



# Rollout Automation with Temporal

How to upgrade 1000+ ClickHouse Clusters

**July / 2024**

# Table of contents

**01**

## **ClickHouse Cloud**

What is “ClickHouse Cloud” and how does it work behind the scenes?

**02**

## **Upgrading ClickHouse in the Cloud**

What’s the challenge of upgrading ClickHouse clusters in the cloud?

**03**

## **Durable Execution & Temporal**

What is Temporal, how does it work and why is perfect for rollout automation?

**04**

## **Our solution**

Some examples how we do rollouts today (compared to one year ago)

01

# ClickHouse Cloud

What is “ClickHouse Cloud” and how does it work behind the scenes?

# What is ClickHouse?

## Open source

Developed since 2009  
OSS 2016  
35k+ Github stars  
1k+ contributors  
300+ releases

## column-oriented

Best for aggregations  
Files per column  
Sorting and indexing  
Background merges

## distributed

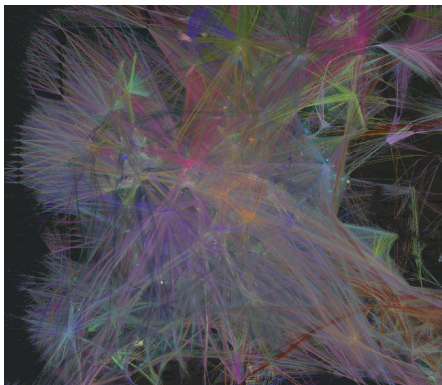
Replication  
Sharding  
Multi-master

## OLAP database

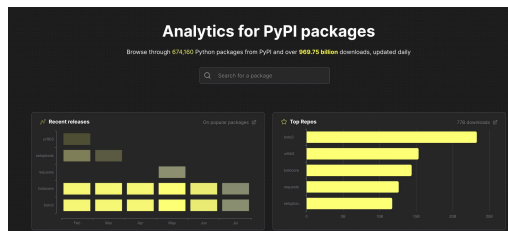
Analytics use cases  
Aggregations  
Visualizations  
Mostly immutable data

# What is ClickHouse?

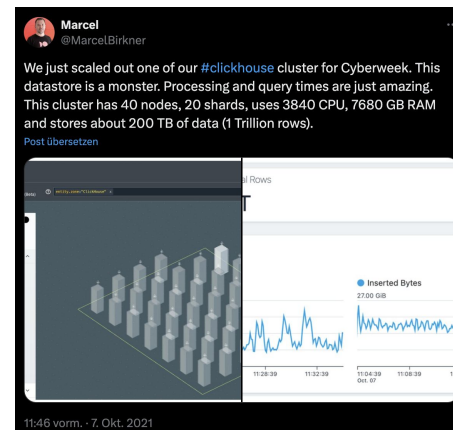
[adzb.exposed](https://adzb.exposed)



[clickpy.clickhouse.com](https://clickpy.clickhouse.com)



Instana

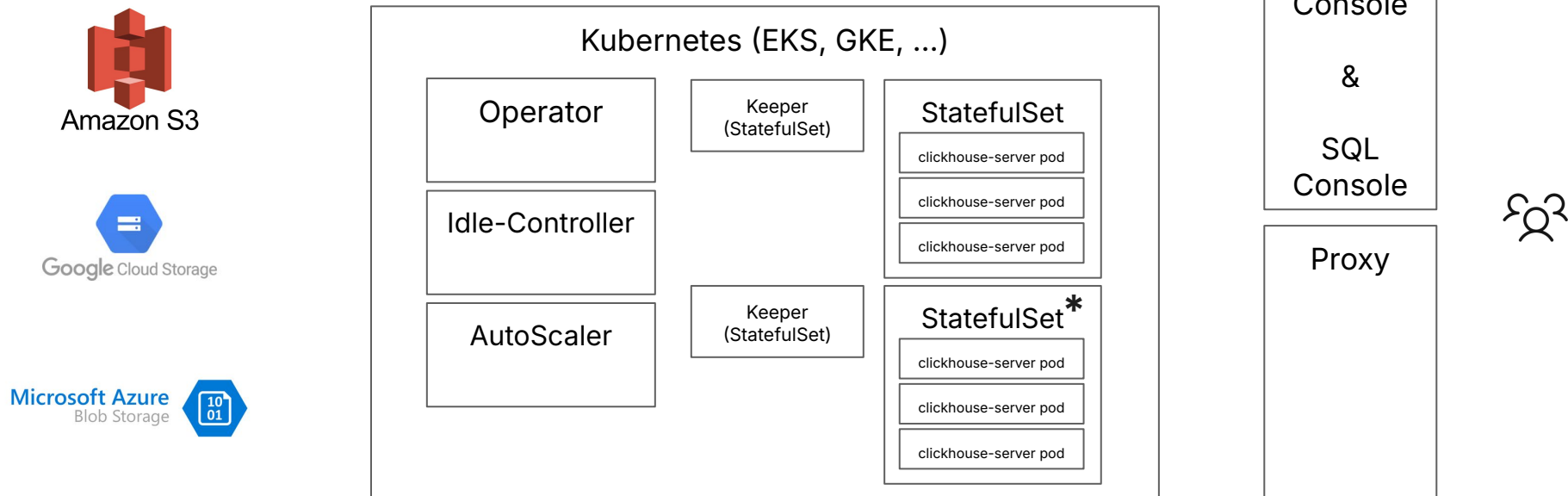


# What is special about ClickHouse “Cloud”?

- It's a **service** - sign up and start using it
- Separation of Storage and Compute
  - Unlimited Storage (S3, GCS, Azure BlobStorage)
  - Compute scales independently

