部署应用程序

NetApp Solutions

NetApp
April 12, 2024

目录

部署应用程序 1
从 GitHub 获取代码 1
配置工作环境 1
部署 Grafana 信息板 13

部署应用程序

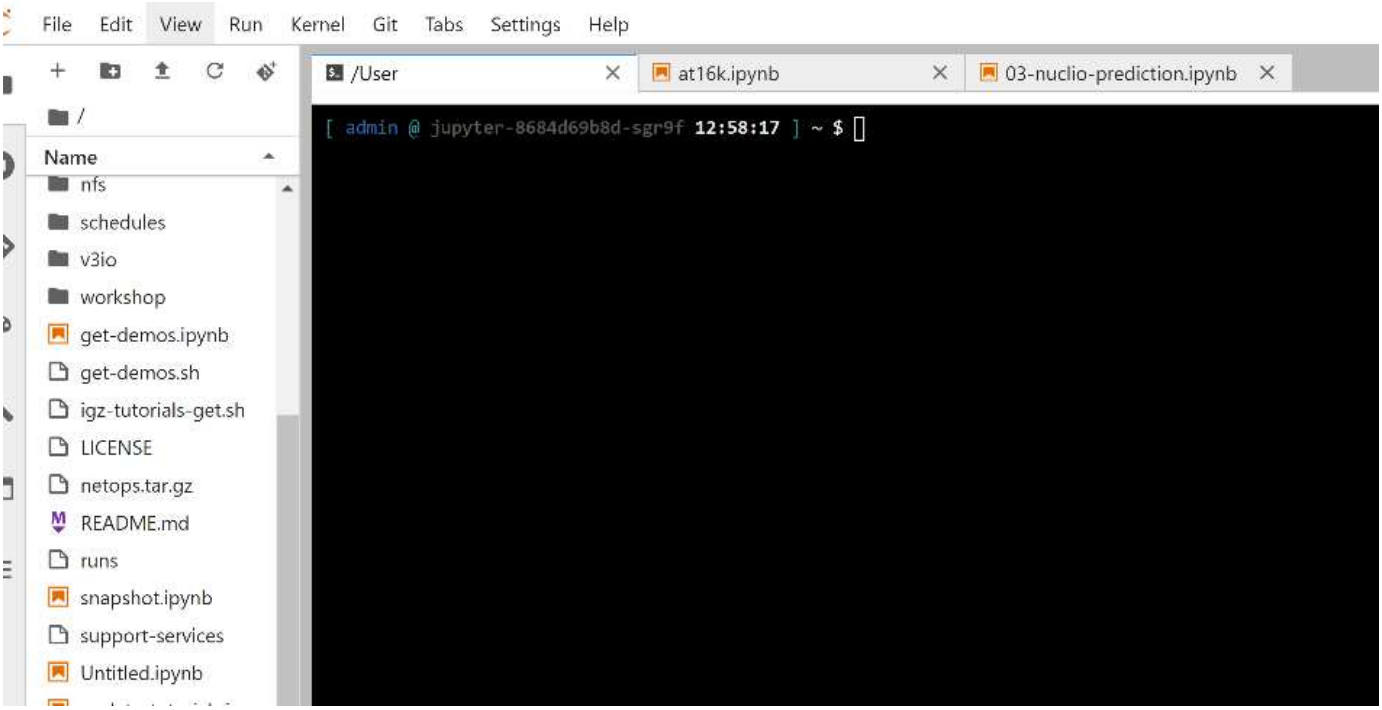
以下各节介绍了如何安装和部署应用程序。

从 GitHub 获取代码

既然 NetApp Cloud Volume 或 NetApp Trident 卷可供 Iguazio 集群和开发人员环境使用，您就可以开始查看该应用程序了。

用户拥有自己的工作空间（目录）。在每个笔记本电脑上，用户目录的路径为 `/User` 。Iguazio 平台负责管理目录。如果按照上述说明进行操作，则可以在 `/NetApp` 目录中找到 NetApp Cloud 卷。

使用 Jupyter 终端从 GitHub 获取代码。



在 Jupyter 终端提示符处，克隆项目。

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

现在，您应在 Jupyter 工作空间的文件树中看到 `NetOps" - "NetApp` " 文件夹。

配置工作环境

将 `Notebook set_env-example.ipynb` 复制为 `set_env.ipynb` 。打开并编辑 `set_env.ipynb` 。此笔记本电脑可为凭据，文件位置和执行驱动程序设置变量。

如果按照上述说明进行操作，则只需执行以下步骤即可：

1. 从 Iguazio 服务信息板获取此值： `docker_regRegistry`

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

2. 将 `admin` 更改为您的 Iguazio 用户名：

   `IGZ_container_path = "/" 用户 /admin"`

   下面是 ONTAP 系统连接详细信息。包括安装 Trident 时生成的卷名称。以下设置适用于内部 ONTAP 集群：

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

   以下设置适用于 Cloud Volumes ONTAP：

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

创建基本 Docker 映像

构建 ML 管道所需的一切都包含在 Iguazio 平台中。开发人员可以定义运行管道和从 Jupyter Notebook 执行映像创建所需的 Docker 映像的规格。打开笔记本 `creation- images.ipynb` 并运行所有单元格。

此笔记本可创建两个我们在管道中使用的映像。

- `igiio/NetApp.` 用于处理 ML 任务。

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/` 渠道 。包含用于处理 NetApp Snapshot 副本的实用程序。

