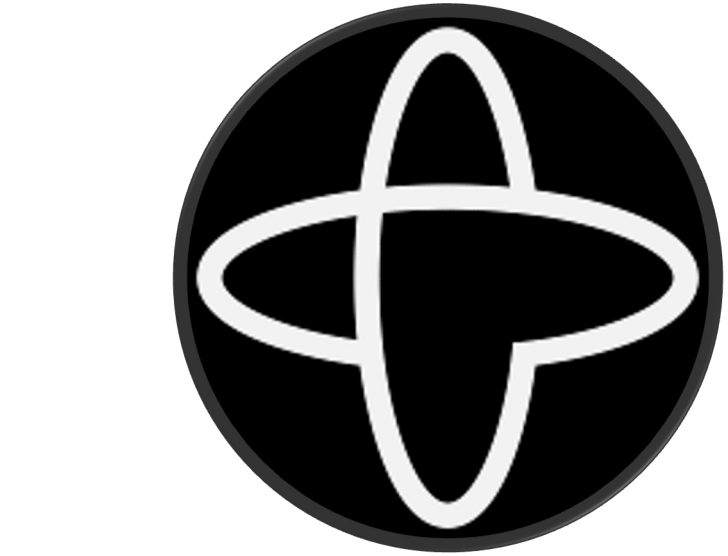


Orchestrate your business with Temporal.io workflows

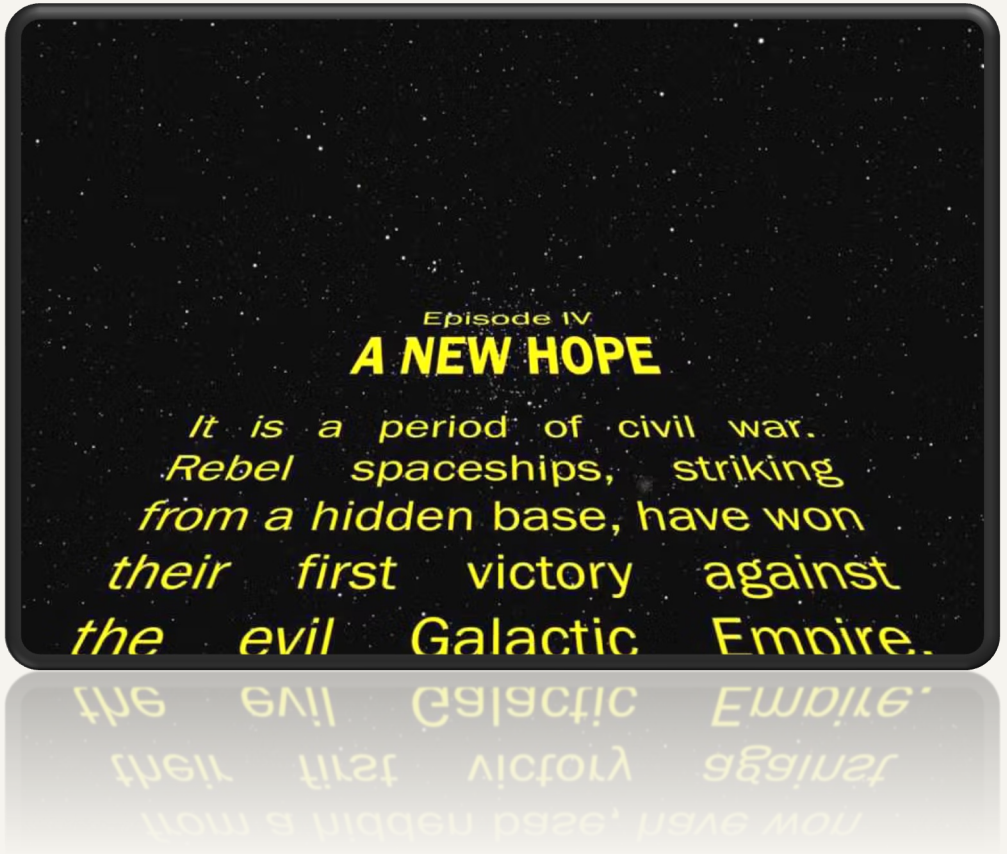
Gwendal Leclerc

OVHcloud

Breizh C@mp

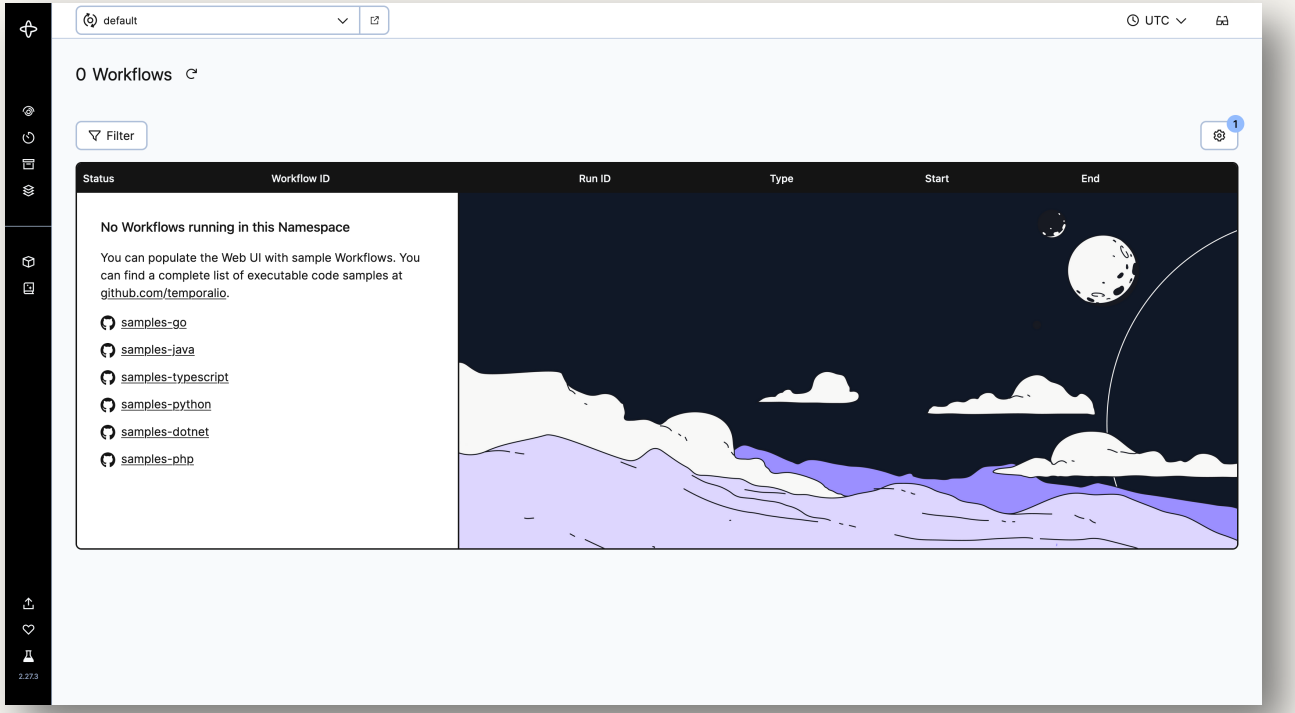
La conférence à l'Ouest

Description



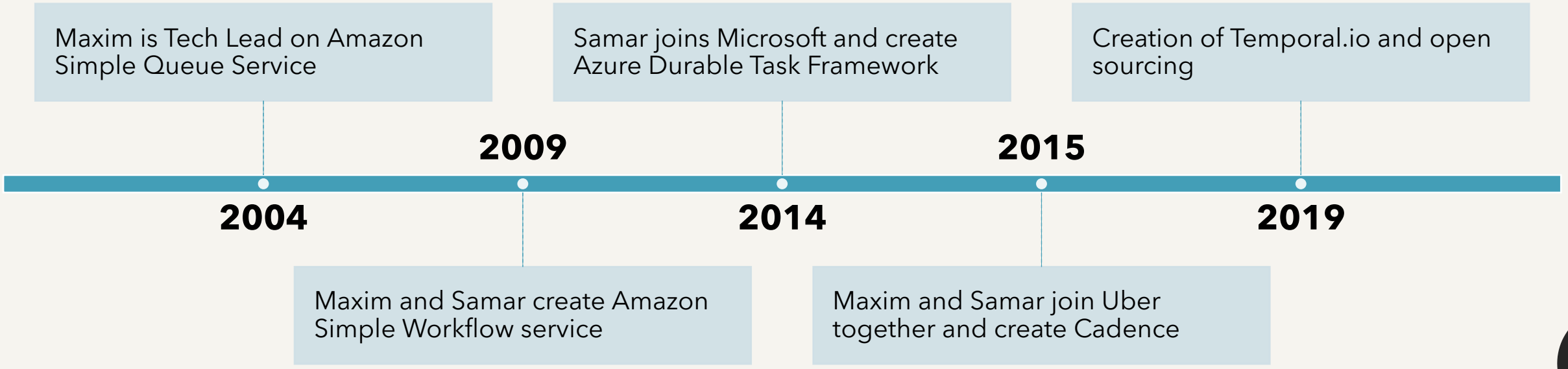
What is it?

- Asynchronous task execution platform
  - *Resilient*
  - *Open source*
  - *Available as SaaS or self-hosted*
  - *Based on event sourcing*
- SDK
  - *Rust, Go, Java, PHP, Python, Typescript, .NET*
- CLI
  - *Client*
  - *Dev Server*
- UI



Where does it come from?

- Created by Maxim Fateev and Samar Abbas in 2019



Overview

Concepts

Activity

- Encapsulates business logic that may fail
- Can be automatically replayed in case of error
