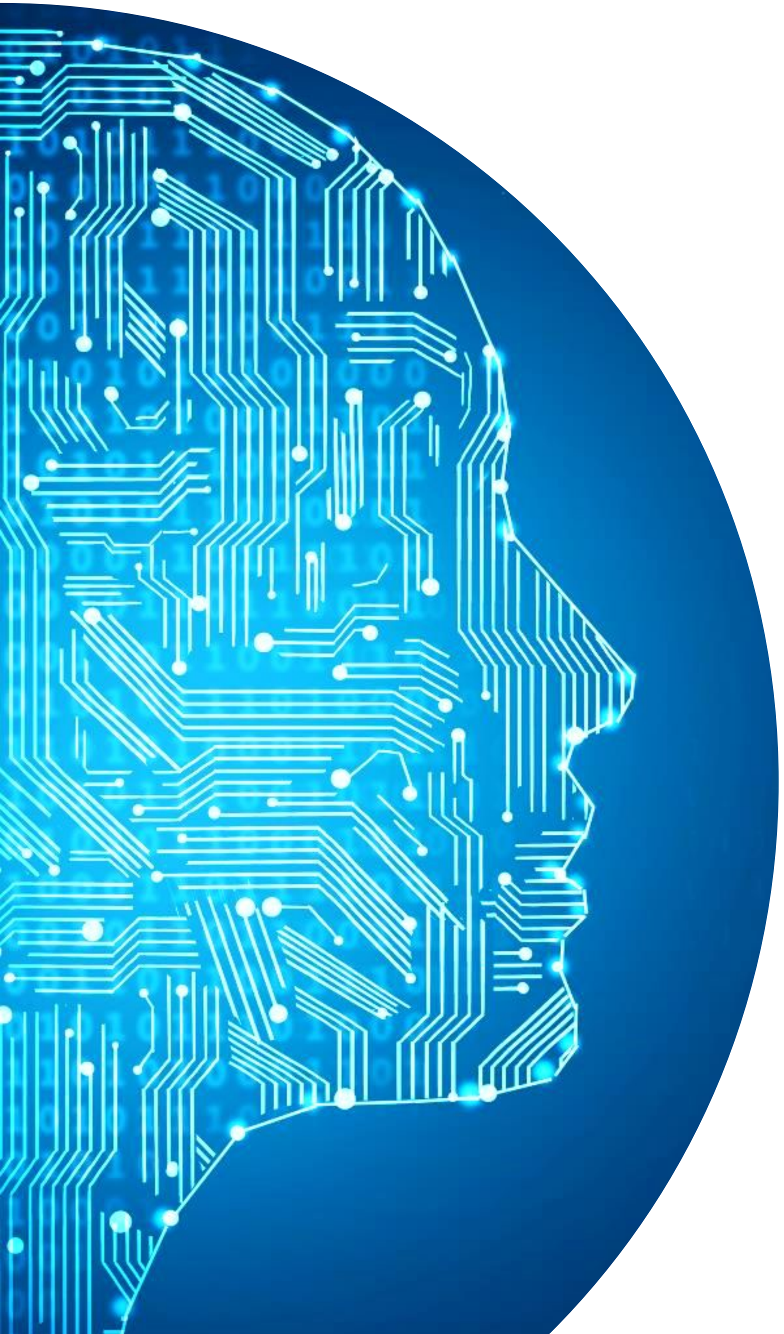
Open Source MANO by ETSI

Managing operations with Temporal in OSM

Mark Beierl (Canonical)
Gulsum Atici (Canonical)

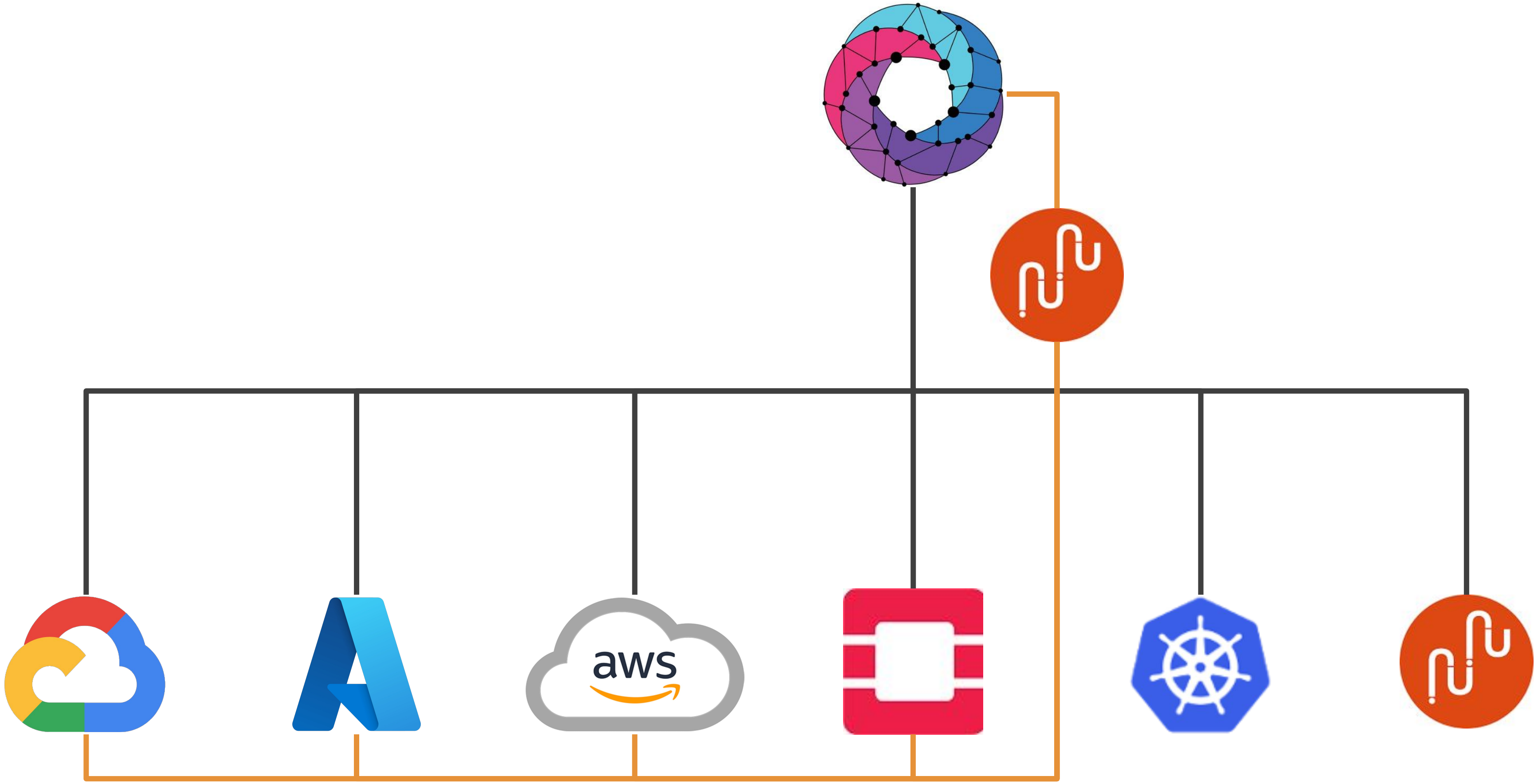
08/03/2023



Agenda

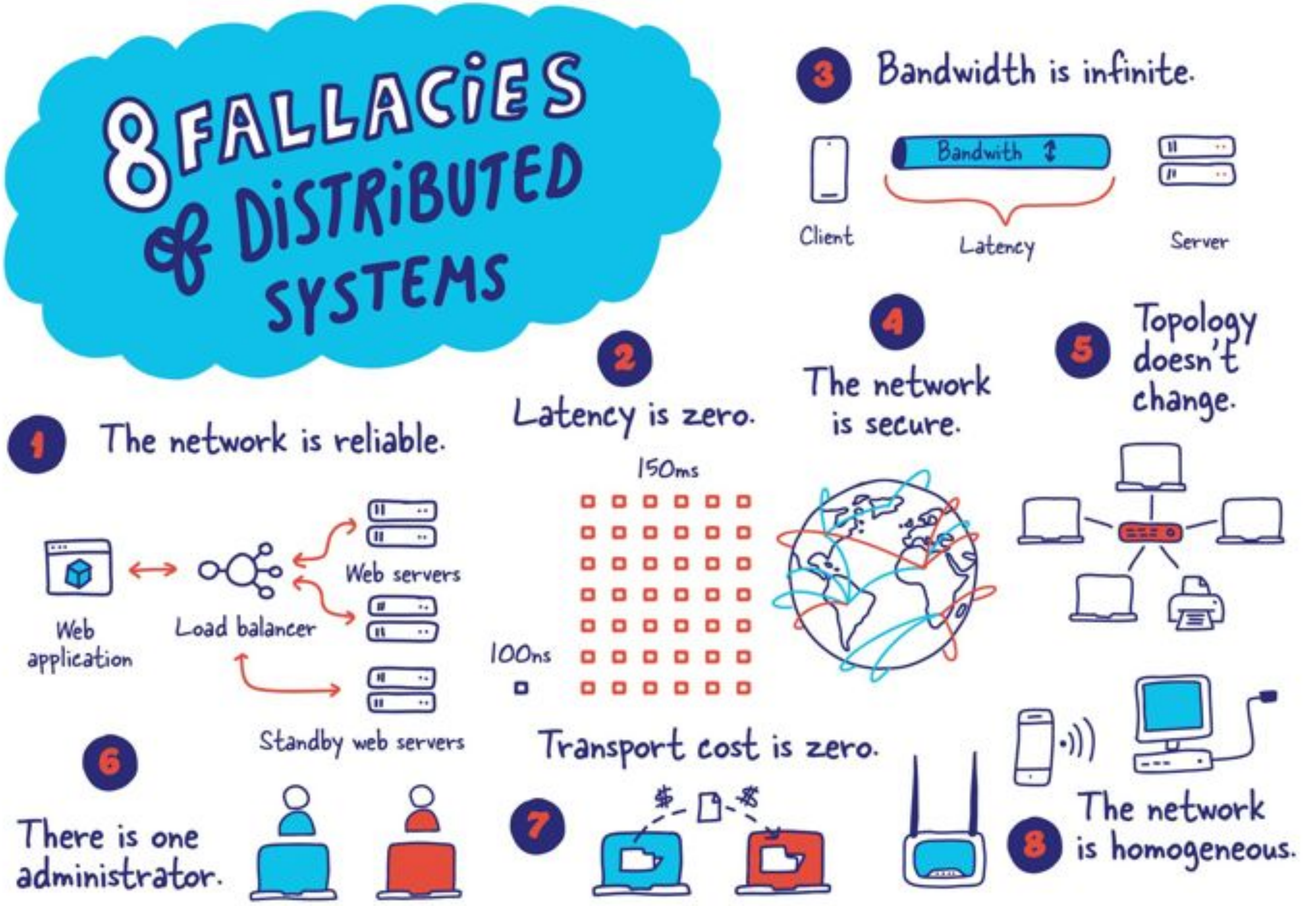
- What is Temporal?
- Managing OSM Operations with Temporal
- Exploring Temporal Concept
- Demo

How Did This Start?