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()
```

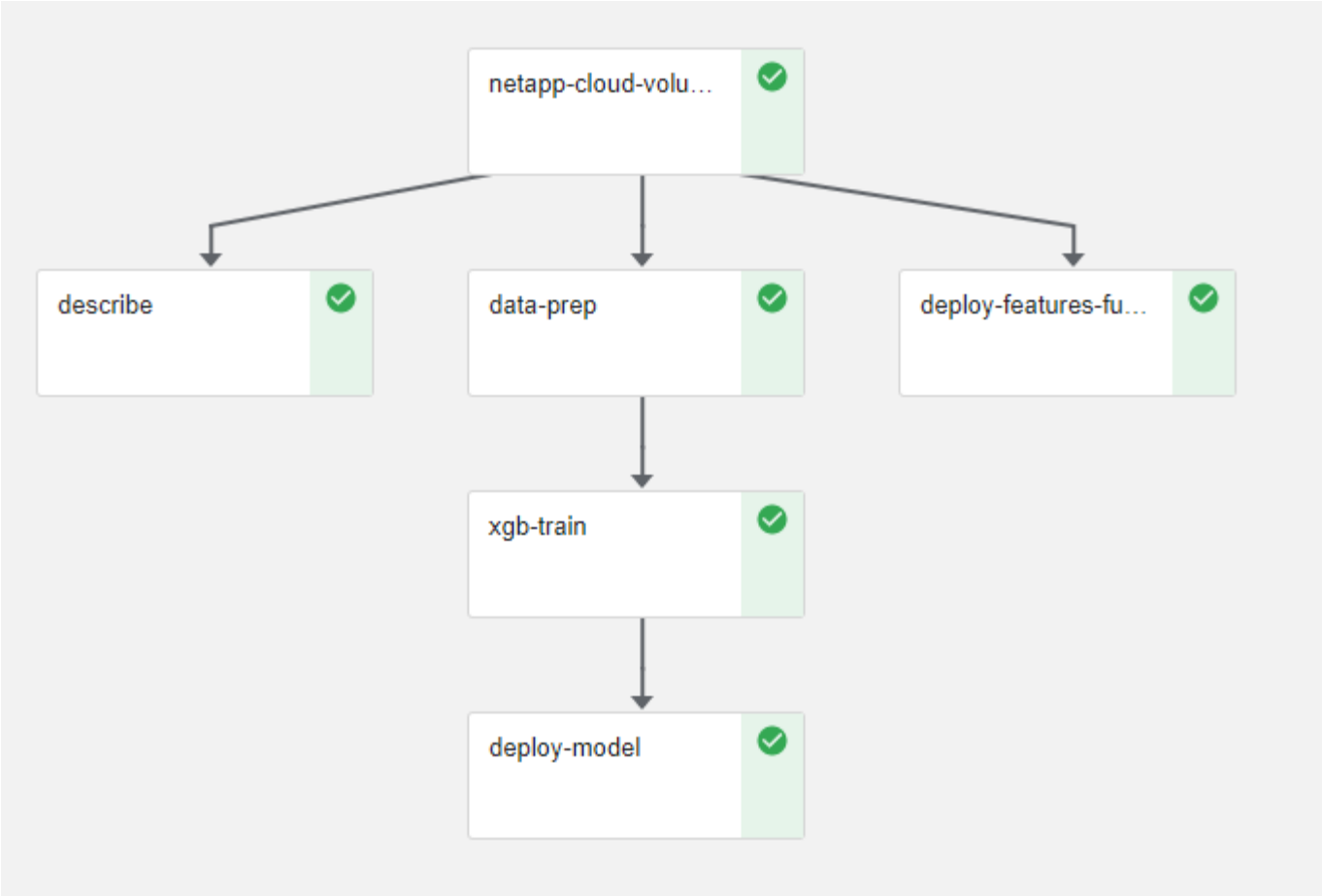
查看各个 Jupyter 笔记本电脑

下表列出了我们用于构建此任务的库和框架。所有这些组件均已与 Iguazio 基于角色的访问和安全控制完全集成。

库 / 框架	Description
MLRun	由 Iguazio 管理的，用于组装，执行和监控 ML/AI 管道。
Nutriio	与 Iguazio 集成的无服务器功能框架。也可作为由 Iguazio 管理的开源项目提供。
Kubeflow	基于 Kubernetes 的框架，用于部署管道。这也是一个开源项目， Iguazio 为此做出了贡献。它与 Iguazio 集成，可提高安全性，并与基础架构的其余部分集成。
Docker	Docker 注册表作为服务在 Iguazio 平台中运行。您也可以更改此设置以连接到注册表。
NetApp Cloud Volumes	通过在 AWS 上运行的 Cloud Volumes ，我们可以访问大量数据，并可以创建 Snapshot 副本来版本用于培训的数据集。
Trident	Trident 是一个由 NetApp 管理的开源项目。它有助于与 Kubernetes 中的存储和计算资源集成。

我们使用多台笔记本电脑来构建 ML 管道。在将每台笔记本电脑整合到管道中之前，可以对其进行单独测试。我们将按照此演示应用程序的部署流程分别介绍每台笔记本电脑。

理想的结果是，通过管道根据数据的 Snapshot 副本训练模型，并部署模型进行推理。下图显示了已完成的 MLRun 管道的结构图。



部署数据生成功能

本节介绍如何使用 Nutrio 无服务器功能生成网络设备数据。此用例是从部署了管道并使用 Iguazio 服务监控和预测网络设备故障的 Iguazio 客户端改编而来的。

我们模拟了来自网络设备的数据。执行 Jupyter 笔记本 `data-generator.ipynb` 可创建一个每 10 分钟运行一次的无服务器功能，并使用新数据生成一个 Parquet 文件。要部署此功能，请运行此笔记本中的所有单元。请参见 "Nutrio 网站" 查看此笔记本中任何不熟悉的组件。

生成函数时，将忽略包含以下注释的单元格。假设笔记本电脑中的每个单元都属于该功能的一部分。导入 Nuclio 模块以启用 ` %nuclio 幻影` 。

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

在函数规范中，我们定义了函数的执行环境，触发方式以及使用的资源。

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

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

初始化 `init_context` 函数时， Noclio 框架会调用该函数。

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

在函数初始化时，将调用不在函数中的任何代码。调用该命令时，系统将执行处理程序功能。您可以更改处理程序的名称并在函数规范中指定它。

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

您可以在部署之前从笔记本电脑测试此功能。

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

该功能可以从笔记本电脑部署，也可以从 CI/CD 管道部署（修改此代码）。

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

渠道笔记本电脑

这些笔记本电脑不能单独执行此设置。这只是对每台笔记本电脑的回顾。我们在管道中调用了这些命令。要分别执行这些操作，请查看 MLRun 文档，将其作为 Kubernetes 作业执行。

Snap_CV.ipynb

此笔记本电脑在管道开始时处理 Cloud Volume Snapshot 副本。它会将卷的名称传递到管道环境。此笔记本会调用 shell 脚本来处理 Snapshot 副本。在管道中运行时，执行上下文包含可帮助查找执行所需的所有文件的变量。编写此代码时，开发人员不必担心执行此代码的容器中的文件位置。如后面所述，此应用程序会随其所有依赖项一起部署，而是通过管道参数的定义来提供执行上下文。

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

创建的 Snapshot 副本位置将放置在 MLRun 上下文中，供管道中的步骤使用。

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

接下来的三台笔记本电脑将并行运行。

data-prep.ipynb

必须将原始指标转换为功能，才能进行模型培训。此笔记本电脑可从 Snapshot 目录读取原始指标，并将模型培训的功能写入 NetApp 卷。

在管道环境中运行时，输入 `DATA_DIR` 包含 Snapshot 副本位置。