# Architecture of ClickHouse Cloud

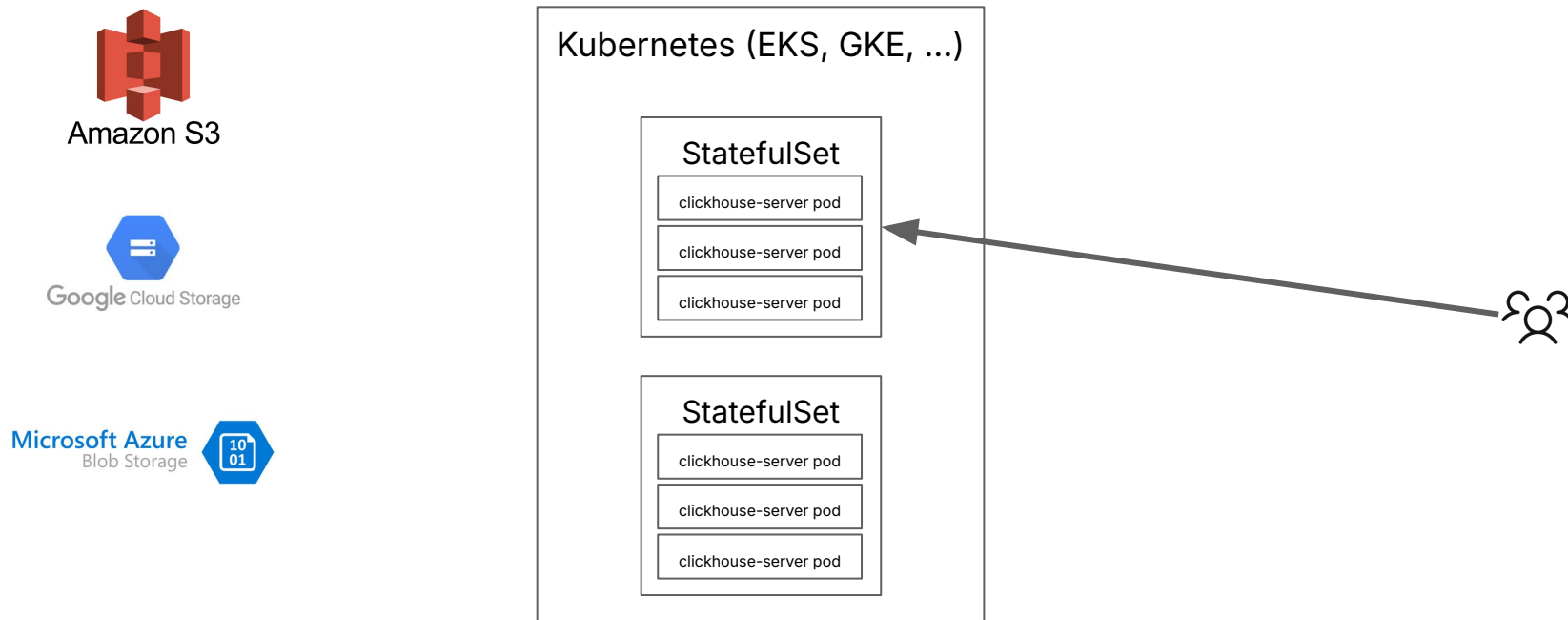
(quite complex)



\* We actually have one StatefulSet per Pod - see Manish's KubeCon talk  
["Fantastic Ordinals and How to Avoid Them: Auto-Scaling Challenges in a Cloud Database"](#)

# Architecture of ClickHouse Cloud

(simplified)





