Argo Workflows 101: Fundamentals

10am-noon PST
22nd Sep 2020
Recording

Pre-requisites

1. Install Kubernetes locally (we recommend use Docker on Desktop + K3D as it supports RBAC):
2. kubectl
3. Add yourself to the sign-in sheet

```
brew install k3d

k3d create ;# or `k3d cluster create`  for newer versions

export KUBECONFIG="$(k3d get-kubeconfig --name='k3s-default')" ;# `or ? for new
version?

kubectl cluster-info
```

Install Argo Workflows

```
kubectl create ns argo

kubectl -n argo apply -f  https://raw.githubusercontent.com/argoproj/argo/master/manifests/quick-start-postgres.yaml

kubectl -n argo patch cm workflow-controller-configmap -p '{"data": {"containerRuntimeExecutor": "pns"}}' ;# needed for K3S

kubectl -n argo get pods --watch ;# takes maybe 2m for all pods to be ready

kubectl -n argo port-forward svc/argo-server 2746:2746

open http://localhost:2746

brew install argo

argo version
```

Fundamentals

Alex

Hands-On:

Using the user interface - submit a workflow that prints “Hi Argo Workshop!”

Hello Argo!

```
argo submit -n argo https://raw.githubusercontent.com/argoproj/argo/master/examples/hello-world.yaml

argo list -n argo

argo get -n argo ...

argo logs -n argo ...
```

Hands-On:

Using the CLI - submit a workflow and wait for it to finish.

Hint: https://argoproj.github.io/argo/cli/argo_submit/

Workflow Service Account

A short detour on running workflows with different service accounts.

```
kubectl create serviceaccount me

kubectl create rolebinding me --serviceaccount=argo:me --role=workflow-role
```

Hands-On:

Submit a workflow that uses a service account.

Anatomy of a Workflow

Simon

Anatomy of a workflow

- Steps
- DAGs
- Containers
- Scripts
- Resources suspend
- Arguments, Inputs, and Outputs
- Artifacts
- Exit handler

Templates

Templates are how we define the work to be done and call other templates to do the work.

All templates are defined under the `templates` field of a Workflow.

Templates act like functions/methods (we will see soon).

Several different templates, but two main kinds: those that define work and those that call on other templates (we will see soon).

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *define* work in methods

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *define* work in methods, give them names

```
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *define* work in methods, give them names, and specify their inputs and outputs

```
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *call* work in code blocks

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *call* work in code blocks by naming our desired definitions

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *call* work in code blocks by naming our desired definitions, passing in and receiving live arguments

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

We *call* work in code blocks by naming our desired definitions, passing in and receiving live arguments, and do some execution control

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
  - - name: addFour
      template: addFour
      arguments: {parameters: [{name: "a", value: "2"}]}
  - - name: sayHello
      template: sayHello
      when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *define* work in methods

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *define* work in methods, give them names

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *define* work in methods, give them names, and specify their inputs and outputs

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
  - - name: addFour
      template: addFour
      arguments: {parameters: [{name: "a", value: "2"}]}
  - - name: sayHello
      template: sayHello
      when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *call* work in code blocks

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello(string toWho) {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *call* work in code blocks by naming our desired definitions

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *call* work in code blocks by naming our desired definitions, passing in and receiving live arguments

```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = addFour(2);
        if (result > 5) {
            sayHello();
        }
    }
    public int addFour(int a) {
        return a + 4;
    }
    public void sayHello() {
        System.out.println("Hello Intuit!");
    }
}
```

```yaml
- name: main
  steps:
    - - name: addFour
        template: addFour
        arguments: {parameters: [{name: "a", value: "2"}]}
    - - name: sayHello
        template: sayHello
        when: "{{steps.addFour.outputs.result}} > 5"
- name: addFour
  inputs: {parameters: [{name: "a"}]}
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
- name: sayHello
  container:
    image: alpine:latest
    command: [sh, -c]
    args: [echo "Hello Intuit!"]
```

We *call* work in code blocks by naming our desired definitions, passing in and receiving live arguments, and do some execution control

Templates

Templates are how we define the work to be done and call other templates to do the work.

Two kinds of templates:

Definition templates:

- `Container` (from the last example)
- `Script`
- `Resource`
- `Suspend`

Execution templates:

- `Steps` (from the last example)
- `DAG`

Inputs and Output Parameters

Input and Output Parameters are how we move data across different steps in a Workflow.

