

《云原生的 ResearchOps/MLops 流水线平台构建》 结项报告

1 项目信息

(1) 项目名称

云原生的 ResearchOps/MLops 流水线平台构建

(2) 方案描述

如图 1 所示，方案基于 openEuler Linux 操作系统部署 Kubernetes 云原生编排系统，使用 Argo 生态圈 Argo CD、Argo Workflow、Argo Events 服务构建了 MLOps 流水线，使用 Kustomization、Helm 进行服务编排文件的渲染工作，涉及的支撑组件包含 Harbor 容器镜像仓库、Cert Manager 证书管理服务、PostgreSQL 结构化数据库服务、Kaniko 非 Docker 运行时容器镜像打包服务、NFS Server 共享存储服务、Let's Encrypt 开源证书签署服务，另外引用的外部服务包含 GCP 用于对象存储、GitHub 用于代码版本管理、Cloudflare 用于域名解析。

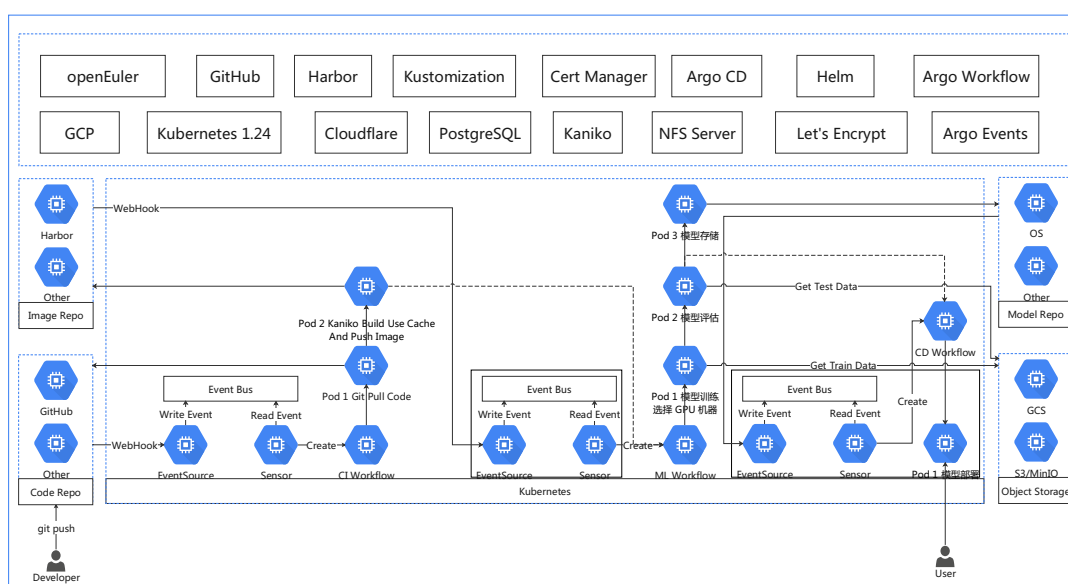


图 1 系统架构图

系统分为 CI（持续集成）、CT（持续训练）、CD（持续部署）三个阶段。

1) CI 阶段

当开发者修改一次代码并提交到 GitHub 时，GitHub 会通过 WebHook 将 Event 发送到 EventSource 模块，并写入事件总线。Sensor 订阅数据总线数据，根据数据启动 CI Workflow，对源代码进行集成，分三个步骤，第一步使用 git 工具获取源代码，第二步使用 Kaniko 进行容器镜像打包，打包镜像前从 Harbor 获取容器镜像缓存层以缩短打包时间，第三步将打包后的容器镜像及其缓存数据存储到 Harbor 系统内。

