



# ArgoCon

North America



# Untangling the threads

November 12, 2024.  
Salt Lake City



Alan Clucas  
Staff Software Engineer



JM (Jason Meridth)  
Senior Software Engineer



# What is a trace?

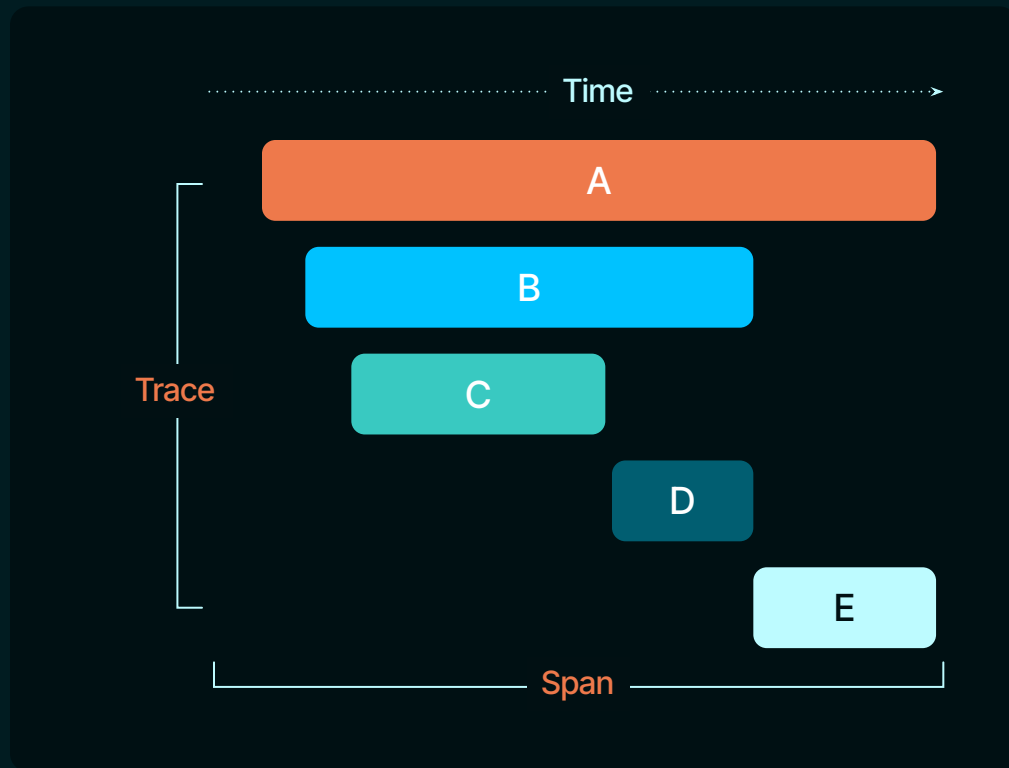
Traces consist of spans

Spans start and stop in time

Spans can have child spans

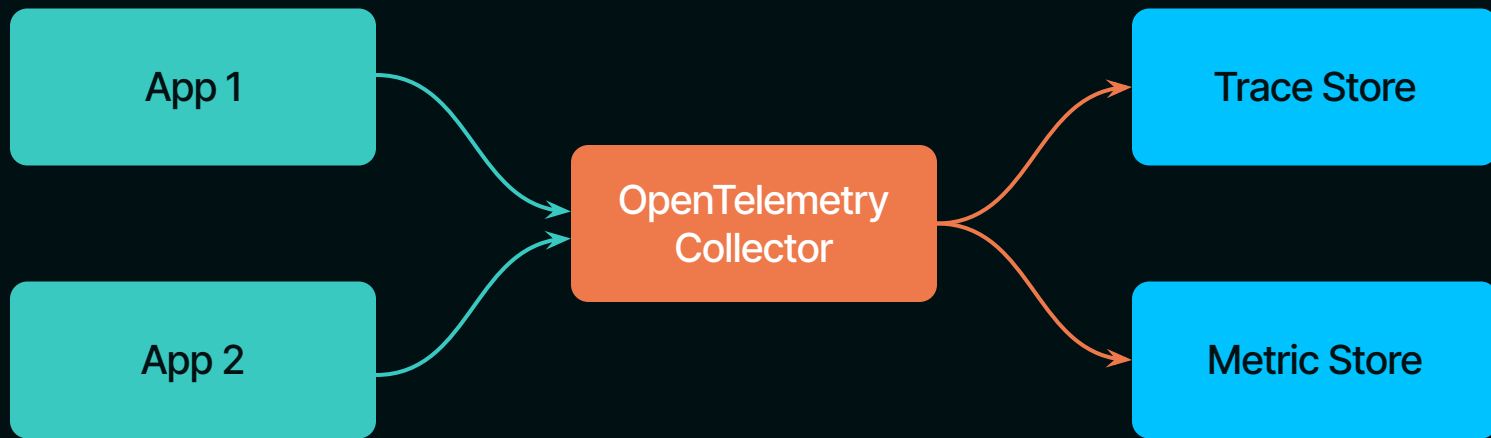