● Inputs are a *definition* of inputs	`addFour(int a)`
● Outputs are a *definition* of outputs	`int addFour(...)`
● Arguments are *live* arguments	`addFour(2)`

Hands On

1. Grab our working example from: http://in/argo-hw or https://bit.ly/2ZyjPmO
2. Modify it as defined in the new Java code (changes highlighted)
3. Challenge: Can you use a DAG template instead of Steps? (Docs and examples in the Argo Repo)

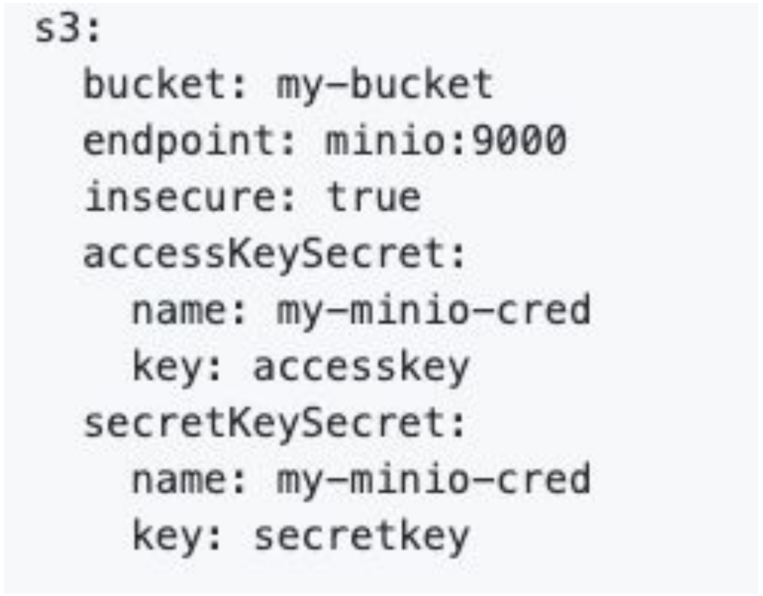
```java
class ArgoDemo {
    public static void main(String[] args) {
        int result = add(2, 5);
        int finalResult = add(result, 10);
        if (finalResult > 5) {
            sayHello(finalResult);
        }
    }
    public int add(int a, int b) {
        return a + b;
    }
    public void sayHello(int result) {
        System.out.println("Result is: " + result);
    }
}
```

