Argo Workflows

- Built on Kubernetes
- Language agnostic with declarative yaml files
- Utilizes containers natively
- You can build workflows **simple and easy**, or **complex and powerful**

Why Argo WF?

(personal opinion)

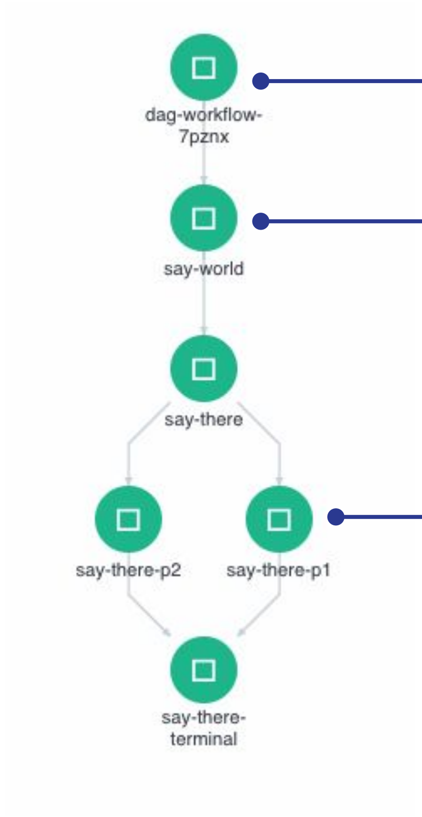
- Does one job, does it well
  Smaller surface area. You don't maintain what you don't need.
- Built on Kubernetes
  When K8s is already available, adoption is easy; and boundlessly scalable.
- Simple things are simple
  Easy to learn, but also allows powerful functionalities.

How about ML?

(personal opinion)

- **Batch computation is popular**
  One of the most popular approaches is scheduled workflows. Argo WF does it well.
- **Artifacts and Metrics**
  I can store, and keep track of the artifacts used or generated, as well as forward the metrics and logs.
- **No limitation**
  It does not limit you how you should arrange your solution. You build your solution in the container, Argo manages the execution.

Anatomy of an Argo WF

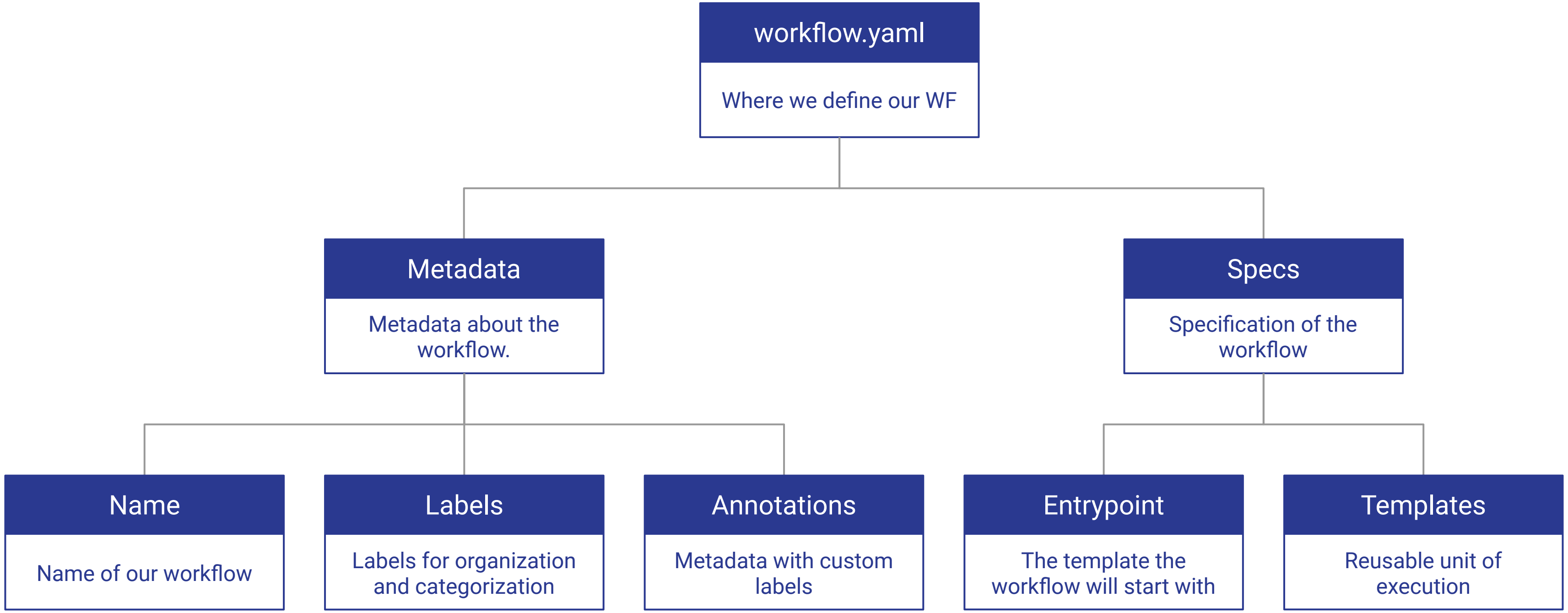


We define steps as part of the WF.