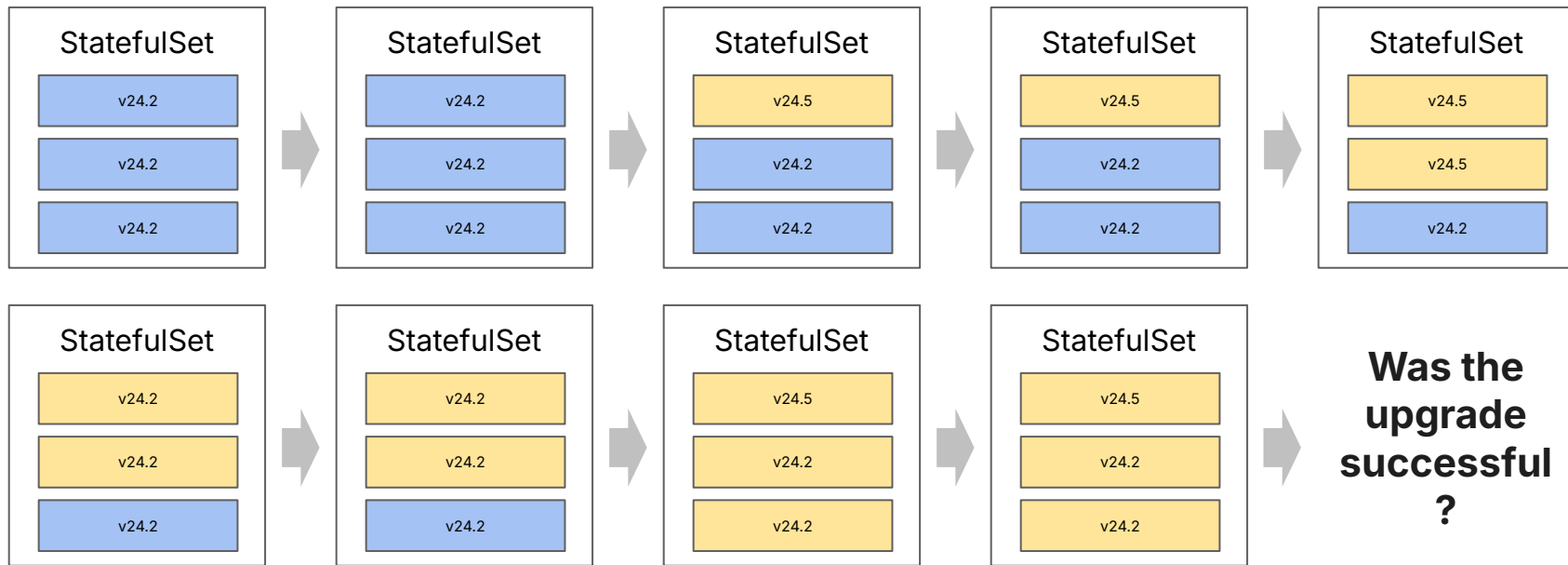
**02**

# Upgrading ClickHouse

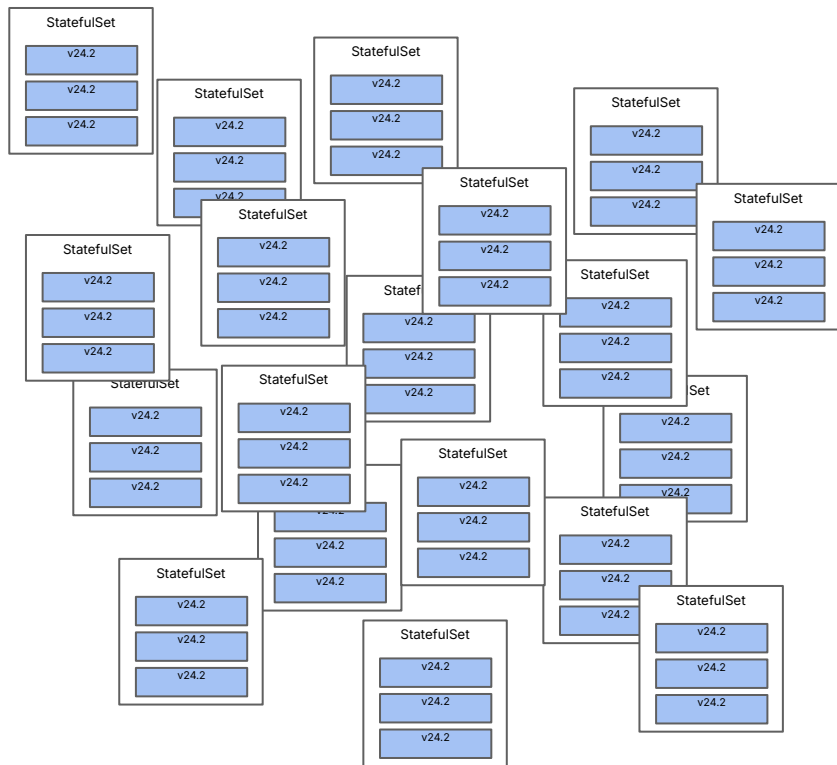
What's the challenge of upgrading ClickHouse clusters in the cloud?