Spans can have attributes

Events are timestamps inside a span



# OpenTelemetry collector

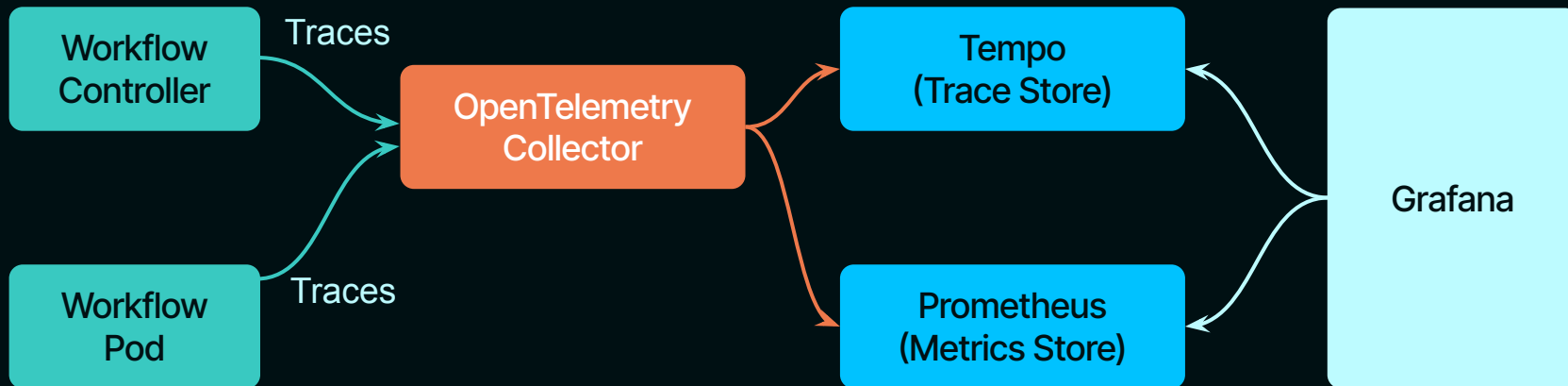
OpenTelemetry recommended architecture



# Architecture details

Specifics:

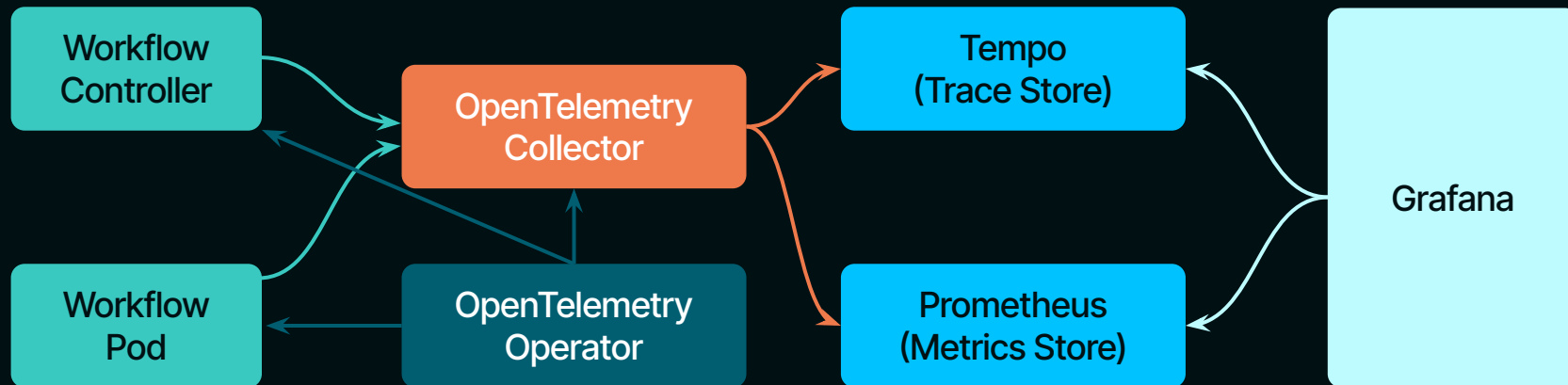
- Sending traces into the collector
- Storing traces and metrics in Grafana stack
- Visualising with Grafana



