Dettagli sulla distribuzione e sulla convalida della soluzione

NetApp Solutions

NetApp
April 26, 2024

Sommario

Dettagli sulla distribuzione e sulla convalida della soluzione

Le sezioni seguenti illustrano i dettagli dell'implementazione e della convalida della soluzione.

Implementazione di ONTAP ai

L'implementazione di ONTAP ai richiede l'installazione e la configurazione dell'hardware di rete, calcolo e storage. Le istruzioni specifiche per l'implementazione dell'infrastruttura ai di ONTAP esulano dall'ambito di questo documento. Per informazioni dettagliate sull'implementazione, vedere "NVA-1121-DEPLOY: NetApp ONTAP ai, basato su NVIDIA".

Per la convalida di questa soluzione, è stato creato un singolo volume e montato sul sistema DGX-1. Tale punto di montaggio è stato quindi montato sui container per rendere i dati accessibili per la formazione. Per implementazioni su larga scala, NetApp Trident automatizza la creazione e il montaggio dei volumi per eliminare i costi amministrativi e consentire la gestione delle risorse da parte dell'utente finale.

Implementazione di Kubernetes

Per implementare e configurare il cluster Kubernetes con NVIDIA DeepOps, eseguire le seguenti operazioni da un host di distribuzione jump:

1. Scaricare NVIDIA DeepOps seguendo le istruzioni sul "Pagina introduttiva" Sul sito NVIDIA DeepOps GitHub.
2. Implementare Kubernetes nel cluster seguendo le istruzioni sul "Guida all'implementazione di Kubernetes" Sul sito NVIDIA DeepOps GitHub.

> Affinché l'implementazione di DeepOps Kubernetes funzioni, lo stesso utente deve esistere su tutti i nodi master e worker di Kubernetes.

Se l'implementazione non riesce, modificare il valore di `kubectl_localhost` a false in `deepops/config/group_vars/k8s-cluster.yml` e ripetere il punto 2. Il `Copy kubectl binary to ansible host` attività, che viene eseguita solo quando il valore di `kubectl_localhost` È vero, si basa sul modulo fetch Ansible, che presenta problemi noti di utilizzo della memoria. A volte, questi problemi di utilizzo della memoria possono causare un errore nell'attività. Se l'operazione non riesce a causa di un problema di memoria, il resto dell'operazione di implementazione non viene completata correttamente.

Se l'implementazione viene completata correttamente dopo aver modificato il valore di `kubectl_localhost` a. `false`, quindi è necessario copiare manualmente `kubectl binary` Da un nodo master Kubernetes all'host di salto per l'implementazione. È possibile trovare la posizione di `kubectl binary` su un nodo master specifico eseguendo `which kubectl` comando direttamente su quel nodo.

Implementazione di cnvrg.io

Implementare il CORE cnvrg utilizzando Helm

Helm è il modo più semplice per implementare rapidamente cnvrg utilizzando qualsiasi cluster, on-premise, Minikube o qualsiasi cluster cloud (come AKS, EKS e GKE). Questa sezione descrive come cnvrg è stato installato su un'istanza on-premise (DGX-1) con Kubernetes installato.

Prerequisiti

Prima di completare l'installazione, è necessario installare e preparare le seguenti dipendenze sul computer locale:

- Kubectl
- Timone 3.x
- Kubernetes cluster 1.15+

Implementazione con Helm

1. Per scaricare i grafici di comando più aggiornati, eseguire il seguente comando:

```
helm repo add cnvrg https://helm.cnvrg.io
helm repo update
```

2. Prima di implementare cnvrg, è necessario disporre dell'indirizzo IP esterno del cluster e del nome del nodo su cui verrà implementato cnvrg. Per implementare cnvrg in un cluster Kubernetes on-premise, eseguire il seguente comando:

```
helm install cnvrg cnvrg/cnvrg --timeout 1500s  --wait \ --set
global.external_ip=<ip_of_cluster> \ --set global.node=<name_of_node>
```

3. Eseguire `helm install` comando. Tutti i servizi e i sistemi vengono installati automaticamente sul cluster. Il processo può richiedere fino a 15 minuti.
4. Il `helm install` il comando può richiedere fino a 10 minuti. Una volta completata l'implementazione, accedere all'URL del cnvrg appena distribuito o aggiungere il nuovo cluster come risorsa all'interno dell'organizzazione. Il `helm` Il comando indica l'URL corretto.

```
Thank you for installing cnvrg.io!
Your installation of cnvrg.io is now available, and can be reached via:
Talk to our team via email at
```

5. Quando lo stato di tutti i container è in esecuzione o completo, cnvrg è stato implementato correttamente. Dovrebbe essere simile al seguente output di esempio:

```
NAME                                READY   STATUS      RESTARTS   AGE
cnvrg-app-69fbb9df98-6xrgf                  1/1     Running     0          2m
cnvrg-sidekiq-b9d54d889-5x4fc               1/1     Running     0          2m
controller-65895b47d4-s96v6                 1/1     Running     0          2m
init-app-vs-config-wv9c4                    0/1     Completed   0          9m
init-gateway-vs-config-2zbpp                0/1     Completed   0          9m
init-minio-vs-config-cd2rg                  0/1     Completed   0          9m
minio-0                                     1/1     Running     0          2m
postgres-0                                  1/1     Running     0          2m
redis-695c49c986-kcbt9                      1/1     Running     0          2m
seeder-wh655                                0/1     Completed   0          2m
speaker-5sghr                               1/1     Running     0          2m
```

