Beyond stand-alone performance: AI Workflow
Session 3

Argo Workflows

"Workflows, the Kubernetes way."

By: CARV at FORTH & Antonis Tapanlis

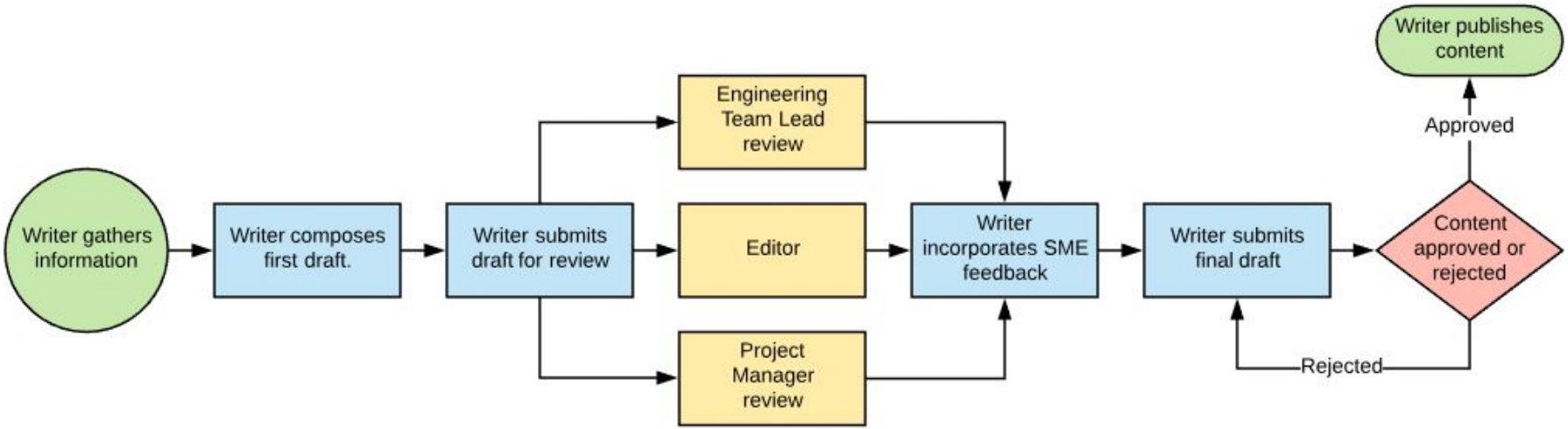
Introduction to Workflows & Argo

Argo Workflows Components

Argo Language & Examples

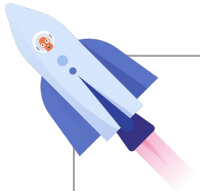
What is a Workflow?

- A workflow is a **series of steps** necessary to complete a task
- Each step in a workflow has a preceding step and a following step (except for the first and last steps)
- Each step may have dependencies, such as files, user actions, or scheduled execution times
- Often modeled as Directed Acyclic Graphs (**DAGs**) or represented as flowcharts

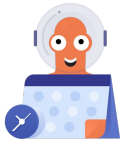


Argo Project

- Open source tools for Kubernetes to run workflows, manage clusters, and do GitOps
- Accepted on CNCF (Cloud Native Computing Foundation) at 2020, Graduated at 2022



Argo CD	Argo Events	Argo Rollouts	Argo Workflows
Declarative continuous delivery with a fully-loaded UI 20k★	Event based dependency management for Kubernetes - 2.5k★	Advanced Kubernetes deployment strategies 3k★	K8s-native workflow engine supporting DAG and step-based workflows - 16k★



Workflow Engine

- Workflow Engine
  - Is a **Language** & a **Runtime System**
  - Defines, orchestrates, and monitors a series of tasks as DAGs
  - Ensures the correct running sequence and the right dependencies

- Kubernetes-native Workflow Engine
  - Leverages Kubernetes features
  - Mainly consists of
    - Kubernetes Objects (CRDs)
    - A Controller (Kubernetes Operator)
  - Each Workflow **task runs in a Container**

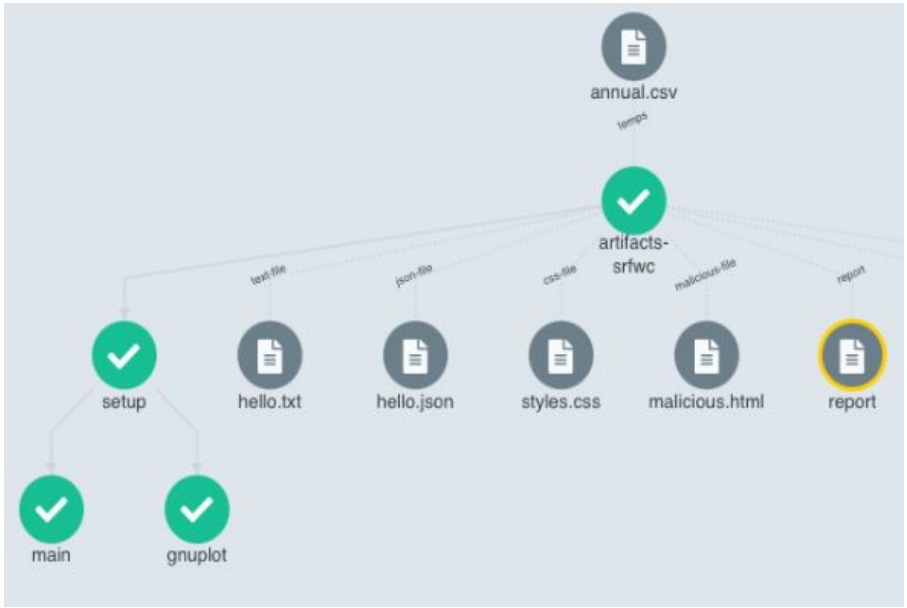
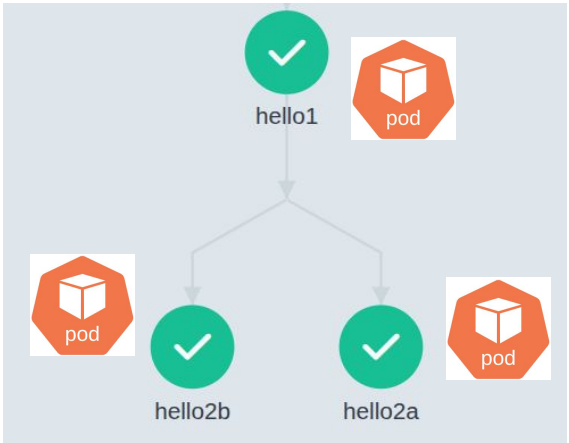


Image source: https://argo-workflows.readthedocs.io/en/latest/artifact-visualization/

Argo Workflows

- Language -> Argo **yaml** (Kubernetes Objects)
- Runtime -> Argo Workflows components running on **Kubernetes**

- Main object -> ***Workflow*** (similar to Job)
- Each step/task of a Workflow is a **pod**
- Use/write **containers** for each step



Why do we need a workflow-tool like this?

