Argo

Prerequisites

Install Argo in your Kubernetes cluster

- A member from the DevOps team can accomplish this.

Have the appropriate permissions in place including:

- access to the kubernetes cluster
- service account permissions for argo (required for creating pods in any namespace)

For the purposes of this research Former user (Deleted) was the one who assisted me. For any extra information about installing Argo, please get in touch with him. Documentation about how to install Argo can be found at: https://github.com/argoproj/argo/blob/master/docs/getting-started.md

Create a YAML file describing the workflow

To replicate the existing behavior which executed all the jobs sequentially a file like this can be created (referred to below as `order-proposals-pipeline.yaml`):

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: order-proposals-pipeline-
spec:
  entrypoint: sequential
  imagePullSecrets:
    - name: my-secret
  templates:
  - name: check-exasol-status
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask check-exasol-status']
      envFrom:
      - secretRef:
          name: production-secret
  - name: daily-import-products
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-products --upsert=True --only-relevant=True'
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-purchase-orders
```

```
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-purchase-orders --interval=1 --upsert=True']
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-customer-orders
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-customer-orders --interval=1 --upsert=True']
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-real-demand
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-real-demand --day=`date --iso-8601` --interv
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-create-proposals
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask create-proposal-for-all-relevant']
      envFrom:
        - secretRef:
            name: production-secret
  - name: sequential
    dag:
      tasks:
      - name: check-exasol-status
        template: check-exasol-status
      - name: daily-import-products
        dependencies: [check-exasol-status]
        template: daily-import-products
      - name: daily-import-purchase-orders
        dependencies: [daily-import-products]
        template: daily-import-purchase-orders
      - name: daily-import-customer-orders
        dependencies: [daily-import-purchase-orders]
        template: daily-import-customer-orders
      - name: daily-import-real-demand
        dependencies: [daily-import-customer-orders]
        template: daily-import-real-demand
      - name: daily-create-proposals
```

```
        dependencies: [daily-import-real-demand]
        template: daily-create-proposals
```

To speed things up a bit, we could think of parallelizing some of the jobs like so (referred to below as `order-proposals-pipeline-parallel.yaml`):

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: order-proposals-pipeline-
spec:
  entrypoint: diamond
  imagePullSecrets:
    - name: my-secret
  templates:
  - name: check-exasol-status
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask check-exasol-status']
      envFrom:
      - secretRef:
          name: production-secret
  - name: daily-import-products
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-products --upsert=True --only-relevant=True'
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-purchase-orders
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-purchase-orders --interval=1 --upsert=True']
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-customer-orders
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask import-customer-orders --interval=1 --upsert=True']
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-import-real-demand
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
```

```
      command: ['/bin/sh',
                '-c',
                'flask import-real-demand --day=`date --iso-8601` --interv
      envFrom:
        - secretRef:
            name: production-secret
  - name: daily-create-proposals
    container:
      image: registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce13e8
      command: ['/bin/sh',
                '-c',
                'flask create-proposal-for-all-relevant']
      envFrom:
        - secretRef:
            name: production-secret
  - name: diamond
    dag:
      tasks:
      - name: check-exasol-status
        template: check-exasol-status
      - name: daily-import-products
        dependencies: [check-exasol-status]
        template: daily-import-products
      - name: daily-import-purchase-orders
        dependencies: [daily-import-products]
        template: daily-import-purchase-orders
      - name: daily-import-customer-orders
        dependencies: [daily-import-products]
        template: daily-import-customer-orders
      - name: daily-import-real-demand
        dependencies: [daily-import-products]
        template: daily-import-real-demand
      - name: daily-create-proposals
        dependencies: [daily-import-purchase-orders, daily-import-customer
        template: daily-create-proposals
```

More examples about how to create certain workflows can be found at: https://github.com/argoproj/argo/blob/master/examples/README.md

Run the workflow using the CLI tool

A workflow can be created by running the following command from the terminal:

```
argo submit --watch order-proposals-pipeline.yaml -n auto-ordering-1391899
```

The `--watch` argument is optional and can be used in order to track the progress of the workflow from the terminal.

The `-n` argument is also optional, but often necessary, and indicated in which namespace the pods for running the tasks should be created. By default the `default` namespace will be used.

