'NETAPP_PVC_CLAIM' : NETAPP_PVC_CLAIM,
           'IGZ_CONTAINER_PATH' : IGZ_CONTAINER_PATH,
           'IGZ_MOUNT_PATH' : IGZ_MOUNT_PATH
            }
```

Une fois que vous avez défini la fonction pour toutes les étapes, vous pouvez construire le pipeline. Nous utilisons le `kfp` module pour définir cette définition. La différence entre l'utilisation de MLRun et le développement de votre propre bâtiment réside dans la simplification et le raccourcissement du codage.

Les fonctions que nous avons définies sont converties en composants STEP à l'aide du `as_step` Fonction de MLRun.

Définition de l'étape d'instantané

Lancez une fonction Snapshot, sortez et montez le v3io comme source :

```
snap = snapfn.as_step(NewTask(handler='handler',params=snap_params),
name='NetApp_Cloud_Volume_Snapshot',outputs=['snapVolumeDetails','training
_parquet_file']).apply(mount_v3io())
```

Paramètres	Détails
Nouvelle tâche	NewTask est la définition de l'exécution de la fonction.
(Module MLRun)	Gestionnaire. Nom de la fonction Python à appeler. Nous avons utilisé le gestionnaire de noms dans l'ordinateur portable, mais il n'est pas nécessaire. params. Les paramètres que nous avons transmis à l'exécution. Dans notre code, nous utilisons Context.get_param («PARAMÈTRE») pour obtenir les valeurs.