Artifacts

Bala

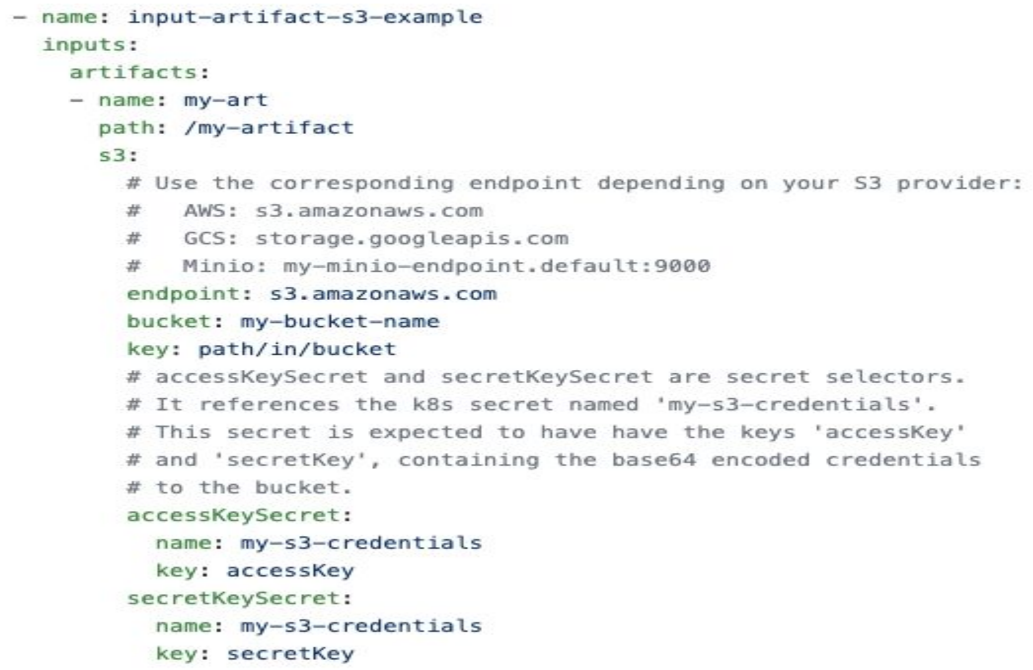
Input and Output Artifacts

- Artifacts can be a single file or directory
- Supported Repositories
  - S3, GCS, OSS, RAW, HDFS, Github, http
- Configure Repository in Argo
  - Controller level configuration in configmap
  - Workflow level configuration
    - ArtifactRepositoryRef
    - Inline configuration



```
s3:
  bucket: my-bucket
  endpoint: minio:9000
  insecure: true
  accessKeySecret:
    name: my-minio-cred
    key: accesskey
  secretKeySecret:
    name: my-minio-cred
    key: secretkey
```

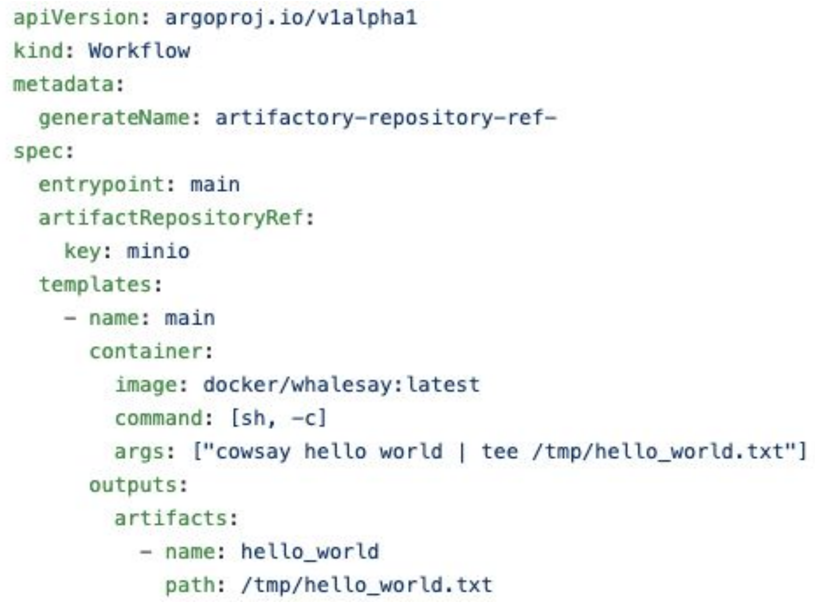
Inline artifact repository



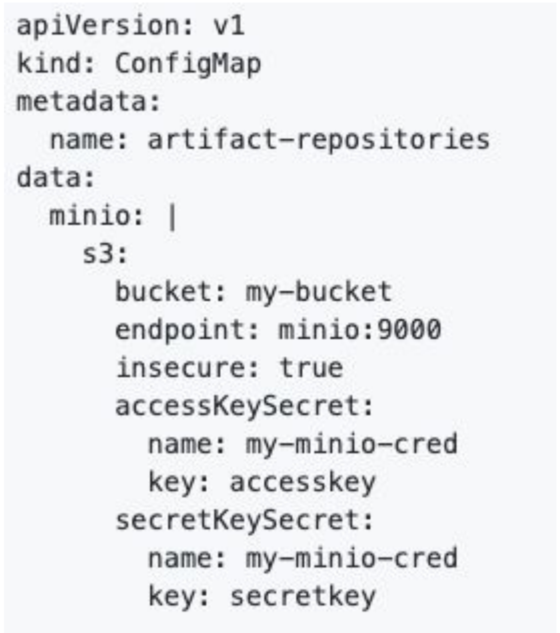
```yaml
- name: input-artifact-s3-example
  inputs:
    artifacts:
    - name: my-art
      path: /my-artifact
      s3:
        # Use the corresponding endpoint depending on your S3 provider:
        #   AWS: s3.amazonaws.com
        #   GCS: storage.googleapis.com
        #   Minio: my-minio-endpoint.default:9000
        endpoint: s3.amazonaws.com
        bucket: my-bucket-name
        key: path/in/bucket
        # accessKeySecret and secretKeySecret are secret selectors.
        # It references the k8s secret named 'my-s3-credentials'.
        # This secret is expected to have have the keys 'accessKey'
        # and 'secretKey', containing the base64 encoded credentials
        # to the bucket.
        accessKeySecret:
          name: my-s3-credentials
          key: accessKey
        secretKeySecret:
          name: my-s3-credentials
          key: secretKey
```