```
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')))
```

描述 **.ipynb**

为了直观显示传入指标，我们部署了一个管道步骤，该步骤可提供通过 Kubeflow 和 MLRun UI 提供的图解和图形。每个执行都有自己版本的此可视化工具。

```
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-feature.ipynb

我们会持续监控指标以查找异常。此笔记本电脑可创建一个无服务器功能，用于生成对传入指标运行预测所需的功能。此笔记本电脑将调用函数的创建。功能代码位于笔记本电脑 `data-prep.ipynb` 中。请注意，我们使用同一笔记本电脑作为管道中的一个步骤。

训练 **.ipynb**

创建功能后，我们将触发模型培训。此步骤的输出为要用于推理的模型。我们还会收集统计信息，以跟踪每个执行情况（实验）。

例如，以下命令会将准确性得分输入到该实验的上下文中。此值在 Kubeflow 和 MLRun 中可见。

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

deploy-inftion-Function.ipynb

管道中的最后一步是将模型部署为无服务器功能，以实现持续推理。此笔记本电脑将调用在 `nuclio-inference - Function .ipynb` 中定义的无服务器功能的创建过程。

审核和构建管道

通过将所有笔记本电脑整合到一个管道中，可以持续运行实验，根据新指标重新评估模型的准确性。首先，打开 `pipeline.ipynb` 笔记本电脑。我们将详细介绍 NetApp 和 Iguazio 如何简化此 ML 管道的部署。

我们使用 MLRun 为管道的每个步骤提供上下文并处理资源分配。MLRun API 服务在 Iguazio 平台中运行，是与 Kubernetes 资源交互的点。每个开发人员都不能直接请求资源； API 负责处理这些请求并启用访问控制。

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

此管道可以与 NetApp Cloud Volumes 和内部卷配合使用。我们构建此演示的目的是使用 Cloud Volumes ，但您可以在代码中看到在内部运行的选项。

```
# 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]
```

将 Jupyter 笔记本电脑转变为 Kubeflow 步骤所需的第一个操作是将代码转换为函数。功能具有运行该笔记本电脑所需的所有规格。向下滚动笔记本电脑时，您可以看到我们为管道中的每个步骤定义了一个函数。

属于笔记本电脑	**Description**
<code_to_Function> （ MLRun 模块的一部分）	函数名称： project name 。用于组织所有项目项目项目。此信息会显示在 MLRun UI 中。好的。在这种情况下，是 Kubernetes 作业。这可以是 dask ， MPI ， spark8s 等。有关详细信息，请参见 MLRun 文档。文件笔记本的名称。此位置也可以是 Git （ HTTP ）中的一个位置。
图像	我们在此步骤中使用的 Docker 映像的名称。我们先前使用 create-image.ipynb 笔记本创建了此版本。
volume_mounts 和 volumes	有关在运行时挂载 NetApp Cloud Volume 的详细信息。

我们还定义了步骤的参数。

```
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
            }
```