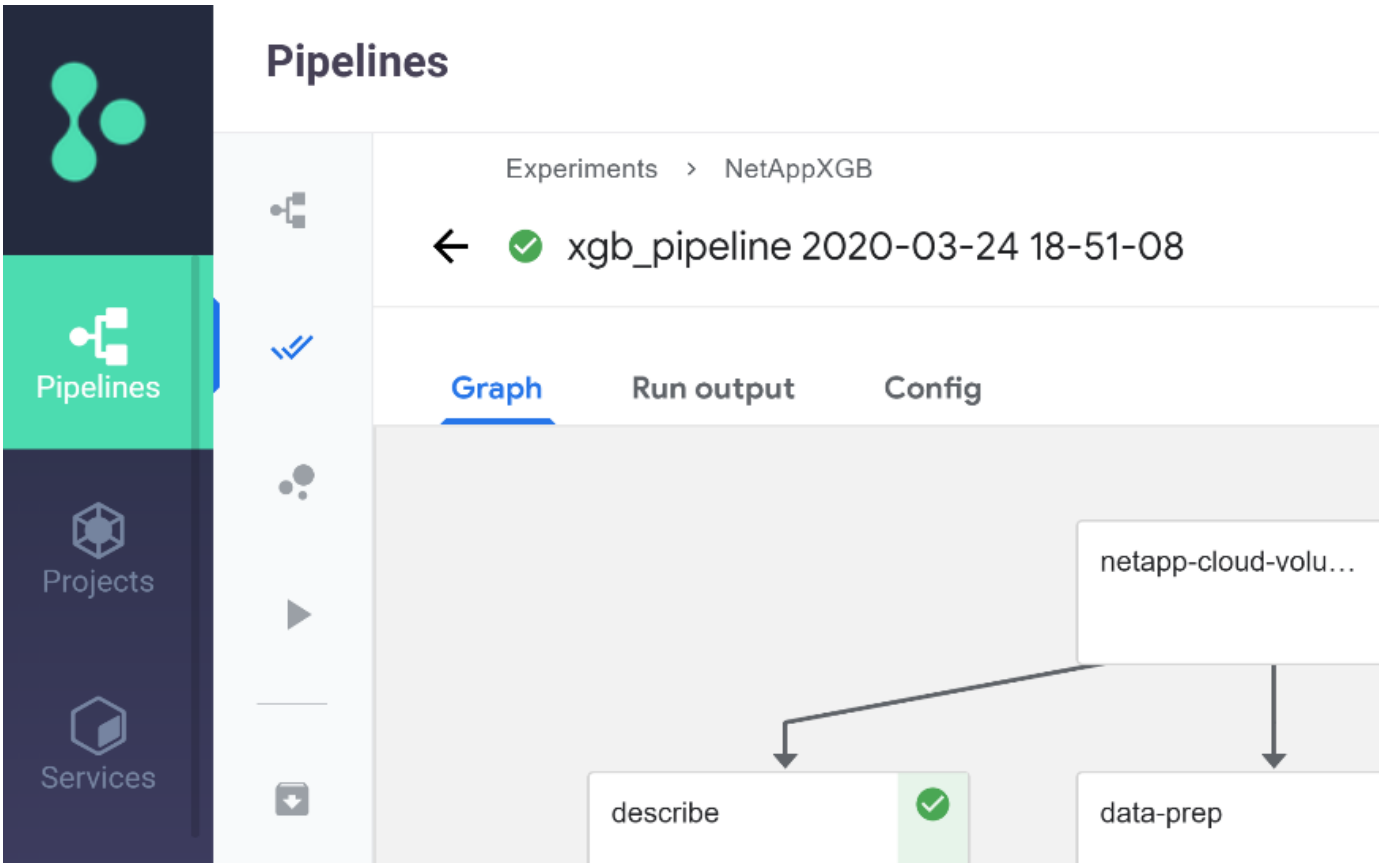
Paramètres	Détails
as_step	Nom. Nom de l'étape du pipeline Kubeflow. sorties. Il s'agit des valeurs que l'étape ajoute au dictionnaire à la fin de l'étude. Regardez le bloc-notes Snap_cv.ipynb. mount_v3io(). Cette opération permet de configurer l'étape pour monter /User pour l'utilisateur exécutant le pipeline.

```
prep = data_prep.as_step(name='data-prep',
handler='handler',params=params,
                          inputs = {'DATA_DIR':
snap.outputs['snapVolumeDetails']} ,

out_path=artifacts_path).apply(mount_v3io()).after(snap)
```

Paramètres	Détails
entrées	Vous pouvez passer à une étape les sorties d'une étape précédente. Dans ce cas, snap.outputs['napVolumeDetails'] correspond au nom de la copie Snapshot que nous avons créée à l'étape d'instantané.
chemin_sortie	Emplacement permettant de placer des artefacts générant à l'aide du module MLRun log_artefacts.

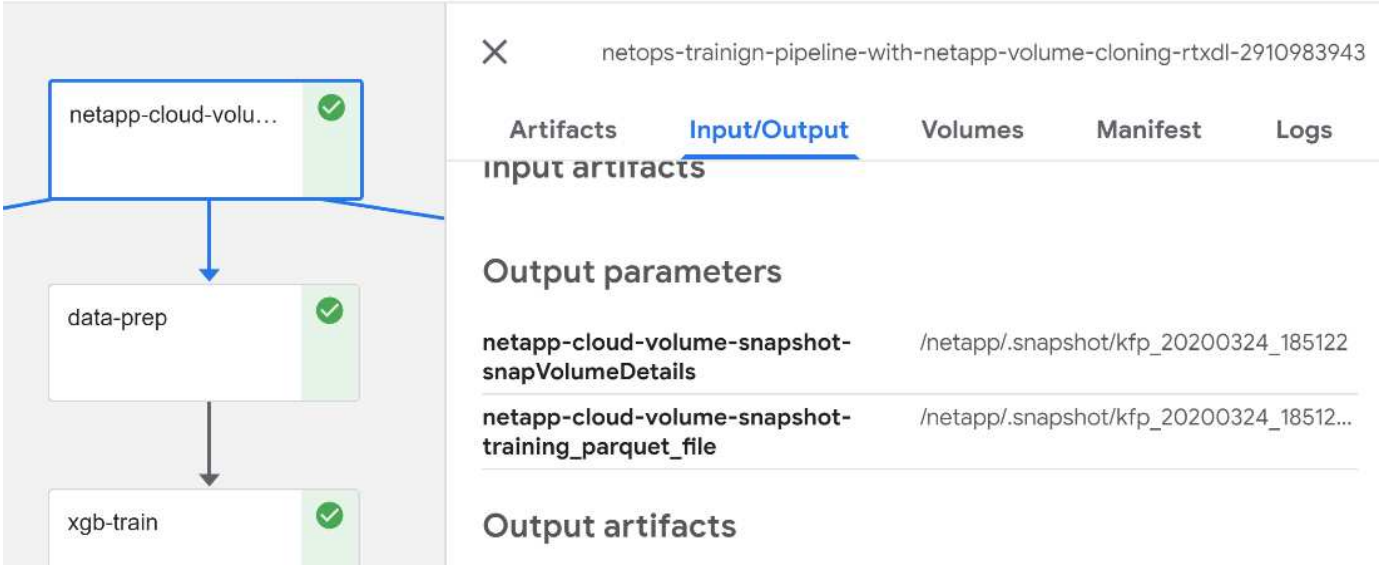
Vous pouvez exécuter `pipeline.ipynb` de haut en bas. Vous pouvez ensuite accéder à l'onglet pipelines à partir du tableau de bord Iguazio pour contrôler la progression comme indiqué dans l'onglet Iguazio Dashboard pipelines.