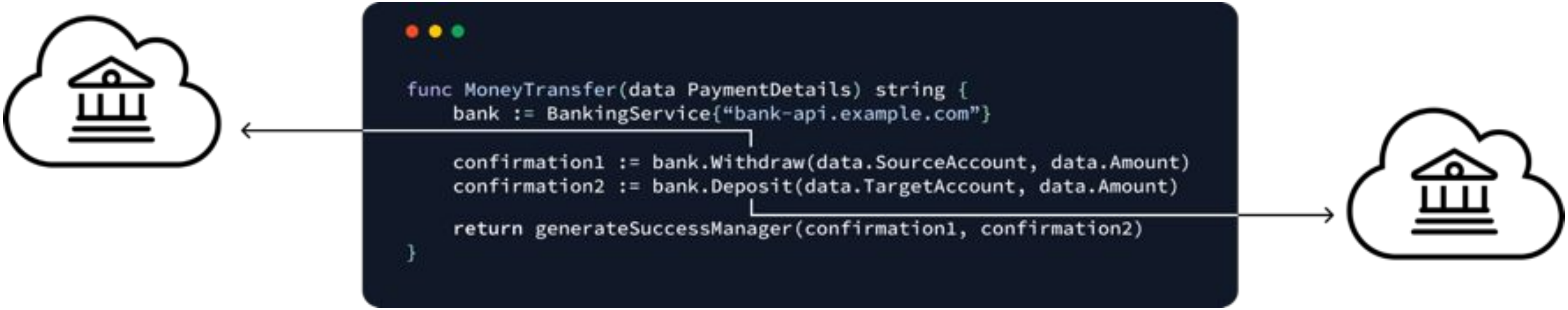


Temporal

Why durable execution changes everything


8 FALLACIES of DISTRIBUTED SYSTEMS
1 The network is reliable.
Web application
Load balancer
Web servers
Standby web servers
2 Latency is zero.
150ms
100ns
3 Bandwidth is infinite.
Bandwith
Client
Latency
Server
4 The network is secure.
5 Topology doesn't change.
6 There is one administrator.
7 Transport cost is zero.
$
$
8 The network is homogeneous.

Engineers have paid the price

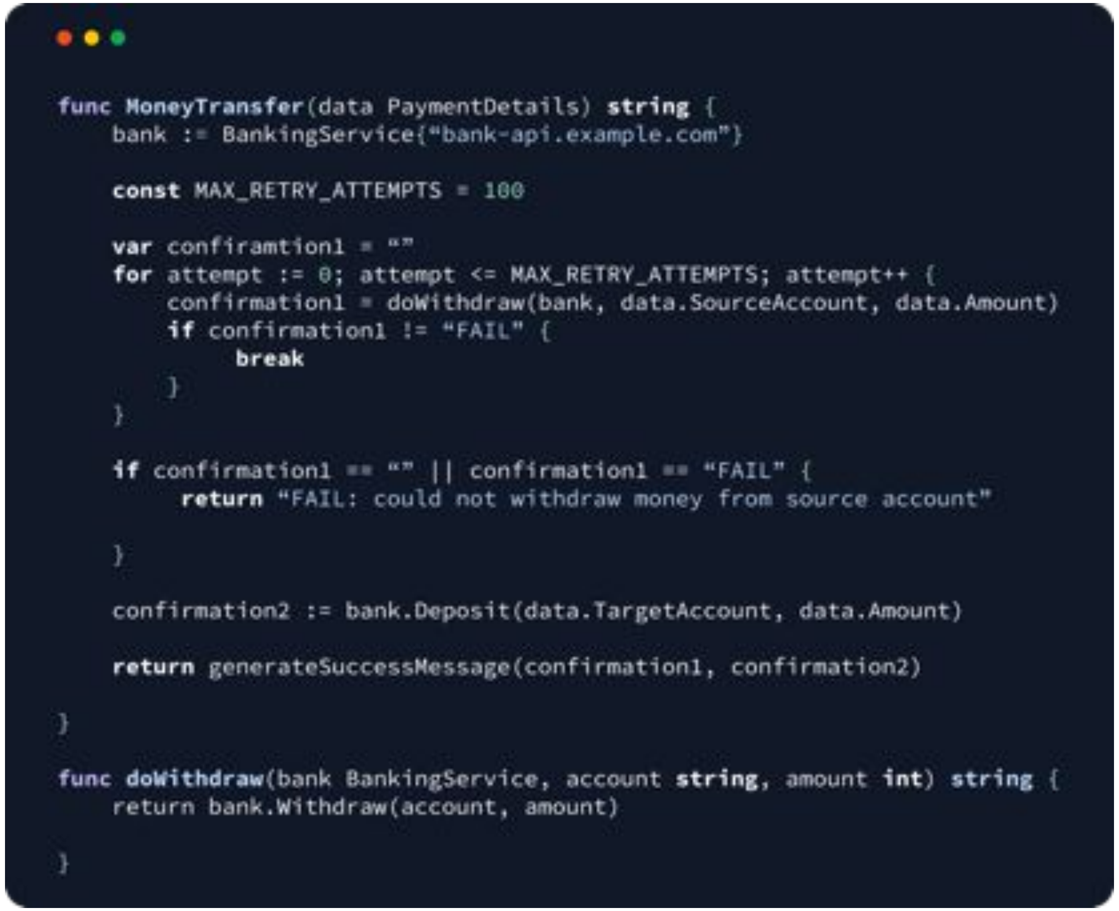


```
func MoneyTransfer(data PaymentDetails) string {
    bank := BankingService{“bank-api.example.com”}

    confirmation1 := bank.Withdraw(data.SourceAccount, data.Amount)
    confirmation2 := bank.Deposit(data.TargetAccount, data.Amount)

    return generateSuccessManager(confirmation1, confirmation2)
}
```

This is a distributed system

Engineers have paid the price again

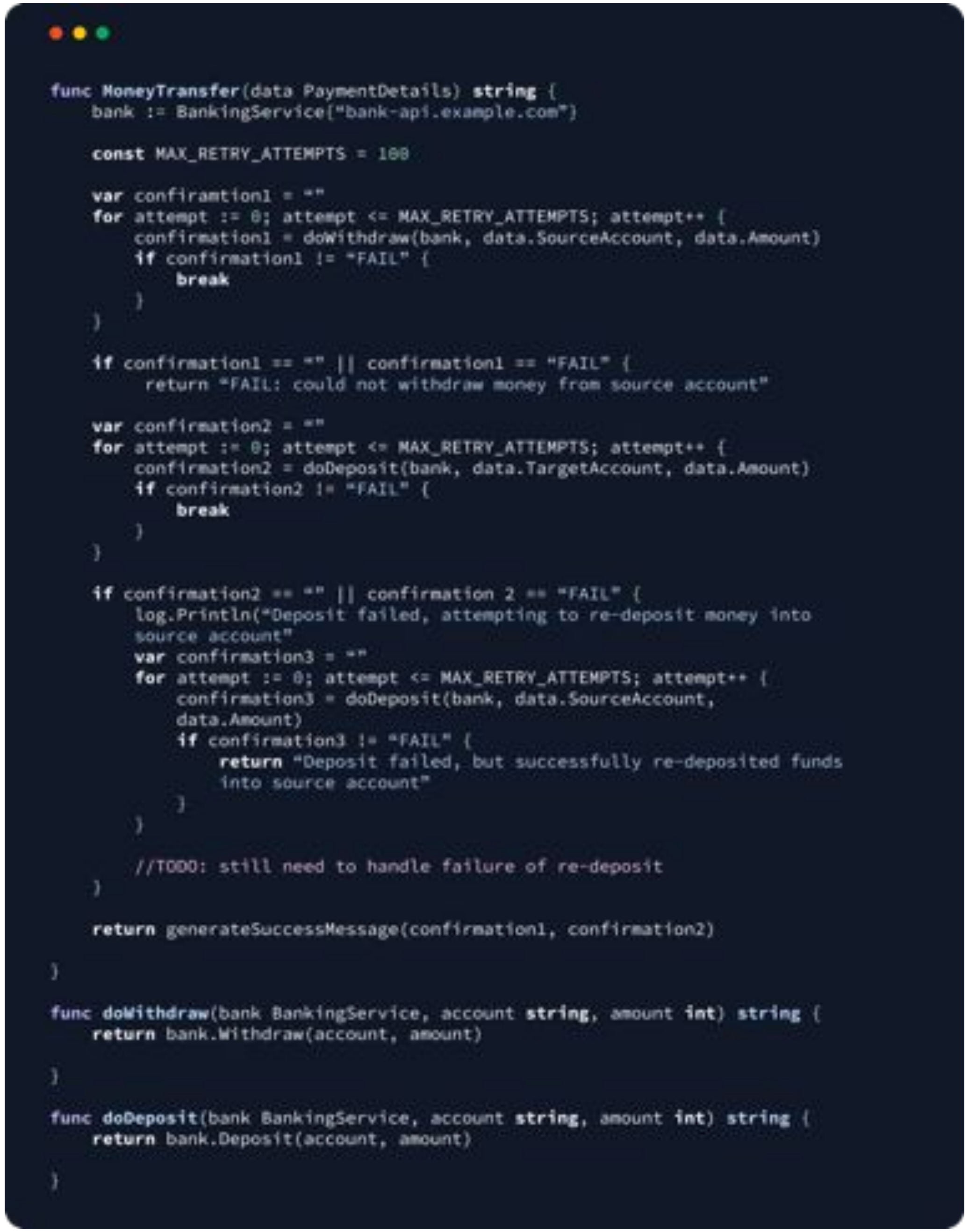
The same code, after adding support for retries during withdrawal



```
func MoneyTransfer(data PaymentDetails) string {
    bank := BankingService{"bank-api.example.com"}

    const MAX_RETRY_ATTEMPTS = 100

    var confiramtion1 = ""
    for attempt := 0; attempt <= MAX_RETRY_ATTEMPTS; attempt++ {
        confirmation1 = doWithdraw(bank, data.SourceAccount, data.Amount)
        if confirmation1 != "FAIL" {
             break
        }
    }

    if confirmation1 == "" || confirmation1 == "FAIL" {
         return "FAIL: could not withdraw money from source account"

    }

    confirmation2 := bank.Deposit(data.TargetAccount, data.Amount)

    return generateSuccessMessage(confirmation1, confirmation2)

}

func doWithdraw(bank BankingService, account string, amount int) string {
    return bank.Withdraw(account, amount)

}
```

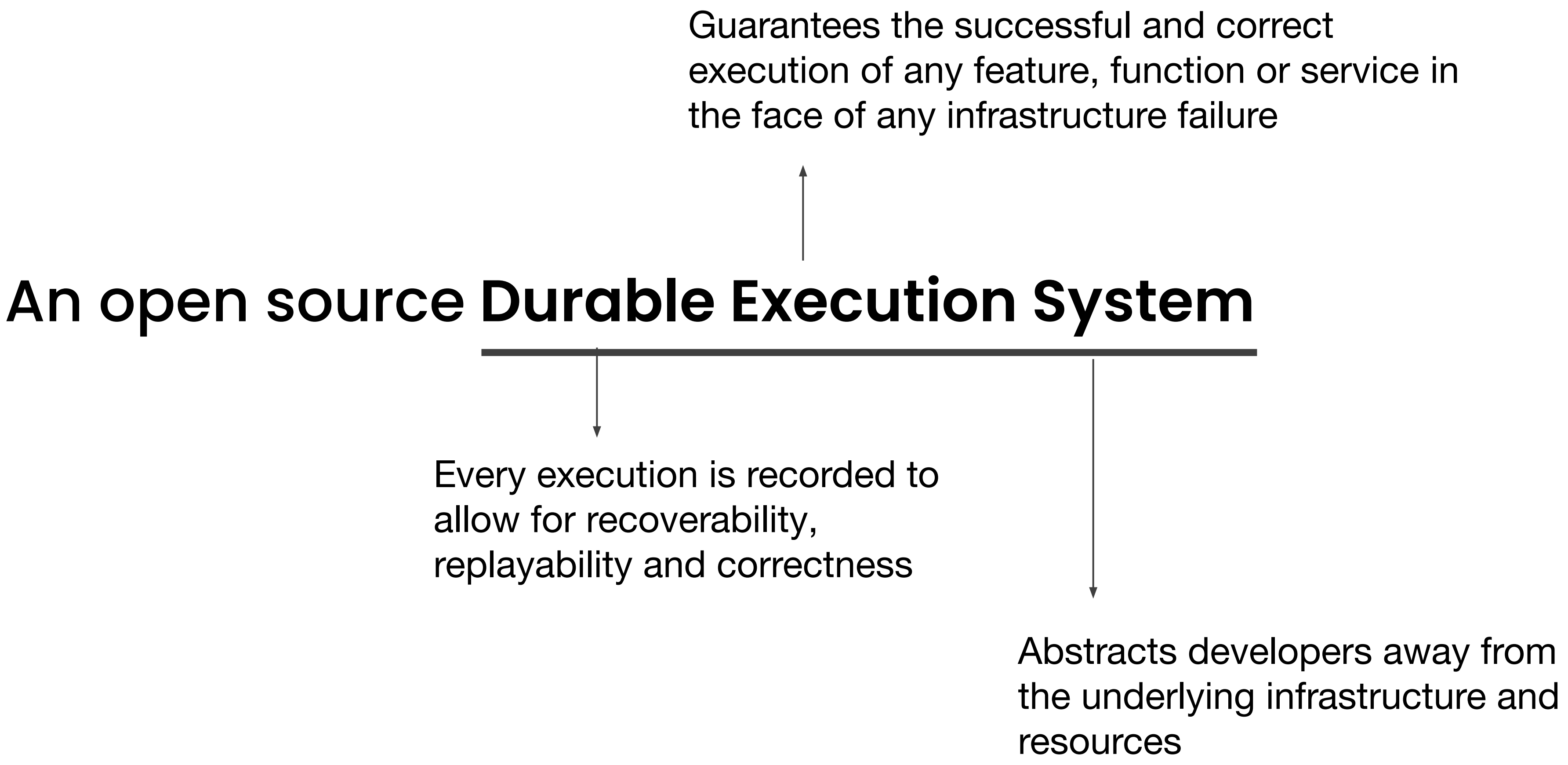
Engineers have paid
the price again and again and again