在为所有步骤定义了函数之后，您可以构建管道。我们使用 `kfp` 模块来定义此定义。使用 MLRun 与自行构建之间的区别在于编码的简化和缩短。

我们定义的函数将使用 MLRun 的 `as_step` 函数转换为步骤组件。

Snapshot 步骤定义

启动 Snapshot 功能，输出并将 v3io 作为源进行挂载：

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

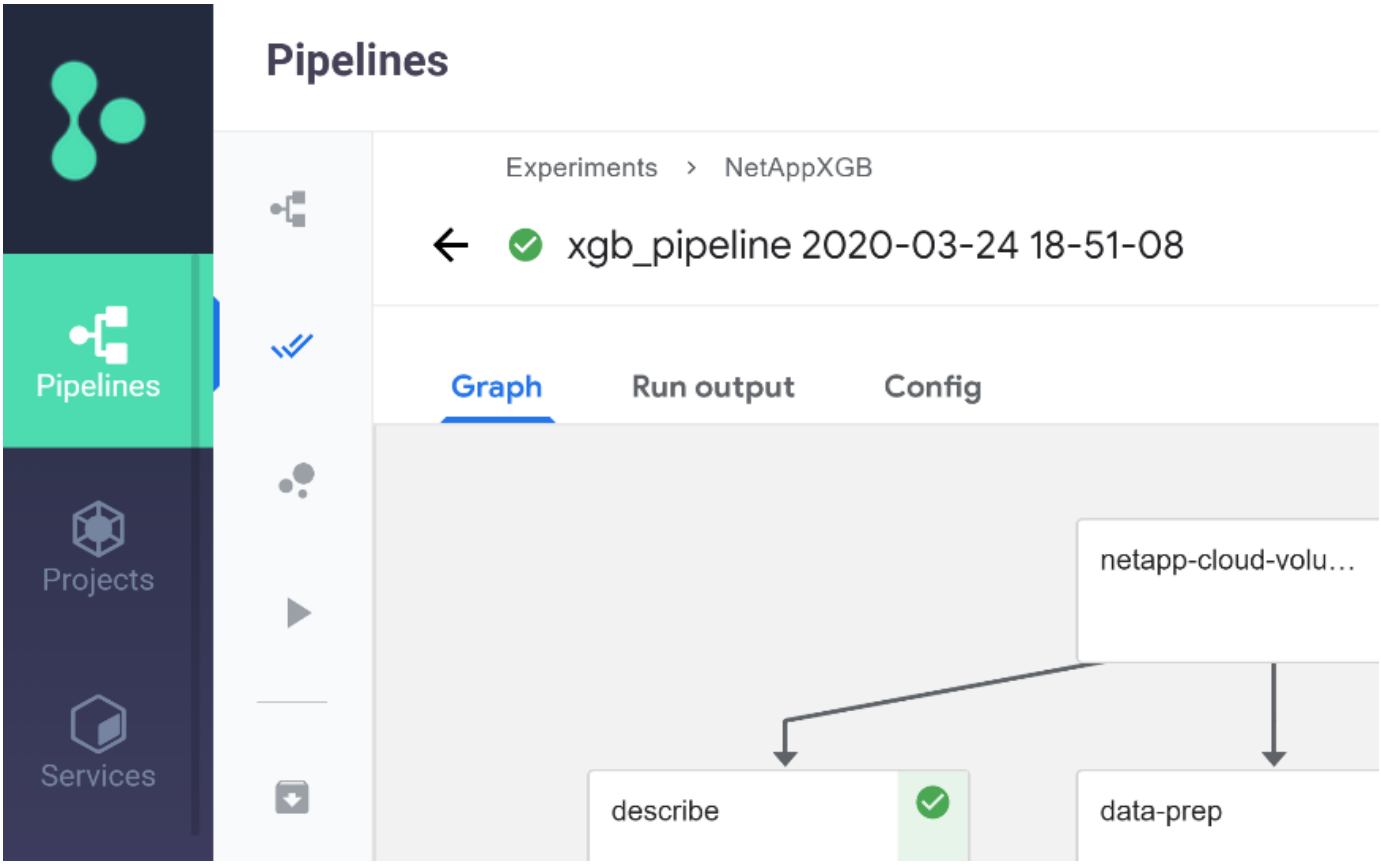
Parameters	详细信息
newtask	newtask 是函数 run 的定义。
（ MLRun 模块）	处理程序。要调用的 Python 函数的名称。我们在笔记本中使用了名称处理程序，但这不是必需的。参数。我们传递给执行的参数。在代码中，我们使用 context.get_param （‘parameter｝ ）来获取值。
as_step	NameKubeflow 管道步骤的名称。输出。这些值是步骤在完成时添加到词典中的值。查看 snap_CV.ipynb 笔记本电脑。mount_v3io （）。此操作将为执行管道的用户配置挂载 /User 的步骤。

```
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)
```

Parameters	详细信息
输入	您可以将上一步的输出传递到步骤。在这种情况下，snap.outputs"snapVolumeDetails" 是我们在快照步骤中创建的 Snapshot 副本的名称。
输出路径	一个位置，用于放置使用 MLRun 模块 log_tools. 生成的项目。