# Upgrading a ClickHouse cluster

```
kubectl patch sts my-clickhouse -p \
  '{"spec": {"template": {"spec":{"containers":[{"name":"clickhouse","tag":"24.5"}]}}}}'
```



# Upgrading 1000+ ClickHouse Clusters



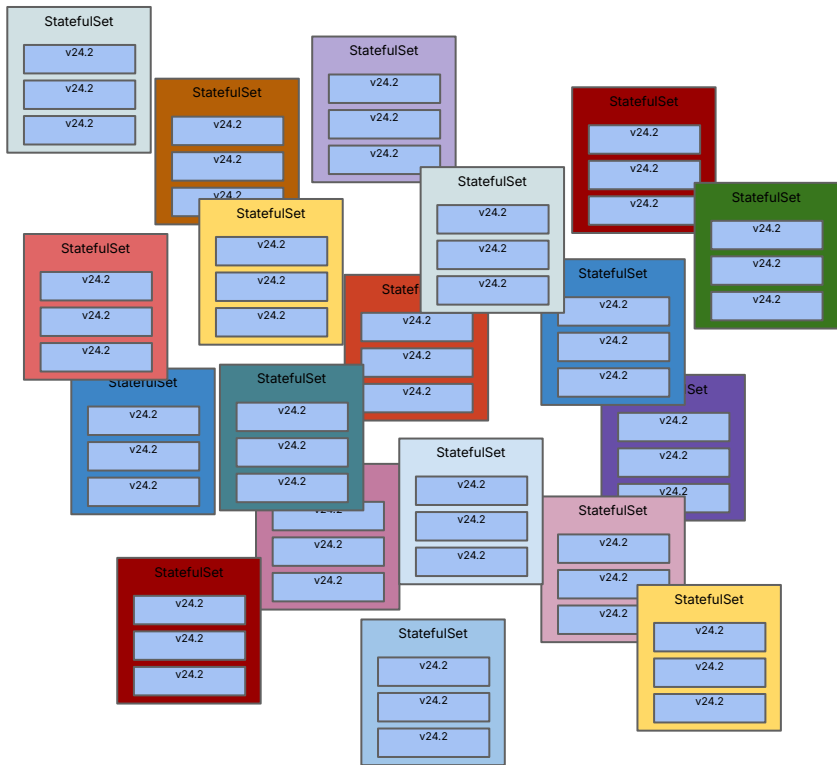
Naive Approach: all at once

- Will overload infrastructure
- Huge blast radius of potential bugs

Naive Approach #2: one after the other

- Too slow

# Upgrading 1000+ ClickHouse Clusters



- **Vastly different use cases** (a new version can be perfectly fine for 98% of the customers and break horribly for the 20 remaining clusters)
- Slightly different configurations and feature flags
- Differences across cloud providers
- ...

# Upgrading 1000+ ClickHouse Clusters

Requirements for a rollout automation system

- As fast as possible, but slow enough to still give time to react
- Runs unattended for several days
- Ability to inspect progress
- Allows interaction with engineers (e.g. pause & resume)
- Can integrate with other systems and tools  
(e2e tests, monitoring, slack, github)
- Flexible

# 03

## Temporal

What is “Durable Execution” and why is perfect for rollout automation?

# How hard can it be ...

- to call an unreliable API reliably
- wait for up to 3h for a result
- repeat these steps 1000 times
- **even if the process is restarted a dozen times in between**

# What is “durable execution”?

<https://temporal.io/blog/building-reliable-distributed-systems-in-node>

Durable execution systems run our code in a way that **persists each step** the code takes. If the process or container running the code dies, the code automatically continues running in another process with all state intact, including call stack and local variables.