- Can be indeterministic

Workflow

- Specifies a sequence of steps and orchestrates the execution of activities
- Must be deterministic

Worker

- Listen on one queue
- Execute the code of workflows and activities

Example: Activity

```go
func GetDomainStatus(ctx context.Context, input DomainStatusInput) (DomainStatusOutput, error) {
    var res DomainStatusOutput
    req, err := http.NewRequestWithContext(ctx, "GET", fmt.Sprintf("http://localhost:8091/domains/%s/status", input.Domain), nil)
    if err != nil {
        return res, err
    }
    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        return res, err
    }
    defer resp.Body.Close()
    if err = errorHook(resp); err != nil {
        return res, err
    }
    err = json.NewDecoder(resp.Body).Decode(&res)
    if err != nil {
        return res, err
    }
    return res, nil
}
```

Example: Workflow

```go
func WorkflowDefinition(ctx workflow.Context, workflowInput RenewInput) error {
    err := validator.New().Struct(&workflowInput)
    if err != nil {
        return err
    }

    var domainStatus DomainStatusOutput
    err = workflow.ExecuteActivity(ctx, GetDomainStatus, DomainStatusInput{Domain: workflowInput.Domain}).Get(ctx, &domainStatus)
    if err != nil {
        return err
    }

    var premiumStatus CheckPremiumStatusOutput
    err = workflow.ExecuteActivity(ctx, CheckPremiumStatus, CheckPremiumStatusInput{Domain: workflowInput.Domain}).Get(ctx, &premiumStatus)
    if err != nil {
        return err
    }

    ...

    return nil
}
```