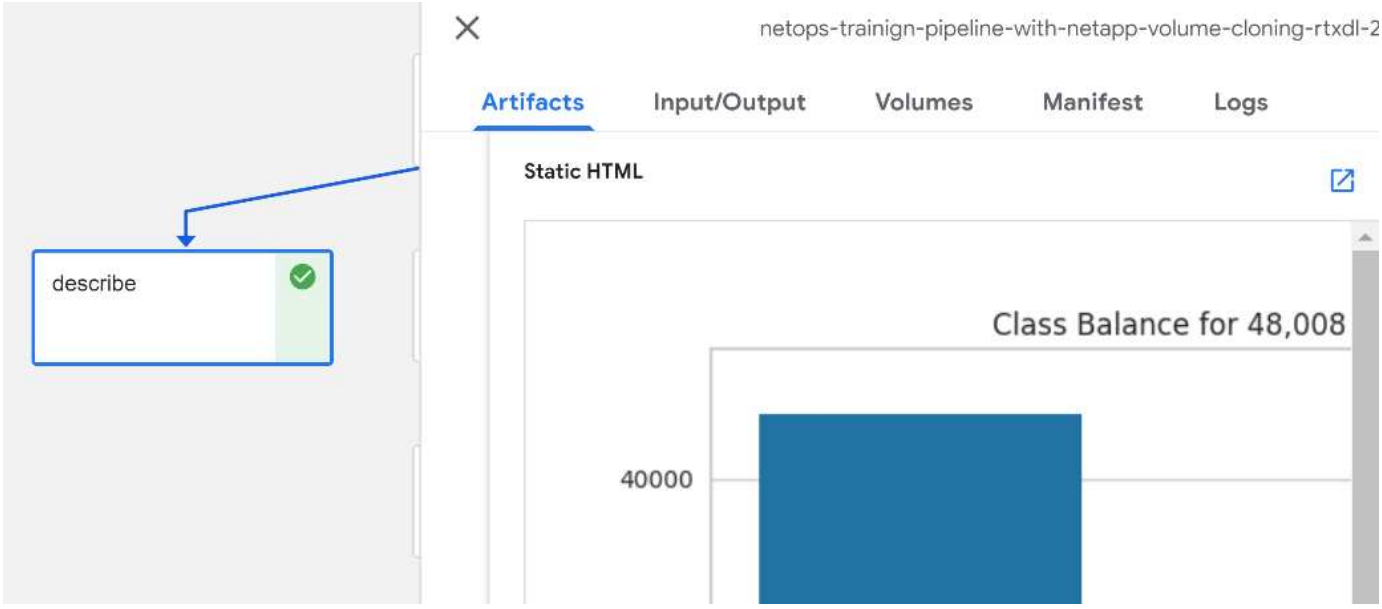
Comme nous avons enregistré la précision de l'étape d'entraînement à chaque série, nous avons un dossier de précision pour chaque expérience, tel qu'indiqué dans le dossier de précision de l'entraînement.

