Argo Workflows Contributors Workshop

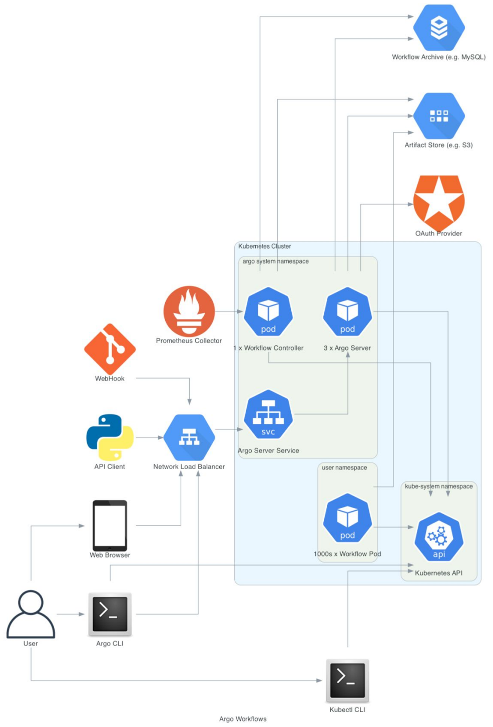
Yuan Tang ([@terrytangyuan]())

Founding Engineer at Akuity & Project Lead of Argo Workflows

Agenda

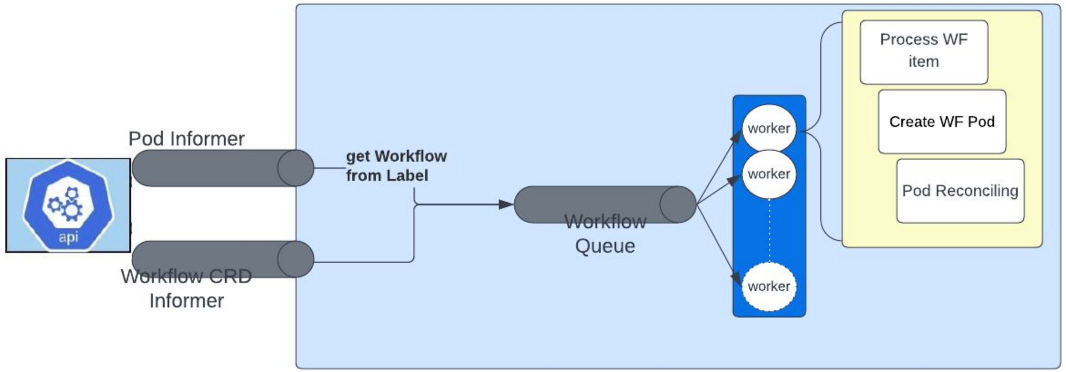
- Architecture overview
- Individual components
  - Controller, operator, workflow pods
  - Server
  - CLI, SDKs
- Hands-on experiments and understanding different components
- Codebase key files walkthrough

Architecture Overview



https://argoproj.github.io/argo-workflows/assets/diagram.png

Controller



Runs reconciliation loops for workflows and other relevant resources.

Controller

Controller includes K8s informers that watches:

- `Workflow`, `WorkflowTemplate`, `ClusterWorkflowTemplate`
- `CronWorkflow`: manages lifecycle of `CronWorkflow` and associated workflows.
- `WorkflowTaskSet`, `WorkflowTaskResult`: manages communication between workflow pods and controller.
- `WorkflowArtifactGCTask`: manages tasks that performs GC on artifacts based on artifactGC strategy.
- Pod: workflow pods lifecycle management.
- `ConfigMap`: for controller configuration, synchronization, memoization cache, etc.

Controller

The controller also includes:

- synchronizationManager: retrieves configurations of synchronization and manages mutex/semaphore locks for running workflows.
- Various garbage collections:
  - Workflows: delete workflows and offloaded node statuses based on specified periodicity and retention policy.
  - Archived workflows: based on specified TTL and periodicity.
  - Pods: based on PodGC strategy
- syncMetrics:
  - Workflow phases counter
  - Pod phases counter
  - Depth of work queues
  - Queue latency
  - Retry counter