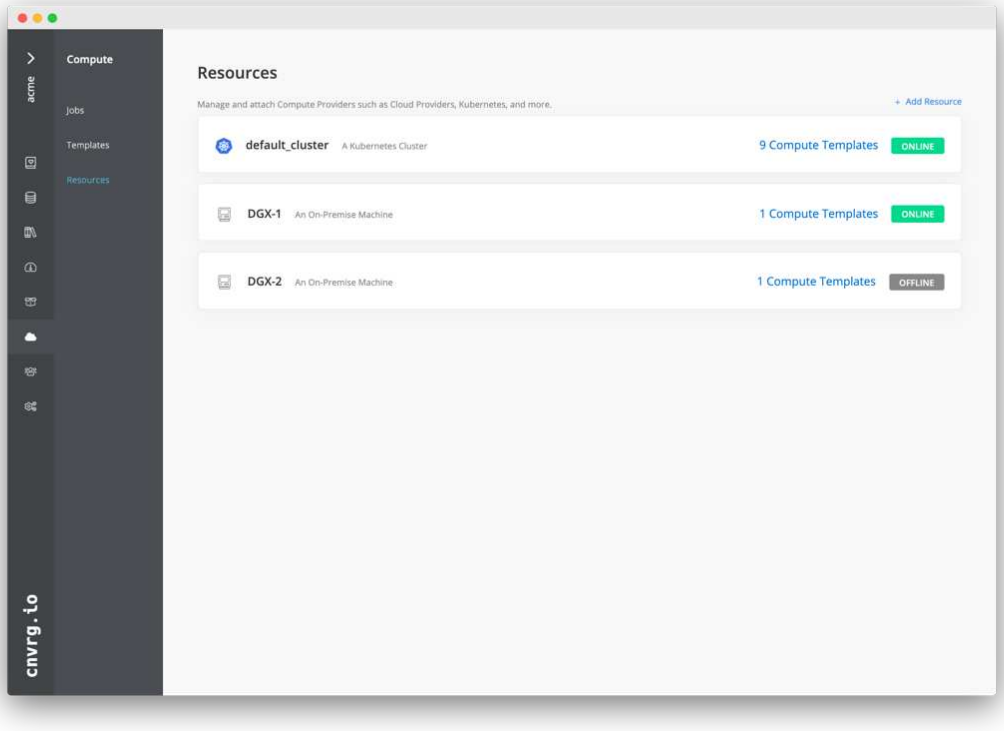
Formazione sul modello di visione artificiale con ResNet50 e il set di dati radiologici Chest

Il sistema operativo ai cnvrg.io è stato implementato su una configurazione Kubernetes su un'architettura NetApp ONTAP ai basata sul sistema NVIDIA DGX. Per la convalida, abbiamo utilizzato il set di dati radiologici NIH Chest costituito da immagini anonimizzate dei raggi X del torace. Le immagini erano in formato PNG. I dati sono stati forniti dal NIH Clinical Center e sono disponibili tramite "Sito di download NIH". Abbiamo utilizzato un campione di 250 GB dei dati con 627, 615 immagini in 15 classi.

Il set di dati è stato caricato sulla piattaforma cnvrg ed è stato memorizzato nella cache di un'esportazione NFS dal sistema di storage NetApp AFF A800.

Impostare le risorse di calcolo

L'architettura cnvrg e la funzionalità di meta-scheduling consentono a tecnici e professionisti IT di collegare diverse risorse di calcolo a una singola piattaforma. Nella nostra configurazione, abbiamo utilizzato lo stesso cluster cnvrg implementato per l'esecuzione dei carichi di lavoro di deep-learning. Se è necessario collegare altri cluster, utilizzare la GUI, come mostrato nella seguente schermata.