2) CT 阶段

CI 阶段成功结束后会将事件写入事件总线，Sensor 订阅到事件后启动 CT 流程。CT 分为三个步骤，分别是数据获取、模型训练和模型评估。目前，数据采用 GCP 对象存储服务，第一个步骤即从对象存储内获取训练集与测试集。第二个步骤通过 Kubernetes 的标签选择器选择 GPU 机器进行模型训练，读取代码仓库内存储的配置文件，读入训练参数以动态调整训练步骤及其他变量。当模型训练结束后，进入模型评估步骤，根据第一个步骤获取到的测试集数据和代码仓库内定义的评估质量阈值，评估若模型训练结果达到阈值则进入模型部署阶段，否则流水线结束，需要重新调整代码或训练参数。第三个步骤则是将代码、模型及数

据存入对象存储进行版本管理和触法 CD 阶段。

3) CD 阶段

在 CD 阶段将评估合格的模型部署通过 Kubernetes 标签选择器选择非 GPU 机器进行服务部署，并验证部署效果。

(3) 时间规划

时间规划见表 1 所示。

表 1 时间规划表

时间	任务	详情
6 月 1 日-6 月 30 日	现状调研	掌握现有基础流水线的部署方式，达到调试水平，调研业界相关研究工作的产品方案及实现方法，着重研究 Argo 及 Kubeflow 开源方案。
7 月 1 日-7 月 31 日	流水线实现	实现完整的 Machine Learning 生命周期管理自动化流水线，覆盖数据接入、验证、处理、特征分析、模型训练、验证与部署等流程。
8 月 1 日-8 月 15 日	用户界面调研设计	完成高度自动化的任务编排、执行、管理服务，设计使用友好的用户界面。
8 月 16 日-8 月 30 日	版本控制功能实现	完成数据、特征、模型、应用独立版本控制能力，用户可基于特定版本触发流水线服务。
9 月 1 日-9 月 15 日	测试报告	设计系统测试方案、进行性能评估，形成质量报告。
9 月 16 日-9 月 30 日	合并代码	编写 README.md 文件，并向社区提交代码。

2 项目总结

(1) 项目产出

项目产出要求及状态见表 2 所示。

项目产出要求	完成状态	成果
提供完整的 ML 生命周期管理自动化流水线，覆盖数据接入、验证、处理、特征分析、模型训练、验证与部署等流程。	已完成	完整的 ML 生命周期管理自动化流水线已实现，且完成了 MVP Demo 演示。
提供高度自动化的任务编排、执行、管理服务，设计使用友好的用户界面。	已完成	使用 Kubernetes 编排、执行、管理服务，复用 Argo 用户界面。
提供数据、特征、模型、应用独立版本控制能力，用户可基于特定版本触发流水线服务。	已完成	版本控制能力基于 Git 标签。
平台设计需满足应用容器化、面向微服务架构、支持容器编排调度、资源可弹性和声明式 API 等特性。	已完成	成果符合要求。
平台支撑 2+AI 服务落地，提升模型部署、迭代效率 300%。	已完成	成果符合要求。

(2) 方案进度

根据原定方案和时间规划，当前进度滞后半个月左右，9 月下旬完成了测试报告，生成 PR，合并代码的工作将按照主办方要求在 10 月底前完成。

(3) 遇到的问题及解决方案

在流水线构建过程中最大的问题是多个开源组件组合编排，版本的同期性尤为重要，采用不同时间段发布的版本可能存在特性不匹配的问题，最终全部采用最新版本的开源组件解

决了问题。另外，在部署流水线的过程中为节省资源，尝试使用多个树莓派组成集群部署，但是在安装 NFS Server 时遇到了官方容器镜像无 ARM 版本问题，自行构建镜像后解决。后在 Kubernetes 启动 Pod 时报错共享存储无法挂载，根据报错信息通过搜索引擎未查结果，经阅读代码库 Issue 获得解决方案，需要在每台 Node 上安装 NFS 库文件才能成功挂载共享存储。

(4) 项目完成质量

此流水线项目较为复杂，预期较高，但在实际开发过程中发现暑期时间难以高标准实现所有功能，均采用简版实现，因此在开发质量上尚为到预期最佳，存在较多方向优化，在开源之夏任务结束后将会继续优化，使其达到预期效果。