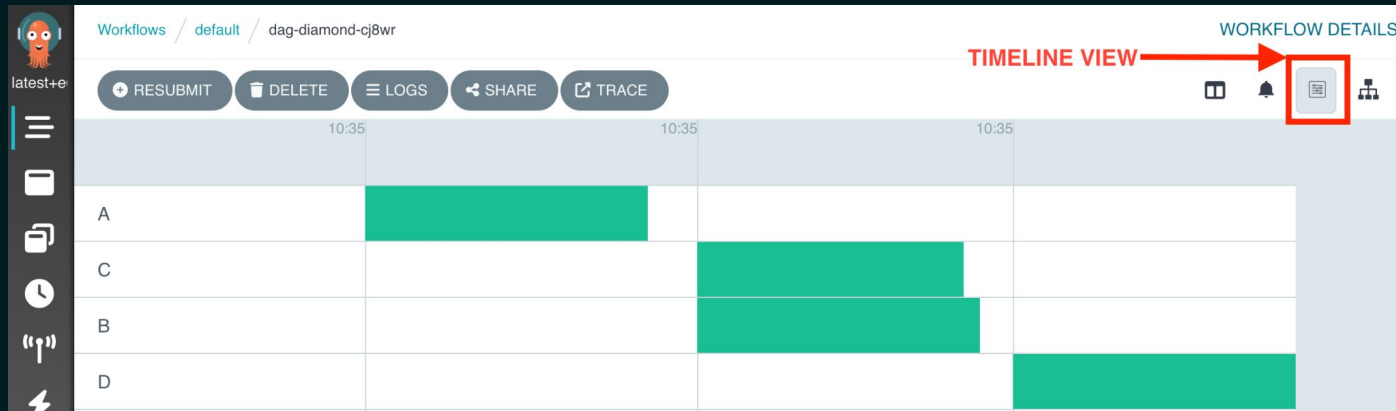
# OpenTelemetry operator

Operator is

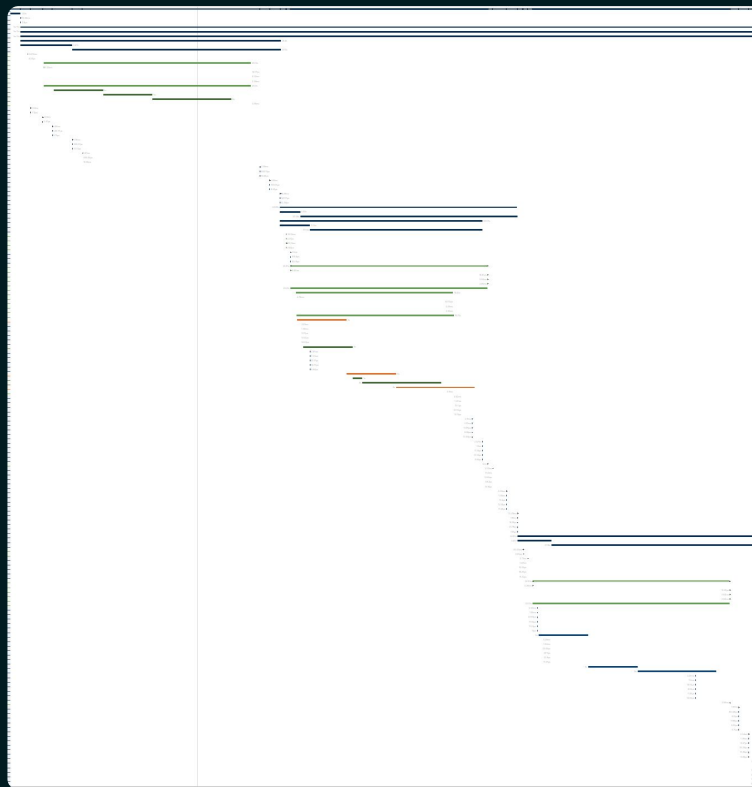
- Managing collector
- And controlling delivery



# Demo DAG Diamond



# A full trace of the same





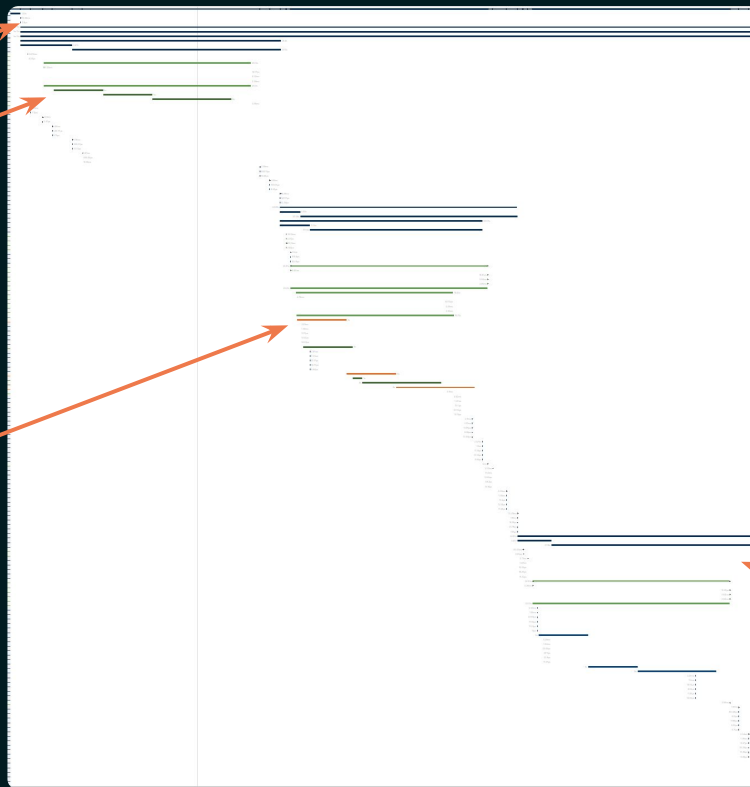
# A full trace of the same

Overall workflow

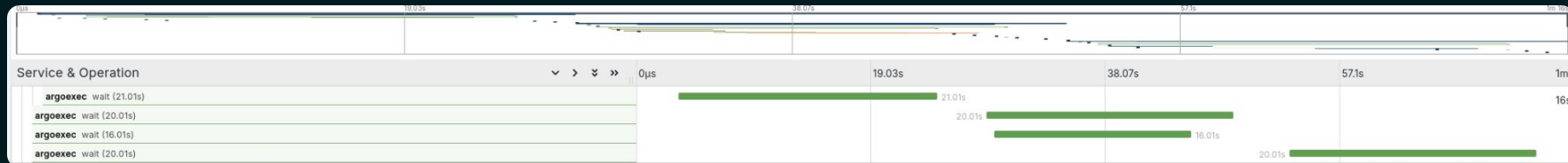
A

B & C

D



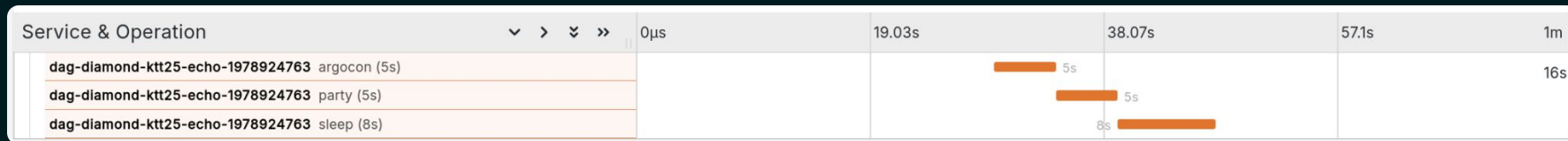
# Just the nodes



# Tracing inside the pods

## Example of tracing in a pod

```
export TRACEPARENT="${ARGO_OTEL_traceparent}"
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
/otel-cli exec --name argocon sleep 5
/otel-cli exec --name party sleep $((1 + $RANDOM % 5))
/otel-cli exec --name sleep sleep 8
```