The same code, after adding support for retries during withdrawal and deposit, and performing a compensation if the withdrawal succeeds but the deposit fails



```
func MoneyTransfer(data PaymentDetails) string {
    bank := BankingService(“bank-api.example.com”)

    const MAX_RETRY_ATTEMPTS = 100

    var confiramtion1 = “”
    for attempt := 0; attempt <= MAX_RETRY_ATTEMPTS; attempt++ {
        confirmation1 = doWithdraw(bank, data.SourceAccount, data.Amount)
        if confirmation1 != “FAIL” {
            break
        }
    }

    if confirmation1 == “” || confirmation1 == “FAIL” {
         return “FAIL: could not withdraw money from source account”

    var confirmation2 = “”
    for attempt := 0; attempt <= MAX_RETRY_ATTEMPTS; attempt++ {
        confirmation2 = doDeposit(bank, data.TargetAccount, data.Amount)
        if confirmation2 != “FAIL” {
            break
        }
    }

    if confirmation2 == “” || confirmation 2 == “FAIL” {
        log.Println(“Deposit failed, attempting to re-deposit money into
        source account”
        var confirmation3 = “”
        for attempt := 0; attempt <= MAX_RETRY_ATTEMPTS; attempt++ {
            confirmation3 = doDeposit(bank, data.SourceAccount,
            data.Amount)
            if confirmation3 != “FAIL” {
                return “Deposit failed, but successfully re-deposited funds
                into source account”
            }
        }

        //TODO: still need to handle failure of re-deposit
    }

    return generateSuccessMessage(confirmation1, confirmation2)

}

func doWithdraw(bank BankingService, account string, amount int) string {
    return bank.Withdraw(account, amount)

}

func doDeposit(bank BankingService, account string, amount int) string {
    return bank.Deposit(account, amount)

}
```

Temporal was created to solve these challenges



Guarantees the successful and correct execution of any feature, function or service in the face of any infrastructure failure

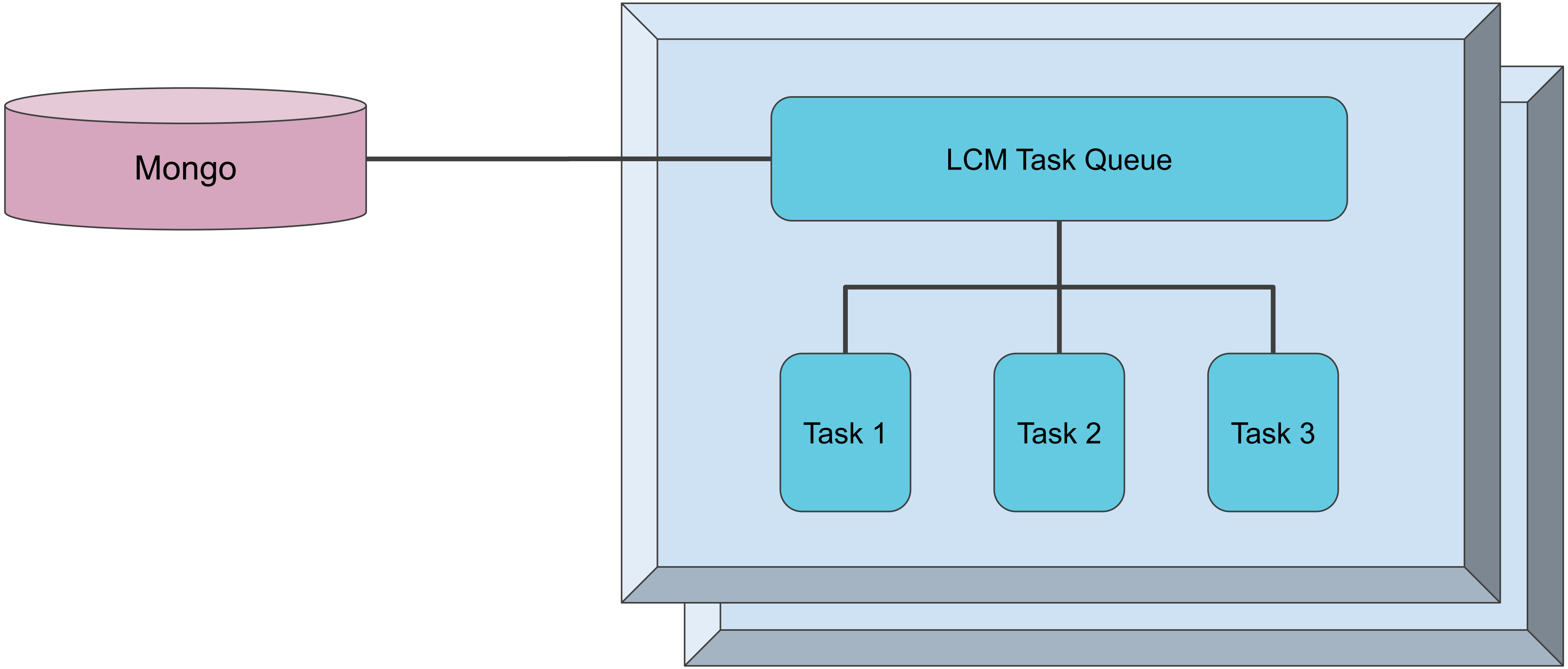
An open source **Durable Execution System**

Every execution is recorded to allow for recoverability, replayability and correctness

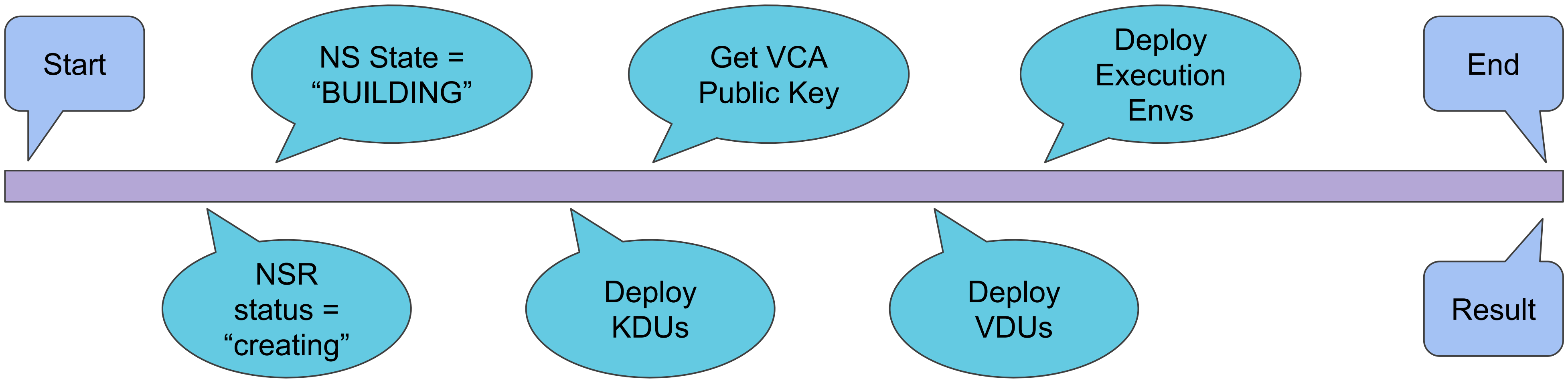
Abstracts developers away from the underlying infrastructure and resources

Managing Operations in OSM

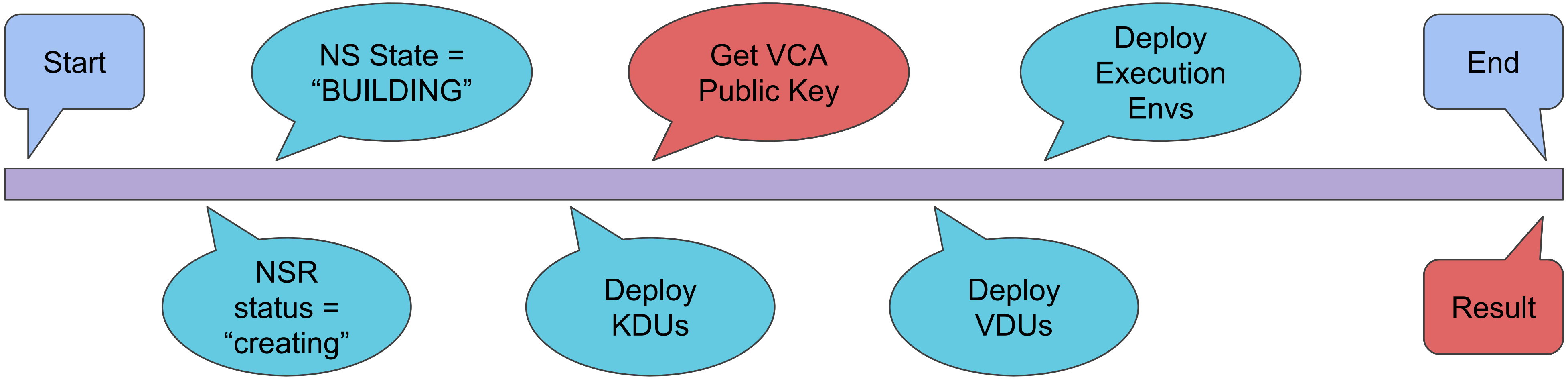
The current state


Mongo
LCM Task Queue
Task 1
Task 2
Task 3

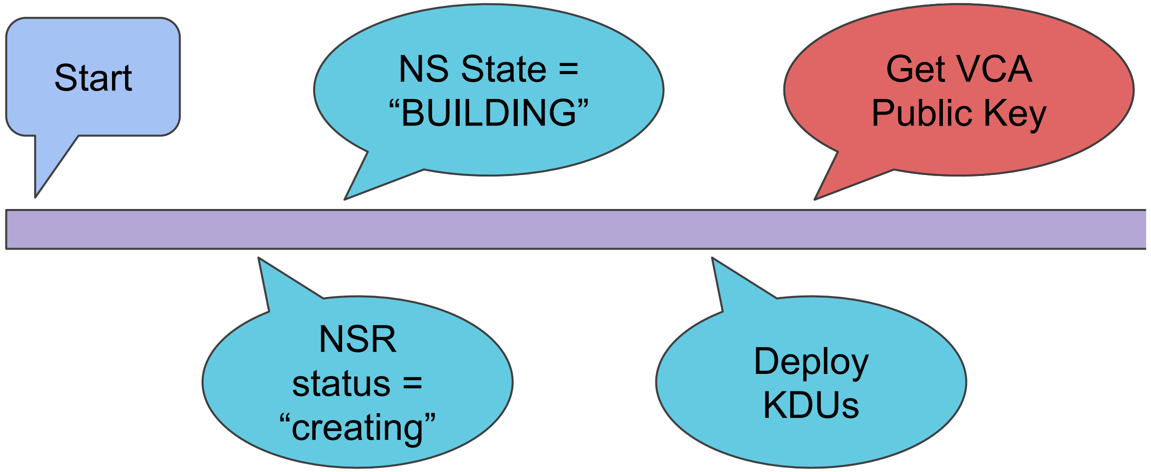
So What Is a Workflow



Expected Failures



Unexpected Failures



Exploration of a Concept

Are NS LCM Operations just Workflows?