(5) 与导师沟通及反馈情况

在项目开展期间，每两周与导师沟通 1 次，并多次参与社区双周例会，汇报项目成果。在最近一次即将结束的双周例会上汇报了项目的最终演示版本，导师评价较好，对项目的完成度给予了认可。

3 项目部署测试报告

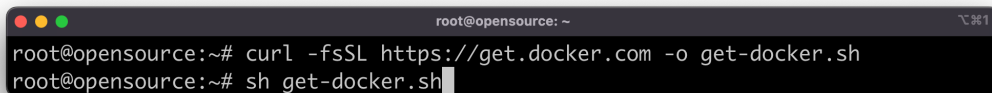
(1) 准备工作

项目部署需要 3 个前置条件，分别是 1 台安装了 openEuler 操作系统的服务器，一个托管在 Cloudflare 的域名，一个 GCP 账户。

测试报告中使用的服务器 IP 为 192.168.0.14，后文以 \${IP} 代替；使用的域名为 abu.pub，在 Cloudflare 中将 *.abu.pub 以 A 记录仅 DNS 方式解析到 \${IP}；使用的 GCP 账户为免费账户。

(2) 在 openEuler 操作系统上部署 Docker 容器引擎服务

部署 Docker 容器引擎服务采用官网提供的脚本部署方式，如图 2 所示。

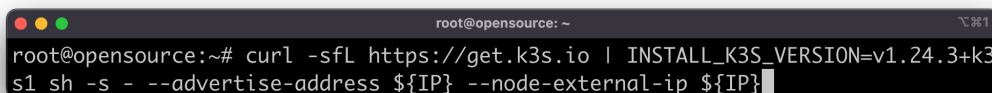


```
root@opensource: ~
root@opensource:~# curl -fsSL https://get.docker.com -o get-docker.sh
root@opensource:~# sh get-docker.sh
```

图 2 在 openEuler 操作系统上部署 Docker 容器引擎服务

(3) 在 openEuler 操作系统上部署 Kubernetes 容器编排服务

部署 Kubernetes 容器编排服务采用 K3S 发行版本，部署 v1.24.3 版本，如图 3 所示。

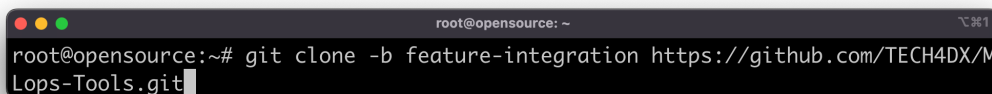


```
root@opensource: ~
root@opensource:~# curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.24.3+k3s1 sh -s - --advertise-address ${IP} --node-external-ip ${IP}
```

图 3 在 openEuler 操作系统上部署 Kubernetes 容器编排服务

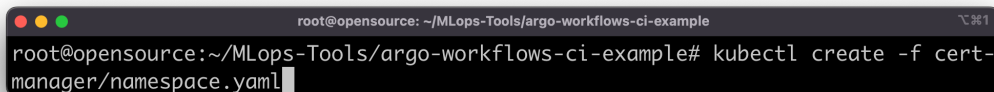
(4) 在 Kubernetes 容器编排服务上部署 cert-manager 证书管理服务

在 Kubernetes 容器编排服务上部署 cert-manager 证书管理服务，如图 4-6 所示。



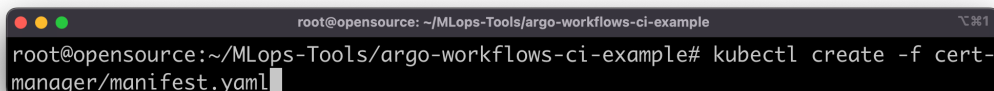
```
root@opensource: ~
root@opensource:~# git clone -b feature-integration https://github.com/TECH4DX/M
Lops-Tools.git
```