Caricare i dati

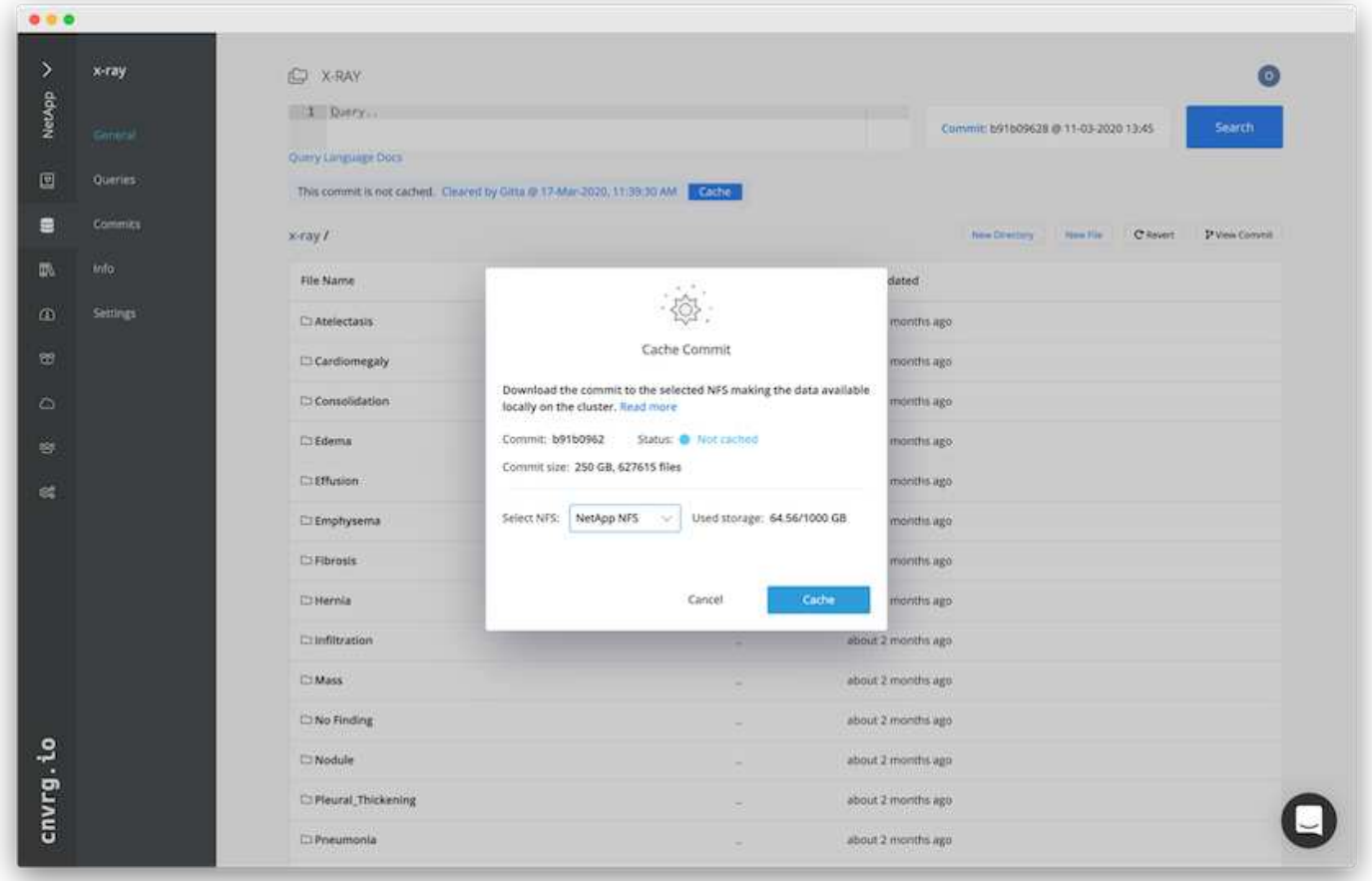
Per caricare i dati sulla piattaforma cnvrg, è possibile utilizzare la GUI o la CLI cnvrg. Per i set di dati di grandi dimensioni, NetApp consiglia di utilizzare CLI perché si tratta di uno strumento potente, scalabile e affidabile in grado di gestire un gran numero di file.

Per caricare i dati, attenersi alla seguente procedura:

1. Scaricare il "CLI cnvrg".
2. accedere alla directory dei raggi x.
3. Inizializzare il set di dati nella piattaforma con `cnvrg data init` comando.
4. Caricare tutti i contenuti della directory nel data Lake centrale con `cnvrg data sync` Command.una volta caricati i dati nell'archivio centrale di oggetti (StorageGRID, S3 o altri), è possibile navigare con la GUI. La figura seguente mostra un file PNG di immagine della fibrosi a raggi X del torace caricato. Inoltre, cnvrg consente di eseguire la versione dei dati in modo che qualsiasi modello creato possa essere riprodotto fino alla versione dei dati.