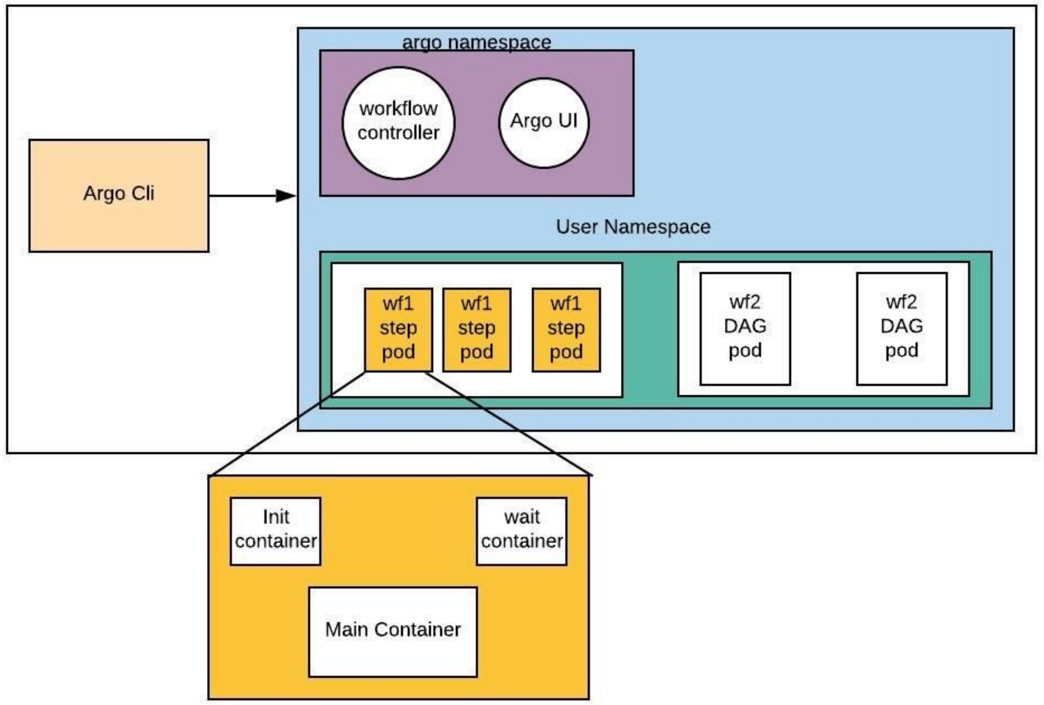
Controller Operator

The controller takes each workflow and performs the following operations:

- Executes individual templates
- Sets up artifact repository
- Initializes and manages synchronization lock
- Collects metrics related to the workflow
- Manages PodDisruptionBudget, PVCs
- Tracks the deadline of the workflow and watches suspend/resume actions
- Configures parallelism for workflow pods
- GC on artifacts
- Execute lifecycle hooks
- Records K8s events

Workflow/Executor Pods

AKUITY



Workflow/Executor Pods

- **main container**: runs the image and command that the user specifies.
- **init container**: fetches artifacts and parameters and makes them available to the main container.
- **wait container**: monitors the progress of the main container, performs tasks that are needed for clean up, including saving output parameters and artifacts, and communicating the result of the step back to the controller.
    - Note that for resource templates, main container will act as the wait container that waits for the status of the K8s resource.

Workflow Pod: Pod Annotations (old)

AKUITY

workflows.argoproj.io/outputs
workflows.argoproj.io/progress

- Pod patch permission is not allowed in some orgs
- Malicious code in main container to exploit this permission
- Version <= v3.3

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: executor
rules:
  - apiGroups:
      - ""
      resources:
      - pods
      verbs:
      - get
      - patch
```

Workflow Pod: WorkflowTaskResult (new)

- Communicate task's result back to the controller
- Result includes: phase, message, outputs, progress
- Version >= 3.4

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: executor
rules:
  - apiGroups:
    - argoproj.io
    resources:
    - workflowtaskresults
    verbs:
    - create
    - patch
```

Lifecycle of WorkflowTaskResult

1. Workflow controller initializes the K8s informer that watches the `WorkflowTaskResult` objects in the cluster.
2. Executor pod (wait container) creates a `WorkflowTaskResult` object in K8s cluster for a node.
3. Wait container continuously watches the progress of the node and patches the `WorkflowTaskResult` object periodically.
4. Controller's informer watches updates on `WorkflowTaskResult` and schedule the next steps based on the results/progress.
5. When workflow is completed, the relevant `WorkflowTaskResults` will be deleted.

Agent Pods

AKUITY

- Execute lightweight HTTP requests/templates independent of the controller.

```
templates:
- name: http-template
  http:
    url: "http://openlibrary.org/people/george08/nofound.json"
```

- Each workflow will start one agent pod for executing all HTTP requests.
- Requires additional RBAC on `WorkflowTaskSet` CRD used for communication between agent pod and controller.
- Agent pod starts a watch on `WorkflowTaskSets` CRs on any updates.

ContainerSet

AKUITY

- Execute multiple user-defined containers within a single executor pod.
- They will be scheduled on the same host.
- You can use cheap and fast empty-dir volumes to share data among containers.
- Limited functionalities: only able to use simple depends logic.

```
 templates:
- name: main
   containerSet:
   containers:
    - name: a
      image: argoproj/argosay:v2
    - name: b
      image: argoproj/argosay:v2
      dependencies: ["a"]
    - name: c
      image: argoproj/argosay:v2
      dependencies: ["a"]
    - name: d
      image: argoproj/argosay:v2
      dependencies: ["b", "c"]
```