- To manage workflows through CLI or Web UI, getting advantage of the Kubernetes power
- To mainly use it for
  - Compute-intensive jobs (ML, data processing)
  - Automating software infrastructure (CI/CD)

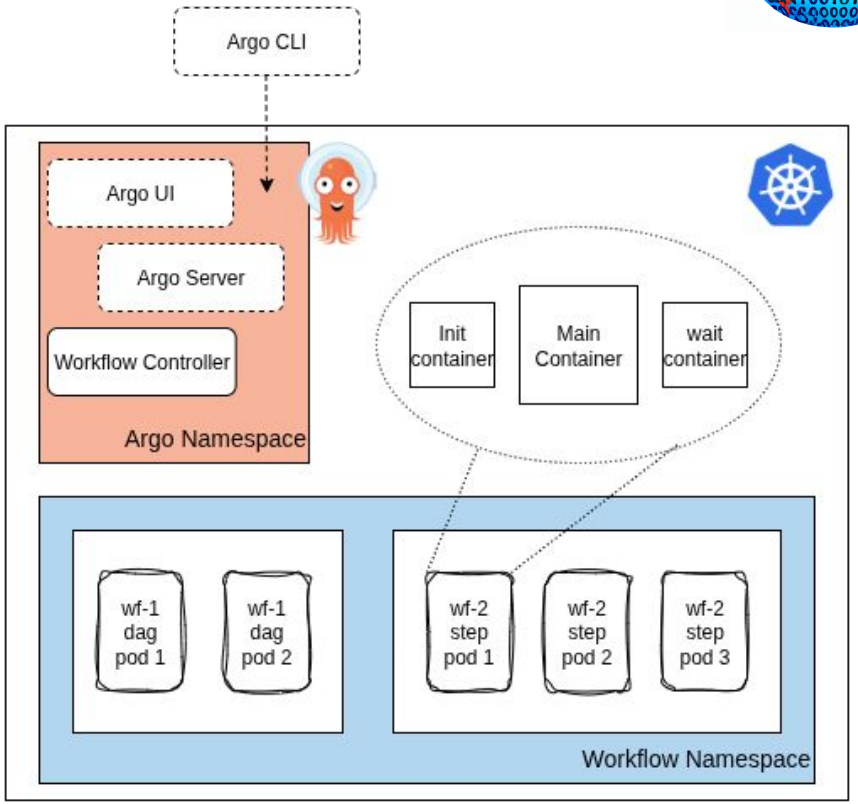
Introduction to Workflows & Argo

Argo Workflows Components

Argo Language & Examples

Argo Workflows Components

- **Argo CLI & UI** (optional use)
  - User endpoint for interaction with Argo
  - Extended functionalities for Argo Objects
- **Argo Server** (optional use)
  - Used to take advantage of CLI and UI
  - Alternatively "*kubectl*" and kubernetes API can be used
- **Workflow Controller**
  - Deploys, monitors and manages workflow objects
  - Pods are spawned per step, based on YAML-defined logic
  - Three containers per Pod



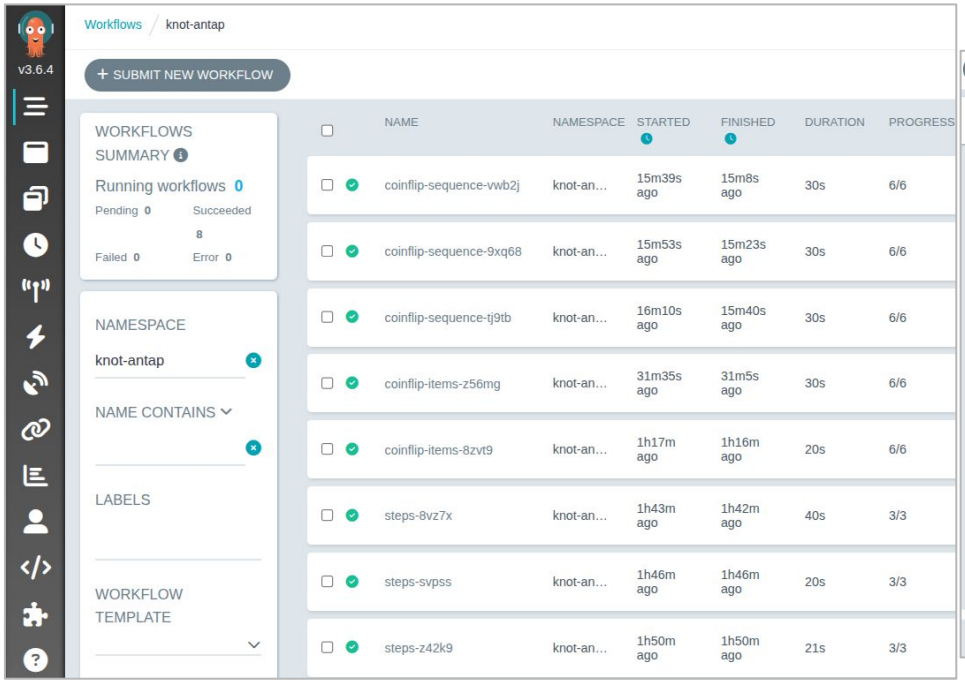
Argo CLI - Basic Commands

- argo **submit** hello-world.yaml -> submit a workflow to Kubernetes
  - --wait -> wait for the workflow to complete
  - --watch -> watch the workflow until it completes
- argo **list** -> list current workflows
  - --running or --completed
  - -n <namespace> or -A (for all namespaces)
- argo **get** hello-world-xxx -> get info about a specific workflow
- argo **logs** hello-world-xxx -> print the logs from a workflow
- argo **watch** hello-world-xxx -> watch the flow in terminal
- argo **stop** hello-world-xxx
- argo **resume** hello-world-xxx
- argo **delete** hello-world-xxx