Dati di cach

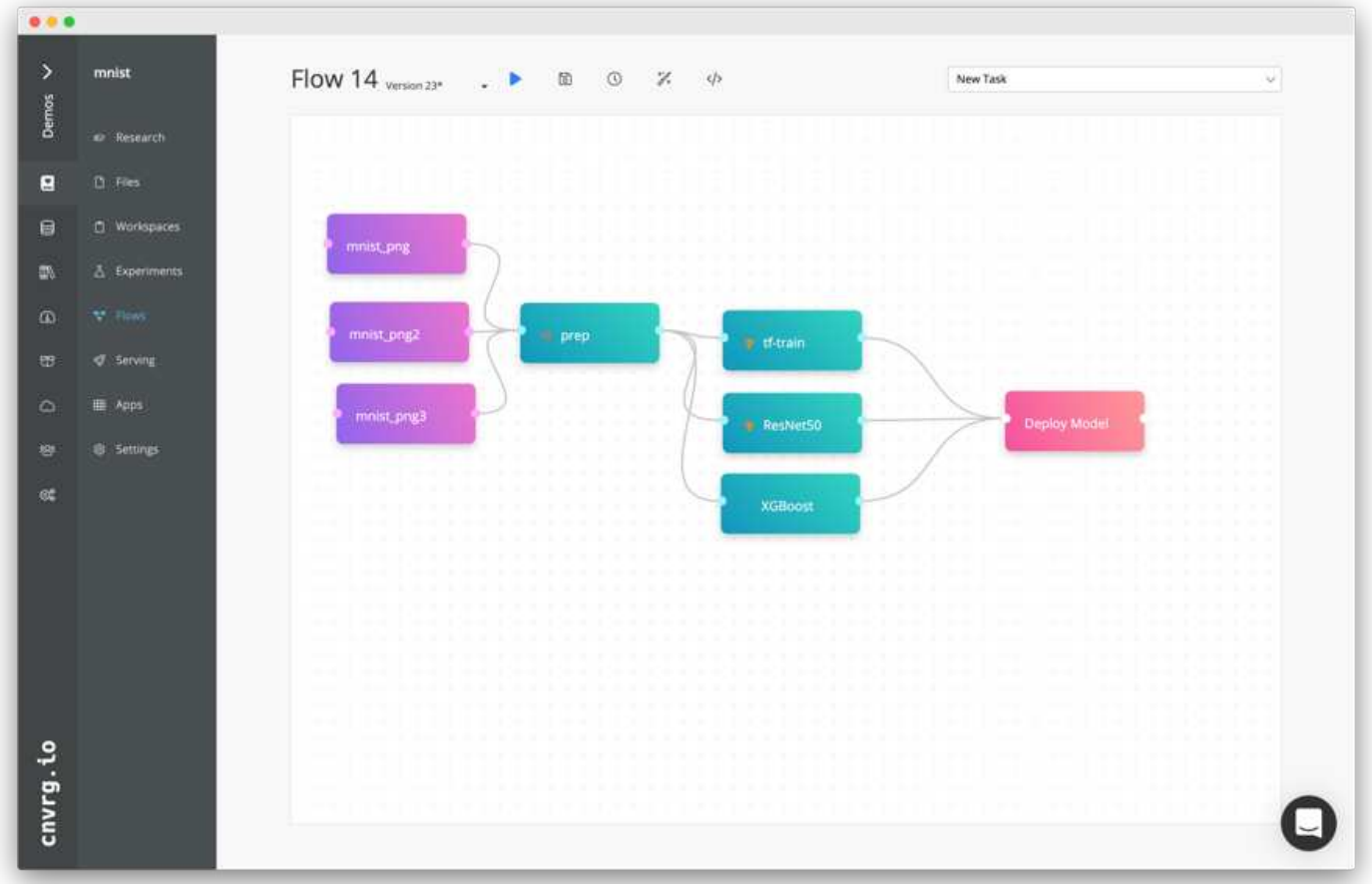
Per accelerare il training ed evitare il download di oltre 600.000 file per ciascun modello di training ed esperimento, abbiamo utilizzato la funzionalità di caching dei dati dopo che i dati sono stati inizialmente caricati nell'archivio centrale di oggetti data-Lake.



Dopo che gli utenti hanno fatto clic su cache, cnvrg scarica i dati nel relativo commit specifico dall'archivio remoto di oggetti e li memorizza nella cache del volume NFS di ONTAP. Al termine, i dati saranno disponibili per il training istantaneo. Inoltre, se i dati non vengono utilizzati per alcuni giorni (ad esempio, per la formazione o l'esplorazione del modello), cnvrg cancella automaticamente la cache.

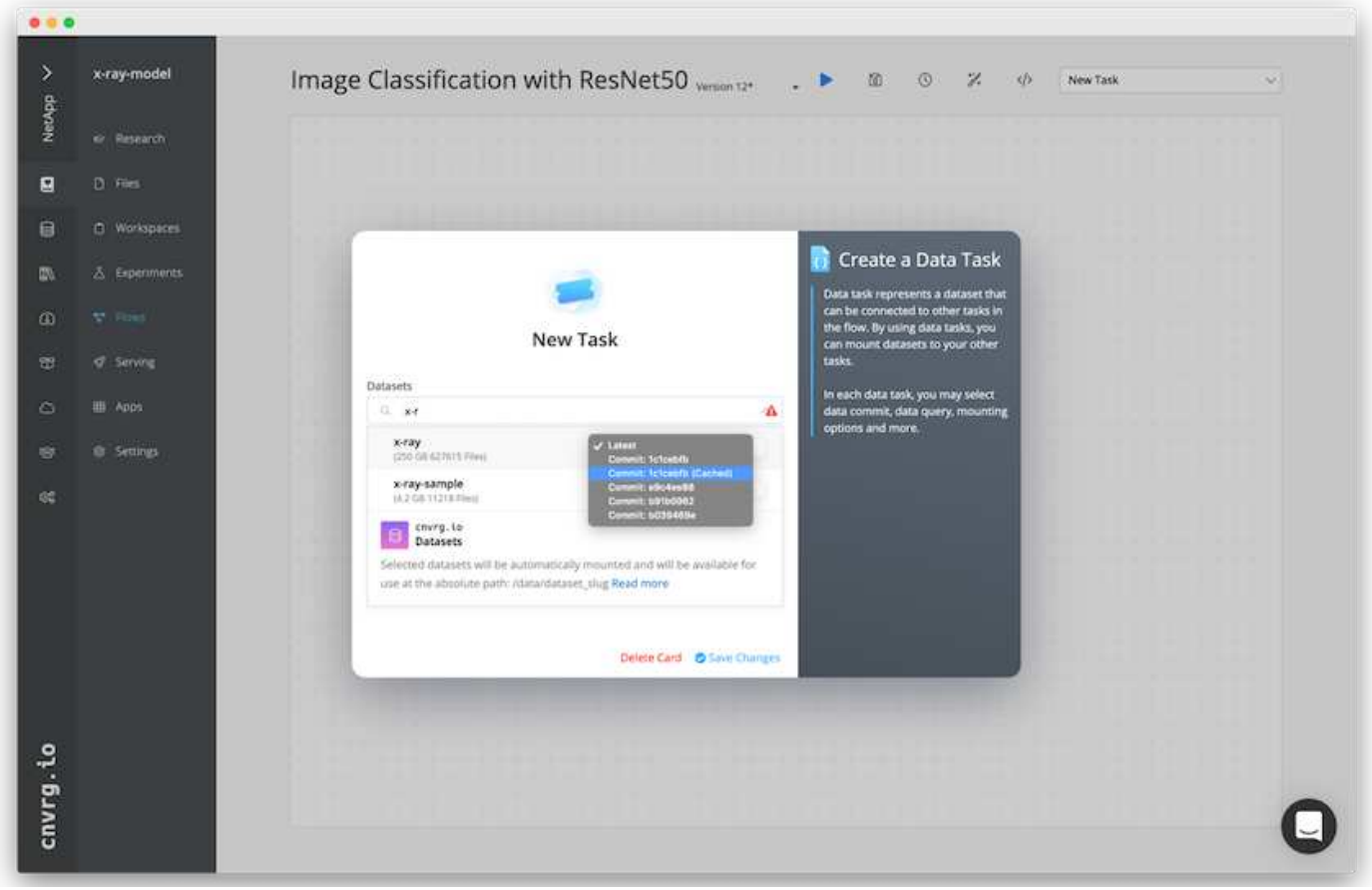
Crea una pipeline ML con i dati memorizzati nella cache