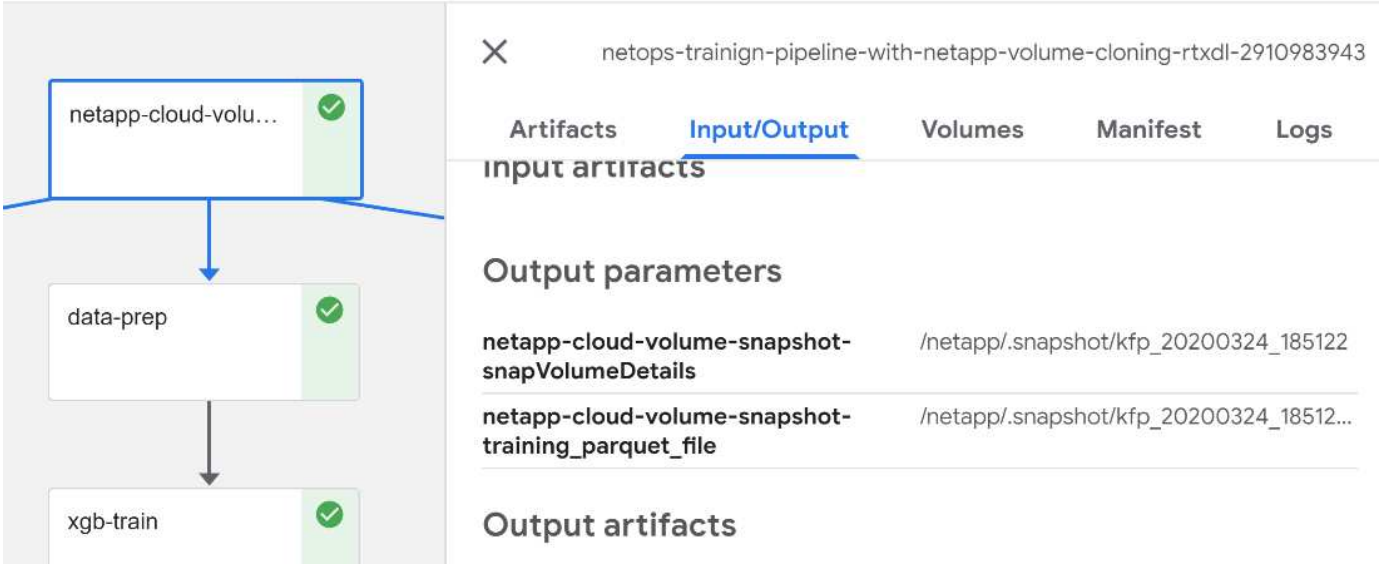
您可以从上至下运行 `pipvip.ipynb` 。然后，您可以转到 Iguazio 信息板中的管道选项卡来监控进度，如 Iguazio 信息板管道选项卡中所示。



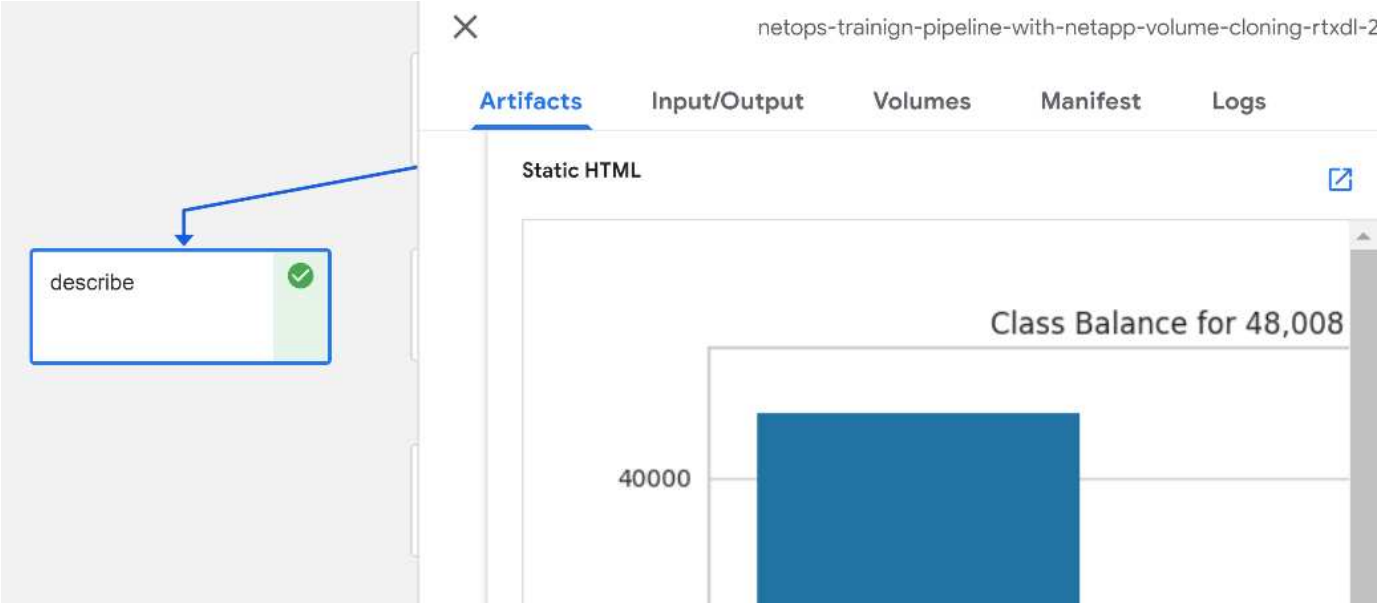
由于我们在每次运行中都记录了训练步骤的准确性，因此我们在每个实验中都有一个准确性记录，如训练准确性记录所示。