Adopt incrementally

Learn

Intro to SDKs

Tutorials

- Hello World
- Money Transfer
- eCommerce

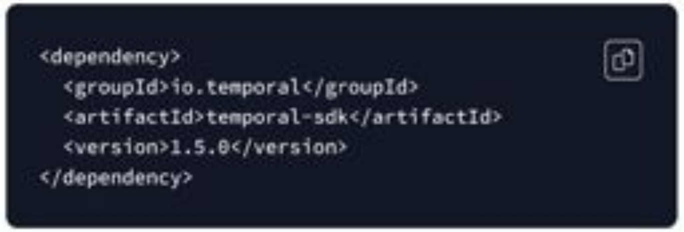
Samples

[temporalio/samples-java]

Refactor (or Write)

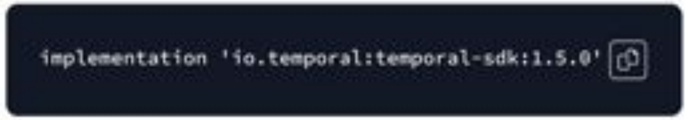
Client

Maven



```
<dependency>
  <groupId>io.temporal</groupId>
  <artifactId>temporal-sdk</artifactId>
  <version>1.5.0</version>
</dependency>
```

Gradle Groovy DSL



```
implementation 'io.temporal:temporal-sdk:1.5.0'
```

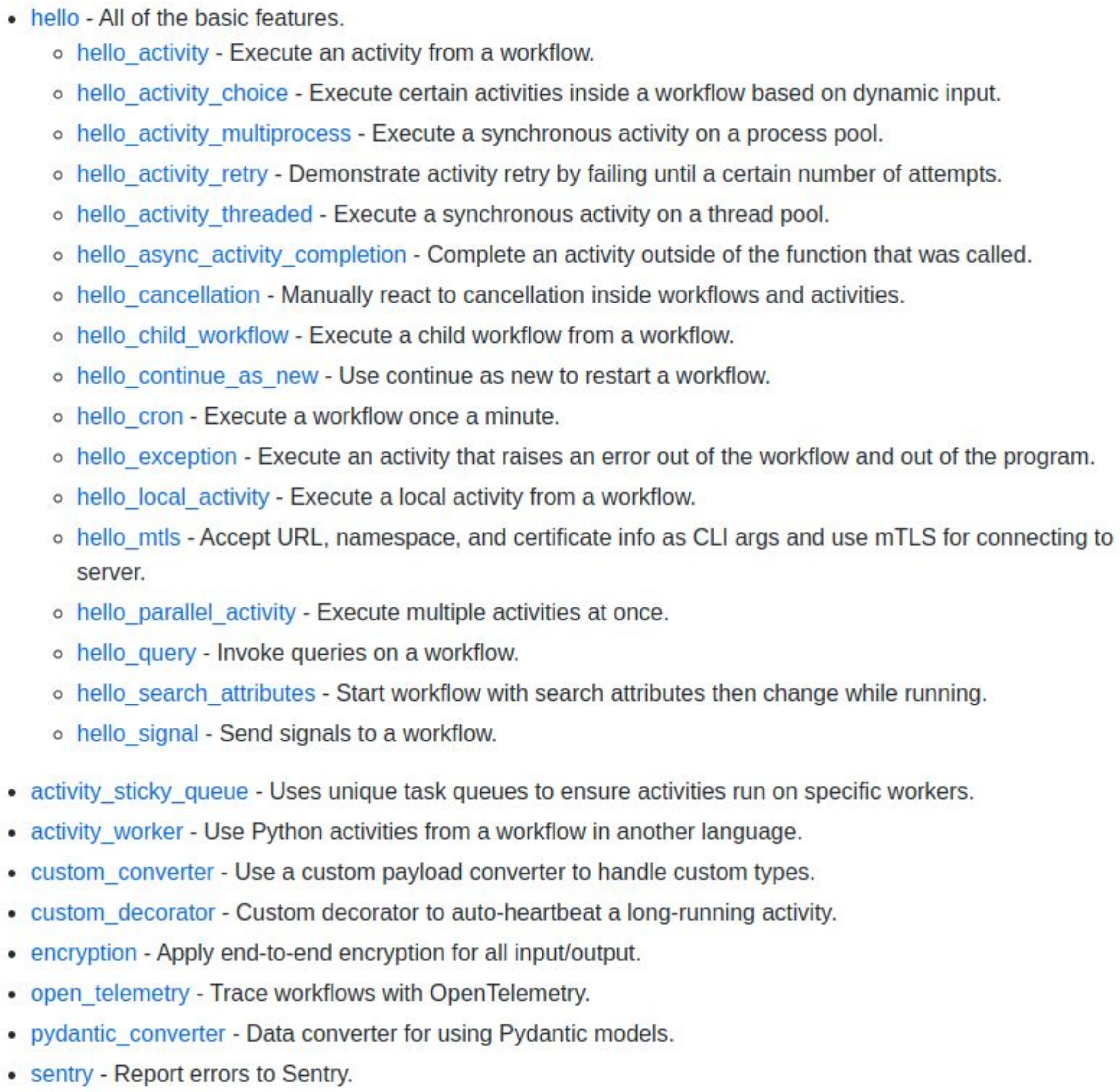
Server

docker compose Gitpod HELM DEPLOY TO RENDER

Learn

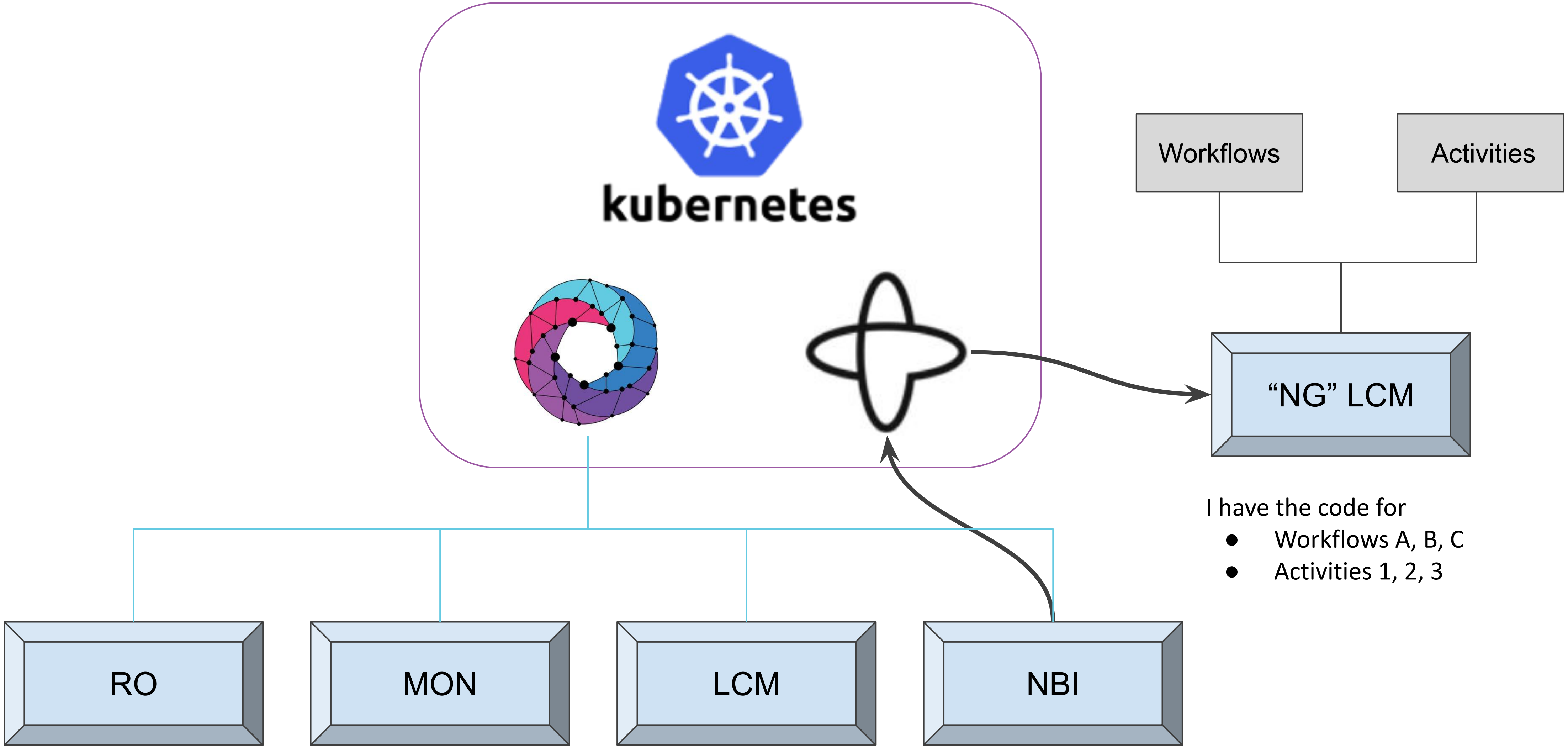
https://github.com/temporalio/samples-python

https://github.com/temporalio/sdk-python



- hello - All of the basic features.
  - hello_activity - Execute an activity from a workflow.
  - hello_activity_choice - Execute certain activities inside a workflow based on dynamic input.
  - hello_activity_multiprocess - Execute a synchronous activity on a process pool.
  - hello_activity_retry - Demonstrate activity retry by failing until a certain number of attempts.
  - hello_activity_threaded - Execute a synchronous activity on a thread pool.
  - hello_async_activity_completion - Complete an activity outside of the function that was called.
  - hello_cancellation - Manually react to cancellation inside workflows and activities.
  - hello_child_workflow - Execute a child workflow from a workflow.
  - hello_continue_as_new - Use continue as new to restart a workflow.
  - hello_cron - Execute a workflow once a minute.
  - hello_exception - Execute an activity that raises an error out of the workflow and out of the program.
  - hello_local_activity - Execute a local activity from a workflow.
  - hello_mtls - Accept URL, namespace, and certificate info as CLI args and use mTLS for connecting to server.
  - hello_parallel_activity - Execute multiple activities at once.
  - hello_query - Invoke queries on a workflow.
  - hello_search_attributes - Start workflow with search attributes then change while running.
  - hello_signal - Send signals to a workflow.
- activity_sticky_queue - Uses unique task queues to ensure activities run on specific workers.
- activity_worker - Use Python activities from a workflow in another language.
- custom_converter - Use a custom payload converter to handle custom types.
- custom_decorator - Custom decorator to auto-heartbeat a long-running activity.
- encryption - Apply end-to-end encryption for all input/output.
- open_telemetry - Trace workflows with OpenTelemetry.
- pydantic_converter - Data converter for using Pydantic models.
- sentry - Report errors to Sentry.