Run name	Status	Duration	Pipeline Version	Recurring ...	Start time	accuracy
xgb_pipeline 2020-03-24 18-51-...	✓	0:08:43	[View pipeline]	-	3/24/2020, 2:51:09 PM	0.985
xgb_pipeline 2020-03-19 13-31-...	✓	0:08:14	[View pipeline]	-	3/19/2020, 9:31:19 AM	0.980
xgb_pipeline 2020-03-18 12-56-...	✓	0:08:11	[View pipeline]	-	3/18/2020, 8:56:08 AM	0.990
xgb_pipeline 2020-03-17 19-49-...	✓	0:08:03	[View pipeline]	-	3/17/2020, 3:49:31 PM	0.985
xgb_pipeline 2020-03-17 18-34-...	✓	0:05:54	[View pipeline]	-	3/17/2020, 2:34:56 PM	0.980
xgb_pipeline 2020-03-17 17-34-...	✓	0:04:48	[View pipeline]	-	3/17/2020, 1:34:16 PM	0.982
xgb_pipeline 2020-03-17 17-01-...	✓	0:05:25	[View pipeline]	-	3/17/2020, 1:01:58 PM	0.987
xgb_pipeline 2020-03-16 16-47-...	✓	0:06:08	[View pipeline]	-	3/16/2020, 12:47:19 ...	0.983
xgb_pipeline 2020-03-16 13-57-...	✓	0:05:18	[View pipeline]	-	3/16/2020, 9:57:03 AM	0.980

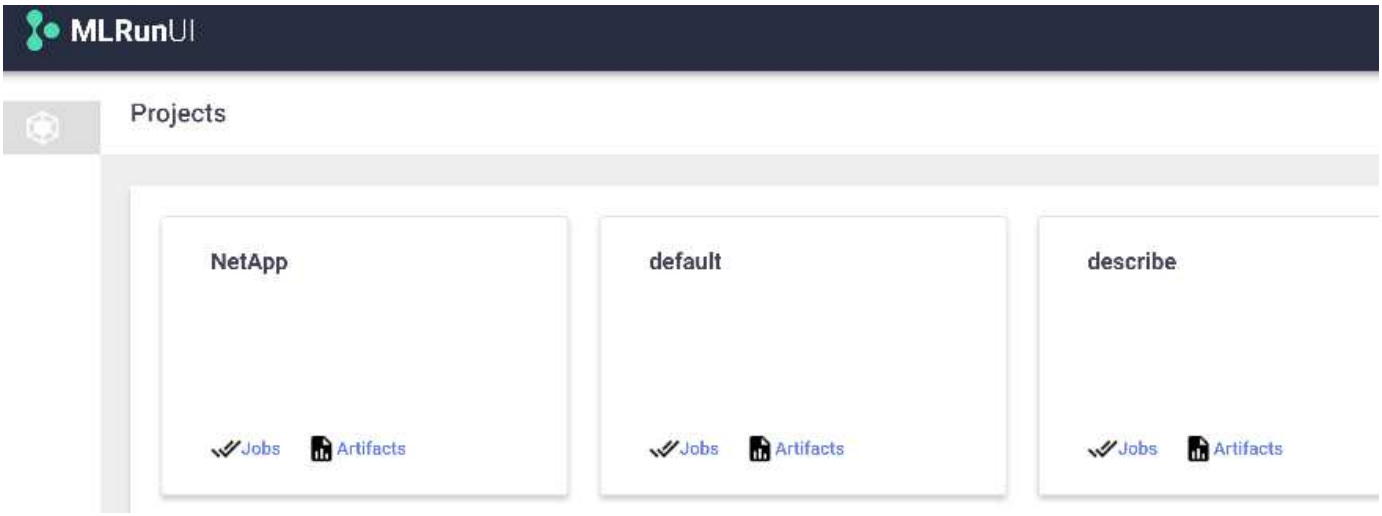
Si vous sélectionnez l'étape Snapshot, le nom de la copie Snapshot utilisée pour exécuter cette expérience s'affiche.