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

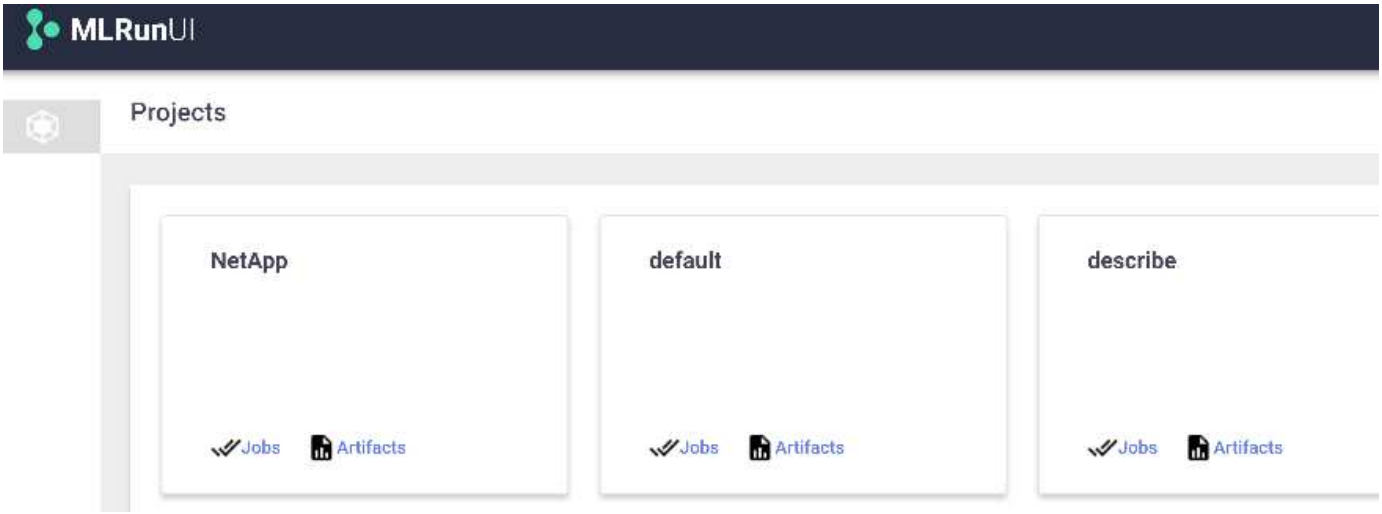
如果选择 Snapshot 步骤，则可以看到用于运行此实验的 Snapshot 副本的名称。



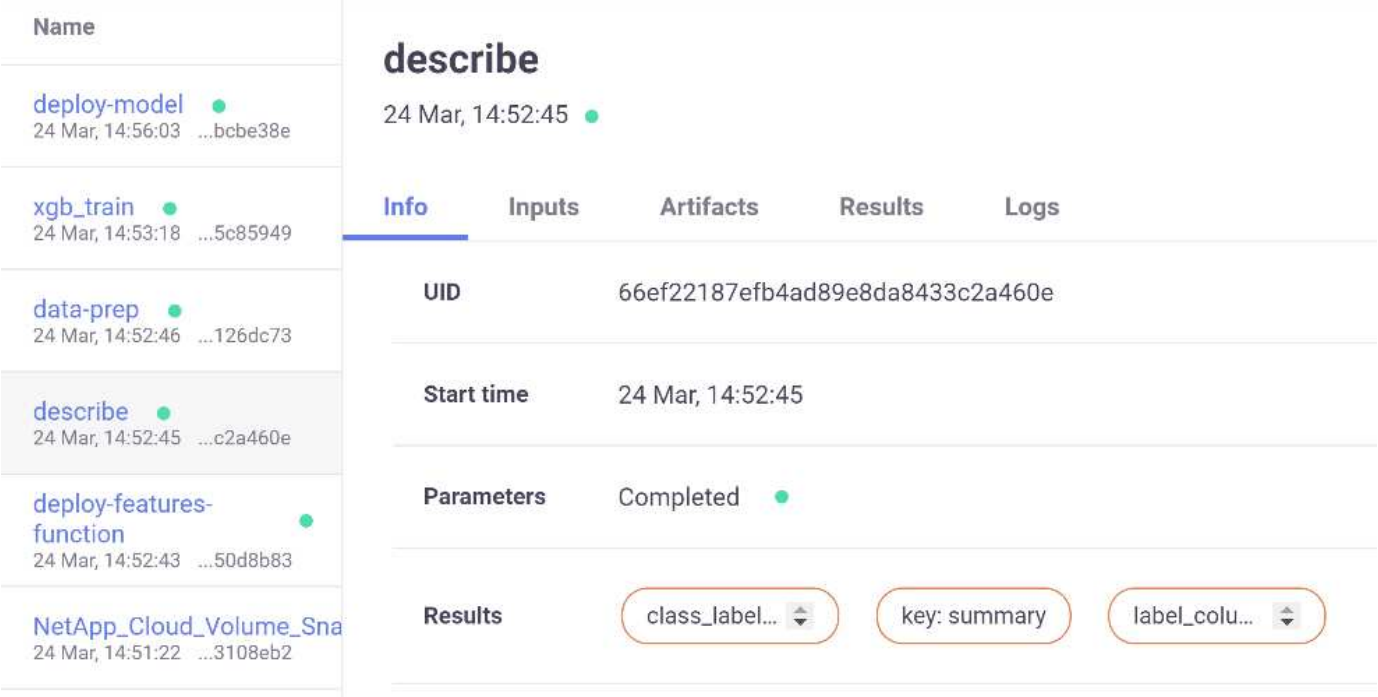
所述步骤具有可视化项目，可用于浏览我们使用的指标。您可以展开以查看完整图，如下图所示。