Write



I have the code for

- Workflows A, B, C
- Activities 1, 2, 3

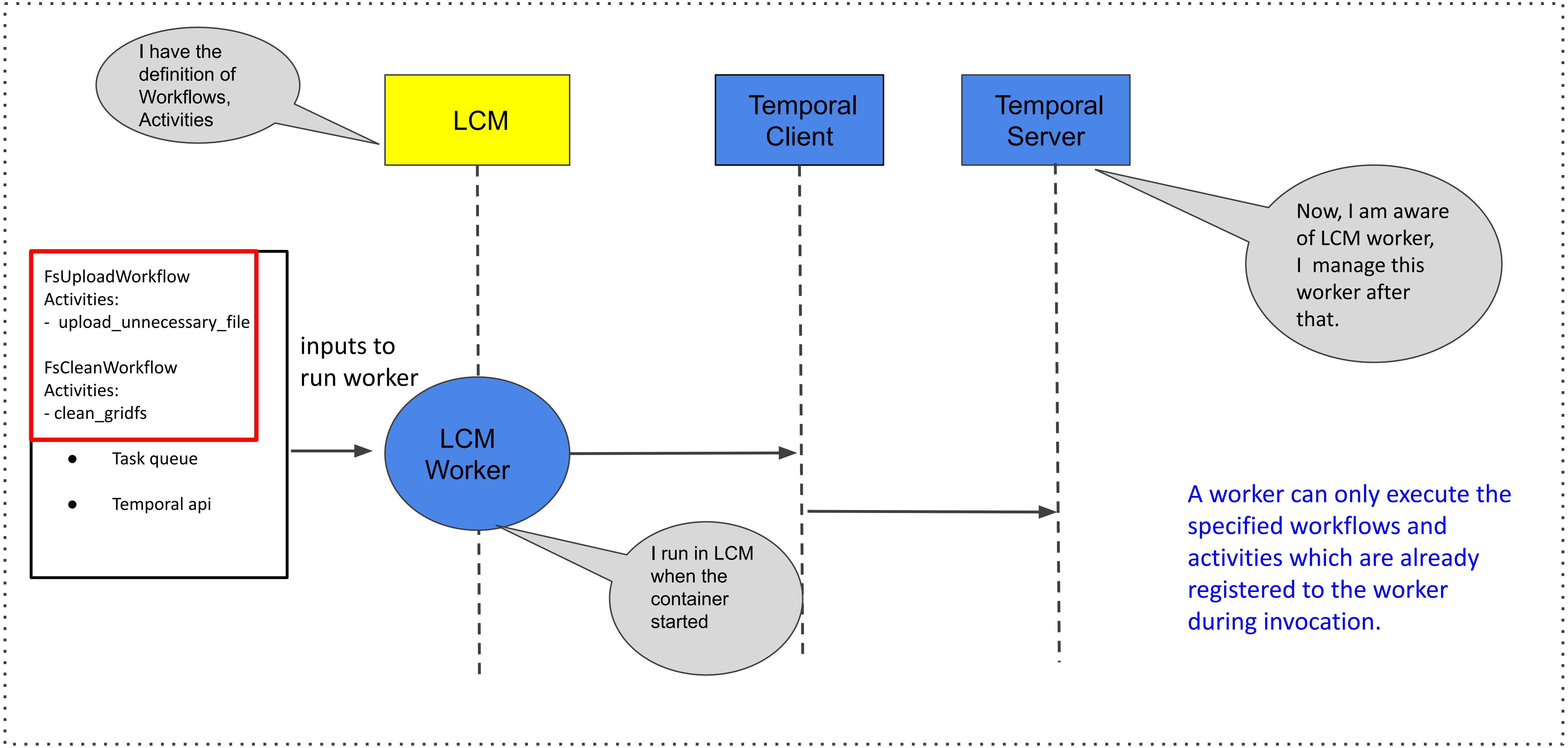
Demo

Periodic Gridfs Cleaning with Temporal Workflows

Why FsClean workflow is created?

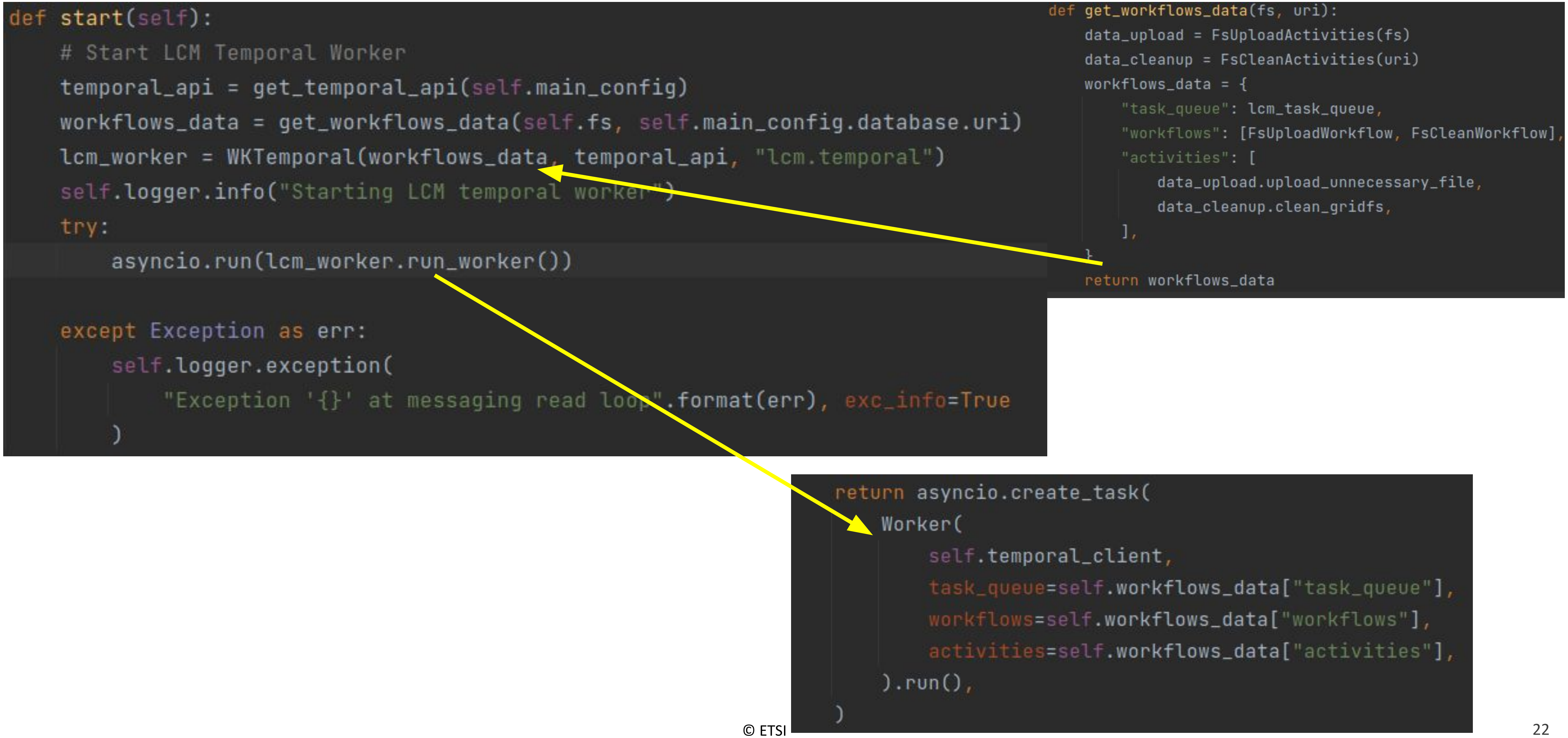
- Existing production issue (https://osm.etsi.org/bugzilla/show_bug.cgi?id=2024)
- NSD/VNFD upload can abandon files (charts, bundles)
- Performance of file synchronization operations can be impacted
- A workflow is created to delete the unused Gridfs files in OSM MongoDB
- The workflow could be scheduled to run periodically so it always keeps the OSM filesystem clean

A Worker invocation Flow


I have the definition of Workflows, Activities
LCM
Temporal Client
Temporal Server
Now, I am aware of LCM worker, I manage this worker after that.
FsUploadWorkflow
Activities:
- upload_unnecessary_file
FsCleanWorkflow
Activities:
- clean_gridfs
● Task queue
● Temporal api
inputs to run worker
LCM Worker
I run in LCM when the container started

A worker can only execute the specified workflows and activities which are already registered to the worker during invocation.

A Worker invocation Flow

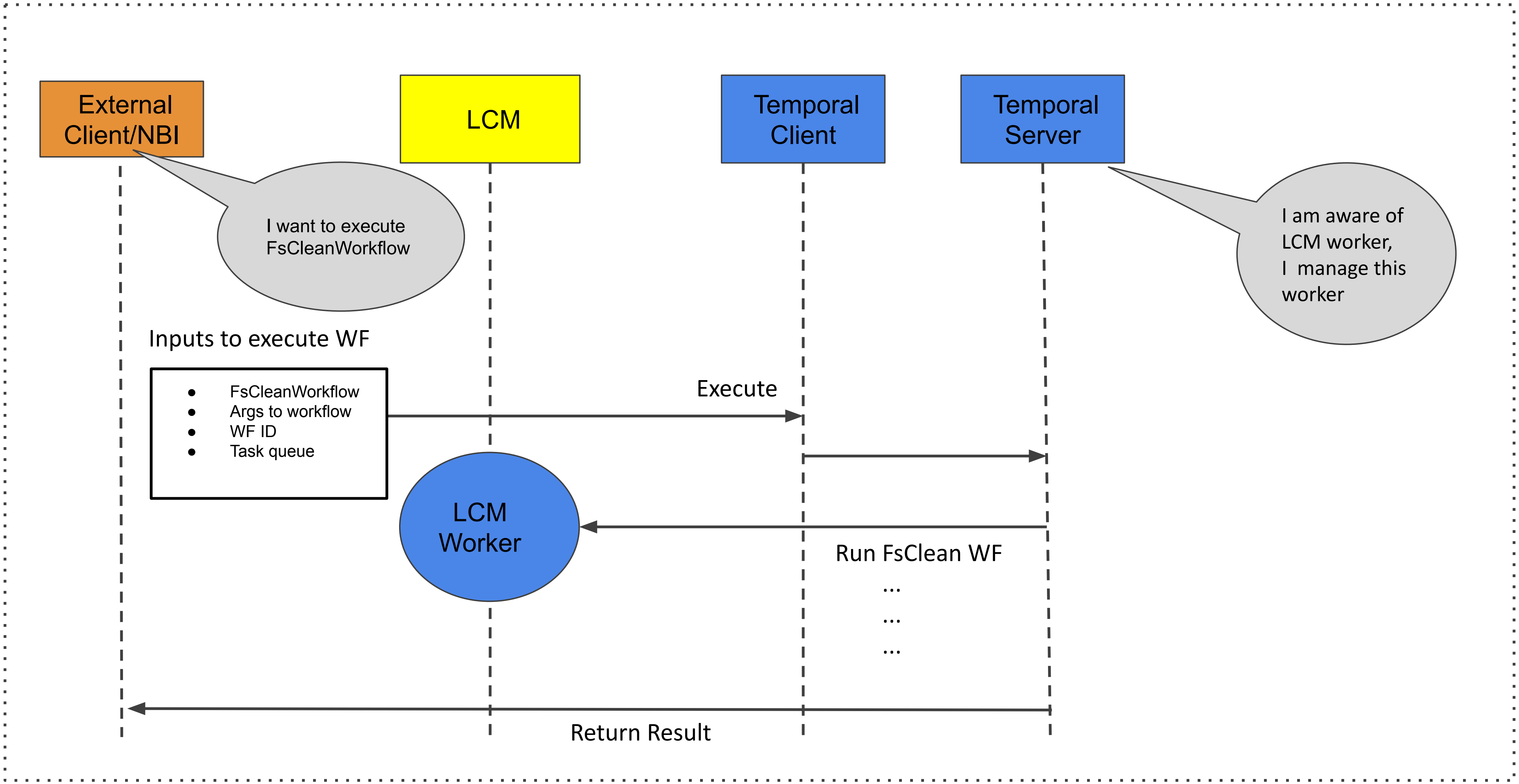


```
def start(self):
    # Start LCM Temporal Worker
    temporal_api = get_temporal_api(self.main_config)
    workflows_data = get_workflows_data(self.fs, self.main_config.database.uri)
    lcm_worker = WKTemporal(workflows_data, temporal_api, "lcm.temporal")
    self.logger.info("Starting LCM temporal worker")
    try:
        asyncio.run(lcm_worker.run_worker())

    except Exception as err:
        self.logger.exception(
            "Exception '{}' at messaging read loop".format(err), exc_info=True
        )
```

```
def get_workflows_data(fs, uri):
    data_upload = FsUploadActivities(fs)
    data_cleanup = FsCleanActivities(uri)
    workflows_data = {
        "task_queue": lcm_task_queue,
        "workflows": [FsUploadWorkflow, FsCleanWorkflow],
        "activities": [
            data_upload.upload_unnecessary_file,
            data_cleanup.clean_gridfs,
        ],
    }
    return workflows_data
```

```
return asyncio.create_task(
    Worker(
        self.temporal_client,
        task_queue=self.workflows_data["task_queue"],
        workflows=self.workflows_data["workflows"],
        activities=self.workflows_data["activities"],
    ).run(),
)
```