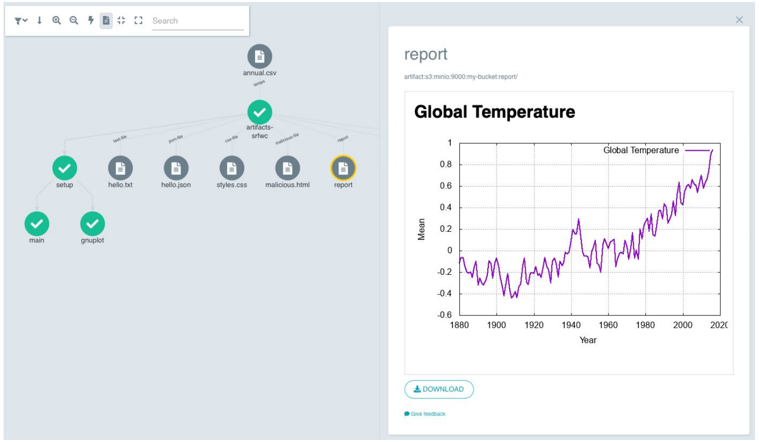
Server

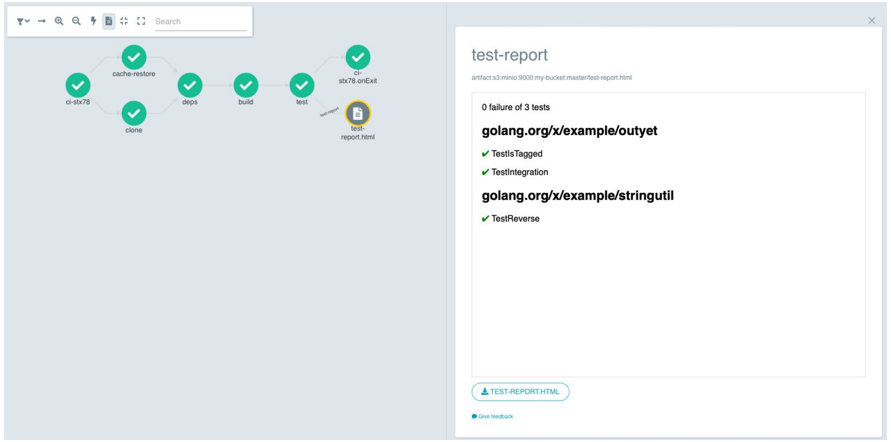
Provides the backend gRPC and RESTful services:

- Workflow server
  - Workflow operations: create, get, list, delete, retry, suspend, etc.
  - Create/update/patch/delete/list Workflow CRs in cluster and then controller reconciles
- Archived workflow server
  - Archived workflows operations: get, list, delete, resubmit, etc.
  - Interacts with the database
- (Cluster) Workflow template server
  - Create, get, update, list WorkflowTemplate CRs
- Cron workflow server:
  - Create, get, list, delete, suspend, resume CronWorkflow CRs

Server

- Artifact server
  - Retrieves input/output artifacts for workflows to be used in the UI





0 failure of 3 tests

golang.org/x/example/outyet

✔ TestIsTagged
✔ TestIntegration

golang.org/x/example/stringutil

✔ TestReverse

Server

AKUITY

- Event, event source, and sensor server
  - create/get/list/update Argo Events resources
- Info server
  - Login info such as SSO claims and service account
  - UI configurations such as first time user prompt and feedback form
- Auth
  - Handles authentication such as SSO

CLI

AKUITY

- **argo**
  - get, create, delete, list, retry, stop, logs, watch …
  - cron: get, create, delete, list, …
  - archive, cluster template, …
  - auth token
  - executor-plugin
  - …
  - Note: some requires a running argo-server
- **server**: starts the server
- **workflow-controller**: starts the controller
- **argoexec**: starts a workflow executor pod. Only use internally.

UI

- Consists of React components
- Wrapper services that send REST requests to Argo Server

UI: Services

```
export const WorkflowsService = {
   create(workflow: Workflow, namespace: string) {
     return requests
          .post(`api/v1/workflows/${namespace}`)
          .send({workflow})
          .then(res => res.body as Workflow);
   }


   get(namespace: string, name: string) {
     return requests
          .get(`api/v1/workflows/${namespace}/${name}`)
          .then(res => res.body as Workflow);
   }

   ...
}
```

UI: Components

```
export const WorkflowDetails = RouteComponentProps<any>) => {

    const [namespace] = useState(match.params.namespace);
    const [workflow, setWorkflow] = useState<Workflow>();

    const renderSummaryTab = () => {...}

    useEffect(() => {
        const retryWatch = new RetryWatch<Workflow>(
            watch: () => services.workflows.watch({name, namespace})
            onError: err => { services.workflows.get(namespace, name) ...

    ...

    return (
        <Page
            title={'Workflow Details'}
            toolbar=...
        </Page>
    );
}
```

SDKs

- Go (native)
- Python (generated by OpenAPI Generator)
    - Published to PyPI here
- Java (generated by OpenAPI Generator)
    - Published to GitHub Packages here

Dive into Components: Hands-on Experiments

AKUITY

https://github.com/terrytangyuan/contributor-workshop

Dive into Codebase: Key Files

Explore key files to different components and core functionalities.

Thanks!
Questions?