Example: Worker

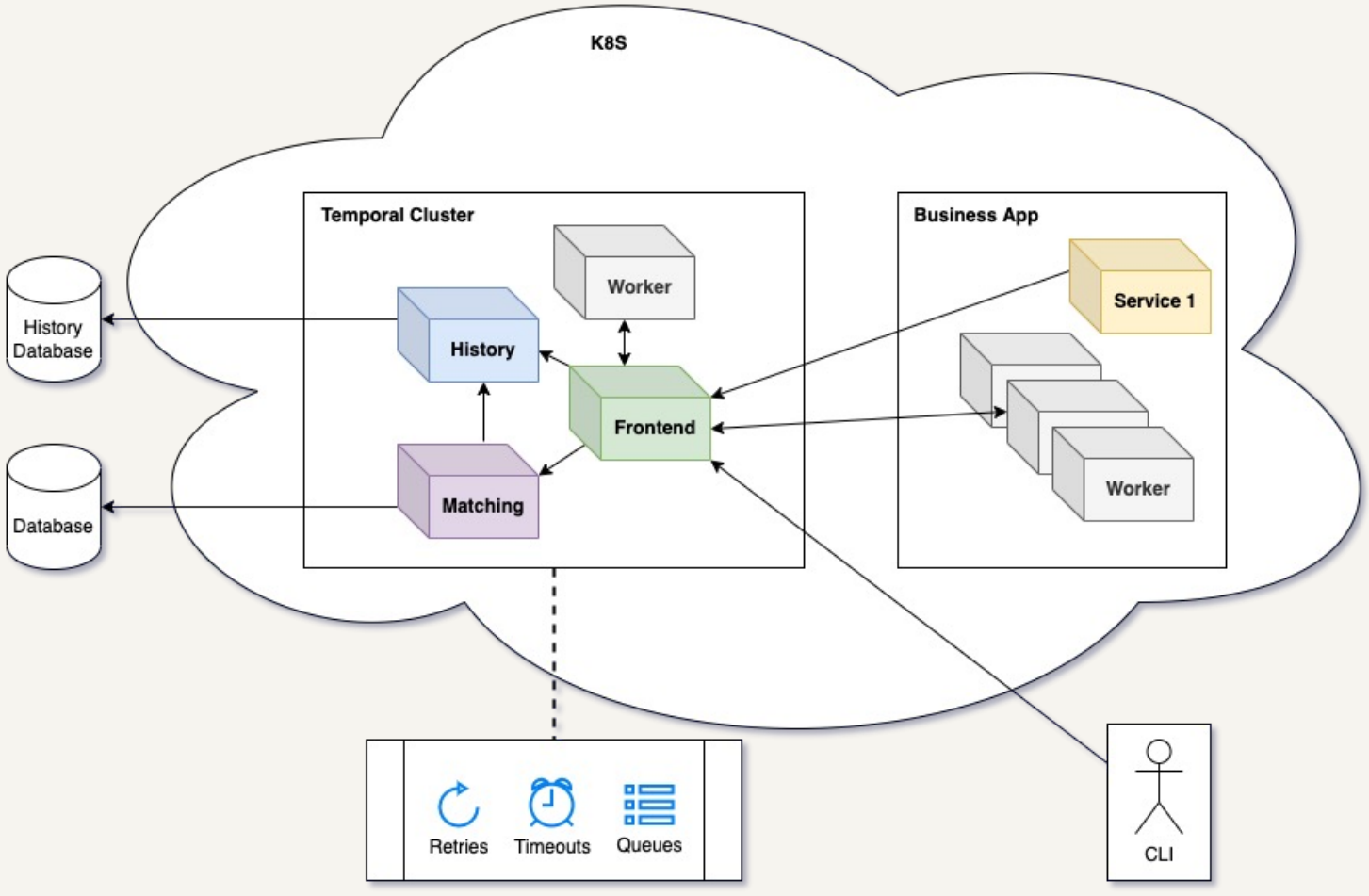
```go
func main() {
    c, err := client.Dial(client.Options{
        HostPort: client.DefaultHostPort,
    })
    if err != nil {
        log.Fatalln("Unable to create Temporal client", err)
    }
    defer c.Close()

    w := worker.New(c, "renew-queue", worker.Options{
        // WorkerActivitiesPerSecond:          config.WorkerActivitiesPerSecond,
        // MaxConcurrentActivityExecutionSize: config.MaxConcurrentActivityExecutionSize,
        // Interceptors: []internal.WorkerInterceptor{...},
    })

    w.RegisterWorkflow(workflows.WorkflowDefinition)
    w.RegisterActivity(workflows.GetDomainStatus)
    w.RegisterActivity(workflows.CheckPremiumStatus)
    ...

    err = w.Run(worker.InterruptCh())
    if err != nil {
        log.Fatalln("Unable to start Worker", err)
    }
}
```