# Temporal Concepts

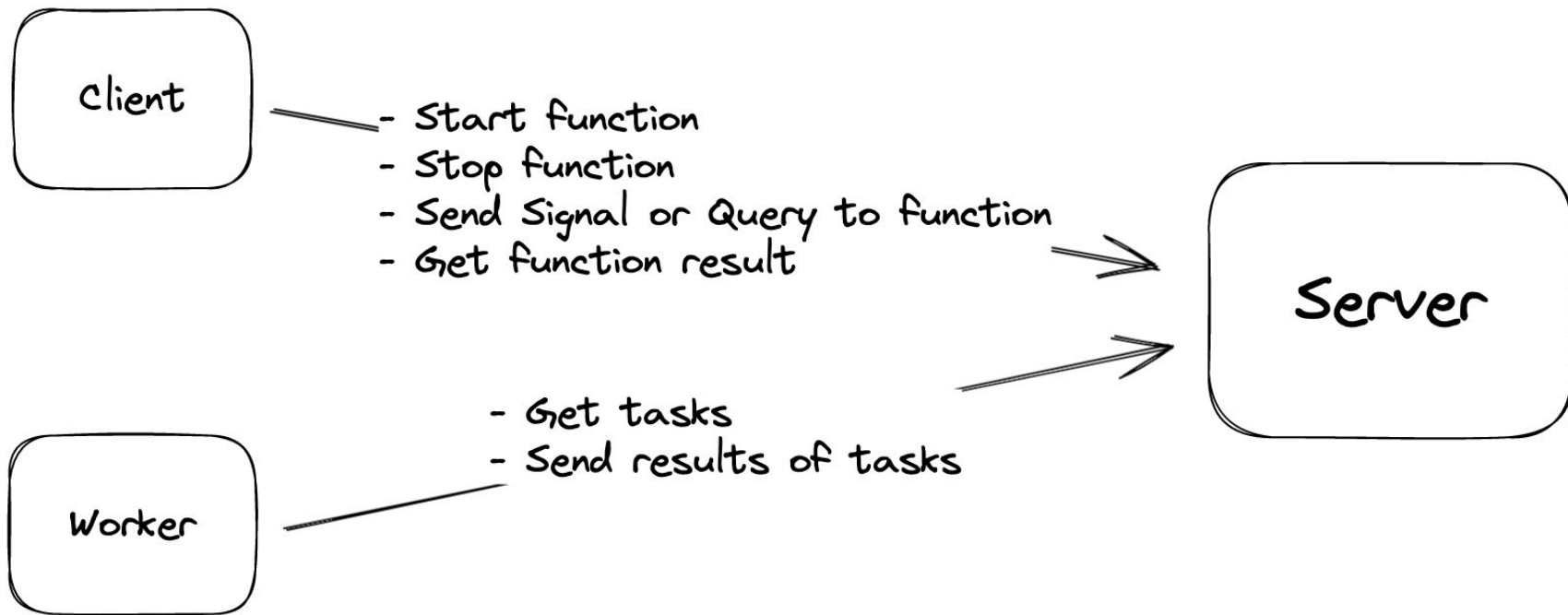
## Workflow

A Temporal Workflow defines the overall flow of the application. Conceptually, a Workflow is a sequence of steps written in a general-purpose programming language.

## Activity

An Activity is an implementation of a task to be performed as part of a larger Workflow. [...] It is in the Activities where all operations that affect the desired results must be implemented.

# How does "durable execution" work?



# Demo

## Demo 1 - Workflow continues after worker interrupted

```
% temporal [temporal]
➤ - temporal server start-dev > /tmp/temporal-dev-server-log

% sch-sre-meetup [-zsh]
➤ munich-sre-meetup git:(step-1-simple-workflow-with-sleep) go run starter/main.go
2024/08/05 17:21:17 INFO No logger configured for temporal client. Created default one.
2024/08/05 17:21:17 Started workflow WorkflowID hello_world_workflowID RunID 3fed9445-41c3-45d7-973b-f60bc4319304
2024/08/05 17:21:37 Workflow result: workflow-result
➤ munich-sre-meetup git:(step-1-simple-workflow-with-sleep) []

% go [main]
➤ munich-sre-meetup git:(step-1-simple-workflow-with-sleep) go run worker/main.go | jq -c '{time, msg}'
{"time":"2024-08-05T17:21:13.099532+02:00","msg":"Started Worker"}

{"time":"2024-08-05T17:21:17.534531+02:00","msg":"HelloWorld workflow started"}
{"time":"2024-08-05T17:21:17.534559+02:00","msg":"It's now 2024-08-05 15:21:17.532057 +0000 UTC"}
{"time":"2024-08-05T17:21:17.534564+02:00","msg":"Sleeping until 2024-08-05 15:21:47.532057 +0000 UTC"}
{"time":"2024-08-05T17:21:17.534609+02:00","msg":"NewTimer"}
^C
➤ munich-sre-meetup git:(step-1-simple-workflow-with-sleep) echo "worker down..."
worker down...
➤ munich-sre-meetup git:(step-1-simple-workflow-with-sleep) go run worker/main.go | jq -c '{time, msg}'
{"time":"2024-08-05T17:21:30.428675+02:00","msg":"Started Worker"}

{"time":"2024-08-05T17:21:37.543168+02:00","msg":"Done with sleeping. It's now 2024-08-05 15:21:37.540191 +0000 UTC"}
[]

% sh [wsl]
Top left: Temporal dev server
Bottom left: Temporal client which starts the workflow

Top right: The important part. The temporal worker which actually does the work. It is interrupted.
But it continues work after starting up again
[]
~
~
~
~
~
~
~
~
~
~
~
```