Cnvrg Flows consente di creare facilmente pipeline ML di produzione. I flussi sono flessibili, possono funzionare per qualsiasi tipo di caso d'utilizzo DI ML e possono essere creati attraverso la GUI o il codice. Ogni componente di un flusso può essere eseguito su una diversa risorsa di calcolo con un'immagine Docker diversa, il che rende possibile la creazione di cloud ibrido e pipeline ML ottimizzate.



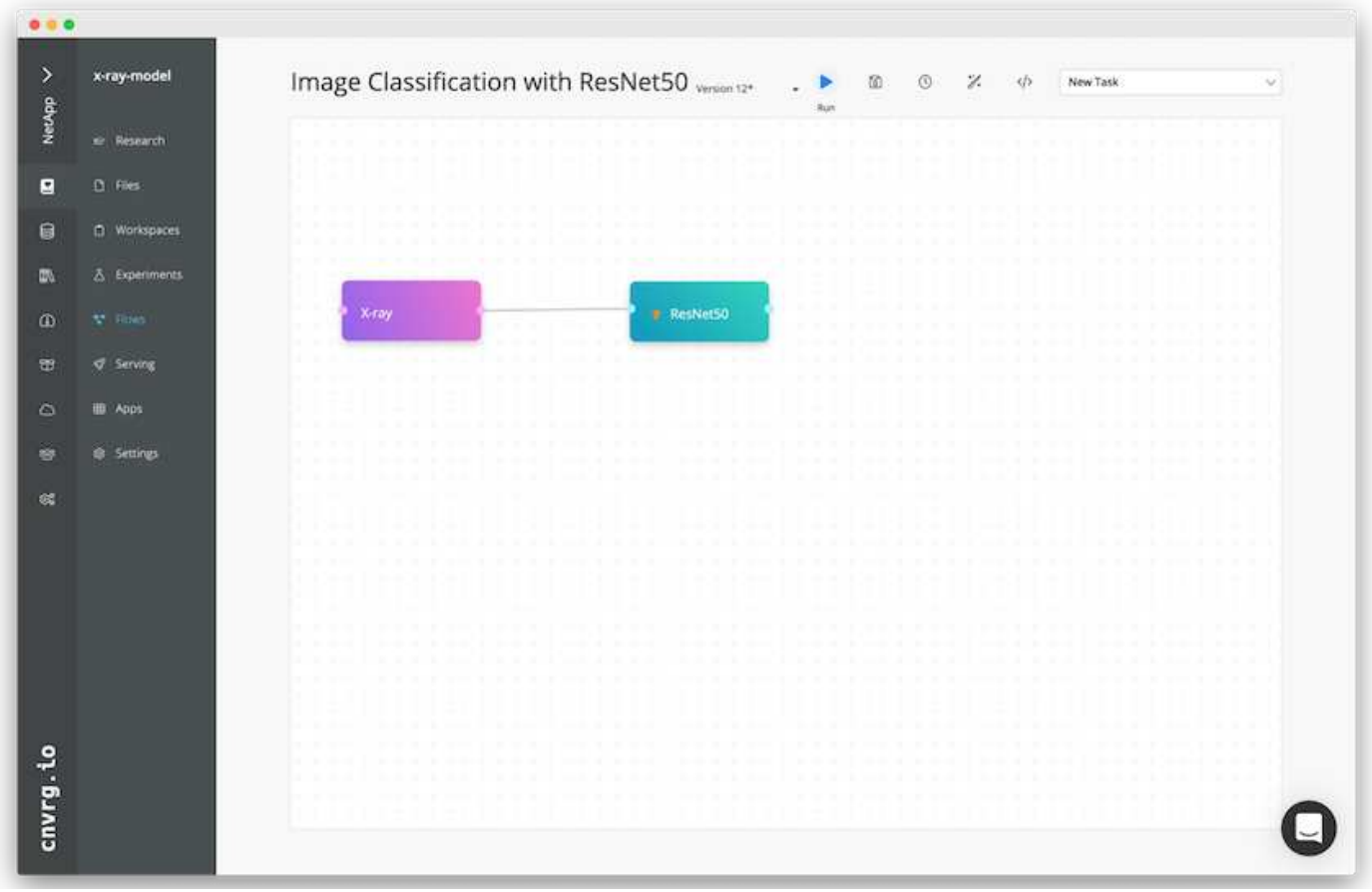
Creazione del flusso di raggi X del torace: Impostazione dei dati

Abbiamo aggiunto il nostro set di dati a un flusso appena creato. Quando si aggiunge il dataset, è possibile selezionare la versione specifica (commit) e indicare se si desidera la versione memorizzata nella cache. In questo esempio, è stato selezionato il commit memorizzato nella cache.