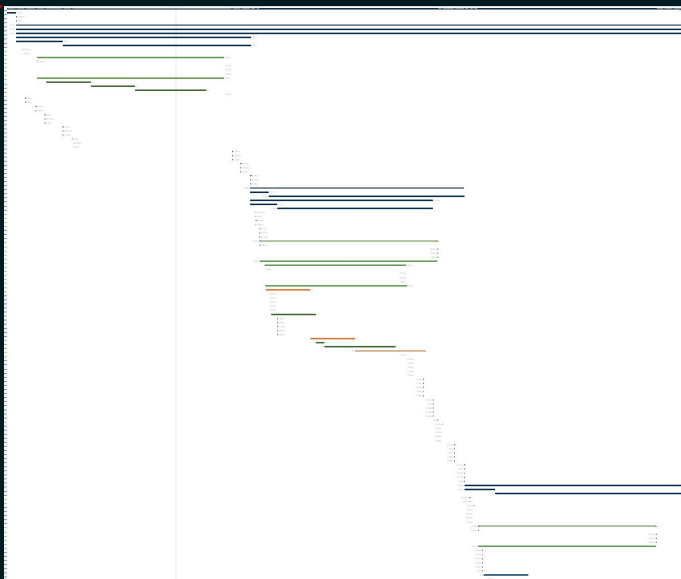
# Inside the pods

dag-diamond-ktt25-echo-1978924763	argocon (5s)	5s	16s																						
dag-diamond-ktt25-echo-1978924763	party (5s)	5s																							
<div><div></div><div><h2>party</h2><p>Service: dag-diamond-ktt25-echo-1978924763   Duration: 5s   Start Time: 34.2s (11:12:57.598)   Kind: client Status: unset   Library Name: github.com/equinix-labs/otel-cli Library Version: 0.4.5 0d4b8a9c49f60a6fc25ed22863259ff573332060 2024-04-01T20:56:07Z</p><div><div>Span Attributes</div><table border="1"><tbody><tr><td>otel</td><td>"true"</td><td></td></tr><tr><td>process.command</td><td>"sleep"</td><td></td></tr><tr><td>process.command_args</td><td>[   "sleep",   "5" ]</td><td></td></tr><tr><td>process.owner</td><td>"root"</td><td></td></tr><tr><td>process.parent_pid</td><td>42</td><td></td></tr><tr><td>process.pid</td><td>35</td><td></td></tr></tbody></table><div>&gt; Resource Attributes: k8s.container.name = main   k8s.namespace.name = default   k8s.node.name = k3d-otel-agent-1..</div><div> SpanID: 9cc1872397f70408</div></div></div></div> <tr><td>dag-diamond-ktt25-echo-1978924763</td><td>sleep (8s)</td><td>8s</td><td></td></tr>				otel	"true"		process.command	"sleep"		process.command_args	[ "sleep", "5" ]		process.owner	"root"		process.parent_pid	42		process.pid	35		dag-diamond-ktt25-echo-1978924763	sleep (8s)	8s	
otel	"true"																								
process.command	"sleep"																								
process.command_args	[ "sleep", "5" ]																								
process.owner	"root"																								
process.parent_pid	42																								
process.pid	35																								
dag-diamond-ktt25-echo-1978924763	sleep (8s)	8s																							

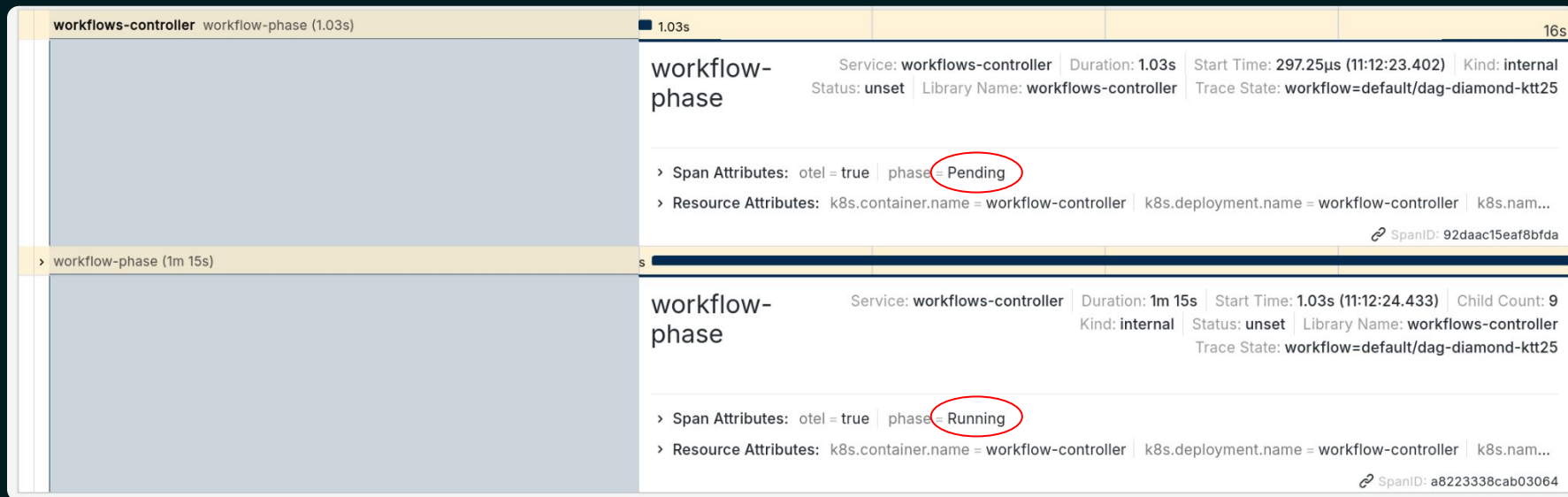
Level of detail is useful and probably 🤑

# Pending workflows

Service & Operation		0µs	19.03s	38.07s	57.1s	1m
workflows-controller	workflow-phase (1.03s)	1.03s				16s
workflow-phase (1m 15s)						



# Pending workflows



The screenshot displays two workflow traces from the 'workflows-controller' service. The first trace, 'workflow-phase (1.03s)', shows a 'Pending' status. The second trace, 'workflow-phase (1m 15s)', shows a 'Running' status. Both spans are circled in red to highlight their current state.