此外， MLRun API 数据库还会跟踪按项目组织的每个运行的输入，输出和项目。下图显示了每个运行的输入，输出和项目示例。



对于每个作业，我们会存储更多详细信息。



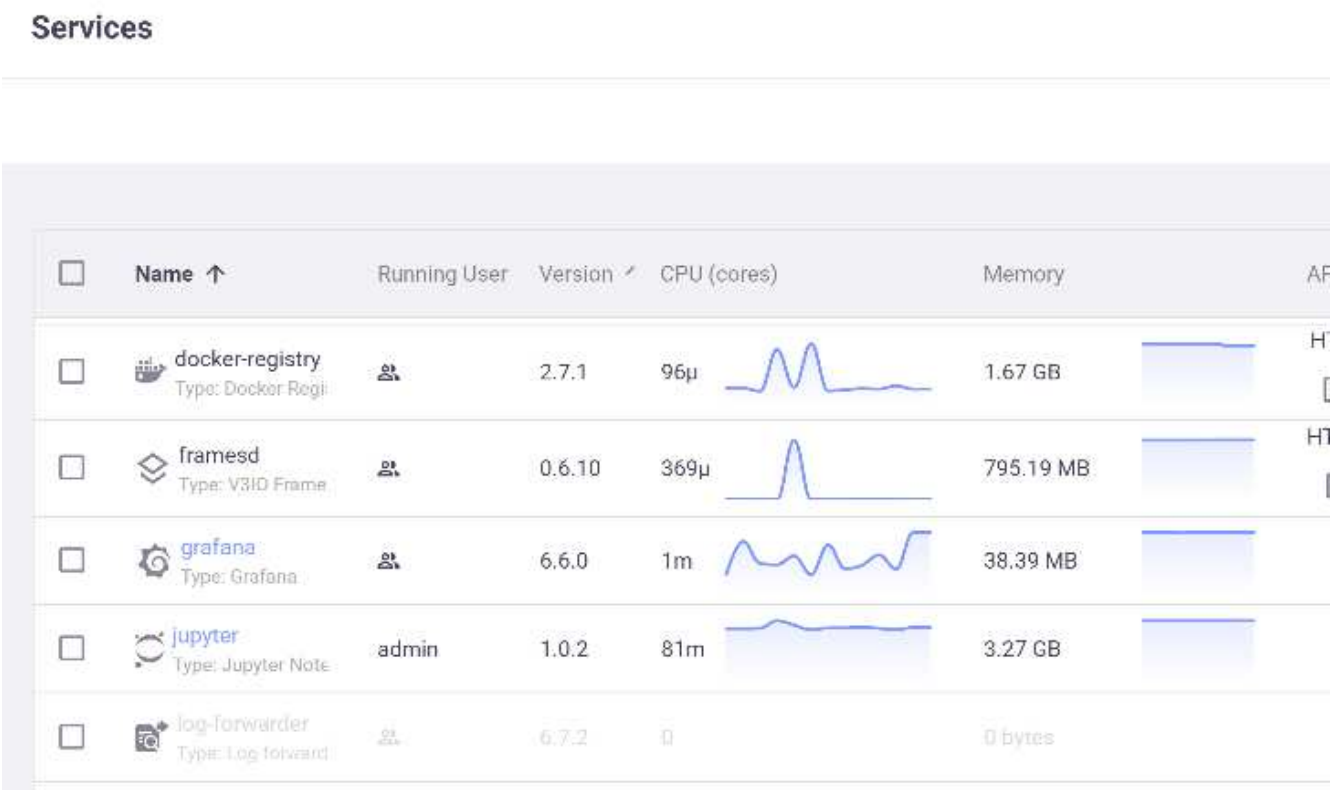
有关 MLRun 的信息比本文档中介绍的信息更多。可以将 AL 项目（包括步骤和功能的定义）保存到 API 数据库中，并进行版本控制，也可以单独调用或作为完整项目调用。此外，还可以保存项目并将其推送到 Git 以供日后使用。我们建议您在中了解更多信息 "MLRun GitHub 站点"。

部署 Grafana 信息板

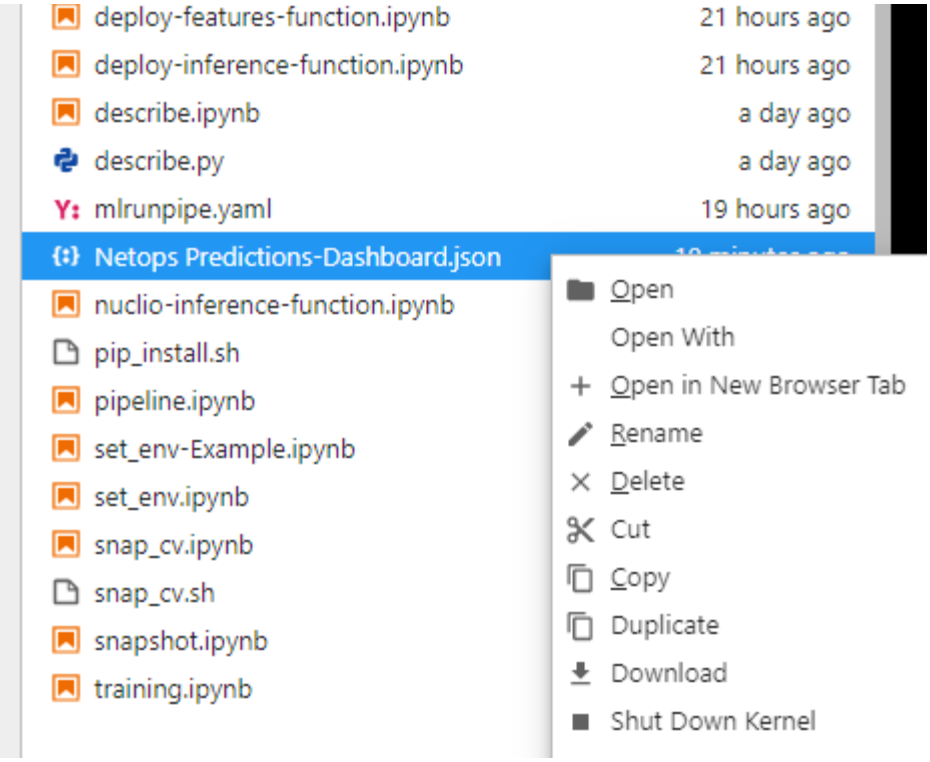
部署完所有内容后，我们会对新数据运行推断。这些型号可预测网络设备故障。预测结果存储在 Iguazio 时间序列表中。您可以在与 Iguazio 的安全和数据访问策略集成的平台中使用 Grafana 来查看结果。