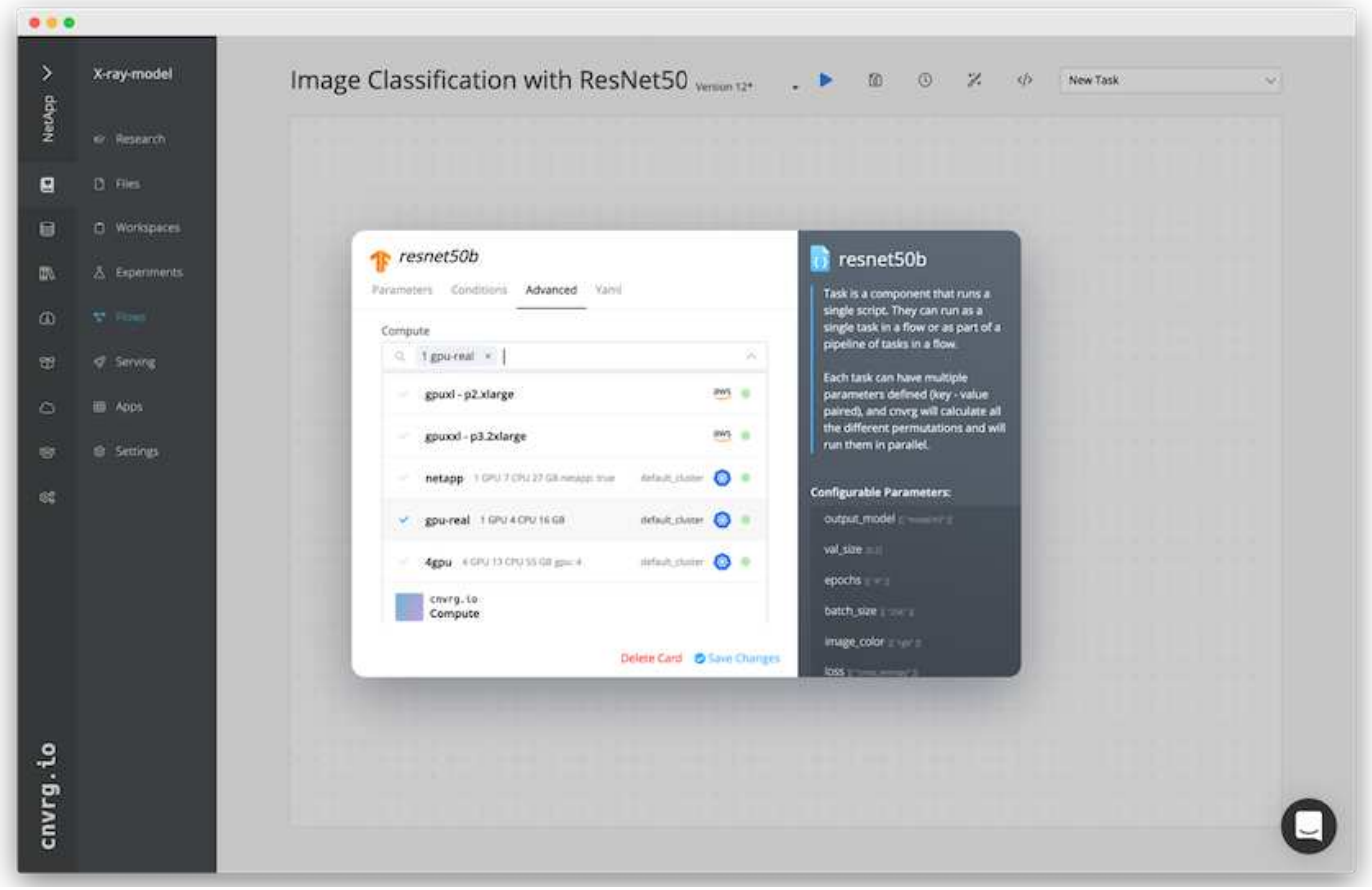
Creazione del flusso di raggi X del torace: Impostazione del modello di training: ResNet50

Nella pipeline, è possibile aggiungere qualsiasi tipo di codice personalizzato desiderato. In cnvrg è disponibile anche la libreria ai, una raccolta di componenti ML riutilizzabili. Nella libreria ai sono presenti algoritmi, script, origini dati e altre soluzioni che possono essere utilizzate in qualsiasi ML o flusso di deep learning. In questo esempio, è stato selezionato il modulo ResNet50 preinstallato. Abbiamo utilizzato parametri predefiniti come batch_size:128, epochs:10 e molto altro ancora. Questi parametri possono essere visualizzati nei documenti della ai Library. La seguente schermata mostra il nuovo flusso con il set di dati radiologici collegato a ResNet50.



