Развертывание на кластере

immediate

1 марта 2025 г.

1 Получите реквизиты доступа к кластеру, проверьте что добавлено домашнее пространство имен.

Пример, после получения реквизитов и их сохранения в файл *.kubeconfig*.

```
export KUBECONFIG=$KUBECONFIG:/path/to/filename.kubeconfig
kubectl get papp -n pu-username
```

Результатом должна быть пустая таблица запущенных приложений.

2 Добавьте в домашнее пространство имён компоненты пользователя:

1. git repo для ветки развертывания
2. secrets с реквизитами доступа
3. docker repo для образов
4. реквизиты для api
5. репозиторий пакетов Python с реквизитами

Все компоненты пользователя, кроме реквизитов для доступа к модулю (auth), добавляются на фреймворк командой

```
kubectl apply -f <filename>
```

2.1 Реквизиты Git-репозитория

```
apiVersion: v1
kind: Secret
metadata:
  namespace: pu-username
  name: git-http-based-url-credentials
data:
  credentials: aHR0cHM6Ly<...>GUtbmFtZS5naXQ=
```

Поле *credentials* можно сгенерировать скриптом:

```
printf "https://username:12345678901234567890@platform-dev-cs-hse.
objectoriented.ru/forgejo/lab-name/bm99-module-name.git" | base64
```

Здесь *12345678901234567890* - это пароль от аккаунта Forgejo или токен приложения, созданный в настройках аккаунта Forgejo.

2.2 Git-репозиторий

```
apiVersion: "unified-platform.cs.hse.ru/v1"
kind: Repository
metadata:
  name: git-source-repository
  namespace: pu-username
spec:
  sourceRepository:
    host: platform-dev-cs-hse.objectoriented.ru
    gitHTTPBasedURLCredentials: git-http-based-url-credentials
```

Поле *spec/sourceRepository/gitHTTPBasedURLCredentials* должно совпадать с полем *metadata/name* компонента реквизитов.

2.3 Реквизиты репозитория пакетов Python

```
apiVersion: v1
kind: Secret
metadata:
  namespace: pu-username
  name: python-http-based-url-credentials
data:
  credentials: aHR0cHM6Ly91c2Vyb<...>by9weXBpL3NpbXBsZQ==
```

Поле *credentials* можно сгенерировать скриптом:

```
printf "https://username:12345678901234567890@platform-dev-cs-
hse.objectoriented.ru/forgejo/api/packages/unified-platform-resources/
pypi/simple" | base64
```

Здесь ссылка всегда одна и та же, и не меняется для других репозиториев.

2.4 Репозиторий пакетов Python

```
apiVersion: "unified-platform.cs.hse.ru/v1"
kind: Repository
metadata:
  name: python-package-registry
  namespace: pu-username
spec:
  packageRegistry:
    host: platform-dev-cs-hse.objectoriented.ru
    pythonHTTPBasedURLCredentials: python-http-based-url-credentials
```

Поле *spec/packageRegistry/pythonHTTPBasedURLCredentials* должно совпадать с полем *metadata/name* компонента реквизитов.

2.5 Реквизиты репозитория образов Docker.

Создать файл с реквизитами можно командой

```
docker login registry-platform-dev-cs-hse.objectoriented.ru
```

Если в файле конфигурации (*.docker/config.json*) есть пароль, то поле *.dockerconfigjson* из файла ниже можно создать через

```
cat .docker/config.json | base64
```

Иначе можно создать такой секрет через *kubectl*:

```
kubectl create secret docker-registry docker-config-json-credentials \
    --dry-run \
    --docker-server=registry-platform-dev-cs-hse.objectoriented.ru \
    --docker-username=<your-forgejo-username> \
    --docker-password=<forgejo-token> \
    --namespace=pu-username \
    -o yaml > docker-secret.yaml
```

В обоих случаях (через kubectl или с ручным редактированием шаблона ниже) должен получиться следующий файл:

```
apiVersion: v1
kind: Secret
metadata:
  name: docker-config-json-credentials
  namespace: pu-username
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: aHR0cHM6Ly91c2VybmFtZ<...>9weXBpL3NpbXBsZQ==
```

Официальная документация создания такого компонента *Secret* - `https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/`

2.6 Компонент репозитория образов Docker

```
apiVersion: "unified-platform.cs.hse.ru/v1"
kind: Repository
metadata:
  name: docker-image-registry
  namespace: pu-username
spec:
  imageRegistry:
    host: registry-platform-dev-cs-hse.objectoriented.ru
    dockerDockerConfigJsonCredentials: docker-config-json-credentials
```

Поле *spec/imageRegistry/dockerDockerConfigJsonCredentials* должно совпадать с полем *metadata/name* компонента реквизитов.