图 4 获取编排文件代码库



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f cert-manager/namespace.yaml
```

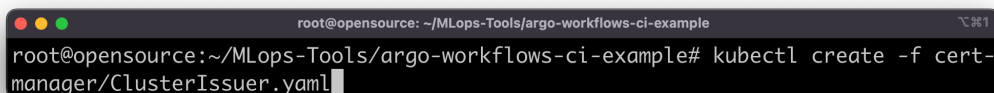
图 5 创建命名空间



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f cert-manager/manifest.yaml
```

图 6 部署 cert-manager 服务

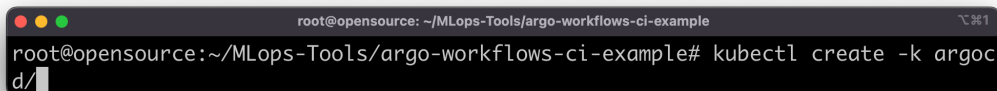
使用 Cloudflare 生成的 Token 创建 ClusterIssuer 资源，如图 7 所示。



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f cert-manager/ClusterIssuer.yaml
```

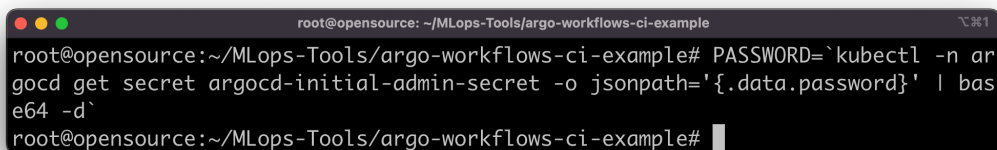
图 7 使用 Cloudflare 生成的 Token 创建 ClusterIssuer 资源

(5) 部署 Argo CD



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -k argocd/
```

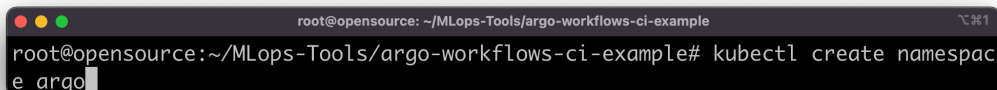
图 8 部署 Argo CD 服务



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# PASSWORD=`kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath='{.data.password}' | base64 -d`
root@opensource:~/MLops-Tools/argo-workflows-ci-example#
```

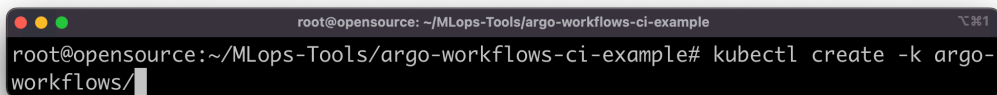
图 9 获取 Argo CD 鉴权信息

(6) 部署 Argo Workflows



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create namespace argo
```