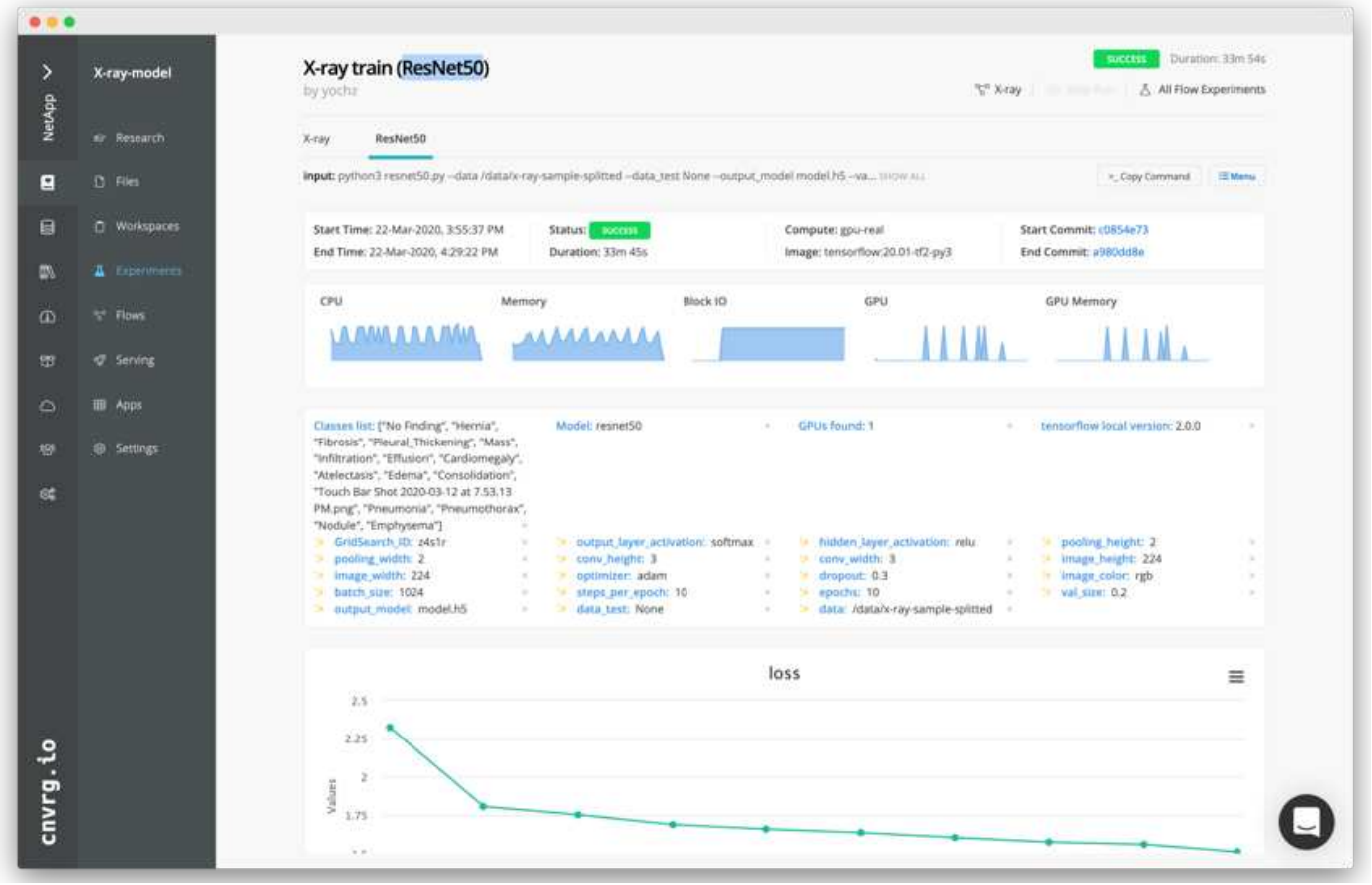
Definire la risorsa di calcolo per ResNet50