Using ArtifactRepositoryRef:

- Configure multiple repositories in configmap and refer them in workflow using `artifactRepositoryRef`:



```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: artifactory-repository-ref-
spec:
  entrypoint: main
  artifactRepositoryRef:
    key: minio
  templates:
    - name: main
      container:
        image: docker/whalesay:latest
        command: [sh, -c]
        args: ["cowsay hello world | tee /tmp/hello_world.txt"]
      outputs:
        artifacts:
          - name: hello_world
            path: /tmp/hello_world.txt
```



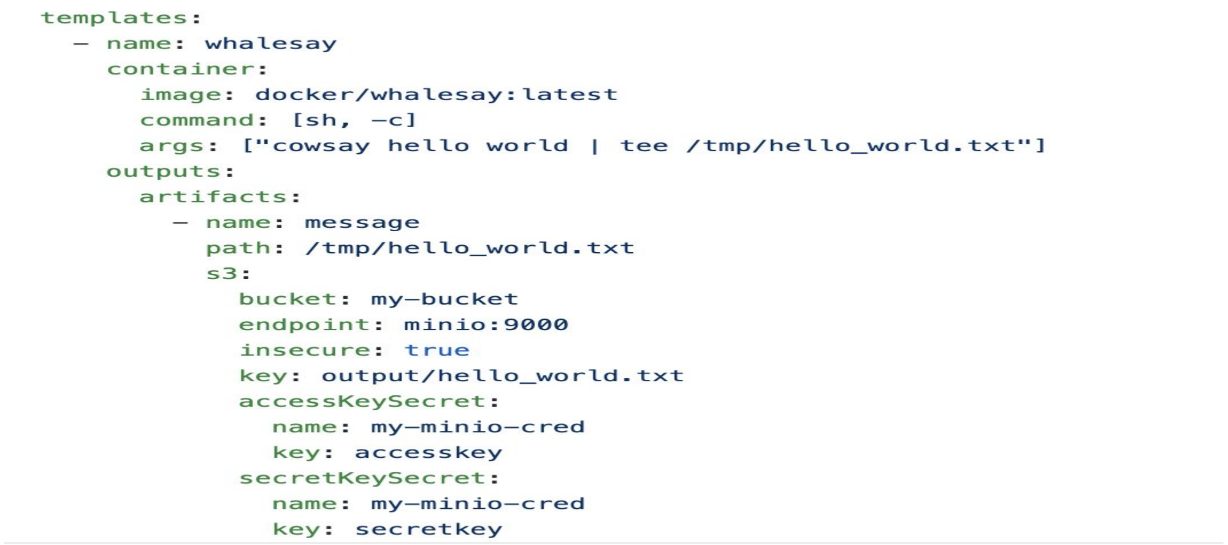
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: artifact-repositories
data:
  minio: |
    s3:
      bucket: my-bucket
      endpoint: minio:9000
      insecure: true
      accessKeySecret:
        name: my-minio-cred
        key: accesskey
      secretKeySecret:
        name: my-minio-cred
        key: secretkey
```

Hands-On:

Submit a workflow that use artifacts.

Output Artifacts

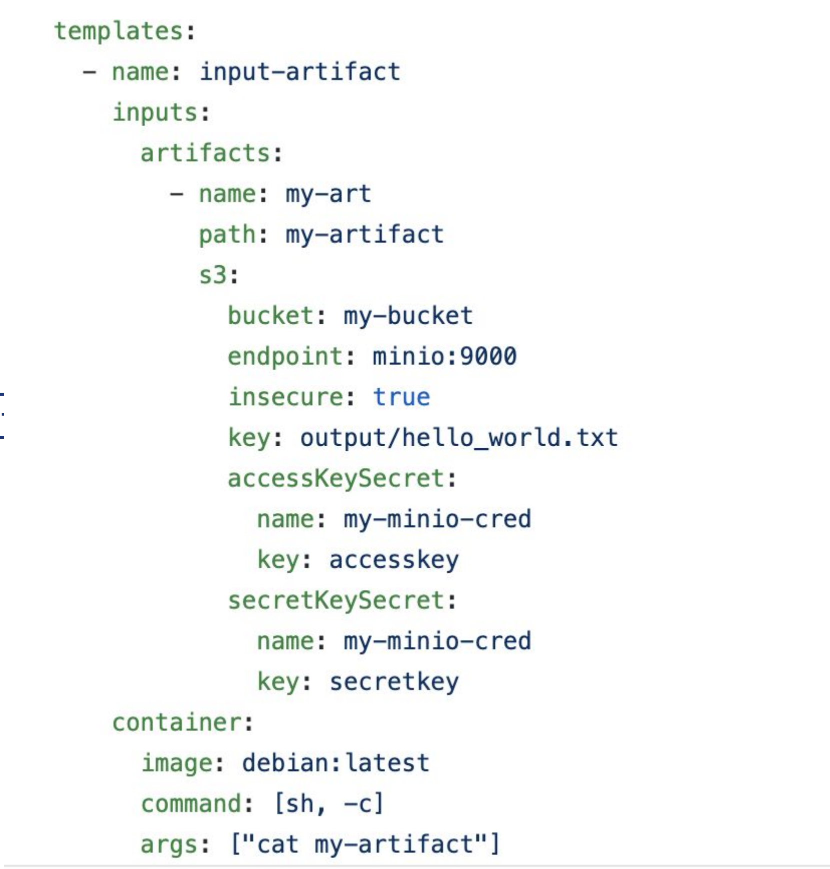
- Output will be pushed to configured Repository once main container is completed
- https://gist.github.com/sarabala1979/a7190fe6f43996f2ee6a2d9877723aaa