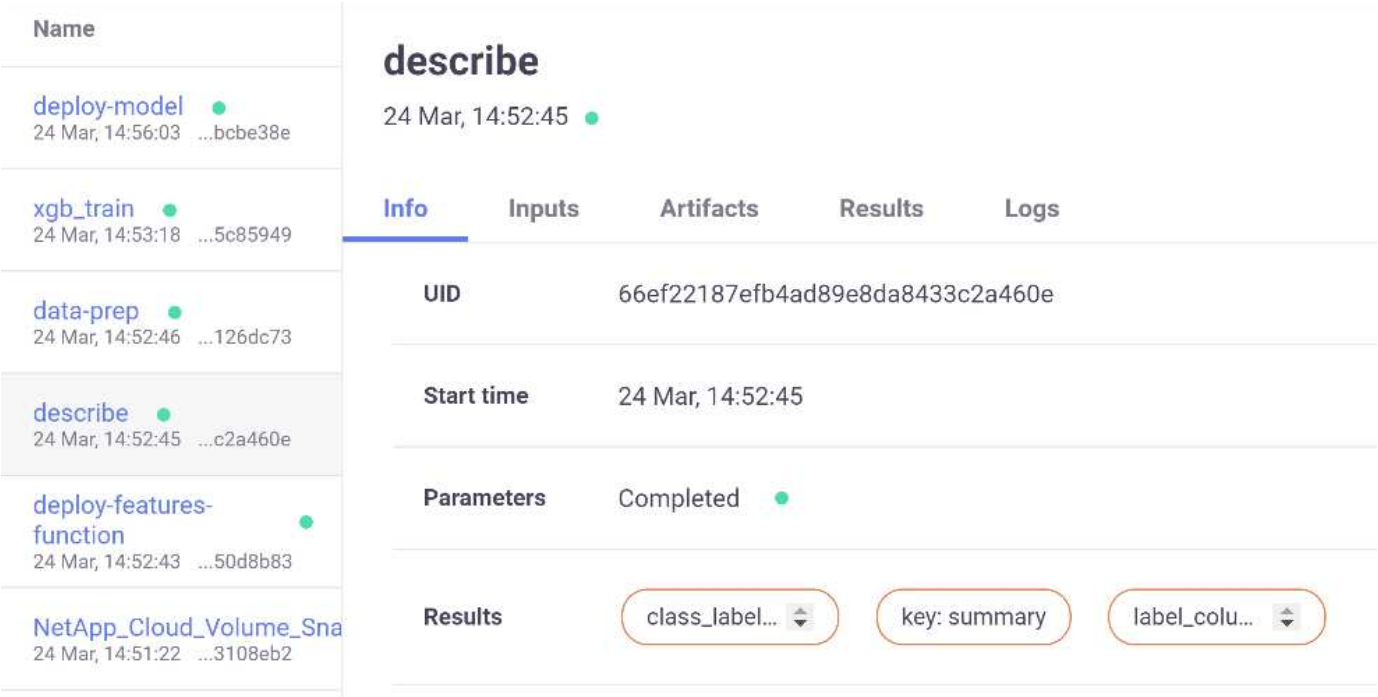
L'étape décrite présente des artéfacts visuels pour explorer les mesures que nous avons utilisées. Vous pouvez développer pour afficher le tracé complet comme illustré dans l'image suivante.



La base de données de l'API MLRun assure également le suivi des entrées, des sorties et des artefacts pour chaque exécution organisée par projet. L'image suivante présente un exemple d'entrées, de sorties et d'artefacts pour chaque séquence.



Pour chaque tâche, nous stockons des détails supplémentaires.