Architecture

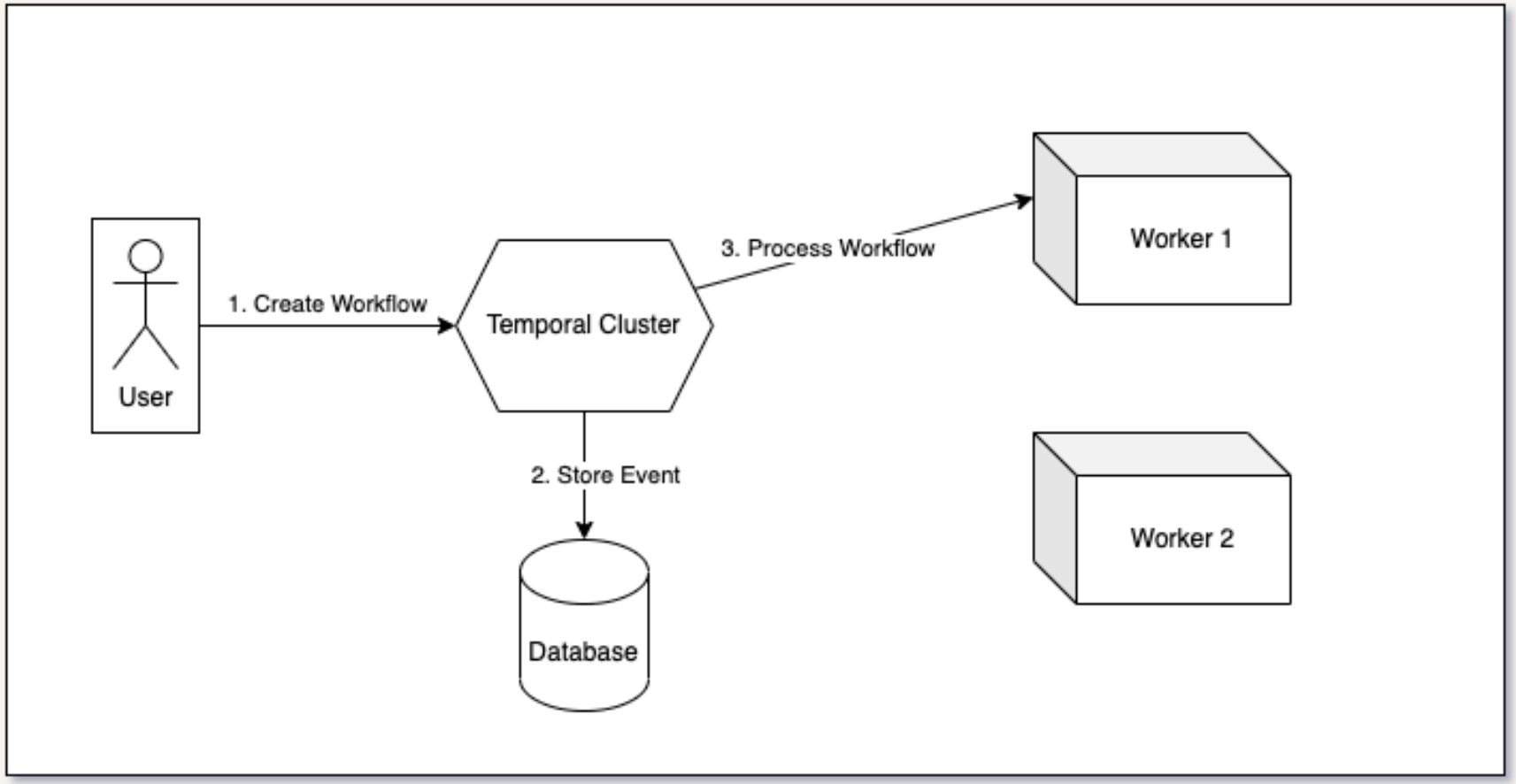


Deterministic workflows?