```yaml
templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [sh, -c]
      args: ["cowsay hello world | tee /tmp/hello_world.txt"]
    outputs:
      artifacts:
        - name: message
          path: /tmp/hello_world.txt
          s3:
            bucket: my-bucket
            endpoint: minio:9000
            insecure: true
            key: output/hello_world.txt
            accessKeySecret:
              name: my-minio-cred
              key: accesskey
            secretKeySecret:
              name: my-minio-cred
              key: secretkey
```

Input Artifacts

- Input artifact will be downloaded from configured Repository and saved in given path for main container to access it.
- https://gist.githubusercontent.com/sarabala1979/a2198b888a31269afb5fe08c0de3af1d/raw/947898cf8f68b544b7c456f0bc9926 27b1a2cca/input-artifact.yaml

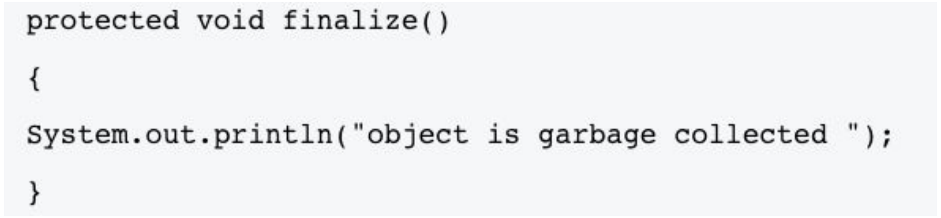


```
templates:
  - name: input-artifact
    inputs:
      artifacts:
        - name: my-art
          path: my-artifact
          s3:
            bucket: my-bucket
            endpoint: minio:9000
            insecure: true
            key: output/hello_world.txt
            accessKeySecret:
              name: my-minio-cred
              key: accesskey
            secretKeySecret:
              name: my-minio-cred
              key: secretkey
    container:
      image: debian:latest
      command: [sh, -c]
      args: ["cat my-artifact"]
```

Passing Artifacts

https://gist.github.com/sarabala1979/03dcc960371851dbc6f7c9ea23abd212

Exit Handlers



```
protected void finalize()
{
System.out.println("object is garbage collected ");
}
```

- It is a destructor of workflow
- You can define Exit handler in Workflow level, Step/Dag level
- https://gist.githubusercontent.com/sarabala1979/f3cacd7fafd2378f577049182beeabaf/raw/126b4ba530bf1b473234dc36f9f569576c865294/exithandler.yaml.

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: exit-handlers-
spec:
 entrypoint: intentional-fail
 onExit: exit-handler
```

```
- name: exit-handler
 container:
   image: docker/whalesay:latest
   command: [cowsay]
   args: ["Exit-Handler"]
```

Hands-On:

Create workflow with an exit handler.