您可以通过将提供的 JSON 文件导入到集群中的 Grafana 接口来部署信息板。

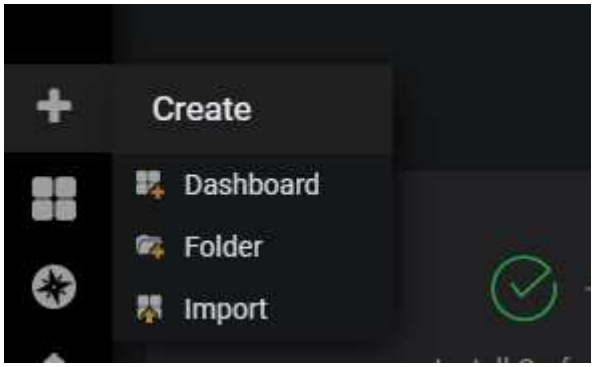
1. 要验证 Grafana 服务是否正在运行，请查看服务下的。



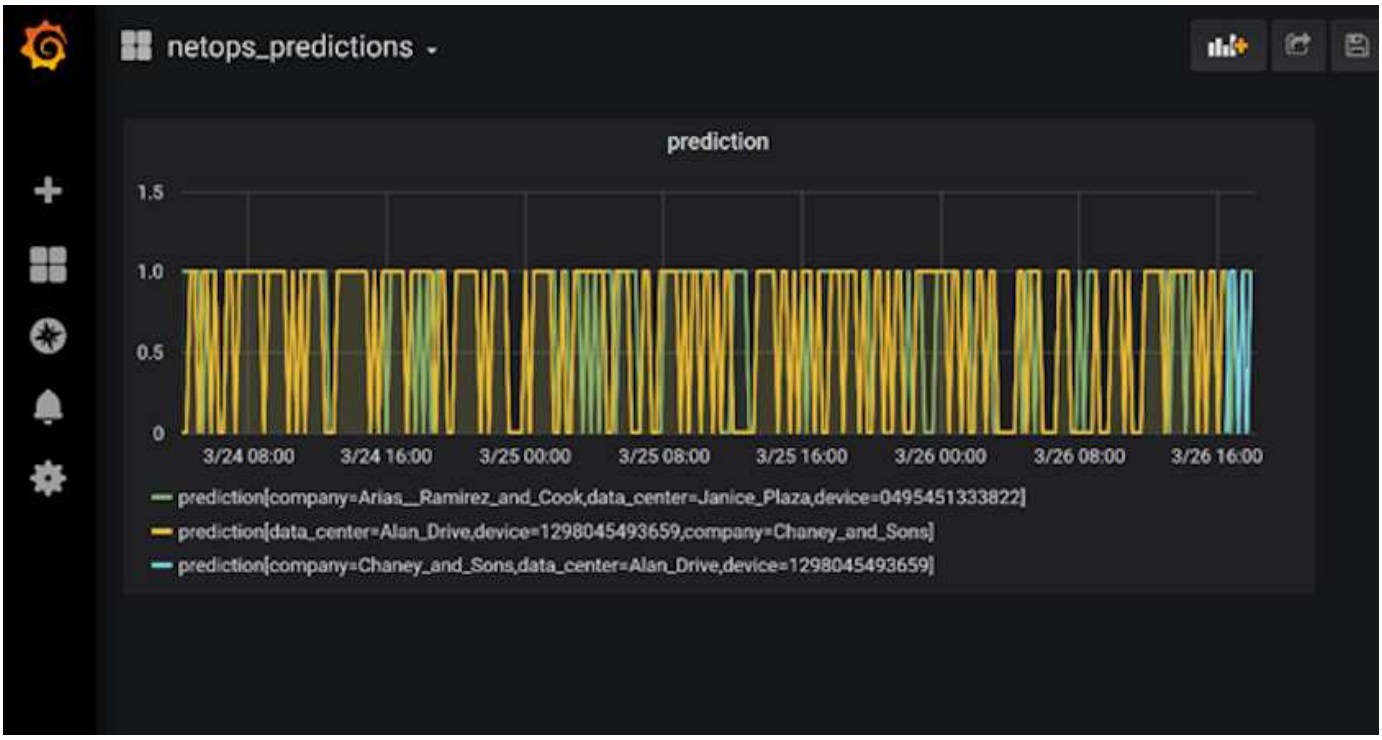
2. 如果不存在此实例，请从服务部分部署此实例：
   a. 单击新建服务。
   b. 从列表中选择 Grafana 。
   c. 接受默认值。
   d. 单击下一步。
   e. 输入您的用户 ID 。
   f. 单击 Save Service 。
   g. 单击顶部的 Apply Changes 。
3. 要部署信息板，请通过 Jupyter 界面下载文件 `NetopsPredictions-Dashboard.json` 。



4. 从服务部分打开 Grafana 并导入信息板。



5. 单击 Upload `* 。 json` File ，然后选择先前下载的文件（`NetopsPredictions-Dashboard.json`）。上传完成后，将显示信息板。



部署清理功能

当您生成大量数据时，保持数据干净有序非常重要。为此，请使用 `cleanup.ipynb` 笔记本部署清理功能。

优势

NetApp 和 Iguazio 通过构建 Kubeflow ， Apache Spark 和 TensorFlow 等基本框架以及 Docker 和 Kubernetes 等业务流程工具，加快并简化 AI 和 ML 应用程序的部署。通过统一端到端数据管道， NetApp 和 Iguazio 可以减少许多高级计算工作负载固有的延迟和复杂性，从而有效地缩小开发和运营之间的差距。在培训阶段，数据科学家可以对大型数据集运行查询，并与授权用户安全地共享数据和算法模型。容器化模型准备好投入生产后，您可以轻松地将其从开发环境迁移到操作环境。