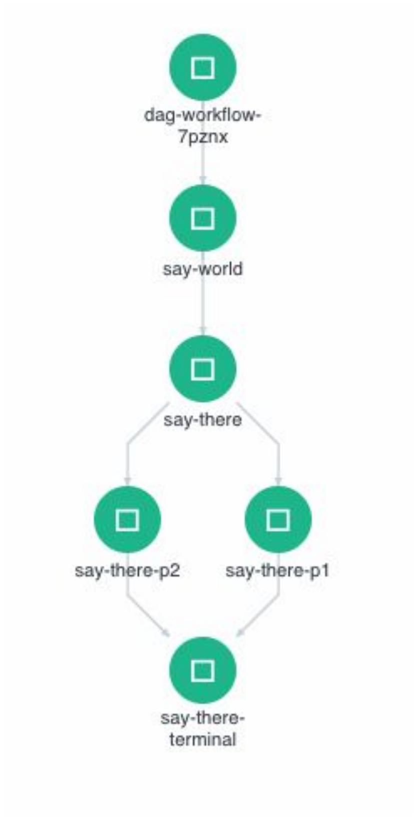
Each step has 2 main elements:
- (Docker) image to run
- Command to run in the container

Steps can fork or merge. Each step will run in a separate pod.

Anatomy of an Argo WF

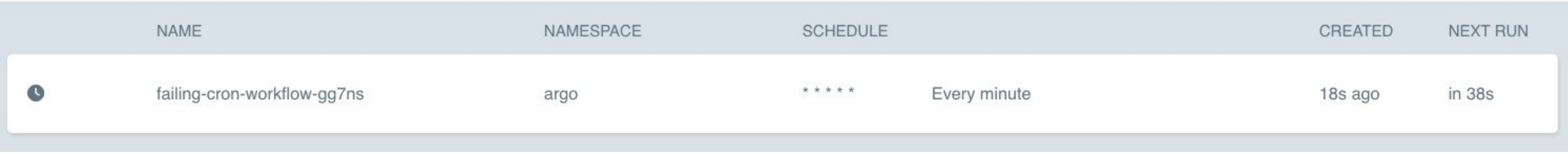

workflow.yaml
Where we define our WF
Metadata
Metadata about the workflow.
Specs
Specification of the workflow
Name
Name of our workflow
Labels
Labels for organization and categorization
Annotations
Metadata with custom labels
Entrypoint
The template the workflow will start with
Templates
Reusable unit of execution

DAGs



- DAGs
  A graph of steps that can create a complex topology.
- Dependencies
  With using “depends”, we can define which step should start after the finish of which ones.
- Parallel Execution
  Depending on the setup, steps can execute in parallel.

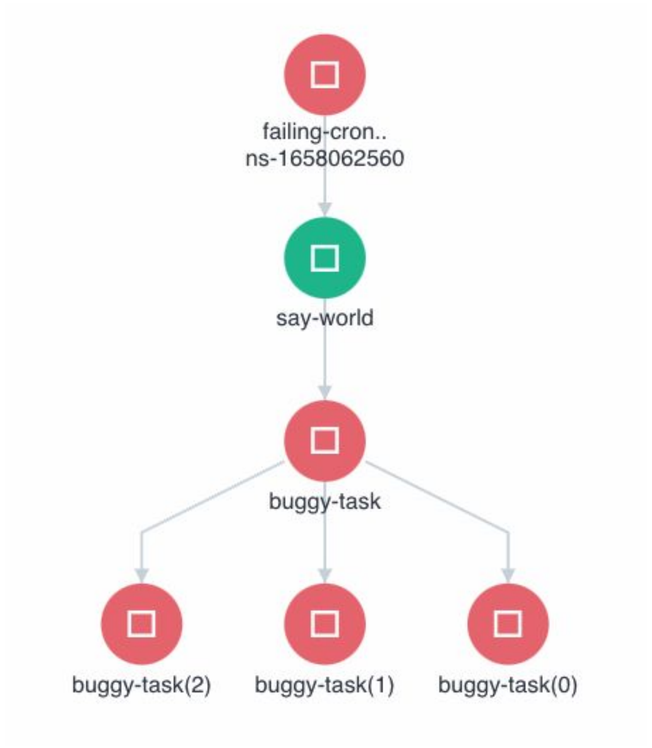
Cron/Scheduled Workflows



	NAME	NAMESPACE	SCHEDULE		CREATED	NEXT RUN
●	failing-cron-workflow-gg7ns	argo	* * * * *	Every minute	18s ago	in 38s

- Scheduling
  We can run any workflow on a schedule.

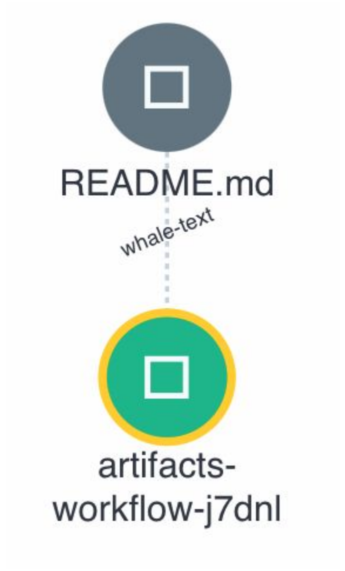
Failure Scenarios



- **Managing Failures**
  We can define what to do in case of which failures

- **Retry**
  Based on conditions, or not, we can retry certain or all steps

Artifacts



- **Artifacts**
  Forwarding artifacts for the steps, and storing them.

- **Debugging**
  Great for building reusable workflows, or just easy debugging upon failures

- **Cloud Agnostic**
  Works with many cloud providers, or just via configmaps.

More Use Cases*

- Automated CT
  Schedule or run model (re)training via CI/CD.
- Backfill Features
  Parameterize your feature extraction pipelines to backfill historical data using the same code.
- Create Templates
  Create workflow templates that users can submit with different parameters, or attach it to a webhook.

*https://argoproj.github.io/argo-workflows/use-cases/