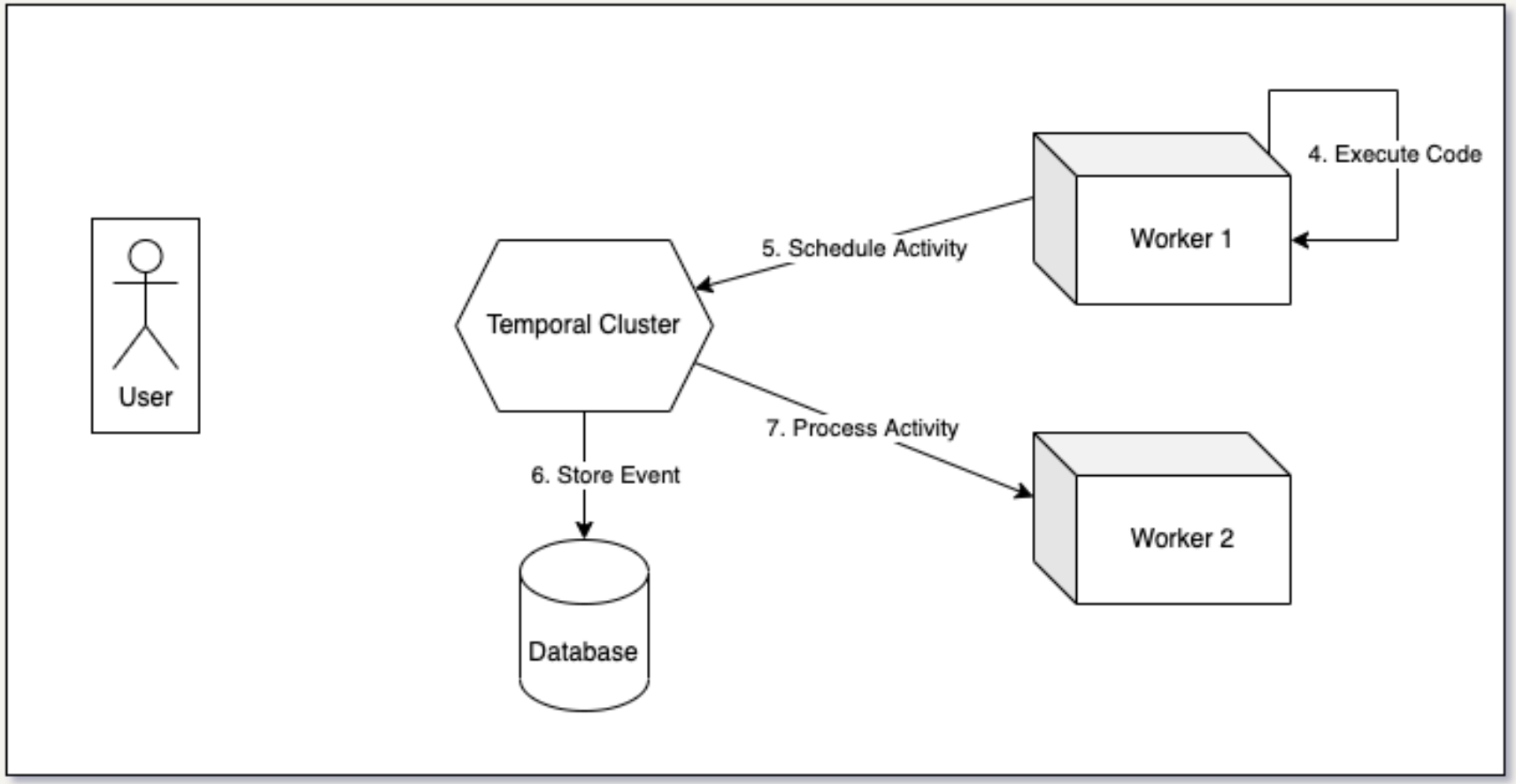
- Workflows are "as Code" => they are functions then
  - *How a function/workflow fall asleep without taking resources ?*
  - *How to resume the Xth activity in a workflow?*
  - *How can another worker take over a workflow that has already started ?*
  - …