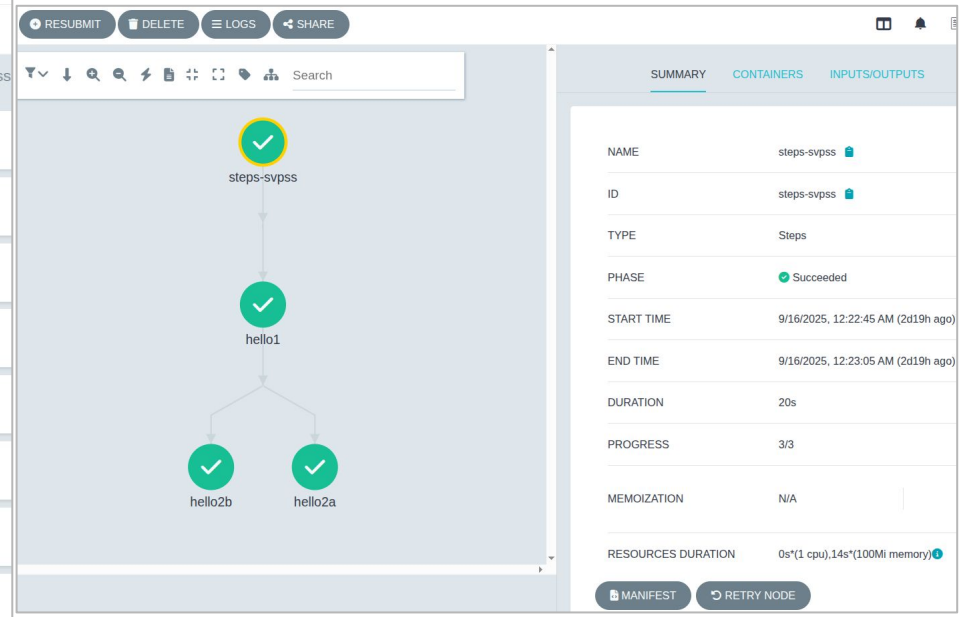
```
STEP              TEMPLATE            PODNAME                    DURATION
 ✔ steps-z2zdn    hello-hello-hello
 ├──✔ hello1      print-message       steps-z2zdn-27420706       2s
 └─┬✔ hello2a     print-message       steps-z2zdn-2006760091     3s
   └─✔ hello2b    print-message       steps-z2zdn-2023537710     3s
```

Argo UI

Home page

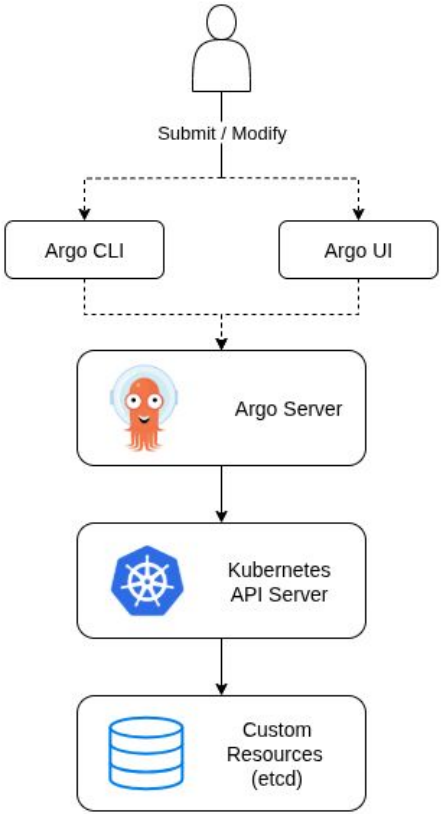


Workflow page



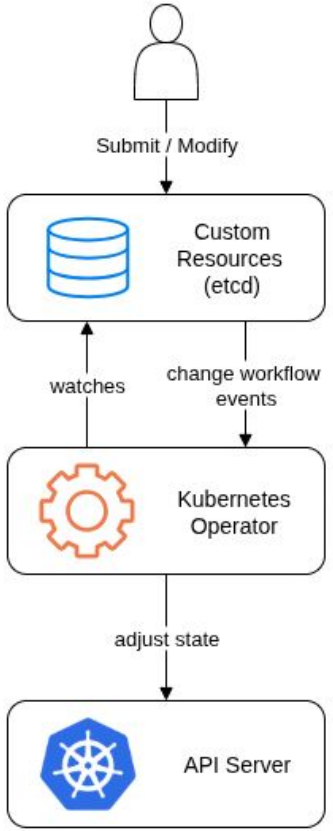
Argo Server

- Mostly used for the communication with the outside world
- Exposes Argo Workflows API
- Provides Argo UI – Considered to make Argo Workflows more accessible to those less familiar with Kubernetes
- Optional use – “kubectl” use for users prefer the Kubernetes way
- Argo CLI talks to the Argo Server