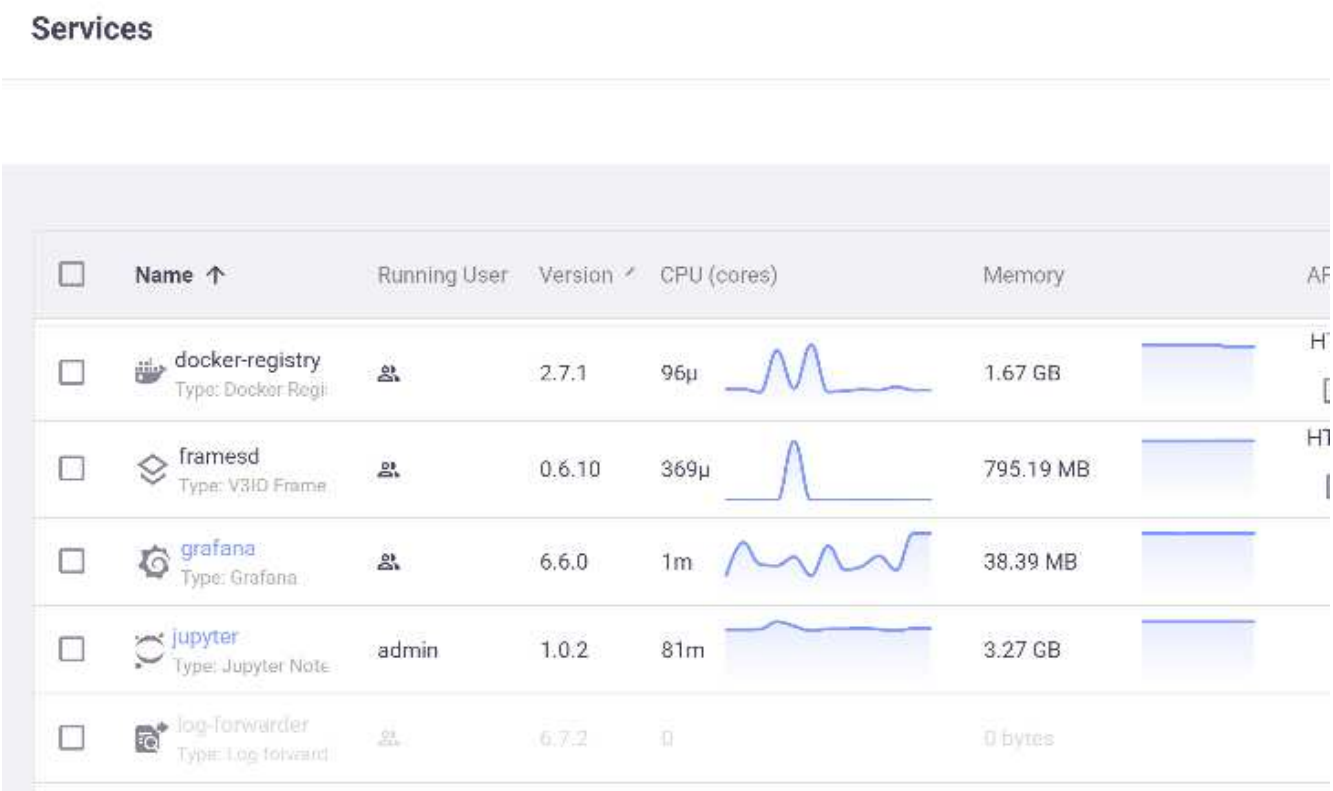
Il y a plus d'informations sur MLRun que ce document. Les artefacts al, y compris la définition des étapes et fonctions, peuvent être enregistrés dans la base de données API, versionnés et appelés individuellement ou en tant que projet complet. Les projets peuvent également être enregistrés et transmis à Git pour une utilisation ultérieure. Nous vous encourageons à en savoir plus sur le "MLRun site GitHub".