Trace Name	Duration	Status	Phase	SpanID
workflow-phase	1.03s	unset	Pending	92daac15eaf8bfda
workflow-phase	1m 15s	unset	Running	a8223338cab03064

**Trace 1: workflow-phase (1.03s)**

- Service: workflows-controller
- Duration: 1.03s
- Start Time: 297.25µs (11:12:23.402)
- Kind: internal
- Status: unset
- Library Name: workflows-controller
- Trace State: workflow=default/dag-diamond-ktt25
- Span Attributes: otel = true | phase = Pending
- Resource Attributes: k8s.container.name = workflow-controller | k8s.deployment.name = workflow-controller | k8s.nam...
- SpanID: 92daac15eaf8bfda

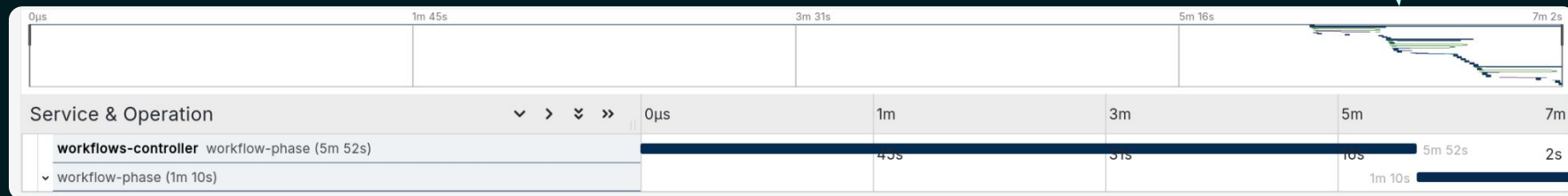
**Trace 2: workflow-phase (1m 15s)**

- Service: workflows-controller
- Duration: 1m 15s
- Start Time: 1.03s (11:12:24.433)
- Child Count: 9
- Kind: internal
- Status: unset
- Library Name: workflows-controller
- Trace State: workflow=default/dag-diamond-ktt25
- Span Attributes: otel = true | phase = Running
- Resource Attributes: k8s.container.name = workflow-controller | k8s.deployment.name = workflow-controller | k8s.nam...
- SpanID: a8223338cab03064

# Pending workflows

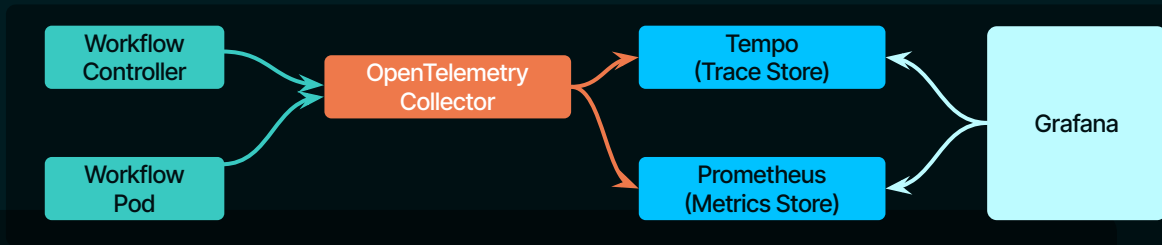
This one takes 7 minutes?

All the action



# Span metrics

Connectors are specialized components that bridge the different pipelines within the OpenTelemetry Collector.

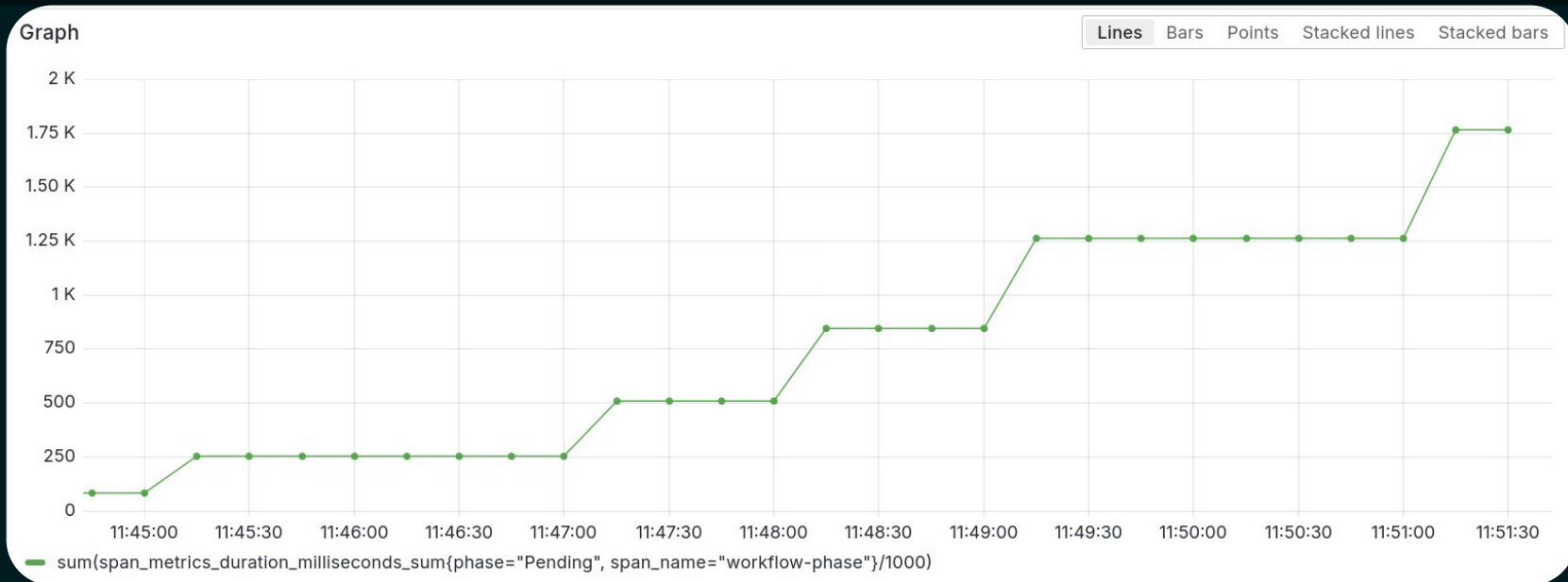


```
connectors:  
  spanmetrics:  
    namespace: span.metrics  
    dimensions:  
      - name: phase
```