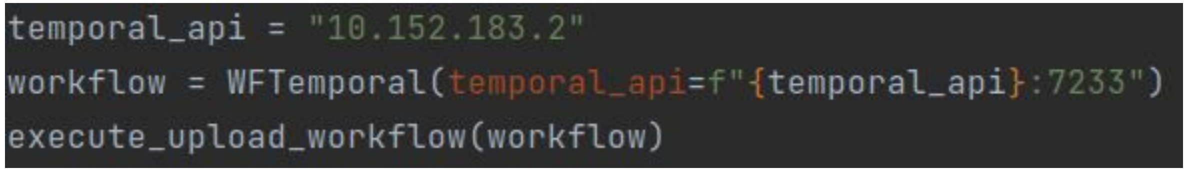
Executing a Workflow using the Temporal client



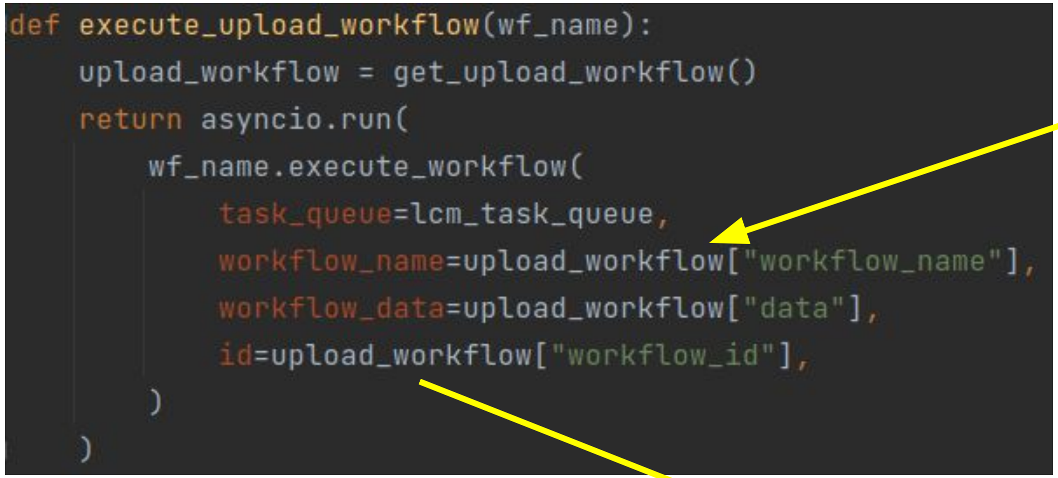
Inputs to execute WF

- FsCleanWorkflow
- Args to workflow
- WF ID
- Task queue

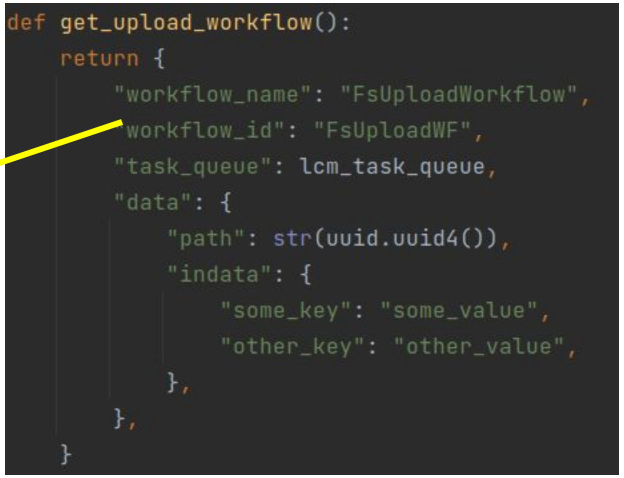
Executing a Workflow using the Temporal client



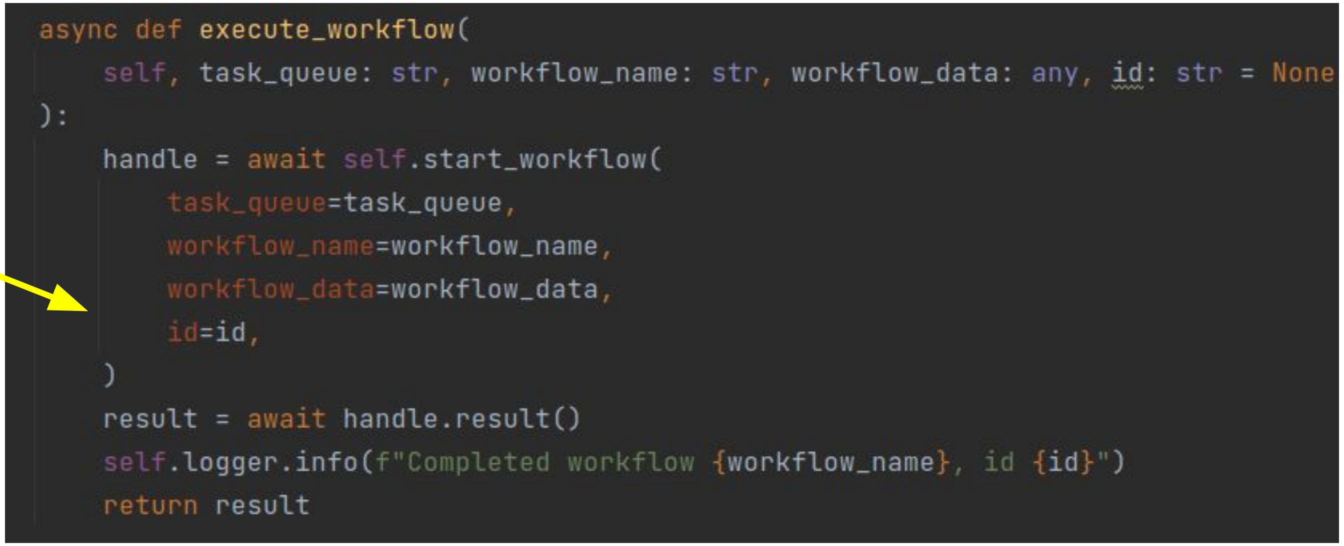
```
temporal_api = "10.152.183.2"
workflow = WFTemporal(temporal_api=f"{temporal_api}:7233")
execute_upload_workflow(workflow)
```



```
def execute_upload_workflow(wf_name):
    upload_workflow = get_upload_workflow()
    return asyncio.run(
        wf_name.execute_workflow(
            task_queue=lcm_task_queue,
            workflow_name=upload_workflow["workflow_name"],
            workflow_data=upload_workflow["data"],
            id=upload_workflow["workflow_id"],
        )
    )
```

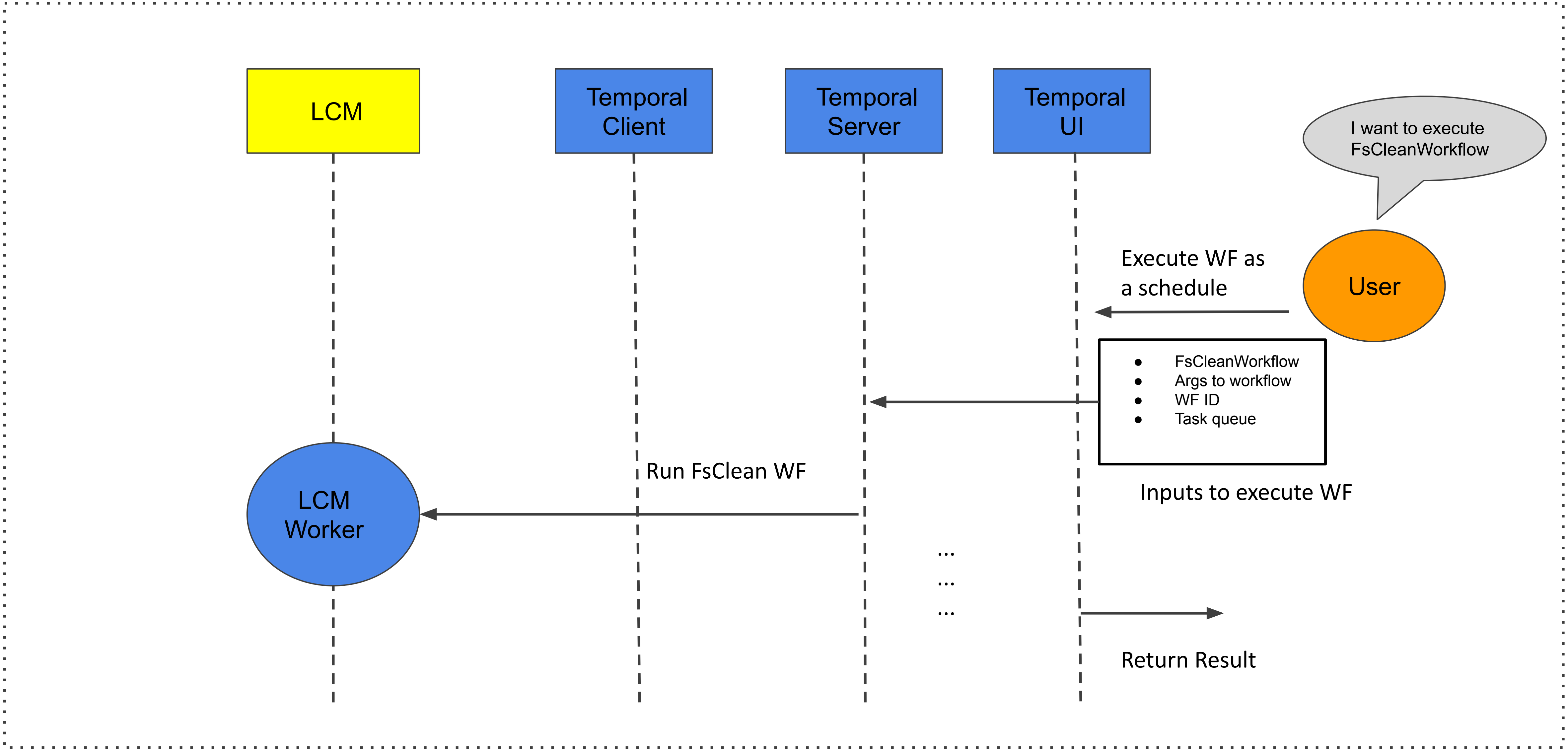


```
def get_upload_workflow():
    return {
        "workflow_name": "FsUploadWorkflow",
        "workflow_id": "FsUploadWF",
        "task_queue": lcm_task_queue,
        "data": {
            "path": str(uuid.uuid4()),
            "indata": {
                "some_key": "some_value",
                "other_key": "other_value",
            },
        },
    }
```