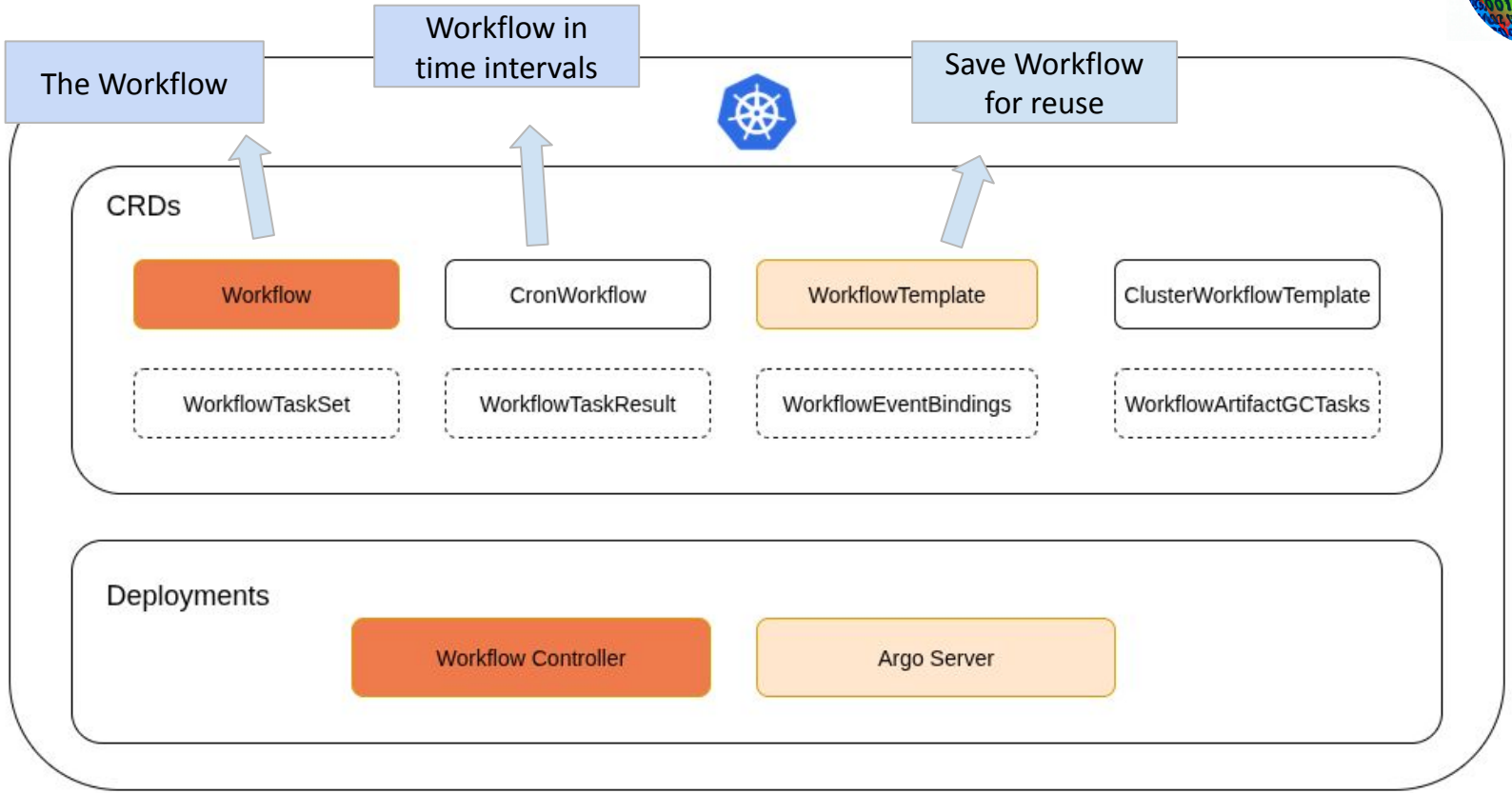


Workflow Controller

- Main component of Argo Workflows
- Manages Workflows and everything related to Workflows (Argo Workflows CRDs)
- Talks with Kubernetes API Server
- Implements the Kubernetes Operator pattern
- Can be modified and/or have multiple deployments (High availability mode, Namespaced mode)



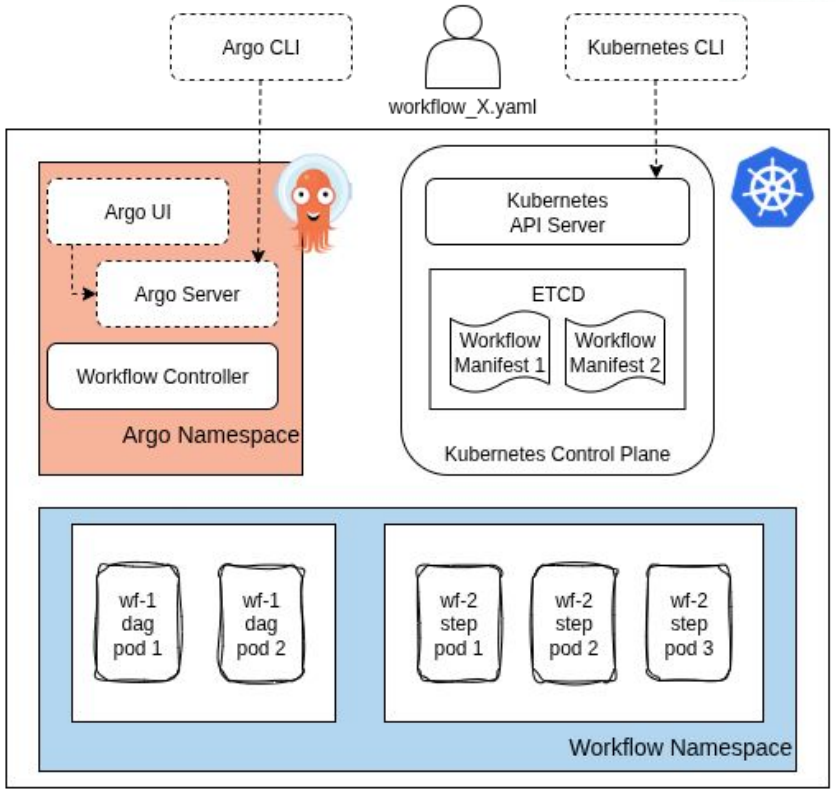
Argo Workflows - Overall



Argo Workflows - Working

1. **User creates** a Workflow manifest using either K8s CLI, Argo CLI, or Argo UI
2. K8s API Server stores the Workflow in **etcd**
3. Workflow Controller **detects it** and starts running
4. Controller asks the Kubernetes API Server to **create the pod(s)** for the read-to-run steps
5. Controller continually asks the Kubernetes API for the **status of the pod(s)**
6. Controller **writes the status** and parameter values of each workflow step back to the Workflow manifest (via k8s API)

- Steps 4–6 are repeated until the workflow is Succeeded or Failed



Introduction to Workflows & Argo

Argo Workflows Components

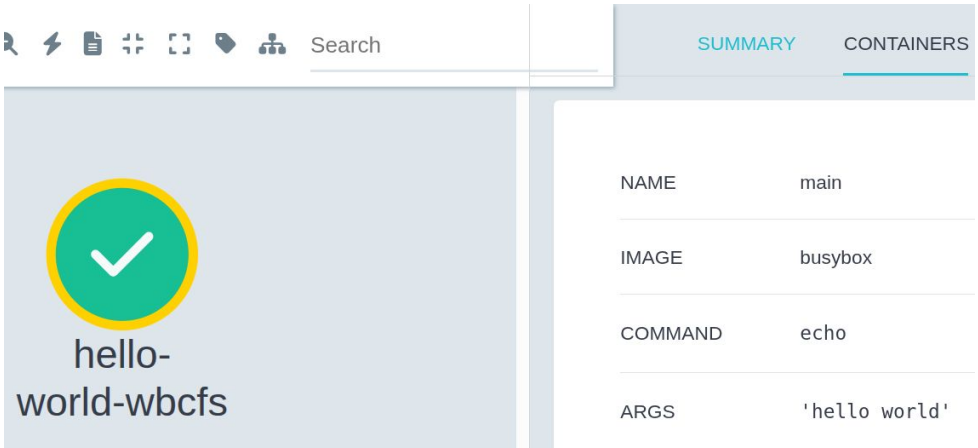
Argo Language & Examples

Language part 1

- Hello World !
- Workflow Structure
  - Workflow Types - (Template Invocators)
  - Step Templates Types - (Template Definitions)
- Hello World !!!!
- Bonus Workflow Structure

Hello World !