# How much are we limited by parallelism


```
sum(span metrics duration milliseconds sum{phase="Pending", span_name="workflow-phase"}/1000)
```



# Node Pending

  
pending-rxn9z

  
A

NAME	pending-rxn9z.A
ID	pending-rxn9z-1885800223
POD NAME	pending-rxn9z-pending-1885800223
HOST NODE NAME	k3d-otel-agent-1
TYPE	Pod
PHASE	 Pending
MESSAGE	ImagePullBackOff: Back-off pulling image "doesnotexist:latest"

operate (1.36ms)

▼ result-reconciliation (506.76µs)

single-result-reconciliation (7.07µs)

node-phase (11.98s)

1.36ms

506.76µs

7.07µs


11.98s

node-phase

Service: workflows-controller | Duration: 11.98s | Start Time: 4.45s (14:19:59.529) | Kind: internal  
Status: unset | Library Name: workflows-controller | Trace State: workflow=default/pending-rxn9z

> Span Attributes: message = ErrImagePull | node = pending-rxn9z-1885800223 | otel = true | phase = Pending

> Resource Attributes: k8s.container.name = workflow-controller | k8s.deployment.name = workflow-controller | k8s.nam...

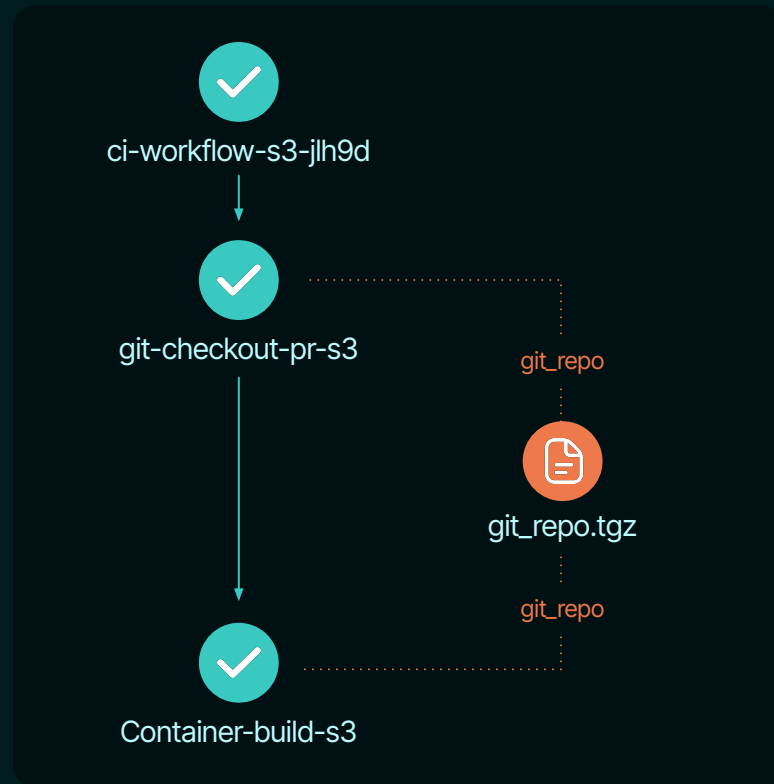
 SpanID: b4bb28b3feae5cf4

# CI Example

Two steps

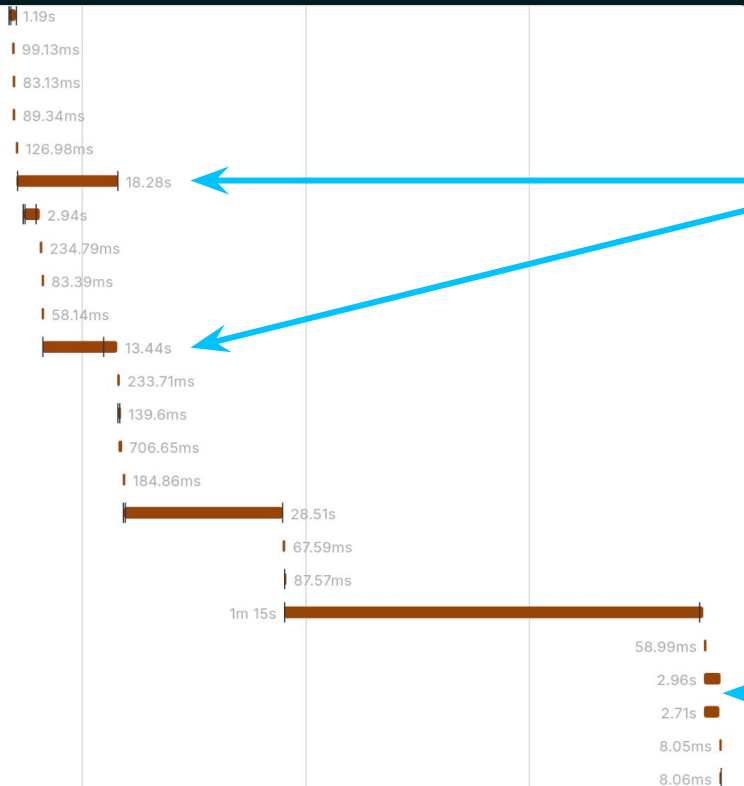
- 1 Checkout Repo
  - Store this as an artifact
- 2 Build docker image using BuildKit
  - Repo comes in as an artifact
  - BuildKit has tracing support

Here we are building workflows CLI+UI  
This is a multistage Dockerfile



# Buildkit output

```
[argo-ui 2/7] RUN apk update && apk add --no-cache git (1.19s)
[argocli 2/5] WORKDIR /home/argo (99.13ms)
[argocli 3/5] COPY hack/ssh_known_hosts /etc/ssh/ (83.13ms)
[argocli 4/5] COPY hack/nsswitch.conf /etc/ (89.34ms)
[argo-ui 3/7] COPY ui/package.json ui/yarn.lock ui/ (126.98ms)
[argo-ui 4/7] RUN --mount=type=cache,target=/root/.yarn YARN_CACH
[builder 2/7] RUN apk update && apk add --no-cache git make ca-certif
[builder 3/7] WORKDIR /go/src/github.com/argoproj/argo-workflows (2
[builder 4/7] COPY go.mod . (83.39ms)
[builder 5/7] COPY go.sum . (58.14ms)
[builder 6/7] RUN --mount=type=cache,target=/go/pkg/mod go mod dc
[builder 7/7] COPY . . (233.71ms)
[argocli-build 1/4] RUN mkdir -p ui/dist (139.6ms)
[argo-ui 5/7] COPY ui ui (706.65ms)