Déployez Grafana Dashboard

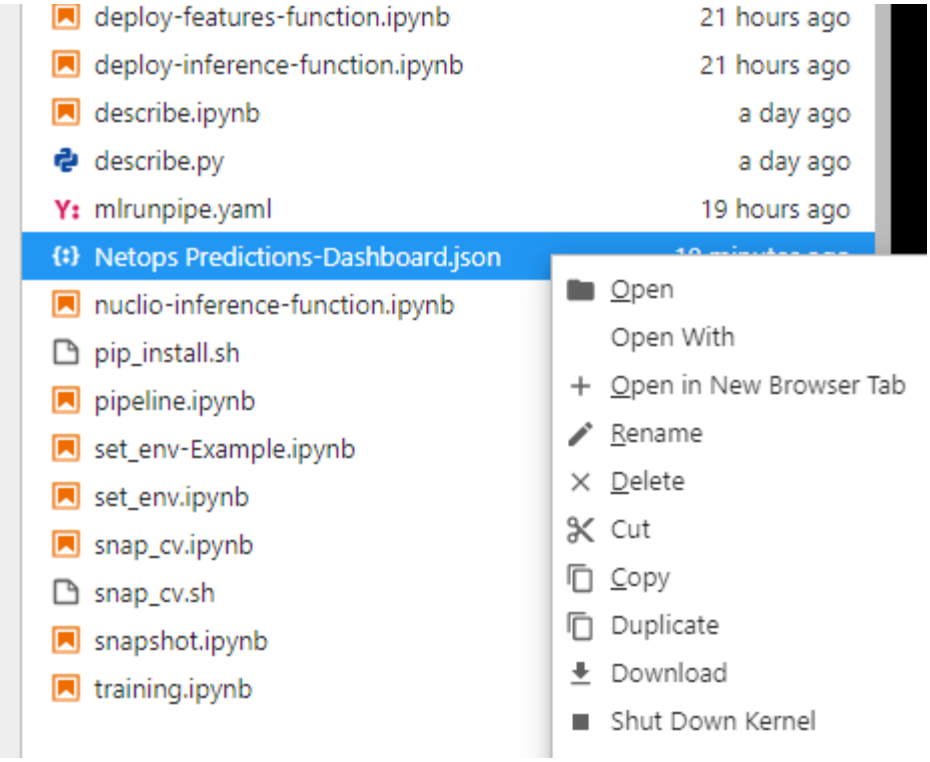
Après le déploiement de tout, nous inférons les nouvelles données. Les modèles prévoient une défaillance sur l'équipement du périphérique réseau. Les résultats de la prédiction sont conservés dans une table de timeseries d'Iguazio. Vous pouvez visualiser les résultats avec Grafana dans la plate-forme intégrée à la politique de sécurité et d'accès aux données d'Iguazio.

Vous pouvez déployer le tableau de bord en important le fichier JSON fourni dans les interfaces de Grafana.

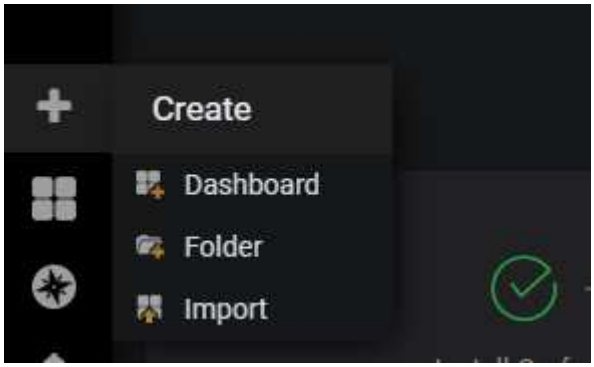
1. Pour vérifier que le service Grafana est exécuté, consultez la section Services.



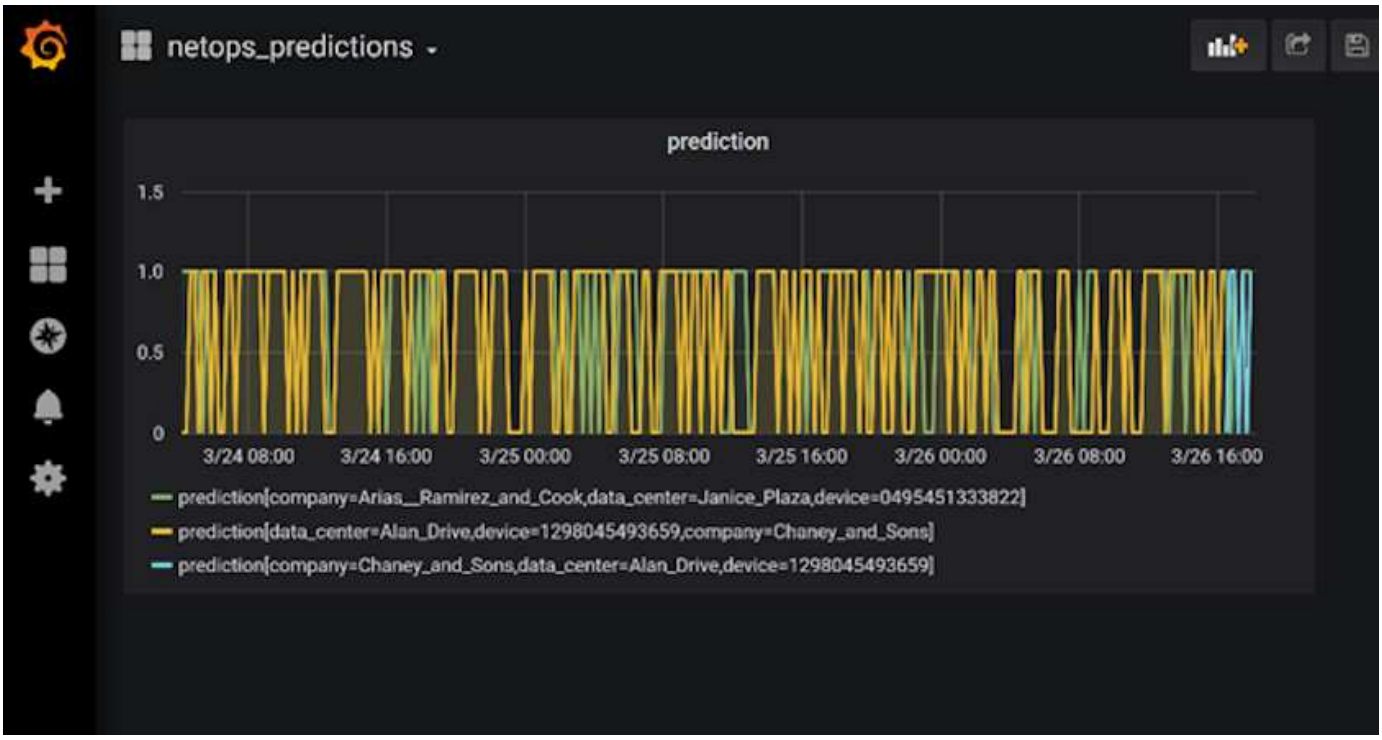
2. Si ce n’est pas le cas, déployez une instance à partir de la section Services :
   a. Cliquez sur Nouveau service.
   b. Sélectionnez Grafana dans la liste.
   c. Acceptez les valeurs par défaut.
   d. Cliquez sur étape suivante.
   e. Entrez votre ID utilisateur.
   f. Cliquez sur Save Service.
   g. Cliquez sur appliquer les modifications en haut.
3. Pour déployer le tableau de bord, téléchargez le fichier `NetopsPredictions-Dashboard.json` Via l’interface Jupyter.



4. Ouvrez Grafana à partir de la section Services et importez le tableau de bord.