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-
  annotations:
    workflows.argoproj.io/description: |
      This is a simple hello world example.
spec:
  entrypoint: hello-world
  templates:
  - name: hello-world
    container:
      image: busybox
      command: [echo]
      args: ["hello world"]
```



Language part 1

- Hello World !
- Workflow Structure
  - Workflow Types - (Template Invocators)
  - Step Templates Types - (Template Definitions)
- Hello World !!!!
- Bonus Workflow Structure

Workflow Types - (Template Invocators)

Single Task

One-task Workflow.
(Hello World)

Step-based

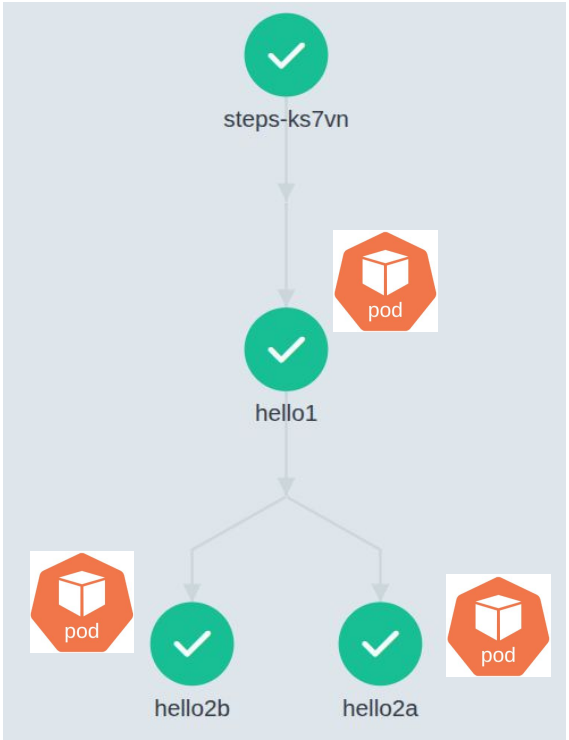
Sequence of steps.
Implicit dependencies

DAG

Graph of tasks.
Explicit dependencies

Workflow Types - Steps

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello
  templates:
  - name: hello-hello-hello
    steps:
    - - name: hello1
        (config)
    - - name: hello2a
        (config)
      - name: hello2b
        (config)
```



Step Template Types - (Template Definitions)

- We need something to implement this steps
- We want reusable code
- We better keep the Workflow “clean”

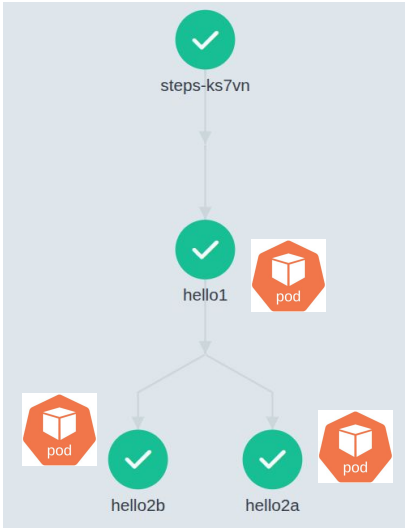
- Solution: Template Definitions!
  - **Container**
  - Script
  - …

```
3   apiVersion: argoproj.io/v1alpha1
4   kind: Workflow
5   metadata:
6     generateName: steps-
7   spec:
8     entrypoint: hello-hello-hello
9     templates:
10    - name: hello-hello-hello
11      steps:
12      - - name: hello1
13
14            (config)
15
16      - - name: hello2a
17
18            (config)
19
20        - name: hello2b
21
22            (config)
23
```

Step Template Types - Container

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello
  templates:
  - name: hello-hello-hello
    steps:
    - - name: hello1
        template: print-message

        (config)
    - - name: hello2a
        template: print-message

        (config)
      - name: hello2b
        template: print-message

        (config)
```

```yaml
- name: print-message

  (config)

  container:
    image: busybox

  (config)
```



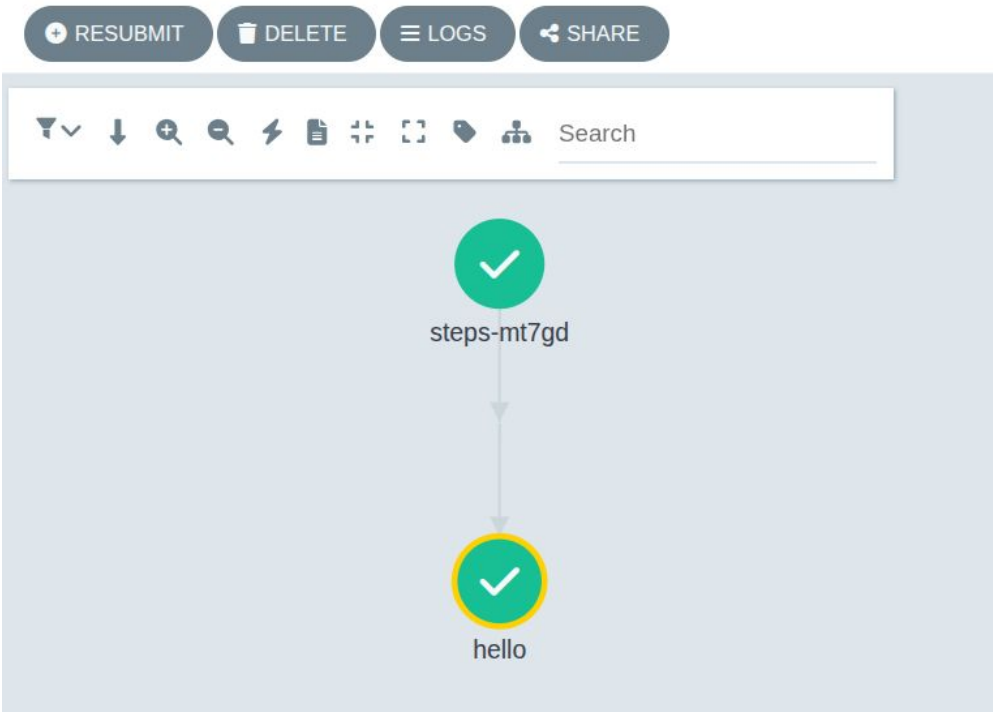
Language part 1

- Hello World !
- Workflow Structure
  - Workflow Types - (Template Invocators)
  - Step Templates Types - (Template Definitions)
- Hello World !!!!
- Bonus Workflow Structure

Hello World !! - (single inline step)

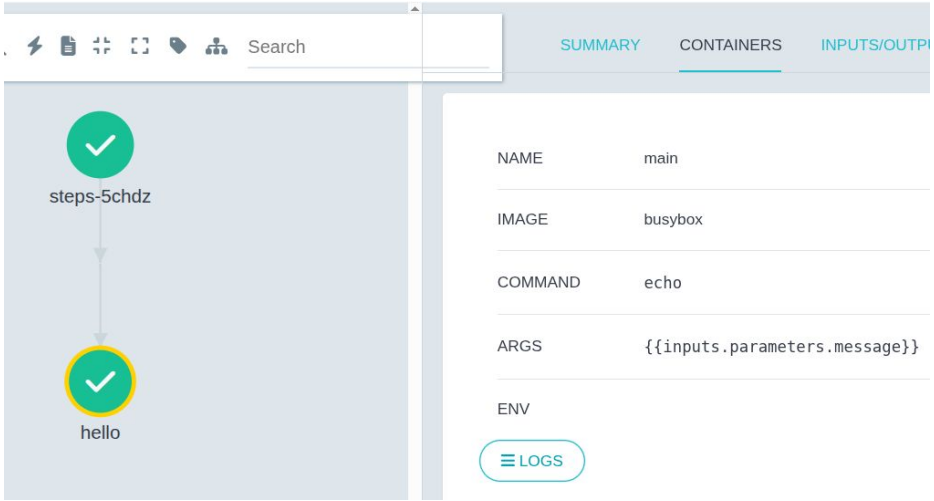
```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello
  templates:
  - name: hello
    steps:
    - - name: hello
        inline:
          container:
            image: busybox
            command: [echo]
            args: [hello world!]
```

(inline: Not the best practice !)



Hello World !!! - (single template step)

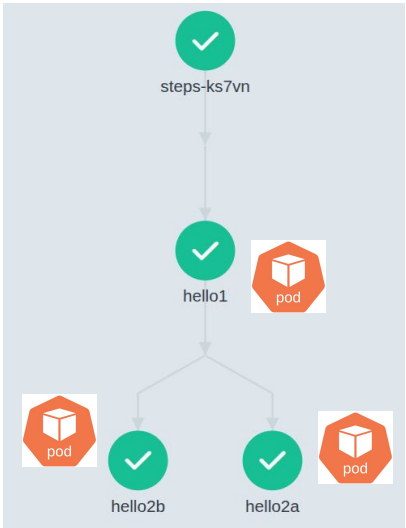
```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello
  templates:
  - name: hello
    steps:
    - - name: hello
        template: print-message
        arguments:
          parameters: [{name: message, value: "hello world!"}]