Workflow Templates

Bala

Change your workflow to a workflow template

- Instead of submitting whole workflow every time.
- You can store the workflow definition in cluster
- You can refer or submit definition multiple time.
- You just change kind from `workflow` to `WorkflowTemplate`
- Cli command for create workflow template
- `argo template create <>`

```
class WorkflowTemplate{

void template(int a, int b){
  // do something
}
}

Class workflow {
Public static void main(String[] args)
   WorkflowTemplate wfTmpl = new
WorkflowTempalte();
   wfTmpl.template(1,2);
wfTmpl.template(3,4);
}
```

Hands-On:

Change your workflow to a template and submit the template.

https://gist.githubusercontent.com/sarabala1979/5f5ecff0976908300d020c5cda9ad53c/raw/239b785009bbab87d161f3ce96bc6d1da392256d/workflowtemplate.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: WorkflowTemplate
metadata:
 name: add-example-template
spec:
 entrypoint: main
 templates:
   - name: main
     steps:
       - - name: addFour
           template: addFour
           arguments: {parameters: [{name: "a", value: "2"}]}
       - - name: sayHello
           template: sayHello
           when: "{{steps.addFour.outputs.result}} > 5"
   - name: addFour
     inputs: {parameters: [{name: "a"}]}
     container:
       image: alpine:latest
       command: [sh, -c]
       args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
   - name: sayHello
     container:
       image: alpine:latest
       command: [sh, -c]
       args: [echo "Hello Intuit!"]
```

Refer workflow template in workflow

https://gist.githubusercontent.com/sarabala1979/9c188e5c425acf0d795eaf0b61e02729/raw/1cf5ba90c1d3e175190121c1c9f948cdd199f9f2/workflowtemplateRef.yaml

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: add-example-
spec:
  workflowTemplateRef:
    name: add-example-template
```

https://gist.githubusercontent.com/sarabala1979/09eb280fdab084ea92455b8f87c70146/raw/b538a8a27983ff53ed4c3e3b8b36752ede61a7d5/templateref.yaml

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: workflow-template-hello-world-
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    steps:
      - - name: call-whalesay-template
          templateRef:
            name: add-example-template
            template: addFour
          arguments:
            parameters:
            - name: a
              value: "3"
```

Submit the workflow from workflow template

`argo submit --from workflowtemplate/add-example-template --watch`

Cluster workflow template

- `argo cluster-template create <>`

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ClusterWorkflowTemplate
metadata:
 name: add-example-template
spec:
 entrypoint: main
 templates:
   - name: main
     steps:
       - - name: addFour
           template: addFour
           arguments: {parameters: [{name: "a", value: "2"}]}
       - - name: sayHello
           template: sayHello
           when: "{{steps.addFour.outputs.result}} > 5"
   - name: addFour
     inputs: {parameters: [{name: "a"}]}
     container:
       image: alpine:latest
       command: [sh, -c]
       args: ["echo $(( {{inputs.parameters.a}} + 4 ))"]
   - name: sayHello
     container:
       image: alpine:latest
       command: [sh, -c]
       args: [echo "Hello Intuit!"]
```

Cron Workflows

Simon

Cron Workflows

CronWorkflows are Regular Workflows that run on a schedule. Converting one is easy.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: hello-world-
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["hello world"]
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: CronWorkflow
metadata:
  generateName: hello-world-
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["hello world"]
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: CronWorkflow
metadata:
  generateName: hello-world-
spec:
  entrypoint: whalesay
  templates:
  - name: whalesay
    container:
      image: docker/whalesay:latest
      command: [cowsay]
      args: ["hello world"]
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: CronWorkflow
metadata:
  generateName: hello-world-
spec:
  workflowSpec:
    entrypoint: whalesay
    templates:
      - name: whalesay
        container:
          image: docker/whalesay:latest
          command: [cowsay]
          args: ["hello world"]
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: CronWorkflow
metadata:
  generateName: hello-world-
spec:
  schedule: "2 * * * *"
  timezone: "America/Los_Angeles"
  concurrencyPolicy: "Replace"
  workflowSpec:
    entrypoint: whalesay
    templates:
      - name: whalesay
        container:
          image: docker/whalesay:latest
          command: [cowsay]
          args: ["hello world"]
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: CronWorkflow
metadata:
  generateName: hello-world-
spec:
  schedule: "2 * * * *"
  timezone: "America/Los_Angeles"
  concurrencyPolicy: "Replace"
  workflowSpec:
    entrypoint: whalesay
    templates:
      - name: whalesay
        container:
          image: docker/whalesay:latest
          command: [cowsay]
          args: ["hello world"]
```