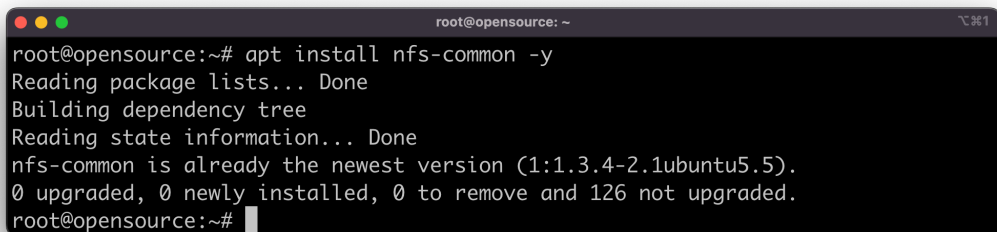
图 10 创建命名空间



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -k argo-workflows/
```

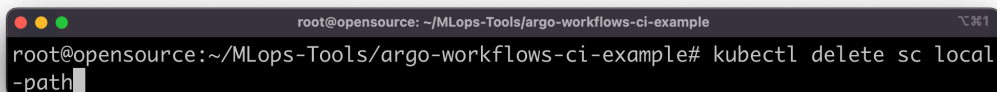
图 11 部署 Argo Workflows 服务

(7) 部署 NFS Server



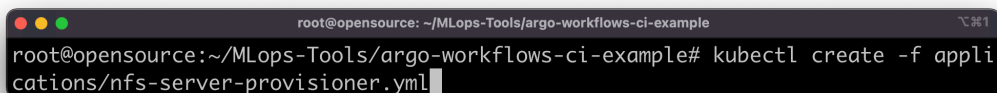
```
root@opensource: ~
root@opensource:~# apt install nfs-common -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
nfs-common is already the newest version (1:1.3.4-2.1ubuntu5.5).
0 upgraded, 0 newly installed, 0 to remove and 126 not upgraded.
root@opensource:~#
```

图 12 安装 NFS Common 软件



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl delete sc local-path
```

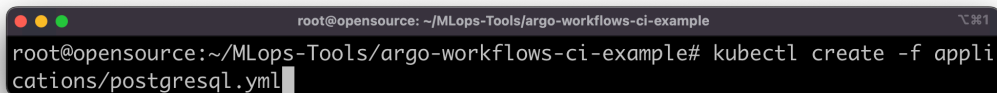
图 13 清理默认 Storage Class 资源



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f applications/nfs-server-provisioner.yml
```

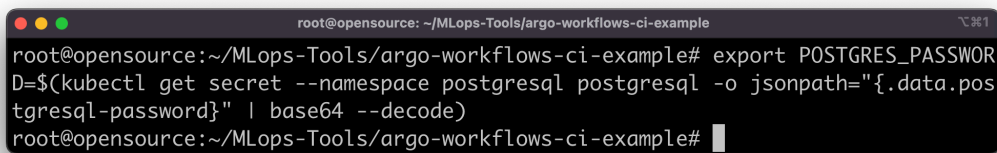
图 14 部署 NFS Server 服务

(8) 部署 PostgreSQL



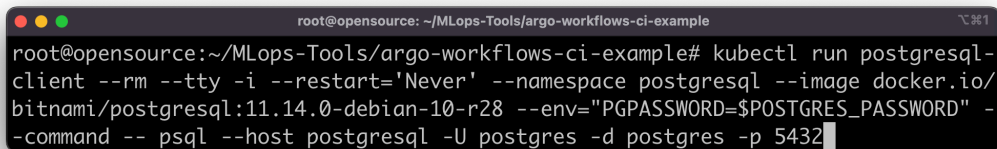
```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f applications/postgresql.yml
```

图 15 部署 PostgreSQL



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# export POSTGRES_PASSWORD=$(kubectl get secret --namespace postgresql postgresql -o jsonpath="{.data.postgresql-password}" | base64 --decode)
root@opensource:~/MLops-Tools/argo-workflows-ci-example#
```

图 16 获取数据库密码

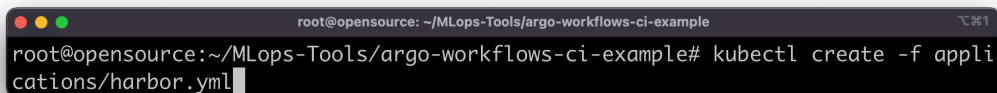


```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl run postgresql-client --rm --tty -i --restart='Never' --namespace postgresql --image docker.io/bitnami/postgresql:11.14.0-debian-10-r28 --env="PGPASSWORD=$POSTGRES_PASSWORD" --command -- psql --host postgresql -U postgres -d postgres -p 5432
```