# Demo 2 - Temporal takes care of retries

```
temporal (temporal)
+ ~ temporal server start-dev > /tmp/temporal-dev-server-log

[ ]

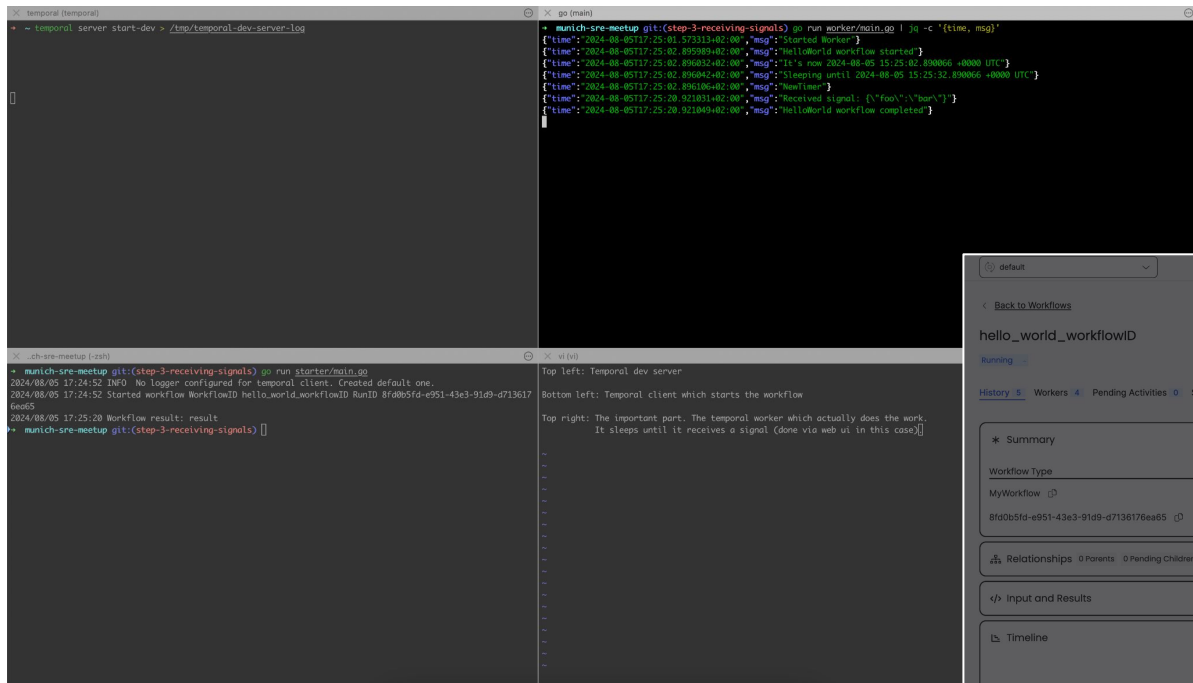
[ ]

ch-sre-meetup (-zsh)
+ ~ munich-sre-meetup git:(step-2-activity-with-retries) go run worker/main.go | jq -c '{time, msg}'
{"time":"2024-08-05T17:23:10.218556+02:00","msg":"Started Worker"}
^C
+ ~ munich-sre-meetup git:(step-2-activity-with-retries) go run starter/main.go
2024/08/05 17:23:20 INFO No logger configured for temporal client. Created default one.
2024/08/05 17:23:20 Started workflow WorkflowID hello_world_workflowID RunID 006a70aa-bef1-4145-9fb5-4c9553a638a8
2024/08/05 17:23:27 Workflow result: foo
+ ~ munich-sre-meetup git:(step-2-activity-with-retries) [ ]

go (main)
+ ~ munich-sre-meetup git:(step-2-activity-with-retries) go run worker/main.go | jq -c '{time, msg}'
{"time":"2024-08-05T17:23:05.729669+02:00","msg":"Started Worker"}
{"time":"2024-08-05T17:23:20.589765+02:00","msg":"HelloWorld workflow started"}
{"time":"2024-08-05T17:23:20.589839+02:00","msg":"ExecuteActivity"}
{"time":"2024-08-05T17:23:20.513873+02:00","msg":"MyActivity - reading /tmp/result"}
{"time":"2024-08-05T17:23:20.513949+02:00","msg":"Activity error."}
{"time":"2024-08-05T17:23:21.517926+02:00","msg":"MyActivity - reading /tmp/result"}
{"time":"2024-08-05T17:23:21.517978+02:00","msg":"Activity error."}
{"time":"2024-08-05T17:23:23.523106+02:00","msg":"MyActivity - reading /tmp/result"}
{"time":"2024-08-05T17:23:23.523472+02:00","msg":"Activity error."}
{"time":"2024-08-05T17:23:27.529844+02:00","msg":"MyActivity - reading /tmp/result"}
{"time":"2024-08-05T17:23:27.531819+02:00","msg":"HelloWorld workflow completed, activity result: foo\n"}

v (v)
Top left: Temporal dev server
Bottom left: Temporal client which starts the workflow
Top right: The important part. The temporal worker which actually does the work.
            It tries to read a file /tmp/result - and retries until the file can actually be read.
```