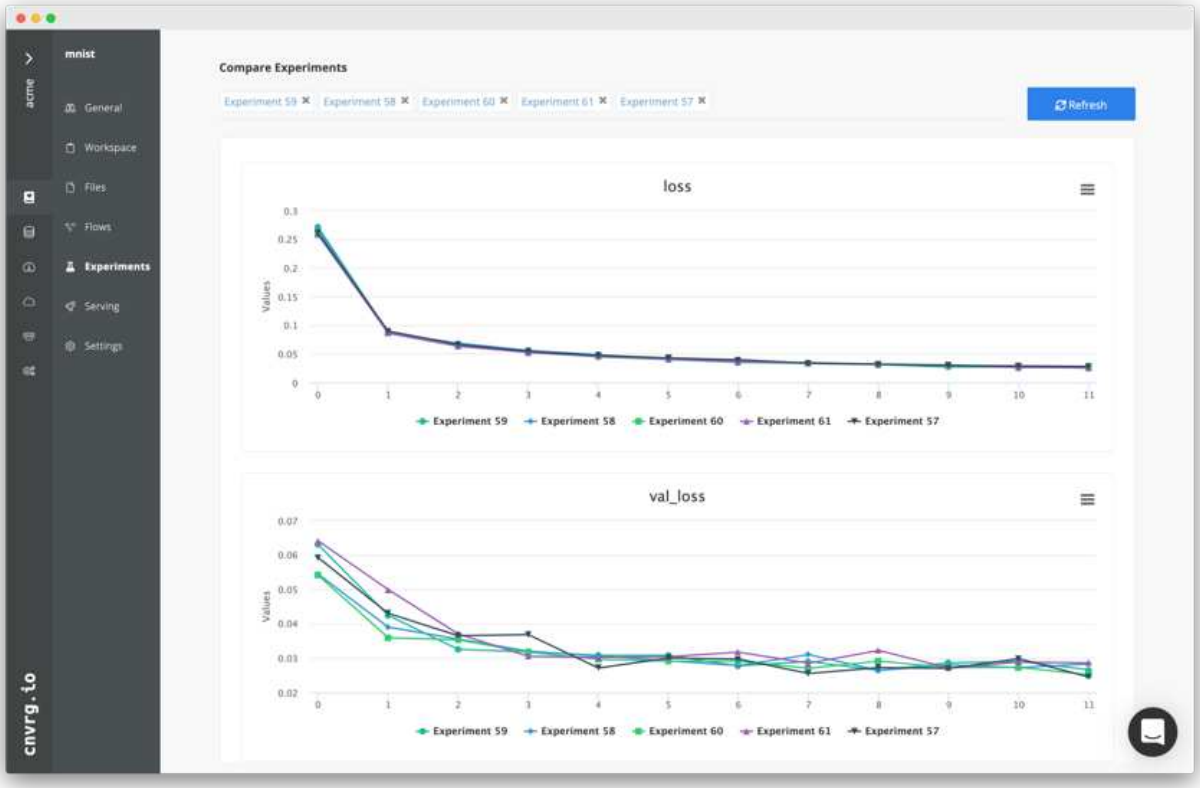
Ogni algoritmo o componente nei flussi cnvrg può essere eseguito su un'istanza di calcolo diversa, con un'immagine Docker diversa. Nella nostra configurazione, volevamo eseguire l'algoritmo di training sui sistemi NVIDIA DGX con l'architettura NetApp ONTAP ai. Nella figura seguente, è stato selezionato `gpu-real`, che è un modello di calcolo e una specifica per il nostro cluster on-premise. Abbiamo anche creato una coda di modelli e selezionato più modelli. In questo modo, se il `gpu-real` non è possibile allocare le risorse (se, ad esempio, altri data scientist le stanno utilizzando), quindi è possibile attivare la diffusione automatica del cloud aggiungendo un modello di cloud provider. La seguente schermata mostra l'utilizzo di gpu-real come nodo di calcolo per ResNet50.



Monitoraggio e monitoraggio dei risultati

Una volta eseguito un flusso, cnvrg attiva il motore di monitoraggio e tracciamento. Ogni esecuzione di un flusso viene documentata e aggiornata automaticamente in tempo reale. Hyperparameters, metriche, utilizzo delle risorse (utilizzo della GPU e altro ancora), versione del codice, artefatti, log, E così via sono disponibili automaticamente nella sezione Experiments (esperimenti), come mostrato nelle due schermate seguenti.


NetApp
X-ray-model
Research
Files
Workspaces
Experiments
Flows
Serving
Apps
Settings
cnvrg.io
X-ray train (ResNet50)
by yochz
SUCCESS
Duration: 33m 54s
X-ray
All Flow Experiments
X-ray
ResNet50
input: python3 resnet50.py --data /data/x-ray-sample-splitted --data_test None --output_model model.h5 --va...
Copy Command
Menu
Start Time: 22-Mar-2020, 3:55:37 PM
End Time: 22-Mar-2020, 4:29:22 PM
Status: SUCCESS
Duration: 33m 45s
Compute: gpu-real
Image: tensorflow:20.01-tf2-py3
Start Commit: c0854e73
End Commit: a980dd8e
CPU
Memory
Block IO
GPU
GPU Memory
Classes list: ["No Finding", "Hernia", "Fibrosis", "Pleural_Thickening", "Mass", "Infiltration", "Effusion", "Cardiomegaly", "Atelectasis", "Edema", "Consolidation", "Touch Bar Shot 2020-03-12 at 7.53.13 PM.png", "Pneumonia", "Pneumothorax", "Nodule", "Emphysema"]
Model: resnet50
GPUs found: 1
tensorflow local version: 2.0.0
GridSearch_ID: z4s1r
pooling_width: 2
image_width: 224
batch_size: 1024
output_model: model.h5
output_layer_activation: softmax
conv_height: 3
optimizer: adam
steps_per_epoch: 10
data_test: None
hidden_layer_activation: relu
conv_width: 3
dropout: 0.3
epochs: 10
data: /data/x-ray-sample-splitted
pooling_height: 2
image_height: 224
image_color: rgb
val_size: 0.2
loss
Values
2.5
2.25
2
1.75


acme
mnist
General
Workspace
Files
Flows
Experiments
Serving
Settings
cnvrg.io
Compare Experiments
Experiment 59
Experiment 58
Experiment 60
Experiment 61
Experiment 57
Refresh
loss
Values
0.3
0.25
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7 8 9 10 11
Experiment 59
Experiment 58
Experiment 60
Experiment 61
Experiment 57
val_loss
Values
0.07
0.06
0.05
0.04
0.03
0.02
0 1 2 3 4 5 6 7 8 9 10 11
Experiment 59
Experiment 58
Experiment 60
Experiment 61
Experiment 57