5. Cliquez sur Télécharger `*.json` Et sélectionnez le fichier que vous avez téléchargé précédemment (`NetopsPredictions-Dashboard.json`). Le tableau de bord s'affiche une fois le téléchargement terminé.



Déployer la fonction nettoyage

Lorsque vous générez un grand nombre de données, il est important de préserver la propreté et l'organisation des données. Pour ce faire, déployez la fonction de nettoyage avec le `cleanup.ipynb` bloc-notes.

Avantages

NetApp et Iguazio accélèrent et simplifient le déploiement des applications d'IA et DE ML en créant dans des frameworks essentiels, comme Kubeflow, Apache Spark et TensorFlow, avec des outils d'orchestration comme Docker et Kubernetes. En unifiant le pipeline de données de bout en bout, NetApp et Iguazio réduisent la latence et la complexité inhérentes à de nombreuses charges de travail informatiques avancées, afin de combler l'écart entre le développement et les opérations. Les data Scientists peuvent exécuter des requêtes sur d'importants jeux de données et partager en toute sécurité les données et les modèles algorithmiques avec les utilisateurs autorisés au cours de la phase d'entraînement. Une fois que les modèles conteneurisés sont prêts pour la production, vous pouvez facilement les déplacer d'environnements de développement à des environnements opérationnels.