  - name: print-message
    inputs:
      parameters:
      - name: message
    container:
      image: busybox
      command: [echo]
      args: ["{{inputs.parameters.message}}"]
```



Hello World !!!! - (3-step Workflow using 1 template)

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello
  templates:
  - name: hello-hello-hello
    steps:
    - - name: hello1
        template: print-message
        arguments:
          parameters: [{name: message, value: "hello1"}]
    - - name: hello2a
        template: print-message
        arguments:
          parameters: [{name: message, value: "hello2a"}]
      - name: hello2b
        template: print-message
        arguments:
          parameters: [{name: message, value: "hello2b"}]
```

```yaml
    - name: print-message
      inputs:
        parameters:
        - name: message
      container:
        image: busybox
        command: [echo]
        args: ["{{inputs.parameters.message}}"]
```

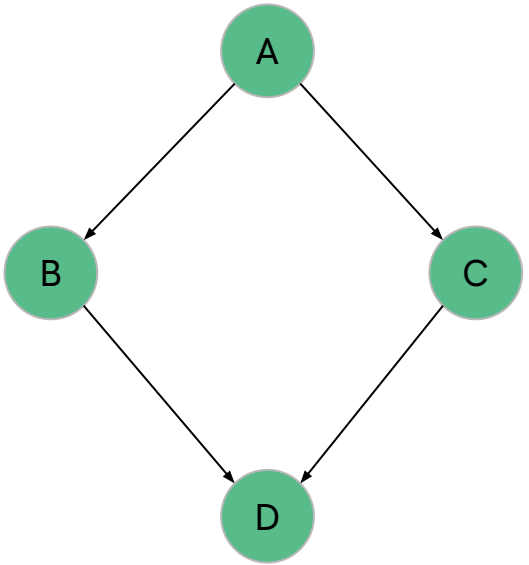


Language part 1

- Hello World !
- Workflow Structure
  - Workflow Types - (Template Invocators)
  - Step Templates Types - (Template Definitions)
- Hello World !!!!
- Bonus Workflow Structure

Workflow Types - DAG

```
13      entrypoint: diamond
14      templates:
15      - name: diamond
16        dag:
17          tasks:
18          - name: A
19
20              (config)
21
22          - name: B
23            depends: "A"
24
25              (config)
26
27          - name: C
28            depends: "A"
29
30              (config)
31
32          - name: D
33            depends: "B && C"
34
35              (config)
36
```



Step Template Types - Script (Container wrapper)

```yaml
entrypoint: bash-script-example
templates:
- name: bash-script-example
  steps:
  - - name: generate
      template: gen-random-int
  - - name: print
      template: print-message
      arguments:
        parameters:
        - name: message
          value: "{{steps.generate.outputs.result}}"

- name: gen-random-int
  script:
    image: debian:9.4
    command: [bash]
    source: |
      cat /dev/urandom | od -N2 -An -i | awk -v f=1 -v r=100 '{printf "%i\n", f + r * $1 / 65536}'

- name: print-message
  inputs:
    parameters:
    - name: message
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo result was: {{inputs.parameters.message}}"]
```

```yaml
- name: gen-random-int
  script:
    image: python:alpine3.6
    command: [python]
    source: |
      import random
      i = random.randint(1, 100)
      print(i)
```

Step Template Types - Some more

More than 1 container per pod

```yaml
entrypoint: main
templates:
  - name: main
    containerSet:
      containers:
        - name: a
          image: argoproj/argosay:v2
        - name: b
          image: argoproj/argosay:v2
```

Kubernetes objects

```yaml
entrypoint: k8s-set-owner-reference
templates:
- name: k8s-set-owner-reference
  resource:
    action: create
    setOwnerReference: true
    manifest: |
      apiVersion: v1
      kind: ConfigMap
      metadata:
        generateName: owned-eg-
      data:
        some: value
```

HTTP request

```yaml
    - name: http
      inputs:
        parameters:
          - name: url
      http:
        timeoutSeconds: 20 # Default 30
        url: "{{inputs.parameters.url}}"
        method: "GET" # Default GET
        headers:
          - name: "x-header-name"
            value: "test-value"
        # Template will succeed if evaluated to true, otherwise will fail
        # Available variables:
        #  request.body: string, the request body
        #  request.headers: map[string][]string, the request headers
        #  response.url: string, the request url
        #  response.method: string, the request method
        #  response.statusCode: int, the response status code
        #  response.body: string, the response body
        #  response.headers: map[string][]string, the response headers
        successCondition: "response.body contains \"google\"" # available
        body: "test body" # Change request body
```