➢ Workflows in progress are replayed by workers

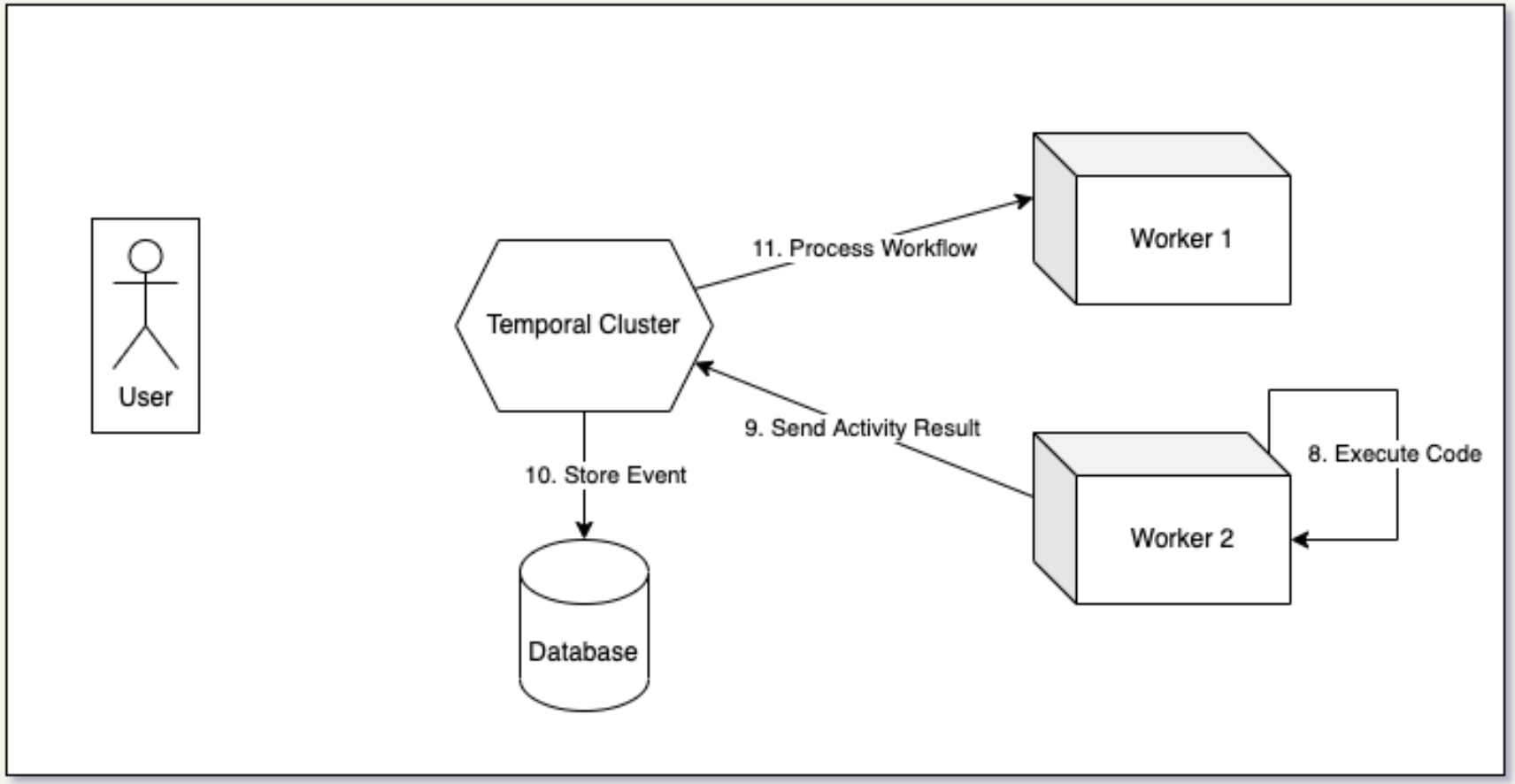
How does it work?