The status of each job and their logs can be seen at the Argo UI. For the purpose of this research, argo was installed in the “dev” kubernetes cluster and the address of the said UI was http://argo.dev.cybersolutions-tech.com

By choosing the workflow from the list you can see something like (additional visualizations can be found in the UI):

Alternatively, a new workflow can be created by running the following command without the possibility of keeping track of the workflow's status:

```
kubectl create -f order-proposals-pipeline.yaml -n argo
```

The parallel example can be run in the same way. There is no extra configuration required.

```
argo submit --watch order-proposals-pipeline-parallel.yaml -n auto-orderin
```

In this case the "tree" will look like this:

Cleaning up

To list all the existing workflows (regardless of their status) you can run (same rules for `-n` as above):

```
argo list -n auto-ordering-13918997-production
```

To terminate (stop the execution of) a particular workflow run (`order-proposals-pipeline-4d9mb` is an example of an automatically generated workflow name):

```
argo terminate order-proposals-pipeline-4d9mb -n auto-ordering-13918997-pr
```

To remove a workflow and clear all the used resources run:

```
argo delete order-proposals-pipeline-4d9mb -n auto-ordering-13918997-produ
```

Running a Workflow at regular intervals (cron)

Currently you can't. There's an ongoing effort to support this however and might soon be available: https://github.com/argoproj/argo/pull/1758 .

Luigi

Prerequisites

Install Luigi in your Kubernetes cluster

- A member from the DevOps team can accomplish this. Former user (Deleted) helped me with this, again.

Have the appropriate permissions in place including:

- access to the kubernetes cluster

Add Luigi to your project

Depending on how you want to use it with respect to your project code, this can be done in several ways:

a) Using Luigi outside your main project (for the purpose of managing kubernetes jobs this might be the best option)

```
pip install luigi
```

b) Using Luigi to manage bits and pieces of your Python code inside the main project (not really our case, but probably what Luigi is best for)

1. include Luigi in your setup.py or requirements.txt file

2. install it alongside your other project dependencies

Write the code for your workflow

The code could look similar to the snippet I included below. You can use the `requires` method in order to define dependencies between tasks. A list can be returned should you wish to design a workflow in which the execution of a task depends on several previous tasks: `return [ATask(), AnotherTask(), OneMore(with_a_parameter)]`.

```
from luigi.contrib.kubernetes import KubernetesJobTask


DOCKER_IMAGE = "registry.gitlab.com/cst-pse/auto-ordering/master:2b25a4ce1


class AutoOrderingKubernetesJobTask(KubernetesJobTask):
    @property
    def kubernetes_namespace(self):
        return 'auto-ordering-13918997-production'

    @property
    def auth_method(self):
        return 'kubeconfig'
        # return 'service-account'


class CheckExasolStatusTask(AutoOrderingKubernetesJobTask):

    @property
    def name(self):
        return 'check-exasol-status'

    @property
    def spec_schema(self):
        return {
            "containers": [{
                "name": "import-products",
                "image": DOCKER_IMAGE,
                "command": ['/bin/sh',
                            '-c',
                            'echo "check-exasol-status"']
            }],
            "restartPolicy": "OnFailure"
        }
```

```
class ImportProductsTask(AutoOrderingKubernetesJobTask):
    @property
    def name(self):
        return 'daily-import-products'

    @property
    def spec_schema(self):
        return {
            "containers": [{
                "name": "import-products",
                "image": DOCKER_IMAGE,
                "command": ['/bin/sh',
                            '-c',
                            'echo "import-products"']
            }],
            "restartPolicy": "OnFailure"
        }

    def requires(self):
        return CheckExasolStatusTask()
```

Run the workflow using the CLI

In order to start the execution of your workflow using the kubernetes scheduler, you can run:

```
luigi --module reordering.jobs.luigi ImportProductsTask --scheduler-url ht
```

Or, alternatively, for development purposes you can test the workflow without sending it to the kubernetes scheduler by running it like this:

```
luigi --module reordering.jobs.luigi ImportProductsTask --local-scheduler
```

Please note that during our efforts to get this running neither method worked for us due to a kubernetes permissions issue we haven't been able to solve.

The problem might be in the code as I was unable to find out how to explicitly name the kubernetes user which should be used in order to create new jobs with Luigi, or it could be something related to the kubernetes setup.

Either way, this is a blocker for any further usage of Luigi.

Comparison

	Argo	**Luigi**
Out of the box scheduling (via cron)	No, but should be available soon	No and it's not planned to have any

Parallelization	Yes	Yes
CLI	Yes	Yes
Ease of use	Excellent	Very poor
Implementation complexity	Some (a YAML file needs to be created)	Some (some Python code needs to be written, but can be commited to a separate repo)
Documentation	Could be worse	Worse
Restart on failure	Yes	Yes
Clean-up	Yes	Yes