https://itnext.io/argo-workflow-engine-for-kubernetes-7ae81eda1cc5

https://codefresh.io/learn/argo-workflows/

What is argo workflows?

Argo Workflows is an open source project that enables CI/CD pipeline management. It is a **workflow engine** that enables the **orchestration** of **parallel jobs** on Kubernetes.

workflow engine - is a type of central resource that defines and manages the execution of tasks in a k8s environment,

orchestration refers to the process of automating and coordinating the execution of tasks or jobs in a defined sequence or flow

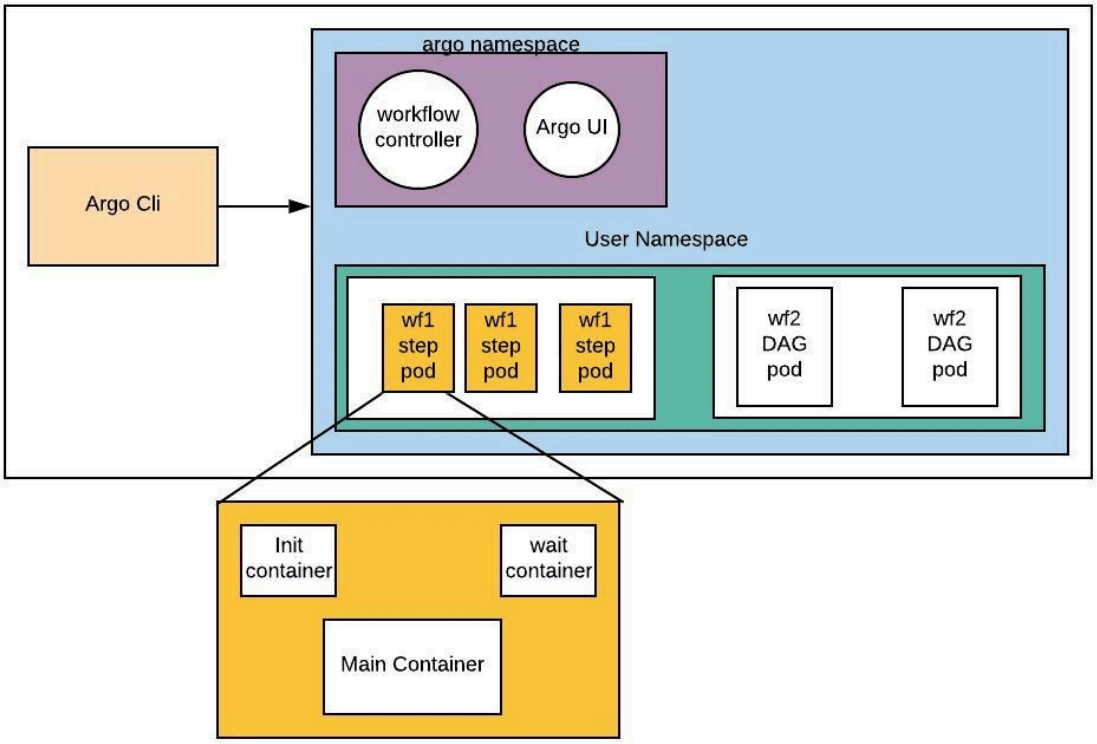
Parallel job refers to a task (or) set of tasks within a workflow that can run simultaneously within a workflow, which optimizes performance by reducing the overall execution time.

Workflow.spec:

- **List of template:**

- **Entrypoint :** `Entrypoint` specifies the primary function of the workflow or the first template to execute.
-

ARGO Workflow Overview



Argo CLI

The Argo CLI is a command-line interface tool that allows users to interact with Argo Workflows. It provides commands to submit, manage, and monitor workflows.

1. Argo Namespace

Kubernetes cluster where the Argo components are deployed. It contains essential components like the workflow controller and the Argo UI.

a. Workflow Controller

- Function: The workflow controller is a Kubernetes controller that manages the execution of workflows. It interprets the workflow definitions, schedules the tasks, and monitors their execution.

- **Responsibilities:**
    a. Reading workflow manifests and creating corresponding Kubernetes resources (e.g., pods).

b. Managing the execution order of tasks based on dependencies.
c. Cleaning up resources after workflow completion.

b. Argo UI

The Argo UI is a web-based user interface for visualizing and managing Argo Workflows.