2.7 Реквизиты API для доступа пользователей к модулю

1. Установите утилиту *htpasswd*. Она часто уже присутствует в системе, но если нет, можно установить HTTP сервер Apache, частью которого она является - `https://httpd.apache.org/download.cgi` * Также есть альтернативы утилите *htpasswd*, например python-скрипт `https://gist.github.com/eculver/1420227`

2. Откройте терминал в папке, где будет создан файл с реквизитами

3. Создайте файл с реквизитами:

   ```
   htpasswd -c auth <name-of-first-user>
   ```

   Должен появиться запрос на ввод пароля, после чего создастся файл *auth*.

4. Добавьте пользователей при необходимости. Желательно добавить пользователя с именем **developer**.

```
htpasswd auth <name-of-second-user>
htpasswd auth developer
htpasswd auth alice
htpasswd auth bob
```

5. Загрузите полученный файл с реквизитами на фреймворк.

```
kubectl create secret generic basic-auth-credentials -n pu-username
--from-file=auth
```

3 Загрузите образ контейнера модуля в реестр

```
docker push registry-platform-dev-cs-hse.objectoriented.ru/
lab-name/bm99-module-container-name:12ab345
```

4 Добавьте изменения в git

Добавьте сами изменения в основную ветку, сделайте запрос на слияние из основной ветки в ветку развертывания, выполните слияние.

В дальнейшем при обновлении модуля работайте в основной ветке и добавляйте в ветку развёртывания уже готовые изменения.

4.0.1 Подготовьте и загрузите компонент приложения

Пример:

```
apiVersion: "unified-platform.cs.hse.ru/v1"
kind: PlatformApp
metadata:
  name: pu-username-pa-bm99
  namespace: pu-username
spec:
  appSourceCode:
    sourceRepositoryRef:
      name: git-source-repository
      sourceRepositoryKind: git
      credentialsKind: HTTPBasedURLCredentials
    repositoryPath: app
    revision: deploy
  repositories:
```

```
  - name: docker-image-registry
    imageRegistryRef:
      name: docker-image-registry
      imageKind: docker
      credentialsKind: DockerConfigJsonCredentials
  - name: python-package-registry
    packageRegistryRef:
      name: python-package-registry
      packageKind: python
      credentialsKind: HTTPBasedURLCredentials
apisCredentials:
  - name: bm99-apis
    secretRef:
      name: basic-auth-credentials
```

Здесь:

- *pu-username-pa-bm99* - пространство имён *приложения*. Выше создавались компоненты *пользователя*. Доступ к репозиториям образов и пакетов есть у Вас как у пользователя. Здесь создаётся пространство имён для компонентов самого приложения. Сами эти компоненты создаются в пункте 6.
- *spec/appSourceCode* - Git-репозиторий с кодом приложения
  - *sourceRepositoryRef* - ссылка на компонент репозитория, созданный выше. Имя (‘name‘) компонента здесь должно совпадать с созданным ранее компонентом репозитория. *git-source-repository* это пример имени.
  - *repositoryPath* - относительный путь внутри репозитория Git к папке, где хранятся манифесты компонентов приложения из пункта 6.
  - *revision* - ветка репозитория, в которой размещаются версии приложения для развёртывания на фреймворке.
- *spec/repositories* - репозитории пакетов Python и образов Docker. Для обоих типов репозиториев есть два имени *name*.
  - "Внешнее"имя, *spec["repositories"][0]["name"]* - это имя, по которому обращаются компоненты приложения. Например, ML-компонент указывает название образа Docker, а так же название компонента репозитория, через который можно найти этот образ. Там указывается именно это название.
  - "Внутреннее"имя, *spec["repositories"][0]["imageRegistryRef"]["name"]* - это имя компонента *пользователя*, который здесь используется. С этим именем компоненты создаются в пунктах выше.

- *spec/apisCredentials* - реквизиты API для предоставления внешним пользователям доступа к развёрнутому приложению.
  - *name* - "внешнее"название компонента API. Компоненты приложения будут обращаться к этому компоненту API как *<name>-creds*. Например, *appname-apis-creds* в данном случае.
  - *secretRef/basic-auth-credentials* - "внутреннее"название компонента, созданного выше, в котором хранятся реквизиты пользователей.

Компонент приложения загружается на фреймворк той же командой, что и компоненты пользователя,

```
kubectl apply -f <filename>
```

5 Проверьте, что приложение создано

Проверьте, что создано пространство имен приложения, получите статус mlcmp через describe

Пример:

```
kubectl get pods -n pu-username-pa-bm99
NAME                                                              READY   STATUS
RESTARTS   AGE
modulename-mlcmp-default-s3-deployment-f5d1234bb-kxh5q  1/1     Running
0          1d

kubectl describe pod modulename-mlcmp-default-s3-deployment-f5d1234bb-
kxh5q -n pu-username-pa-bm99
```

6 Загрузите используемые данные (временный пункт, в дальнейшем будет переход на Git LFS)

Пример загрузки параметров модели через сервис *files*.