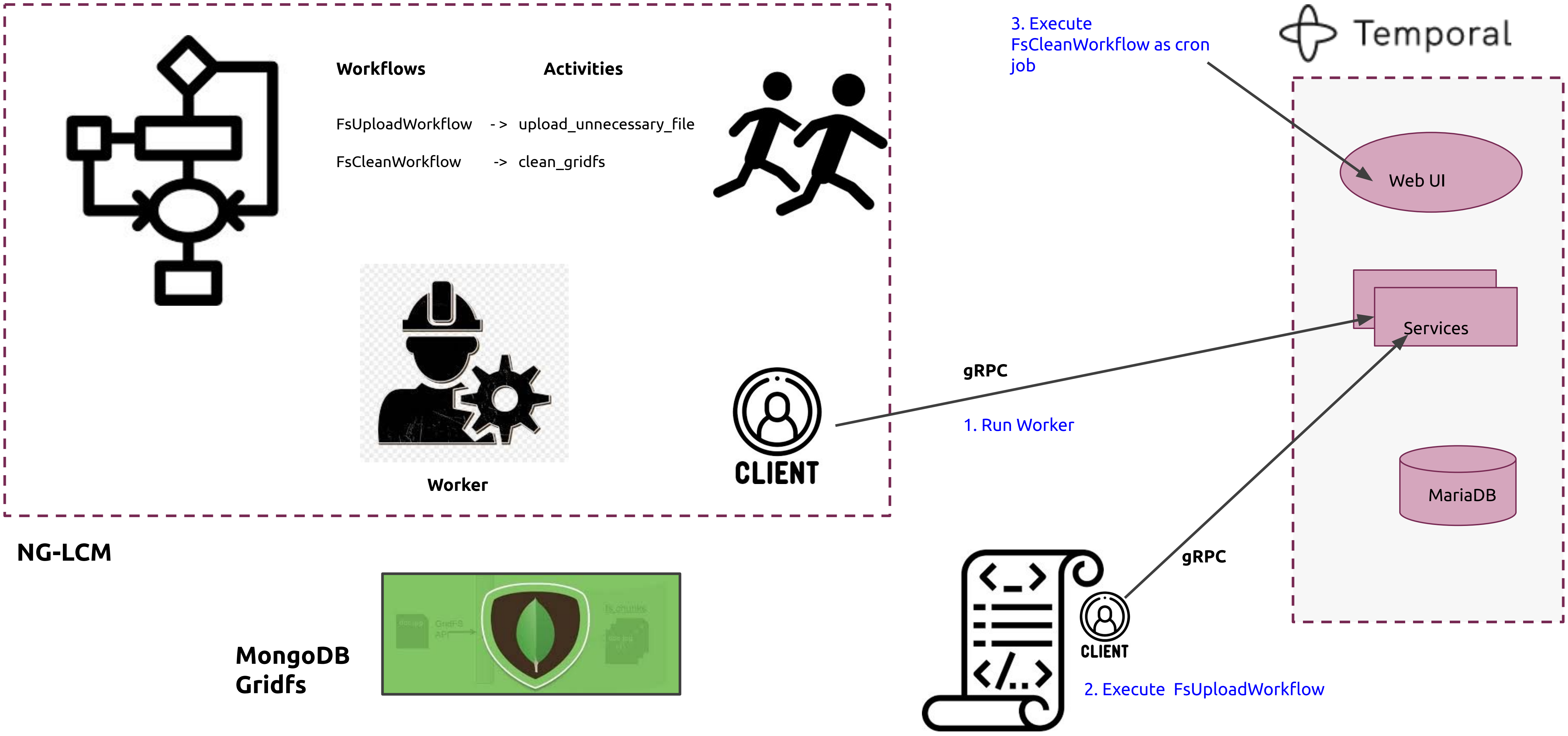


```
async def execute_workflow(
    self, task_queue: str, workflow_name: str, workflow_data: any, id: str = None
):
    handle = await self.start_workflow(
        task_queue=task_queue,
        workflow_name=workflow_name,
        workflow_data=workflow_data,
        id=id,
    )
    result = await handle.result()
    self.logger.info(f"Completed workflow {workflow_name}, id {id}")
    return result
```

Executing a Scheduled Workflow by Temporal UI



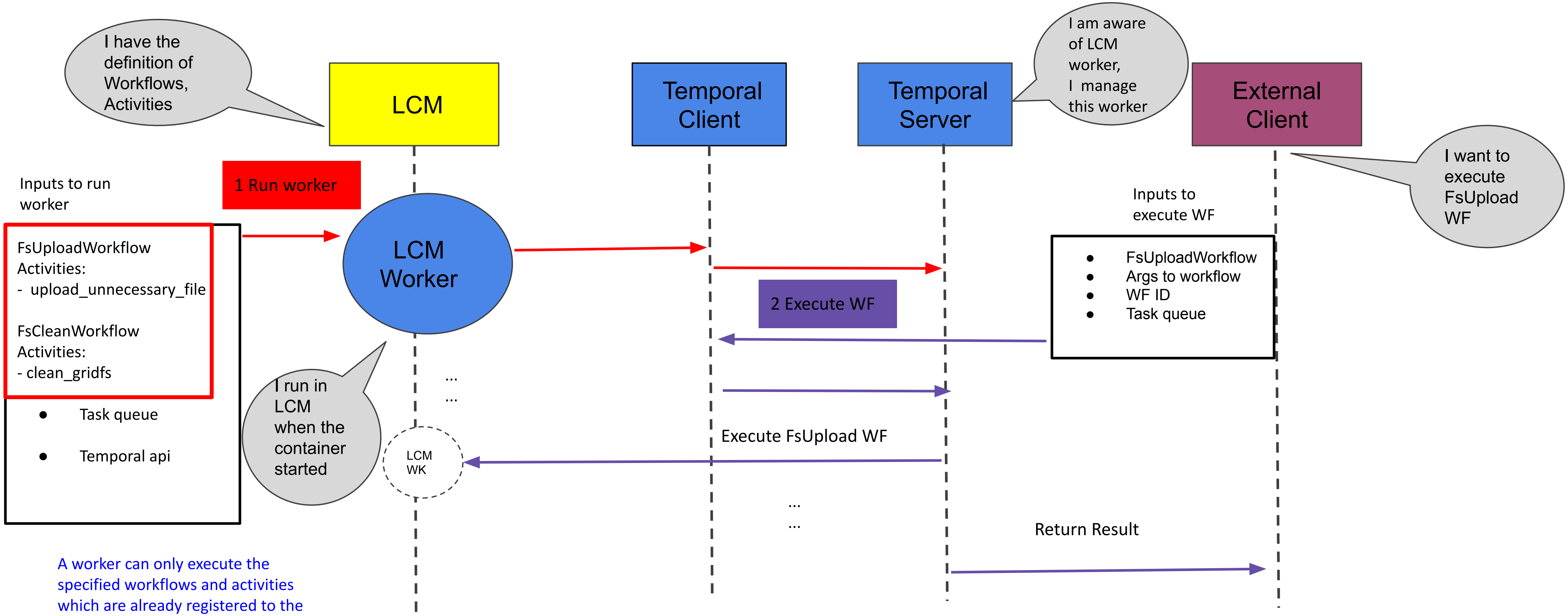
Abstract overview of the workflow for the demo



Workflows		Activities
FsUploadWorkflow	- >	upload_unnecessary_file
FsCleanWorkflow	->	clean_gridfs

Demo Flow

Open Source MANO



I have the definition of Workflows, Activities

LCM

Temporal Client

Temporal Server

I am aware of LCM worker, I manage this worker

External Client

I want to execute FsUpload WF

Inputs to run worker

1 Run worker

FsUploadWorkflow
Activities:
- upload_unnecessary_file

FsCleanWorkflow
Activities:
- clean_gridfs

- Task queue
- Temporal api

LCM Worker

I run in LCM when the container started

LCM WK

...
...

Inputs to execute WF

- FsUploadWorkflow
- Args to workflow
- WF ID
- Task queue

2 Execute WF

Execute FsUpload WF

...
...

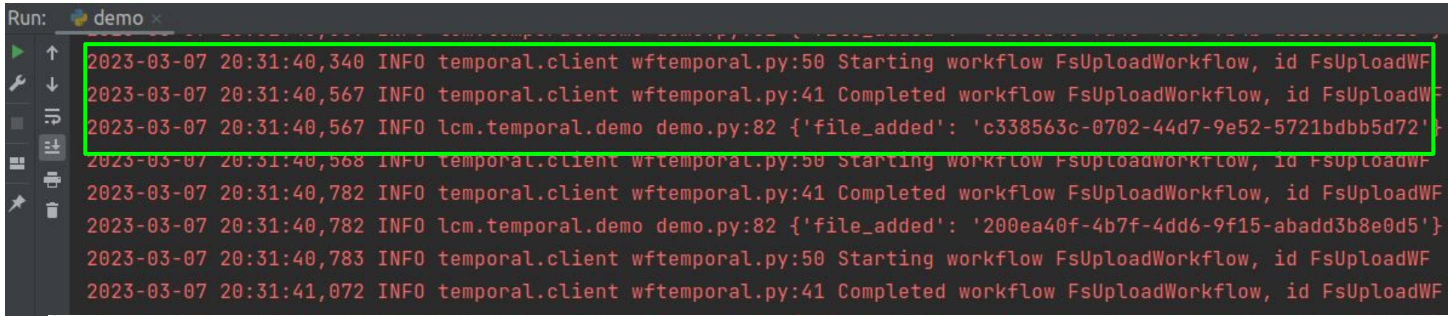
Return Result

A worker can only execute the specified workflows and activities which are already registered to the worker during invocation.

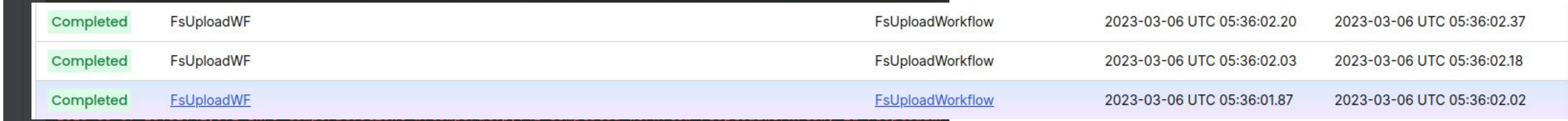
© ETSI

Execute FsUploadWorkflow

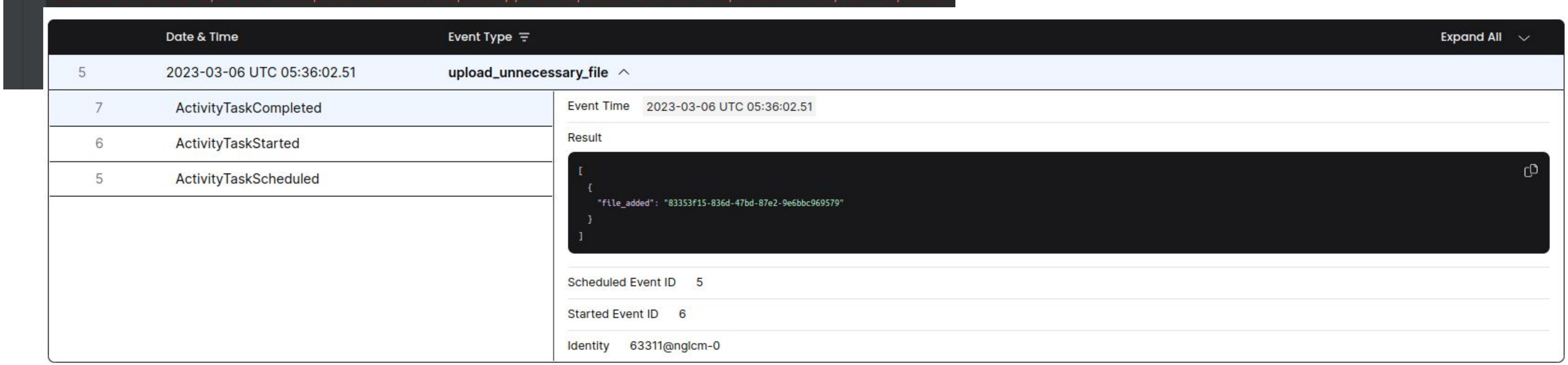
Execute FsUploadWorkflow in order to **upload unnecessary** files to Gridfs



```
2023-03-07 20:31:40,340 INFO temporal.client wftemporal.py:50 Starting workflow FsUploadWorkflow, id FsUploadWF
2023-03-07 20:31:40,567 INFO temporal.client wftemporal.py:41 Completed workflow FsUploadWorkflow, id FsUploadWF
2023-03-07 20:31:40,567 INFO lcm.temporal.demo demo.py:82 {'file_added': 'c338563c-0702-44d7-9e52-5721bdbb5d72'}
2023-03-07 20:31:40,568 INFO temporal.client wftemporal.py:50 Starting workflow FsUploadWorkflow, id FsUploadWF
2023-03-07 20:31:40,782 INFO temporal.client wftemporal.py:41 Completed workflow FsUploadWorkflow, id FsUploadWF
2023-03-07 20:31:40,782 INFO lcm.temporal.demo demo.py:82 {'file_added': '200ea40f-4b7f-4dd6-9f15-abadd3b8e0d5'}
2023-03-07 20:31:40,783 INFO temporal.client wftemporal.py:50 Starting workflow FsUploadWorkflow, id FsUploadWF
2023-03-07 20:31:41,072 INFO temporal.client wftemporal.py:41 Completed workflow FsUploadWorkflow, id FsUploadWF
```