# Demo 3 - Interacting with the workflow using signals

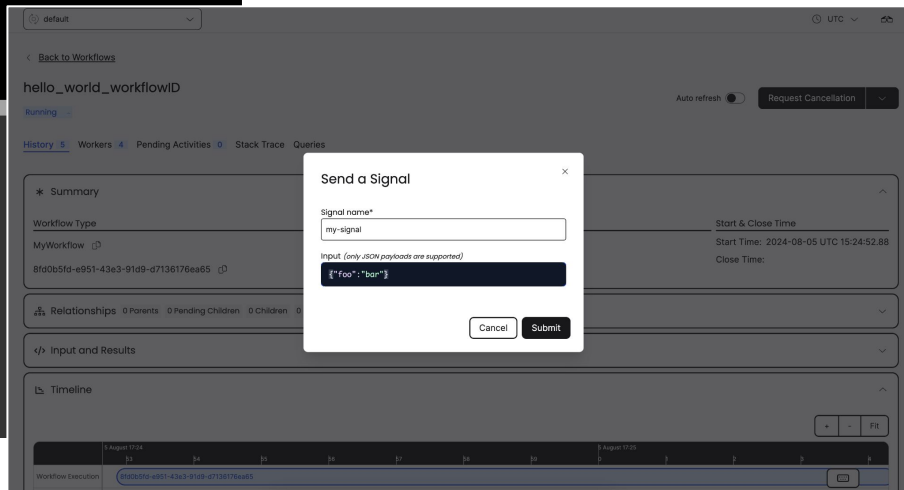


The image shows a terminal window with two panes. The left pane shows the Temporal server logs, and the right pane shows the worker logs. The worker logs indicate that the workflow received a signal and completed successfully.

```
temporal-server start-dev > /tmp/temporal-dev-server-log

+ Munich-sre-meetup git:(step-3-receiving-signals) go run worker/main.go i jq -c '{time, msg}'
{"time": "2024-08-05T17:25:01.57313+02:00", "msg": "Started Worker"}
{"time": "2024-08-05T17:25:02.895989+02:00", "msg": "HelloWorld workflow started"}
{"time": "2024-08-05T17:25:02.896024+02:00", "msg": "It's now 2024-08-05 15:25:02.896066 +0000 UTC"}
{"time": "2024-08-05T17:25:02.896042+02:00", "msg": "Sleeping until 2024-08-05 15:25:32.896066 +0000 UTC"}
{"time": "2024-08-05T17:25:02.896106+02:00", "msg": "NewTimer"}
{"time": "2024-08-05T17:25:20.521831+02:00", "msg": "Received signal: {\"foo\":\"bar\"}"}
{"time": "2024-08-05T17:25:20.921849+02:00", "msg": "HelloWorld workflow completed"}

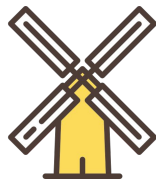
+ Munich-sre-meetup git:(step-3-receiving-signals) go run starter/main.go
2024/08/05 17:24:52 INFO No logger configured for temporal client. Created default one.
2024/08/05 17:24:52 Started workflow WorkflowID hello_world_workflowID RunID 8f0b5fd-e951-43e3-91d9-d7136176e055
2024/08/05 17:25:20 Workflow result: result
+ Munich-sre-meetup git:(step-3-receiving-signals) []
```



# 04

## Our solution

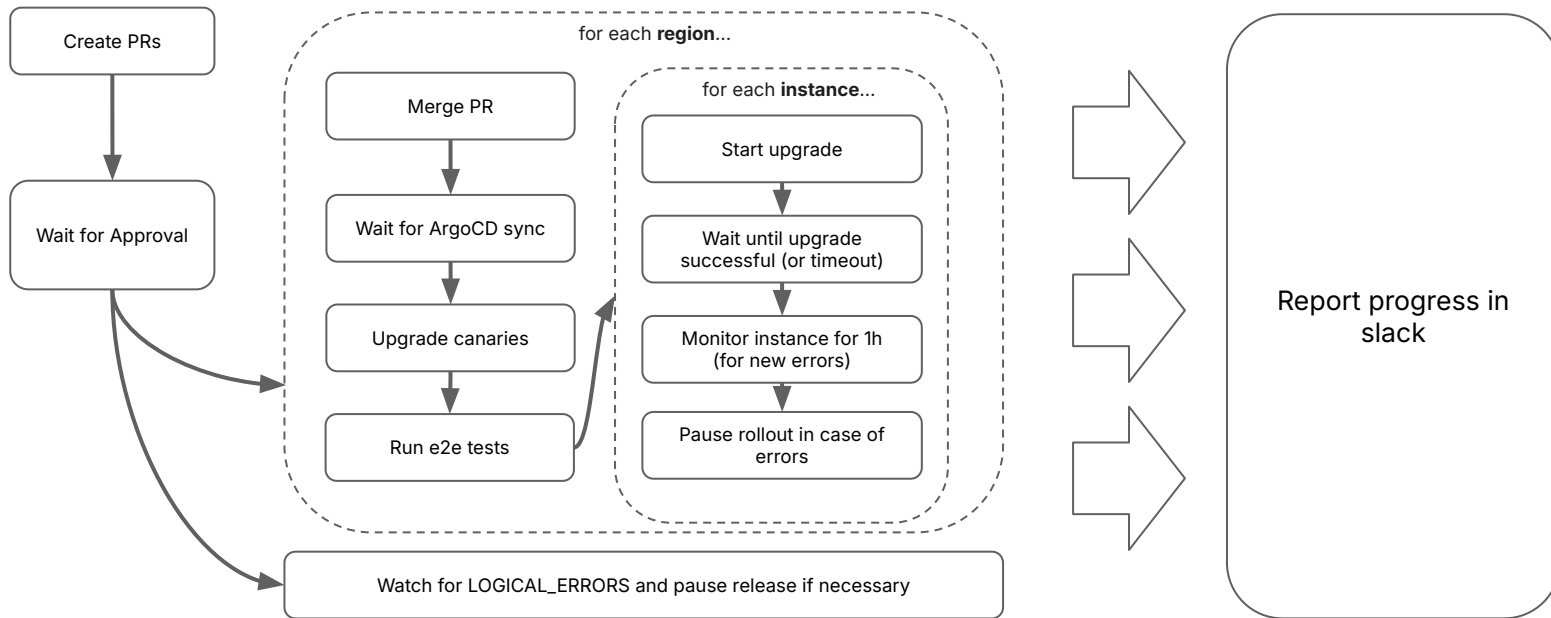
Some examples how we do rollouts today (compared to one year ago)



