ArgoCon

Managing Artifacts at Scale for CI and Data Processing

Caelan Urquhart, Pipekit & Julie Vogelman, Intuit

#ArgoCon2023

April 18, 2023

pipekit

INTUIT

turbotax creditkarma quickbooks mailchimp

Intro & Goals for the Talk

ArgoCon

#ArgoCon2023

GOALS FOR THE TALK

- Understand the different types of artifacts used with Argo Workflows
- Understand what artifact storage options work with Argo Workflows and when to use them
- Learn the key features to scale artifact management with Argo Workflows

ABOUT US

Caelan Urquhart
Co-founder, CEO @ Pipekit

Contributor to the Argo Workflows project

Focus: Data engineering architectures, product development

Julie Vogelman
Staff Software Engineer @ Intuit

Maintainer of the Argo Workflows project

Focus: Backend engineering & architecture for high scale, low latency applications

Background: Artifacts and Storage Options

ArgoCon

#ArgoConEU2023

WHAT ARE ARTIFACTS?

- For data processing: tabular data, image/video files, geospatial data, vector data, etc.
- For CI: dockerfiles, git repos, built binaries, binary caches, etc.
- For machine learning: training datasets, trained ML models, feature stores, etc.

TYPES OF DATA & ARTIFACTS

- Transient data: Passed between workflow steps; data not needed beyond life of the workflow
- Semi-persistent data: Relevant across multiple runs of workflows; yet, easy to reinstate if lost, like module/ binary/ dockerfile caches
- Persistent data: Data we want to keep, often the final output from a running workflow

STORAGE TYPES

	Pros	Cons
Blob	• Easy setup • Easily queryable • Useful data archiving settings • Artifacts are visible in the Argo Workflows UI	• Files are tarred by Workflows before uploading - takes time and resources. • Copies are downloaded in the init container of each workflow task • Not Read-Write-Many compatible
Block	• Easy setup • Some performance advantages over blob • Lower latency • Easy data backup and restoration	• Intended for per-instance storage use cases • Not Read-Write-Many compatible
Network file system	• Read-Write-Many compatible • No tarring required, so faster transfer between steps • Auto-scales well for high-throughput needs	• Can’t be dynamically-provisioned with a PVC • Slow with parallel read/writes at scale • Only useful if you’re in the cloud

PROBLEMS

Core problem: How do I pass data between my workflow steps?

Parameters

- Passing strings or commands between steps
- Passing small, stringified .json, .txt files
- Passing small script outputs
- Biggest challenge: there's a memory limit per workflow

Blob storage

- More persistent artifacts, like data tables, trained models, etc.
- Overall, you need to package up outputs between steps

Network file system

- You don't need to package up your outputs between steps
- You want read-write-many functionality
- Save runtime

PROBLEMS

Before you start passing artifacts around, you need to configure your artifact management properly.

- Centralize the artifact repository configuration for scale & security
- Implement smart artifact naming patterns
- Managing small vs. large artifacts
- Artifact garbage collection (GC)

Solutions

ArgoCon

#ArgoCon2023

CENTRALIZE ARTIFACT CONFIGURATION

- Problems with specifying artifact configs in your workflow
  - Adds redundant lines of YAML
    - Harder to read
    - Tedious to maintain

```yaml
- name: generate
  container:
    image: argoproj/argosay:v2
    args: [ echo, hello, /mnt/file ]
  outputs:
    artifacts:
      - name: file
        path: /mnt/file
        s3:
          key: my-file
          bucket: my-bucket
          endpoint: minio:9000
          insecure: true
          accessKeySecret:
            name: my-minio-cred
            key: accesskey
          secretKeySecret:
            name: my-minio-cred
            key: secretkey

- name: consume
  container:
    image: argoproj/argosay:v2
    args: [cat, /tmp/file]
  inputs:
    artifacts:
      - name: file
        path: /tmp/file
        s3:
          key: my-file
          endpoint: minio:9000
          insecure: true
          accessKeySecret:
            name: my-minio-cred
            key: accesskey
          secretKeySecret:
            name: my-minio-cred
            key: secretkey
```