图 17 登陆数据库

创建 registry、notary_signer、notary_server 三个数据库，用于 Harbor 存储数据。

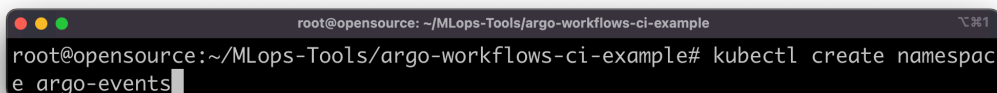
(9) 部署 Harbor



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f applications/harbor.yml
```

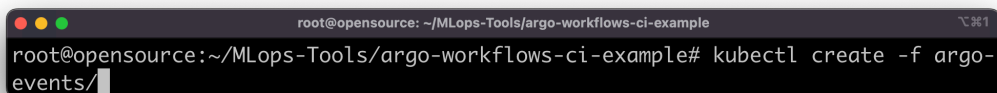
图 18 部署 Harbor

(10) 部署 Argo Events



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create namespace argo-events
```

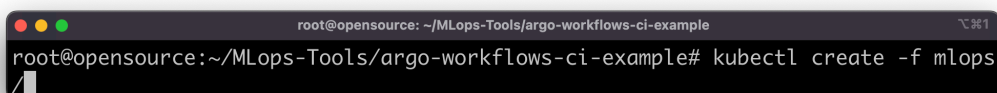
图 19 创建命名空间



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f argo-events/
```

图 20 部署 Argo Events

(11) 部署 ML0ps



```
root@opensource: ~/MLops-Tools/argo-workflows-ci-example
root@opensource:~/MLops-Tools/argo-workflows-ci-example# kubectl create -f mlops/
```

图 21 部署 ML0ps

(12) 部署 MNIST 手写数字识别的流水线

```
root@opensource: ~/Mlops-Tools/argo-workflows-ci-example
root@opensource:~/Mlops-Tools/argo-workflows-ci-example# kubectl create -f mnist
```

图 22 部署流水线

(13) 触发 MNIST 手写数字识别的流水线

通过修改代码，提交变更到 GitHub 的方式触发流水线，如图 23 所示。