Completed	FsUploadWF	FsUploadWorkflow	2023-03-06 UTC 05:36:02.20	2023-03-06 UTC 05:36:02.37
Completed	FsUploadWF	FsUploadWorkflow	2023-03-06 UTC 05:36:02.03	2023-03-06 UTC 05:36:02.18
Completed	FsUploadWF	FsUploadWorkflow	2023-03-06 UTC 05:36:01.87	2023-03-06 UTC 05:36:02.02



	Date & Time	Event Type
5	2023-03-06 UTC 05:36:02.51	upload_unnecessary_file

7	ActivityTaskCompleted
6	ActivityTaskStarted
5	ActivityTaskScheduled

Event Time 2023-03-06 UTC 05:36:02.51

Result

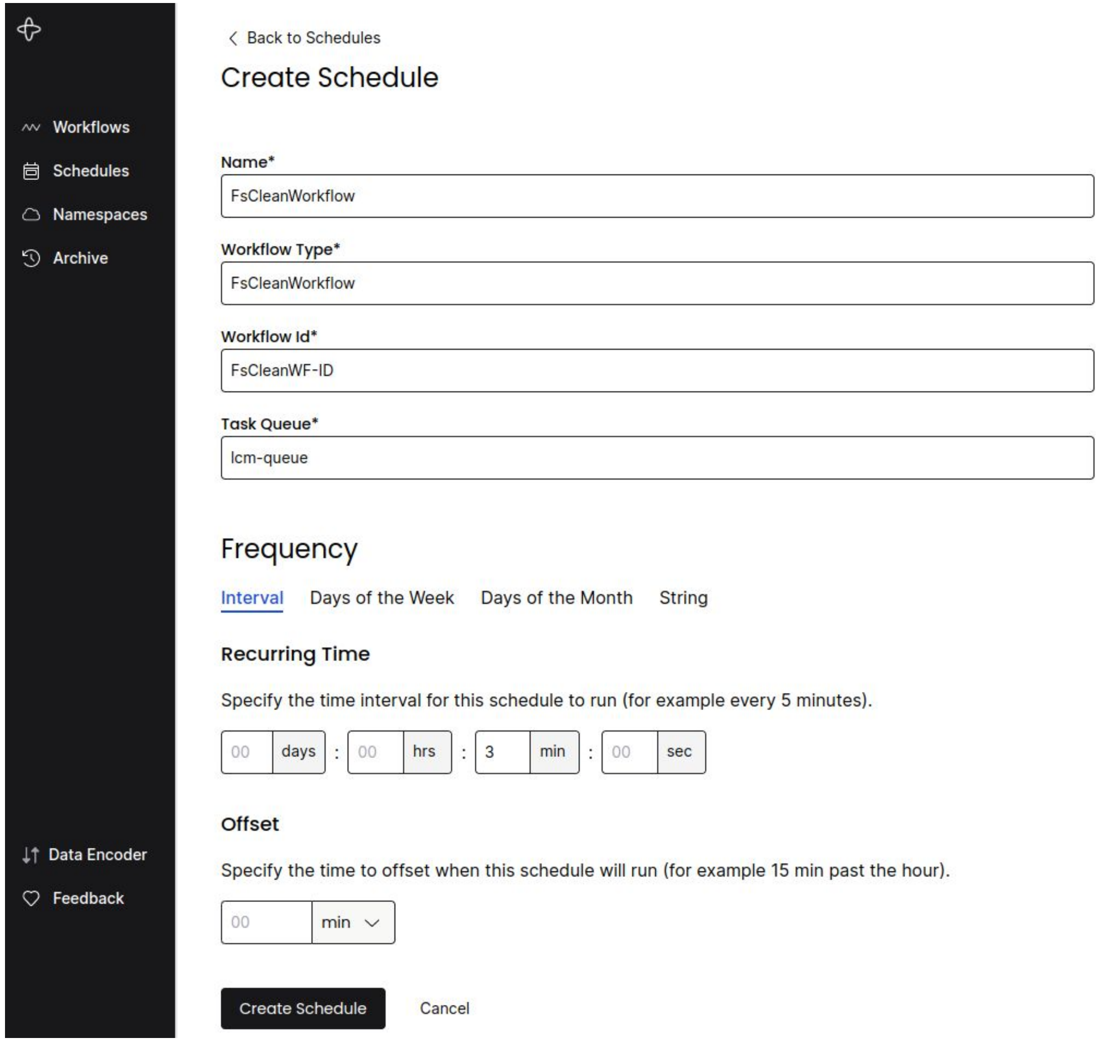
```
[
  {
    "file_added": "83353f15-836d-47bd-87e2-9e6bbc969579"
  }
]
```

Scheduled Event ID 5

Started Event ID 6

Identity 63311@nglcm-0

Create Schedule for FsCleanWorkflow



Create Schedule

Name*: FsCleanWorkflow

Workflow Type*: FsCleanWorkflow

Workflow Id*: FsCleanWF-ID

Task Queue*: lcm-queue

Frequency

Recurring Time

Specify the time interval for this schedule to run (for example every 5 minutes).

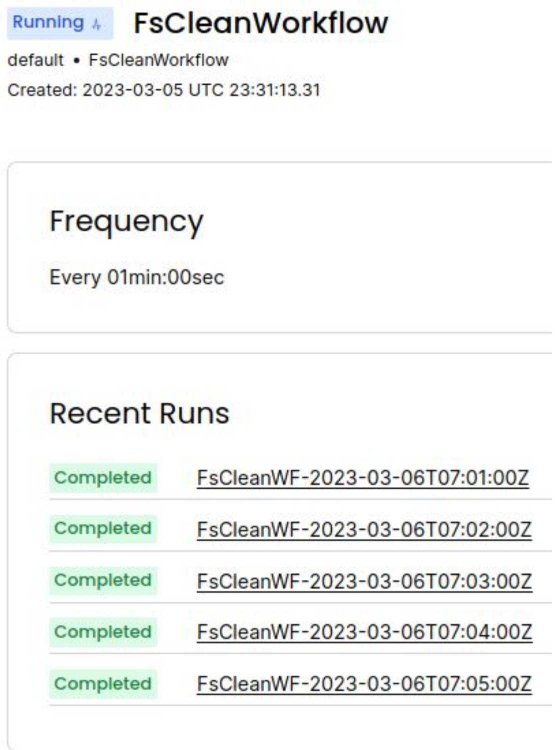
00 days : 00 hrs : 3 min : 00 sec

Offset

Specify the time to offset when this schedule will run (for example 15 min past the hour).

00 min

Scheduled FsCleanWorkflow cleans



Running FsCleanWorkflow

default • FsCleanWorkflow

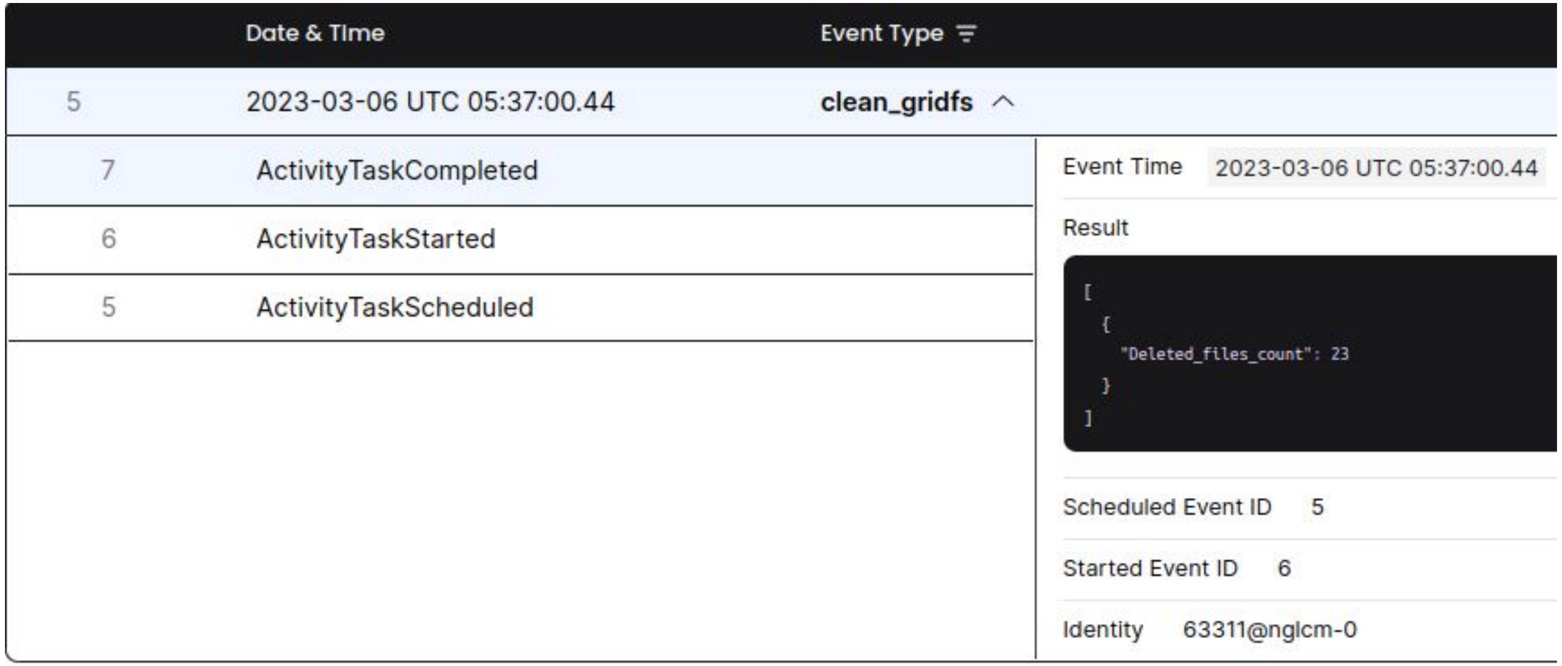
Created: 2023-03-05 UTC 23:31:13.31

Frequency

Every 01min:00sec

Recent Runs

Completed	FsCleanWF-2023-03-06T07:01:00Z
Completed	FsCleanWF-2023-03-06T07:02:00Z
Completed	FsCleanWF-2023-03-06T07:03:00Z
Completed	FsCleanWF-2023-03-06T07:04:00Z
Completed	FsCleanWF-2023-03-06T07:05:00Z



	Date & Time	Event Type
5	2023-03-06 UTC 05:37:00.44	clean_gridfs
7	ActivityTaskCompleted	
6	ActivityTaskStarted	
5	ActivityTaskScheduled	

Event Time 2023-03-06 UTC 05:37:00.44

Result

```
[
  {
    "Deleted_files_count": 23
  }
]
```

Scheduled Event ID 5

Started Event ID 6

Identity 63311@nglcm-0

Open Source MANO by ETSI

Thank You!