CENTRALIZE ARTIFACT CONFIGURATION

What if we could remove these redundant lines of YAML?

```yaml
- name: generate
  container:
    image: argoproj/argosay:v2
    args: [ echo, hello, /mnt/file ]
  outputs:
    artifacts:
      - name: file
        path: /mnt/file
        s3:
          key: my-file
          bucket: my-bucket
          endpoint: minio:9000
          insecure: true
          accessKeySecret:
            name: my-minio-cred
            key: accesskey
          secretKeySecret:
            name: my-minio-cred
            key: secretkey

- name: consume
  container:
    image: argoproj/argosay:v2
    args: [cat, /tmp/file]
  inputs:
    artifacts:
      - name: file
        path: /tmp/file
        s3:
          key: my-file
          endpoint: minio:9000
          insecure: true
          accessKeySecret:
            name: my-minio-cred
            key: accesskey
          secretKeySecret:
            name: my-minio-cred
            key: secretkey
```

```yaml
- name: generate
  container:
    image: argoproj/argosay:v2
    args: [ echo, hello, /mnt/file ]
  outputs:
    artifacts:
      - name: file
        path: /mnt/file
        s3:
          key: my-file
- name: consume
  container:
    image: argoproj/argosay:v2
    args: [cat, /tmp/file]
  inputs:
    artifacts:
      - name: file
        path: /tmp/file
        s3:
          key: my-file
```

CENTRALIZE ARTIFACT CONFIGURATION

Use the ArtifactRepositoryRef & Key-only Artifacts features

Step 1: Define your artifact repo settings in one (or more) ConfigMaps

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-artifact-repository
  annotations:
    # v3.0 and after - if you want to use a specific key, put that key into this annotation.
    workflows.argoproj.io/default-artifact-repository: default-v1-s3-artifact-repository
data:
  default-v1-s3-artifact-repository: |
    s3:
      bucket: my-bucket
      endpoint: minio:9000
      insecure: true
      accessKeySecret:
        name: my-minio-cred
        key: accesskey
      secretKeySecret:
        name: my-minio-cred
        key: secretkey
  v2-s3-artifact-repository: |
    s3:
      ...
```

CENTRALIZE ARTIFACT CONFIGURATION

Use the ArtifactRepositoryRef & Key-only Artifacts features

Step 2: In the workflow, reference the ConfigMap key

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: key-only-artifacts-
spec:
  artifactRepositoryRef:
    configMap: my-artifact-repository
    key: v2-s3-artifact-repository
  entrypoint: main
  templates:
```

CENTRALIZE ARTIFACT CONFIGURATION

Alternative default: the artifact-repositories ConfigMap

If no ArtifactRepositoryRef is defined, the workflow will use the artifact-repositories ConfigMap as a default.

```yaml
kind: ConfigMap
metadata:
  annotations:
    workflows.argoproj.io/default-artifact-repository: default-v1
  name: artifact-repositories
apiVersion: v1
data:
  default-v1: |
    archiveLogs: true
    s3:
      bucket: my-bucket
      endpoint: minio:9000
      insecure: true
      accessKeySecret:
        name: my-minio-cred
        key: accesskey
      secretKeySecret:
        name: my-minio-cred
        key: secretkey
```

ARTIFACT NAMING PATTERNS

Parameterize your artifact keys using {{workflow.uid}}

Avoid overwriting or deleting an artifact when multiple users run the same workflow concurrently

```yaml
templates:
  - name: main
    container:
      image: argoproj/argosay:v2
      command:
        - sh
        - -c
      args:
        - |
          echo "my first artifact" > /tmp/first-artifact.txt
          echo "and my second artifact" > /tmp/second-artifact.txt
    outputs:
      artifacts:
        - name: first-artifact
          path: /tmp/first-artifact.txt
          s3:
            key: first-artifact-{{workflow.uid}}.txt
        - name: second-artifact
          path: /tmp/second-artifact.txt
          s3:
            key: second-artifact-{{workflow.uid}}.txt
```

ARTIFACT NAMING PATTERNS

Help end-users find their artifacts using other parameters, too

- {{workflow.name}}
- {{workflow.namespace}}
- {{workflow.labels.<NAME>}}
- {{workflow.creationTimestamp}}
  - Note: not helpful if many workflows running concurrently
- {{pod.name}}

SMALL VS. LARGE ARTIFACTS

- Use podSpecPatch to request template-level resources to accommodate larger artifacts

```
- name: ingest-large-artifact
  podSpecPatch: |
    initContainers:
      - name: init
        resources:
          requests:
            memory: 1Gi
            cpu: 200m
  inputs:
    artifacts:
    - name: data
      path: /tmp/large-file
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["cat /tmp/large-file"]
```

SMALL VS. LARGE ARTIFACTS

- Reminder: Provision enough resources for your workflow steps - including artifacts - in the ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: workflow-controller-configmap
data:
  executor: |
    imagePullPolicy: Always
    resources:
      requests:
        memory: 2Gi
        cpu: 300m
```

SMALL VS. LARGE ARTIFACTS

- Add an archive strategy to adjust an artifact's compression