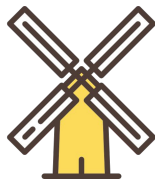
# ClickHouse Molen



Listen to Pause / Reset / Continue signals from engineers







# ClickHouse Molen



**Data-plane bot** APP 12:50  
 [Main-distracted\\_northcutt](#)  
Started new ClickHouse release: distracted\_northcutt  
Release Engineer: [nikitamikhaylov@clickhouse.com](mailto:nikitamikhaylov@clickhouse.com)

Please review PRs  
To start rollout send **Continue** signal to the main workflow. Send **Cancel** to cancel the rollout and cleanup PRs

Release Schedule:  
\* Production small regions  
\* Prod AWS eu-central-1, wait 1h0m0s  
\* Prod AWS eu-west-1, wait 1h0m0s  
\* Prod GCP europe-west4, wait 1h0m0s

```
- description: Update ClickHouse version
operation: add_or_update
path: clickhouse.version
value: "23.9.2.47465"
- description: Change user-level setting
operation: add_or_update
path: extraConfig.users.profiles.default.cluster_for_parallel_replicas
value: "default"
```

47 replies Last reply today at 14:47

**Data-plane bot** APP 4 days ago  
 [Release-adoring\\_turing](#)  
Starting next stage 'Staging dev tier' in 0s at 16 Nov 23 15:17 UTC

**Data-plane bot** APP 4 days ago  
 [Release-adoring\\_turing](#)  
Starting stage: Staging dev tier

**Data-plane bot** APP 4 days ago  
 [Group\\_rollout-adoring\\_turing-staging\\_dev\\_tier-staging\\_dev\\_tier](#)  
Starting updating instances in group: Staging dev tier

**Data-plane bot** APP 4 days ago  
 [Group\\_rollout-adoring\\_turing-staging\\_dev\\_tier-staging\\_dev\\_tier](#)  
Instance update failedInstance tan-dn-76 in staging/aws/eu-west-1 failed to update with error: 'initialState: degraded, instance update marked as failed as it exceeded the autoupdate\_ttl timeout (type: NonRetryableError, retryable: false)'

**Data-plane bot** APP 4 days ago  
 [Group\\_rollout-adoring\\_turing-staging\\_dev\\_tier-staging\\_dev\\_tier](#)  
Instances in group Staging dev tier updated.

**Data-plane bot** APP 4 days ago  
 [Release-adoring\\_turing](#)  
"Stage Staging dev tier is done"Starting next stage 'Staging prod tier' in 0s at 16 Nov 23 15:59 UTC

**Data-plane bot** APP 4 days ago  
 [Release-adoring\\_turing](#)  
Starting stage: Staging prod tier

**Data-plane bot** APP 4 days ago  
 [E2E-distracted\\_northcutt-aws-production-eu-central-1](#)  
E2E tests finished for: aws/production/eu-central-1Seed: 2652460134  
Success: false  
To check failed specs follow [this link](#).  
Logs: [logs in DD](#)

**Data-plane bot** APP 4 days ago  
 [Update\\_base\\_configurations-distracted\\_northcutt-prod\\_aws\\_eu\\_central\\_1-prod\\_aws\\_eu\\_central\\_1](#)  
"Child workflow E2E-distracted\_northcutt-aws-production-eu-central-1 failed."To ignore this failure and continue release send **Continue** signal to [this workflow](#), otherwise send **Cancel** signal.

**Data-plane bot** APP 4 days ago  
 [Errors\\_checks-distracted\\_northcutt-prod\\_aws\\_eu\\_central\\_1](#)  
"Logical errors detected"Found logical **errors** in Cloud: aws Environment: production Region: eu-central-1 Cluster type: data-plane Cluster name: prod-production-eu-central-1-data-plane Namespace: ns-bisque-wa-97

**Data-plane bot** APP 2 minutes ago  
 [Group\\_rollout-adoring\\_turing-staging\\_prod\\_tier-staging\\_prod\\_tier](#)  
Rollout paused for all groups in stageStage will be paused because error rate 12 is over the limit 10. Send 'Continue' or 'Reset' to resume the rollout

**Data-plane bot** APP 3 days ago  
 [tada Main-zealous\\_mclean](#)  
ClickHouse release zealous\_mclean is done

# Temporal

## Great

- It's code  
testing, tooling, CI 🎉
- Active development  
test SDK, replay testing didn't exist when we started
- Flexible  
recently added better error monitoring and soft/hard timeouts

## Challenging

- Workflow versioning  
"nondeterministic workflow definition code or incompatible change"
- Limits  
Actions per Workflow, Child workflows, "ContinueAsNew", Size of Activity Input/Output
- Things work different than in normal go code - You need to embrace the paradigm!