[argo-ui 6/7] COPY api api (184.86ms)
[argo-ui 7/7] RUN --mount=type=cache,target=/root/.yarn YARN_CACH
[argocli-build 2/4] COPY --from=argo-ui ui/dist/app ui/dist/app (67.59m
[argocli-build 3/4] RUN touch ui/dist/app/index.html (87.57ms)
[argocli-build 4/4] RUN --mount=type=cache,target=/go/pkg/mod --mc
[argocli 5/5] COPY --from=argocli-build /go/src/github.com/argoproj/ar
exporting to image (2.96s)
  export layers (2.71s)
remotes.docker.resolver.HTTPRequest (8.05ms)
  HTTP HEAD (8.06ms)
```



Parallel  
build steps

Export

# Buildkit output

This will get  
expensive

```
[builder 6/7] RUN --mount=type=cache,target=/go/pkg/mod go mod download
```

Service: ci-workflow-s3-7hfds-container-build-2620049006

Duration: 13.44s

Start Time: 32.94s (08:58:58.656) | Kind: internal

Status: unset

Library Name: go.opentelemetry.io/otel/sdk/tracer

## Span Attributes

otel "true"

vertex "sha256:637780026229faf7a9a7465b8cc522370120c2c6b23e0d8cd09384a203afc1ae"

Resource Attributes: k8s.container.name = main | k8s.namespace.name = default | k8s.node.name = ...

## Events (4)

- > 32.94s: message = ExecOp started
- > 32.99s: message = Container created
- > 32.99s: message = Container started
- > 43.79s: message = Container exited | exit.code = 0

Log timestamps are relative to the start time of the full trace.

SpanID: e2f82857639010ee

# Artifact transfer time

```
argoexec load artifacts (13.64μs)
```

13.64μs

```
argoexec save artifacts (266.17ms)
```

266.17ms

```
argoexec load artifacts (824.49ms)
```

824.49ms

The artifact here is the git repo.  
Curious that it takes **3 times** as long to load as it does to save.



# Artifacts again



20.57s

**load artifacts**

Service: argoexec | Duration: 20.57s | Start Time: 47.91s (09:34:34.281)  
Kind: internal | Status: unset | Library Name: argoexec  
Trace State: workflow=default/artifact-passing-7qhzf

> Span Attributes: otel = true

> Resource Attributes: k8s.container.name = wait | k8s.namespace.name = default | k8s.node.nam...

▼ Events (2)

- > 47.91s: message = download artifact | file = foo
- > 51.76s: message = download artifact | file = bar