- **Features:**
  a. Provides a graphical view of workflow executions, showing the status and dependencies of tasks.
  b. Allows users to submit new workflows, monitor ongoing workflows, and inspect logs of completed tasks.
  c. Facilitates debugging and troubleshooting by providing detailed information on task failures and retries.

2. User Namespace

The user namespace is where the actual workflows are executed. Each workflow runs in its own namespace, allowing for isolation and resource management specific to each workflow.

a. Workflow Pods (wf1 step pods)

Each step in a workflow is executed in its own pod. These pods have a specific structure to handle various aspects of task execution and monitoring.

- **Init Container:**
  a. The purpose of an init container is to perform setup or initialization tasks that need to be completed before the main containers start their execution. Init containers run **sequentially**, and each must complete successfully before the next one starts or before the main application container begins.
  b. An **init container** is a special type of container in Kubernetes that runs before the main application containers in a pod, performing setup or initialization tasks. It runs sequentially, and each init container must complete successfully before the next one starts or before the main containers begin execution.

- **Wait Container:**
  a. **Wait Container** is a specialized container in a workflow that monitors and tracks the status of the main container, ensuring accurate reporting of task execution status and handling the lifecycle of the main container by managing its progress, retries, and cleanup.

- **Main Container:**
  a. The **Main Container** executes the primary task defined in the workflow step, running the actual commands or scripts specified in the workflow definition and performing the core operations required by the workflow.

b. Workflow DAG Pods (wf2 DAG pods)

- **DAG (Directed Acyclic Graph):**
    a. **DAG (Directed Acyclic Graph)** is a graph used to represent workflows where tasks are nodes, and edges between nodes represent dependencies or relationships between tasks.

- **DAG Pods:**
    a. **DAG Pods** refer to the execution pods in a workflow that follow the **Directed Acyclic Graph (DAG)** structure, where each pod corresponds to a task or node in the DAG and is managed based on the task dependencies.

Sample single workflow example

```
apiVersion: argoproj.io/v1alpha1  # Specifies the API version for Argo Workflows.

kind: Workflow  # Defines the kind of resource, which is a Workflow in this case.

metadata:

  name: hello-argo-workflow  # A unique name for the workflow, used to identify it in the Kubernetes environment.
```

spec: # The spec defines the workflow's configuration and behavior.

entrypoint: hello-argo # The entrypoint specifies the first template to execute in the workflow.

templates: # This section defines the templates (tasks) that the workflow will execute.

- name: hello-argo # Name of the template referenced in the entrypoint. This is the first task to run.

container: # Defines the container configuration to run this task.

image: busybox # Specifies the container image to use (busybox in this case, which is a minimal Linux image).

command: ["sh", "-c", "echo Hello, Naveen!"] #The shell command to execute. "sh -c" allows the execution of a string as a shell command.

Different types of templates:

1) **Template definition**
    a) **Container**
    b) **Script**
    c) **Resources**
    d) **suspend**

https://github.com/devopshobbies/argocd-tutorial/blob/main/v15-argo-workflows-getting-started/template-types/templates.yml

https://github.com/devopshobbies/argocd-tutorial/blob/main/v15-argo-workflows-getting-started/template-types/templates.yml

2) Template invocators
    a) Steps
    b) Dag

Inner list: will runs the parallel manner
Outer list: will runs the sequential manner

DAG Template

apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
generateName: argo-wf- # Name of this Workflow
spec:

```
  entrypoint: dag-template # Defines "dag-template" as the "main"
template
  templates:
  - name: dag-template # Defining the "dag template" as a Template
Invocator
    dag:
      tasks:
      - name: A
        template: container-template
      - name: B
        template: script-template
        dependencies: [A]
      - name: C
        template: resource-template
        dependencies: [A]
      - name: D
        template: delay-template
        dependencies: [B, C]

  - name: container-template # Defining the "container template" as
a Template Definition
    container:
      image: alpine
      command: ["/bin/sh", "-c"]
      args: ["echo Hello from container template"]

  - name: script-template # Defining the "script template" as a
Template Definition
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import random
        i = random.randint(1, 100)
        print(i)
```

```
  - name: resource-template # Defining the "resource template" as a
Template Definition
    resource:
      action: create
      manifest: |
        apiVersion: v1
        kind: ConfigMap
        metadata:
          generateName: argo-wf-resource-template-cm-
        data:
          created_by: argo-workflows

  - name: delay-template # Defining the "delay template" as a
Template Definition
    suspend:
      duration: "20s"
```