```
outputs:
  artifacts:
    # disable
   - name: do-not-compress-container
    path: /tmp/image.tar
    archive:
      none: {}

    # customize the compression
  - name: hello-art-3
    path: /tmp/hello_world.txt
    archive:
      tar:
        compressionLevel: 1
```

ArgoCon

#ArgoCon2023

ARTIFACT GC

- Problem: users have to manually delete artifacts to avoid incurring storage costs
- Solution: Artifact Garbage Collection (GC)
  - **Configure your workflow to automatically delete artifacts**
    - When workflow completes
    - When workflow is deleted
  - **Works for S3, GCS, Azure artifacts**
    - Help us support more, PRs welcome!
  - **New pods will start up in user's namespace to do GC**

ARTIFACT GC

- Set a default for the workflow, optionally override for specific artifacts

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: artifact-gc-
spec:
  entrypoint: main
  artifactGC:
    strategy: OnWorkflowDeletion
  templates:
    - name: main
      container:
        image: argoproj/argosay:v2
        command:
          - sh
          - -c
        args:
          - |
            echo "can throw this away" > /tmp/temporary-artifact.txt
            echo "keep this" > /tmp/keep-this.txt
      outputs:
        artifacts:
          - name: temporary-artifact
            path: /tmp/temporary-artifact.txt
            s3:
              key: temporary-artifact-{{workflow.uid}}.txt
          - name: keep-this
            path: /tmp/keep-this.txt
            s3:
              key: keep-this-{{workflow.uid}}.txt
            artifactGC:
              strategy: Never
```

default strategy applies to all artifacts

but can be overridden here

ARTIFACT GC

- Use annotations (or ServiceAccount) for pods to access your bucket

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: artifact-gc-
spec:
  entrypoint: main
  artifactGC:
    strategy: OnWorkflowDeletion
    metadata:
      annotations:
        - eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/iam-role-name
  templates:
    - name: main
      container:
        image: argoproj/argosay:v2
        command:
          - sh
          - -c
        args:
          - |
            echo "can throw this away" > /tmp/temporary-artifact.txt
            echo "keep this" > /tmp/keep-this.txt
      outputs:
        artifacts:
          - name: temporary-artifact
            path: /tmp/temporary-artifact.txt
            s3:
              key: temporary-artifact.txt
          - name: keep-this
            path: /tmp/keep-this.txt
            s3:
              key: keep-this.txt
            artifactGC:
              strategy: Never
```

Use an annotation to specify an IAM role giving access to your bucket for all artifacts (can override at artifact level)

ARTIFACT GC

- Use annotations (or ServiceAccount) for pods to access your bucket

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: art-gc-demo-
spec:
  entrypoint: main
  artifactGC:
    strategy: OnWorkflowDeletion
    serviceAccountName: gc-service-account
  templates:
    - name: main
      container:
        image: argoproj/argosay:v2
        command:
          - sh
          - -c
        args:
          - |
            echo "can throw this away" > /tmp/temporary-artifact.txt
            echo "keep this" > /tmp/keep-this.txt
      outputs:
        artifacts:
          - name: temporary-artifact
            path: /tmp/temporary-artifact.txt
            s3:
              key: temporary-artifact.txt
          - name: keep-this
            path: /tmp/keep-this.txt
            s3:
              key: keep-this.txt
            artifactGC:
              strategy: Never
```

← Or use a ServiceAccount

Demos & How-to

ArgoCon

#ArgoCon2023

DEMOS & HOW-TO

- **Data processing demo: fan-out, fan-in example**
- **CI demo: build and deploy with Argo CD**

Requirements

- K8s cluster running locally (K3d, Docker Desktop, Minikube, etc.)
- Argo Workflows 3.4 or later
- Argo CD 2.6 or later
- MinIO installed

View repo on GitHub: https://github.com/pipekit/talk-demos/tree/main/argocon-demos

ArgoCon

#ArgoCon2023

Next Steps & Resources

ArgoCon

#ArgoCon2023

NEXT STEPS & RESOURCES

- Visit our repo: https://github.com/pipekit/talk-demos/tree/main/argocon-demos
- Check out upcoming Lightening Talk: **Configuring Volumes for Parallel Workflow Reads and Writes**
- PR for “forceFinalizerRemoval” for Artifact GC
- Argo Workflows artifact docs

ArgoCon

#ArgoCon2023

ArgoCon

pipekit

INTUIT

turbotax creditkarma quickbooks mailchimp