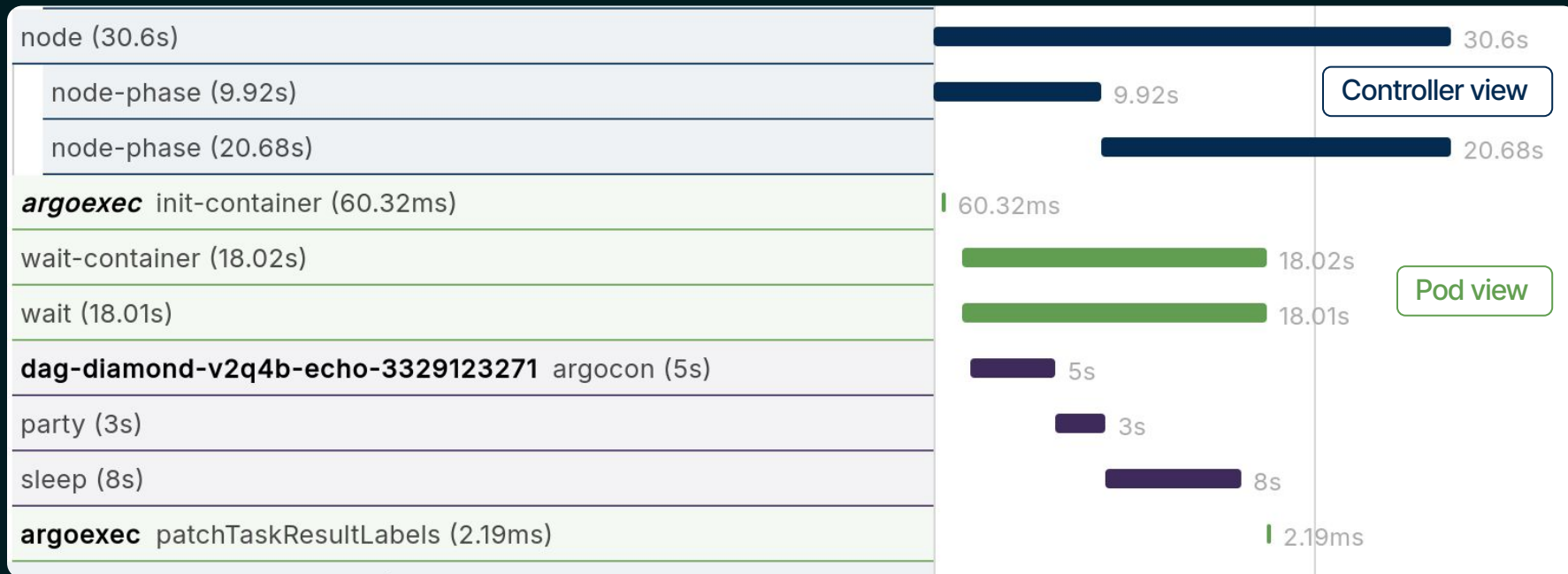
Log timestamps are relative to the start time of the full trace.

SpanID: 23e82e3d80762573

Events: timestamp + data within a span

# As a developer I'd like to know more...

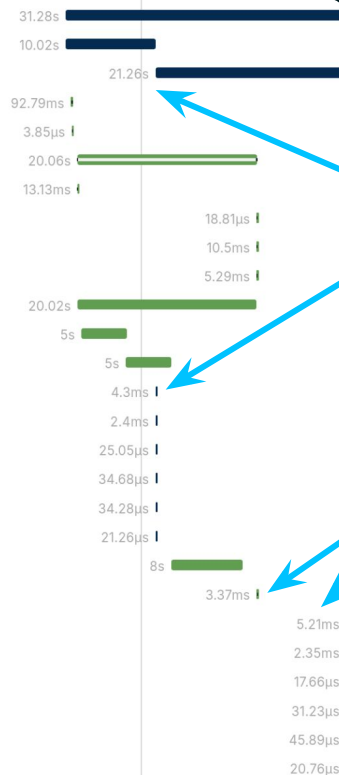
Going back to the initial DAG Diamond...





# As a developer I'd like to know more...

```
node (31.28s)
node-phase (10.02s)
node-phase (21.26s)
argoexec init-container (92.79ms)
load artifacts (3.85µs)
wait-container (20.06s)
createTaskResult (13.13ms)
save artifacts (18.81µs)
createTaskResult (10.5ms)
patchTaskResult (5.29ms)
wait (20.02s)
dag-diamond-mk2js-echo-8163044 argocon (5s)
party (5s)
workflows-controller operate (4.3ms)
result-reconciliation (2.4ms)
single-result-reconciliation (25.05µs)
single-result-reconciliation (34.68µs)
single-result-reconciliation (34.28µs)
single-result-reconciliation (21.26µs)
dag-diamond-mk2js-echo-8163044 sleep (8s)
argoexec patchTaskResultLabels (3.37ms)
workflows-controller operate (5.21ms)
result-reconciliation (2.35ms)
single-result-reconciliation (17.66µs)
single-result-reconciliation (31.23µs)
single-result-reconciliation (45.89µs)
single-result-reconciliation (20.76µs)
```



Reconcile

Why this delay

# What next



<https://github.com/pipekit/argo-workflows/tree/tracing-base>

<https://github.com/Joibel/otel-deploy>

Aiming for release in 3.7

How do we control traces - can be super noisy

Which things get traced?

Turn it on for this workflow only.

Re-submit with tracing on?

Standards compliance - semconv

- <https://opentelemetry.io/docs/specs/semconv/attributes-registry/cicd/>



# Issues during implementation

OpenTelemetry operator doesn't  
annotate initContainers

**Issue #3308**





OpenTelemetry SDK expects spans to start  
and end in the same running binary

**Not necessarily the case  
with the controller restarting**

# About Pipekit



## Scale Argo & Kubernetes with Pipekit

-  Direct support from 40% of the active Argo Workflows maintainers in the world
-  Save engineering time and up to 60% on compute costs
-  Add 3 Argo maintainers and 7 Argo contributors to your team
-  Serving startups & Fortune 500 enterprises since 2021:

### Enterprise Support for Argo:

Ideal for Platform Eng teams scaling with Argo

### Control Plane for Argo Workflows:

Ideal for data teams, granular RBAC, and multi-cluster architectures

# Find us at the Argo stand in the Project Pavillion

Free Argo/Infrastructure Help & Advice:

👤 Booth T33

📅 Regular Office Hours [@pipekit.io/office-hours](https://pipekit.io/office-hours)