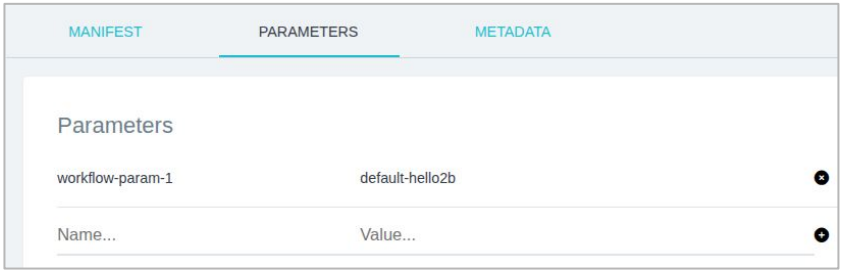
Language part 2

- Runtime Parameters
- WorkflowTemplates
- Conditionals + Step Output
- Loops

Runtime Parameters

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello
  arguments:
    parameters:
    - name: workflow-param-1
      value: "default-hello2b"  # default value if none is passed
  templates:
  - name: hello-hello-hello
    steps:
```

argo submit example.yaml
-p 'workflow-param-1="zdravo!"'

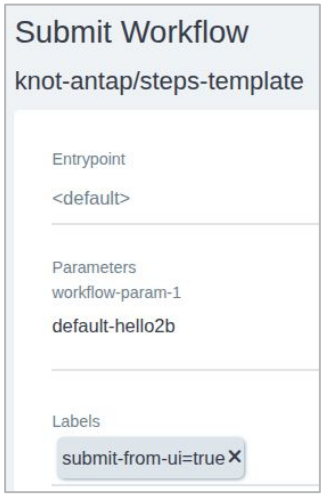


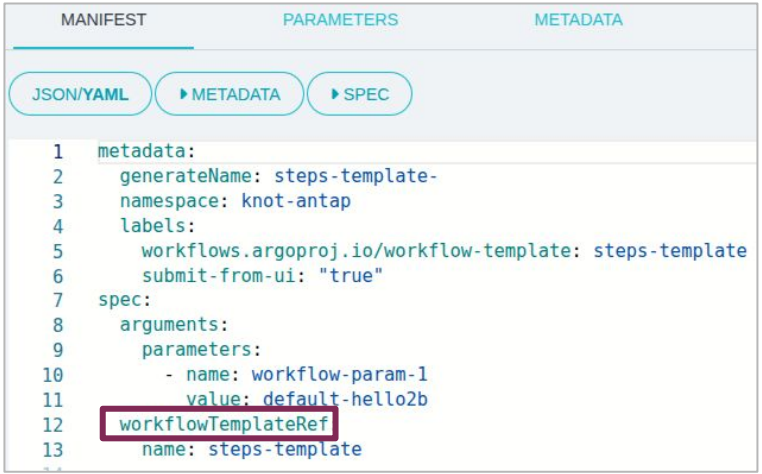

```yaml
    - - name: hello2a
        template: print-message
        arguments:
          parameters: [{name: message, value: "hello2a"}]
      - name: hello2b
        template: print-message
        arguments:
          parameters:
          - name: message
            value: "{{workflow.parameters.workflow-param-1}}"

- name: print-message
  inputs:
    parameters:
    - name: message
  container:
    image: busybox
    command: [echo]
    args: ["{{inputs.parameters.message}}"]
```

WorkflowTemplate

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: steps-
spec:
  entrypoint: hello-hello-hello
  arguments:
    parameters:
    - name: workflow-param-1
      value: "default-hello2b"   # default value if none is passed
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate
metadata:
  name: steps-template
spec:
  entrypoint: hello-hello-hello
```



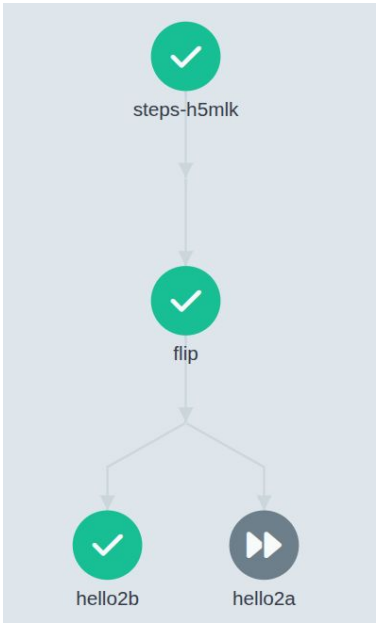


Conditionals + Step Output

```
8       - name: coin-flip-workflow
9         steps:
10        - - name: flip
11            template: flip-coin
12        - - name: hello2a
13            template: print-message
14            arguments:
15              parameters: [{name: message, value: "hello2a"}]
16            when: "{{steps.flip.outputs.result}} == heads"
17          - name: hello2b
18            template: print-message
19            arguments:
20              parameters: [{name: message, value: "hello2b"}]
21            when: "{{steps.flip.outputs.result}} == tails"
```

```
23      # Template for flipping a coin
24      - name: flip-coin
25        script:
26          image: python:alpine3.6
27          command: [python]
28          source: |
29            import random
30            result = "heads" if random.randint(0,1) == 0 else "tails"
31            print(result)
```

steps.flip.output.result
=
steps.<step_name>.outputs.STDOUT



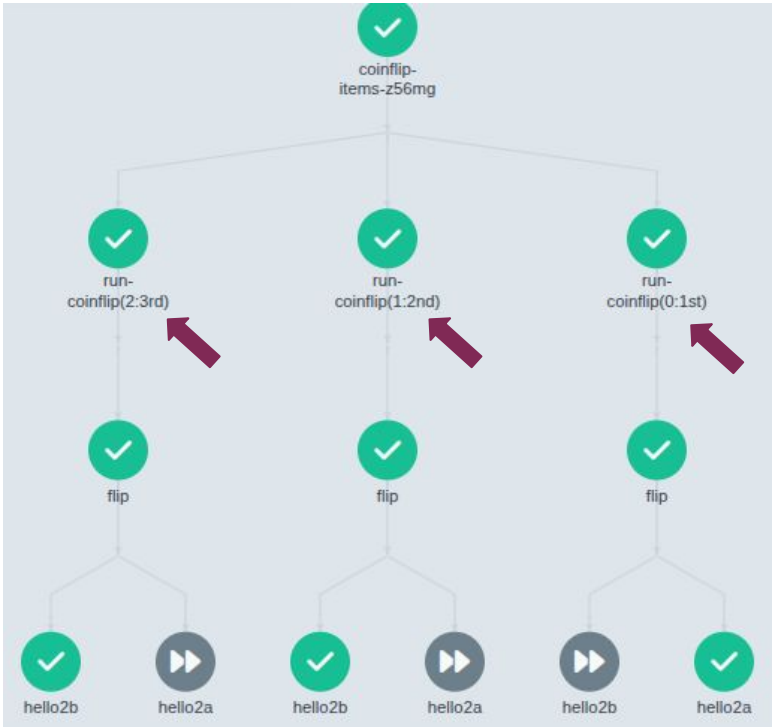
Loop (for)

```yaml
    - name: coinflip-experiment
      steps:
      - - name: run-coinflip
          template: coinflip-round
          withSequence:
            count: 3   # repeat experiment 3 times

    # One round of coin flip + conditional steps
    - name: coinflip-round
      steps:
      - - name: flip
          template: flip-coin
      - - name: hello2a
          template: print-message
          arguments:
            parameters: [{name: message, value: "heads chosen"}]
          when: "{{steps.flip.outputs.result}} == heads"
        - name: hello2b
          template: print-message
          arguments:
            parameters: [{name: message, value: "tails chosen"}]
          when: "{{steps.flip.outputs.result}} == tails"
```