Step template

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: argo-wf-  # Name of this Workflow
spec:
  entrypoint: steps-template # Defines "steps-template" as the "main" template
  templates:
  - name: steps-template # Defining the "steps template" as a Template Invocator
    steps:
    - - name: step1
        template: container-template
    - - name: step2a
        template: script-template
      - name: step2b
        template: resource-template
    - - name: step3
        template: delay-template

  - name: container-template # Defining the "container template" as a Template
Definition
    container:
      image: alpine
      command: ["/bin/sh", "-c"]
      args: ["echo Hello from container template"]

  - name: script-template # Defining the "script template" as a Template Definition
    script:
      image: python:alpine3.6
      command: [python]
      source: |
        import random
        i = random.randint(1, 100)
        print(i)

  - name: resource-template # Defining the "resource template" as a Template
Definition
    resource:
      action: create
      manifest: |
        apiVersion: v1
        kind: ConfigMap
        metadata:
          generateName: argo-wf-resource-template-cm-
        data:
          created_by: argo-workflows
```

```
  - name: delay-template # Defining the "delay template" as a Template Definition
    suspend:
      duration: "20s"
```

step-example:

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: steps-example  # Name of the workflow
spec:
  entrypoint: steps-example  # Entry point is the first template to execute
  templates:
    - name: stepsl-example  # Template that contains the steps to execute
      steps:
        - - name: step-a  # First step
            template: step-a  # Refers to the step-a template
        - - name: step-b  # Second step
            template: step-b  # Refers to the step-b template
        - - name: step-c  # Third step
            template: step-c  # Refers to the step-c template

    - name: step-a  # Template for step-a
      container:  # Specifies the container to run this step
        image: busybox  # The container image to use (busybox)
        command: ["sh", "-c", "echo Step A"]  # The shell command to run (prints "Step A")

    - name: step-b  # Template for step-b
      container:  # Specifies the container to run this step
        image: busybox  # The container image to use (busybox)
        command: ["sh", "-c", "echo Step B"]  # The shell command to run (prints "Step B")

    - name: step-c  # Template for step-c
      container:  # Specifies the container to run this step
        image: busybox  # The container image to use (busybox)
        command: ["sh", "-c", "echo Step C"]  # The shell command to run (prints "Step C")
```

In Argo Workflows, sensors are part of Argo Events and are used to listen to various event sources and trigger workflows based on those events.

- DockerSensor,
- GitSensor
- CronSensor

- **DockerSensor**

  The DockerSensor listens for Docker-related events. It can be configured to respond to events like container start, stop, or image push events.

- **GitSensor**

  The GitSensor listens for Git events such as push, pull request, or tag events. It's useful for triggering workflows based on changes in a Git repository.

- **CronSensor**

  The CronSensor is used to trigger workflows based on a cron schedule. It's useful for scheduling workflows to run at specific times or intervals.

Summary

- **DockerSensor**: Listens for Docker-related events (e.g., container start, stop) and triggers workflows.

- **GitSensor**: Listens for Git events (e.g., push, pull request) and triggers workflows.

- **CronSensor**: Triggers workflows based on a cron schedule.

Argo CD: Focuses on continuous delivery and GitOps-based deployment of applications in Kubernetes.

Argo Workflows: Manages complex, container-native workflows and batch jobs on Kubernetes.

Argo Events: Provides an event-based framework to trigger actions and workflows in response to various events.

Argo Sensors: A sub-component of Argo Events, responsible for listening to events and triggering corresponding actions or workflows.

workflow engine - is a type of central resource that defines and manages the execution of tasks in a k8s environment,

orchestration refers to the process of automating and coordinating the execution of tasks or jobs in a defined sequence or flow

Parallel job refers to a task (or) set of tasks within a workflow that can run simultaneously within a workflow, which optimizes performance by reducing the overall execution time.

Matrix Parallel-example

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  name: matrix-parallel-example
spec:
  entrypoint: matrix-example
  templates:
    - name: matrix-example
      matrix:
        parameters:
          - name: color
            values: [red, green, blue]
      steps:
        - - name: step
            template: process-color
            arguments:
              parameters:
                - name: color
                  value: "{{item}}"

    - name: process-color
      inputs:
        parameters:
          - name: color
      container:
        image: busybox
        command: ["sh", "-c", "echo Processing {{inputs.parameters.color}}"]
```