Согласно манифесту ML-компонента, известно следующее:

- Контейнер считывает веса из локального пути */home/path/to/model.joblib*, что указано в *spec/mlService/inference/model/modelPath*. - Ящик *model-box* монтируется к локальному пути внутри контейнера */home/path/to*, что указано в *spec/connectedBoxes/[0]/path* - В ящике *model-box* монтируется путь внутри ящика *users/developer/file_groups/models*, что указано в *spec/connectedBoxes/[0]/mountS3Box/subPath*. - Ящик *model-box* соответствует компоненту *box* с "внешним"именем *mymodule-data-box*.

Также, фреймворк работает таким образом, что

- Если пользователь *developer* загружает свои данные в любой ящик, внутри ящика путь к этим данным будет *users/developer/file_ groups/....* - В корне ящика не должно быть файлов, только папки. Файлы в ящике должны помещаться уже далее в папки или подпапки.

Соответственно, если пользователь *developer* загрузит данные в путь *models/model.joblib*, то контейнер должен правильно прочитать веса.

1. Полный путь к данным внутри ящика становится *users/developer/file_ groups/models/model.joblib*.
2. При монтировании часть пути *users/developer/file_ groups/models* остаётся внутри ящика, а часть *model.joblib* добавляется к пути внутри контейнера.
3. Начало пути внутри контейнера - */home/path/to/* , что вместе с путём в смонтированной папке даёт */home/path/to/model.joblib*.
4. Именно по этому пути сервис ожидает считать веса модели.

Для загрузки весов можно использовать следующие запросы:

```
curl -X PUT https://platform-dev-cs-hse.objectoriented.ru/pu-username-pa-bm99/
files/mymodule-data-box/models/model.joblib -H \
"Content-Type: application/json" -u "developer:<password>"
```

Это запрос:

1. К приложению *pu-username-pa-bm99*.
2. К файловому API (у которого endpoint всегда *files*).
3. К ящику *mymodule-data-box*.
4. К пути *models/model.joblib* внутри ящика.
5. От имени пользователя *developer*, пароль которого - *<password>*.

Если доступ к API есть, то должен прийти ответ в виде JSON

```
{
    "name": "pu-username-pa-bm99/files/mymodule-data-box/models/model.joblib",
    "presigned_put_url": "<long pre-signed uploading link>"
}
```

Теперь можно загрузить сами веса по pre-signed ссылке.

```
curl -X PUT --data-binary @local/path/to/model.joblib \
"<long pre-signed uploading link>"
```

Здесь,

- *local/path/to/model.joblib* - локальный путь к файлу с весами относительно текущей папки, из которой делается запрос.

Если файл с моделью большой, то можно использовать другую команду:

```
curl -X PUT -T local/path/to/model.joblib \
"<long pre-signed uploading link>"
```

Здесь перед путём к имени файла нет символа '@'.

6.0.1 Проверка модуля и доступ через API

1. Проверьте, что в пространстве имен приложения развернуты deployment и в их describe нет ошибок

2. Узнайте URL развернутого mlcmp по шаблону

   ```
   https://platform-dev-cs-hse.objectoriented.ru/pu-username-pa-bm99/
   API_NAME/predict
   ```

   - Здесь *API_NAME* - это пункт *spec/restfulAPI/path* из манифеста API-компонента вызова расчётов.

3. Реализуйте пробный скрипт вызова API из jupyter или curl по примеру demo-project с теми же данными, что и тест на pytest ранее

   Пример:

   (a) Загрузить данные в S3 через API 'files'.

   (b) Создать файл с описанием входных и выходных данных запуска, здесь 'data.json'.

   ```
   {
     "inputs": [
       {
         "name": "image",
         "data": "uploads/test_image.png",
         "datatype": "FILE",
         "content_type": "image/png",
         "shape": [128, 128]
       }
     ],
     "output_fields": [
       {
         "name": "predict",
         "datatype": "INT32"
       }
     ]
   }
   ```

(c) Отправить запрос, передав ‘data.json‘ в его теле.

```
curl -X POST -d @data.json https://platform-dev-cs-hse.objectoriented
.ru/pu-username-pa-bm99/API_NAME/predict -H
"Content-Type: application/json" -u "<username>:<password>"
```

Здесь *<username>:<password>* - это данные, создаваемые в пункте "Развёртывание на кластере"в компоненте *basic-auth-credentials*.

4. Проверьте, что модуль через API возвращает ожидаемые результаты, такие же как в локальном тесте.