```
root@opensource: ~/mnist-serving
root@opensource:~/mnist-serving# git add README.md
root@opensource:~/mnist-serving# git commit -m "Demo Test."
[master 9807792] Demo Test.
 1 file changed, 1 insertion(+), 1 deletion(-)
root@opensource:~/mnist-serving# git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 289 bytes | 289.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:abuxliu/mnist-serving.git
   f9cba6f..9807792 master -> master
root@opensource:~/mnist-serving#
```

图 23 触发流水线

(14) 查看流水线 CI 阶段状态

通过 <https://argo.abu.pub> 用户界面查看流水线 CI 阶段状态，此用户界面未配置鉴权信息，界面如图 24 所示。

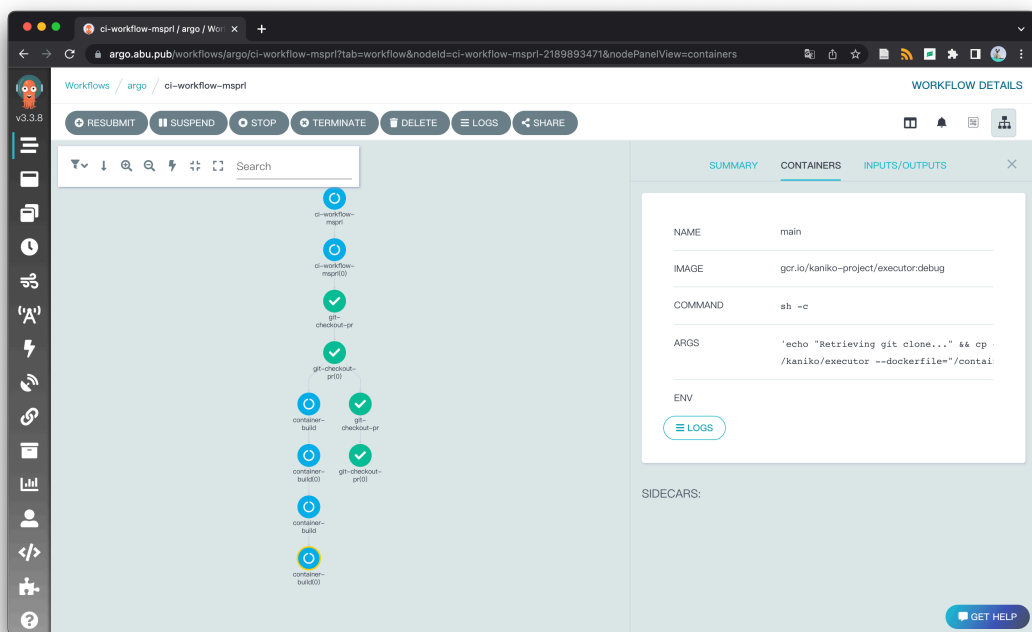


图 24 查看流水线 CI 阶段状态

(15) 查看流水线 CT 阶段状态

通过 <https://argo.abu.pub> 用户界面查看流水线 CT 阶段状态，此用户界面未配置鉴权信息，界面如图 25 所示。

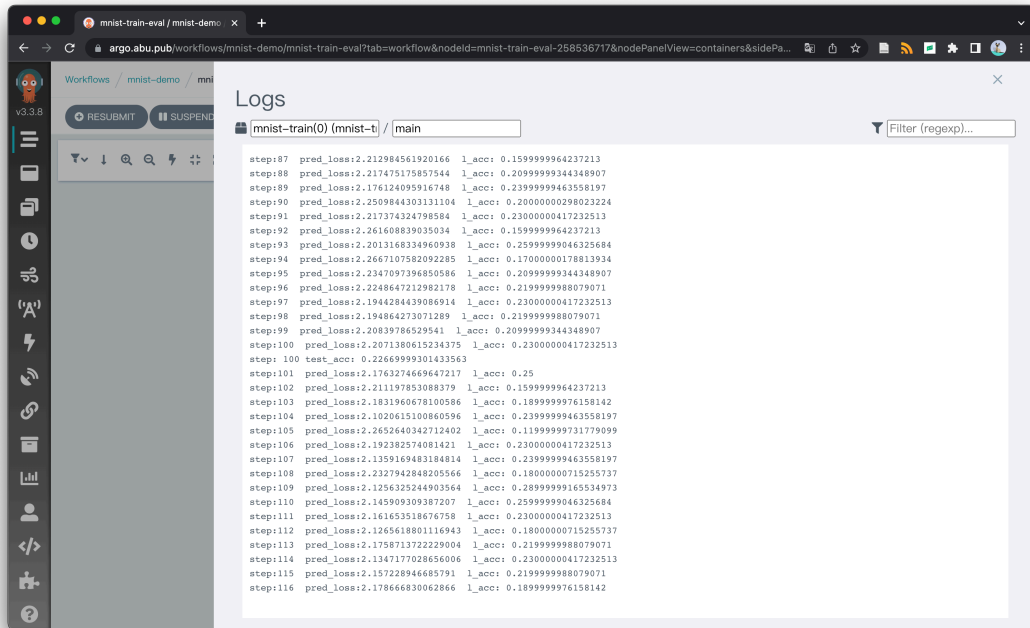


图 25 查看流水线 CT 阶段状态

(16) 查看流水线 CD 阶段状态

通过 <https://argocd.abu.pub> 用户界面查看流水线 CD 阶段状态，此用户界面鉴权信息用户名为 admin，密码为图 9 获取的密码，界面如图 26 所示。

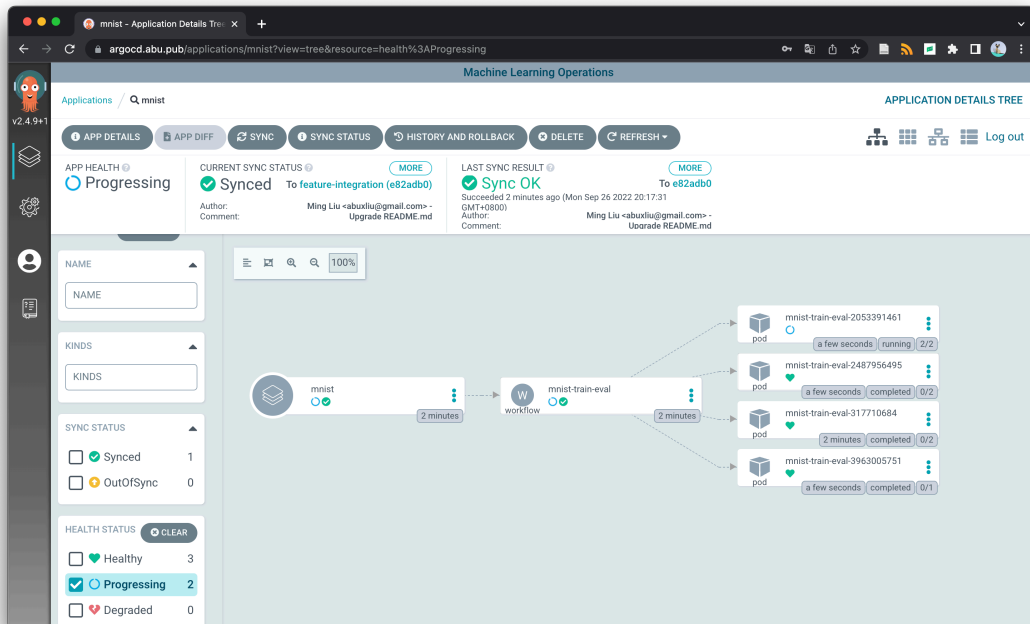


图 26 查看流水线 CD 阶段状态

（17）验证部署完成的模型

通过 <https://mnist.abu.pub> 用户界面查看流水线 CD 阶段状态，此用户界面未配置鉴权信息，界面如图 27 所示。

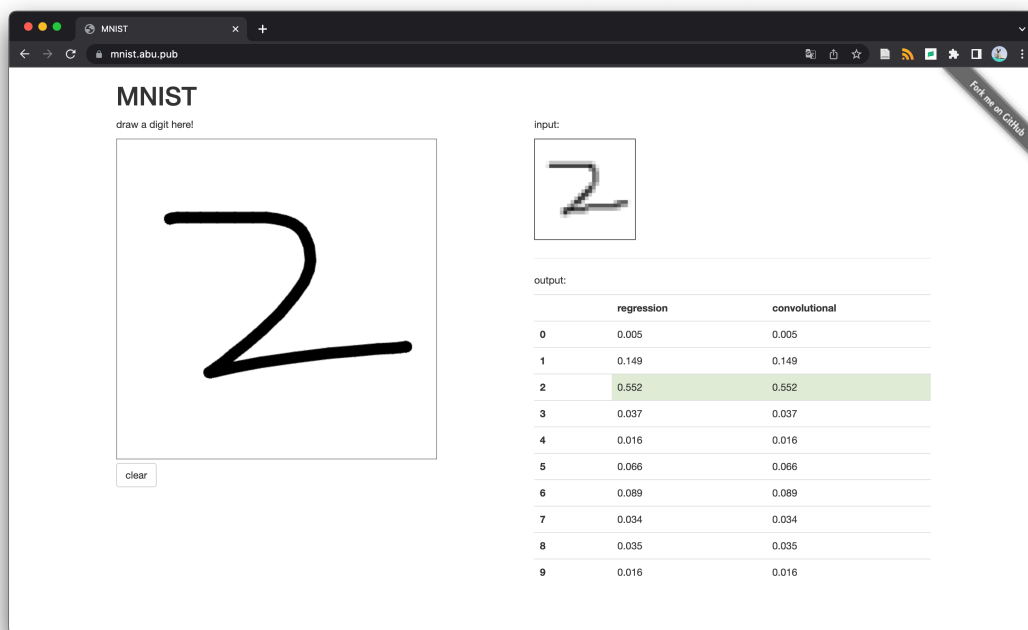


图 27 验证部署完成的模型