Loop (with items)

```yaml
- name: coinflip-round
  inputs:
    parameters:
    - name: round
  steps:
  - - name: flip
      template: flip-coin
  - - name: hello2a
      template: print-message
      arguments:
        parameters:
        - name: message
          value: "{{inputs.parameters.round}} round: heads chosen"
      when: "{{steps.flip.outputs.result}} == heads"
    - name: hello2b
      template: print-message
      arguments:
        parameters:
        - name: message
          value: "{{inputs.parameters.round}} round: tails chosen"
      when: "{{steps.flip.outputs.result}} == tails"
```

```yaml
- name: coinflip-experiment
  steps:
  - - name: run-coinflip
      template: coinflip-round
      arguments:
        parameters:
        - name: round
          value: "{{item}}"
      withItems:
      - "1st"
      - "2nd"
      - "3rd"
```



Language part 3

- Data between Steps - Input/Output
  - Artifacts - Input/Output files + Artifact Repository
  - Hardwired artifacts
  - Files - Shared Filesystem

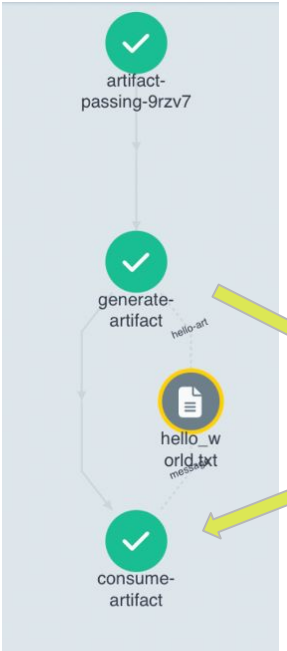
Data between Steps - Input/Output

Parameters	**Artifacts**	**Volumes**	**Shared Filesystem**
Booleans, loop values, strings/JSON (e.g. the output of a python script). [~256 KB cap]	Files, Large files (Argo way)	Files, Large files, XLarge files (Kubernetes way)	Files, Large files, XLarge files (Filesystem attached to all containers)
Easy use. No setup is needed	Moderate use. - For **Output** files, set up of Artifact Repository (or Cloud Storage) is needed. - No setup is needed for **Input** files	Advanced use. Filesystem-style sharing	Easy to use once it's set up. User just writes the **filepath** Existing in Knot system

Artifacts - Input/Output files + Artifact Repository

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: artifact-passing-
spec:
  entrypoint: artifact-example
  templates:
  - name: artifact-example
    steps:
    - - name: generate-artifact
        template: hello-world-to-file
    - - name: consume-artifact
        template: print-message-from-file
        arguments:
          artifacts:
          # bind message to the hello artifact
          # generated by the generate-artifact step
          - name: message
            from: "{{steps.generate-artifact.outputs.artifacts.hello}}"
```

```yaml
    - name: hello-world-to-file
      container:
        image: busybox
        command: [sh, -c]
        args: ["echo hello world | tee /tmp/hello_world.txt"]
      outputs:
        artifacts:
        # generate hello artifact from /tmp/hello_world.txt
        # artifacts can be directories as well as files
        - name: hello
          path: /tmp/hello_world.txt

    - name: print-message-from-file
      inputs:
        artifacts:
        # unpack the message input artifact
        # and put it at /tmp/message
        - name: message
          path: /tmp/message
      container:
        image: alpine:latest
        command: [sh, -c]
        args: ["cat /tmp/message"]
```



Artifact Repository

Artifacts - Hardwired [git, http, S3]

```yaml
    inputs:
      artifacts:
      # Check out the main branch of the argo repo and place it at /src
      # revision can be anything that git checkout accepts: branch, commit, tag, etc.
      - name: argo-source
        path: /src
        git:
          repo: https://github.com/argoproj/argo-workflows.git
          revision: "main"
      # Download kubectl 1.8.0 and place it at /bin/kubectl
      - name: kubectl
        path: /bin/kubectl
        mode: 0755
        http:
          url: https://storage.googleapis.com/kubernetes-release/release/v1.8.0/bin/linux/amd64/kubectl
      # Copy an s3 compatible artifact repository bucket (such as AWS, GCS and MinIO) and place it at /s3
      - name: objects
        path: /s3
        s3:
          endpoint: storage.googleapis.com
          bucket: my-bucket-name
          key: path/in/bucket
          accessKeySecret:
            name: my-s3-credentials
            key: accessKey
          secretKeySecret:
            name: my-s3-credentials
            key: secretKey
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hardwired-artifact-
spec:
  entrypoint: hardwired-artifact
  templates:
  - name: hardwired-artifact
    container:
        image: debian
        command: [sh, -c]
        args: ["ls -l /src /bin/kubectl /s3"]
```

AWS S3 acts as Artifact Repository and can be used for output Artifacts too!

Files - Shared Filesystem

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: catalog-demo-
spec:
  entrypoint: run-catalog
  templates:
  - name: run-catalog
    steps:
    - - name: foo-step
        template: process-foo
      - name: bar-step
        template: process-bar
```

```yaml
  - name: process-foo
    inputs:
      parameters:
      - name: in-file
        value: "/files/private/tiffs/foo.tiff"
    metadata: {}
    container:
      image: alpine:3.19
      command: ["sh","-c"]
      args: ["echo Processing {{inputs.parameters.in-file}}"]

  - name: process-bar
    inputs:
      parameters:
      - name: in2-file
        value: "/files/private/tiffs/bar.tiff"
    metadata: {}
    container:
      image: alpine:3.19
      command: ["sh","-c"]
      args: ["echo Processing {{inputs.parameters.in2-file}}"]
```

"Take-Home Message"

Argo lets you focus on workflows, not infrastructure.

👉 Run complex workflows with Kubernetes power.

No deep Kubernetes knowledge required !

References

- https://argoproj.github.io/
- https://argo-workflows.readthedocs.io/
- https://github.com/argoproj/argo-workflows
- https://www.youtube.com/watch?v=FBRMURQYbgw
- https://dafab-platform.eu/
- https://github.com/AntonisT/Dafab-Summer-School-Argo
- https://github.com/argoproj/argo-workflows/discussions/7338
- https://argo-workflows.readthedocs.io/en/latest/configure-artifact-repository/#accessing-non-default-artifact-repositories


Questions ?
antap@ics.forth.gr