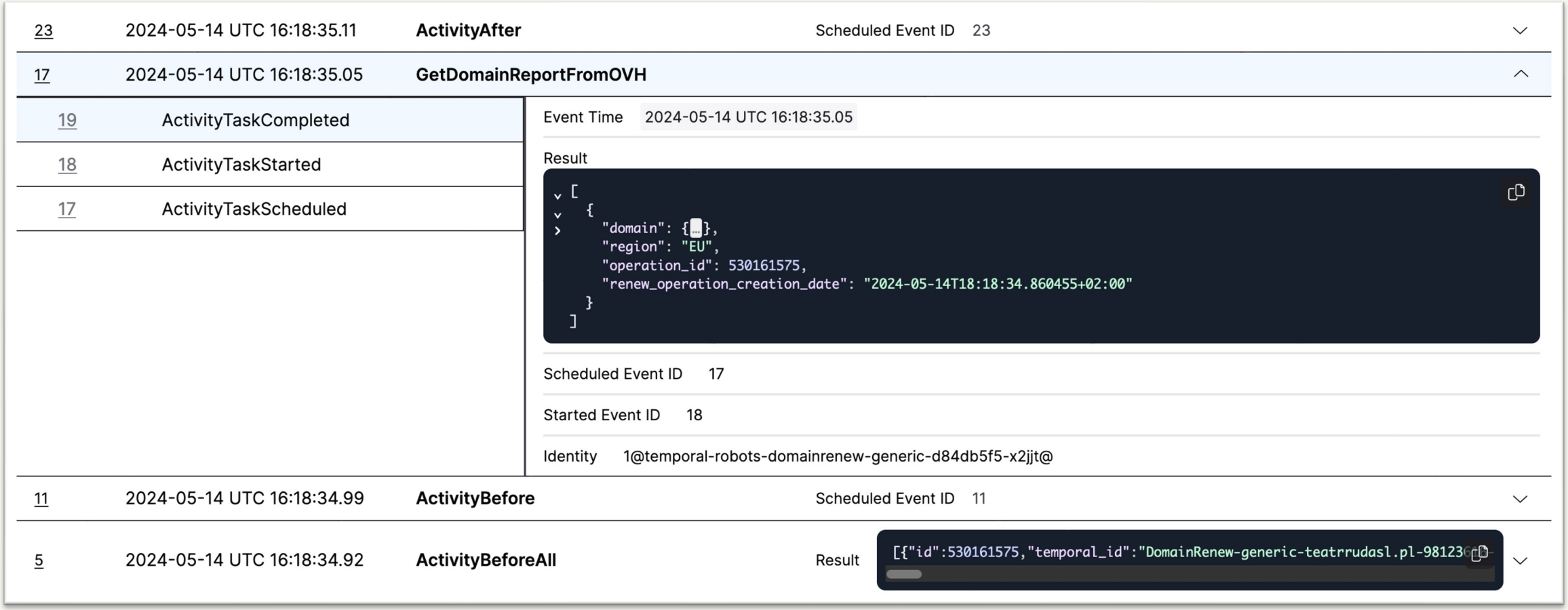
How does it work?



How does it work?



How does it work?



Concrete Case

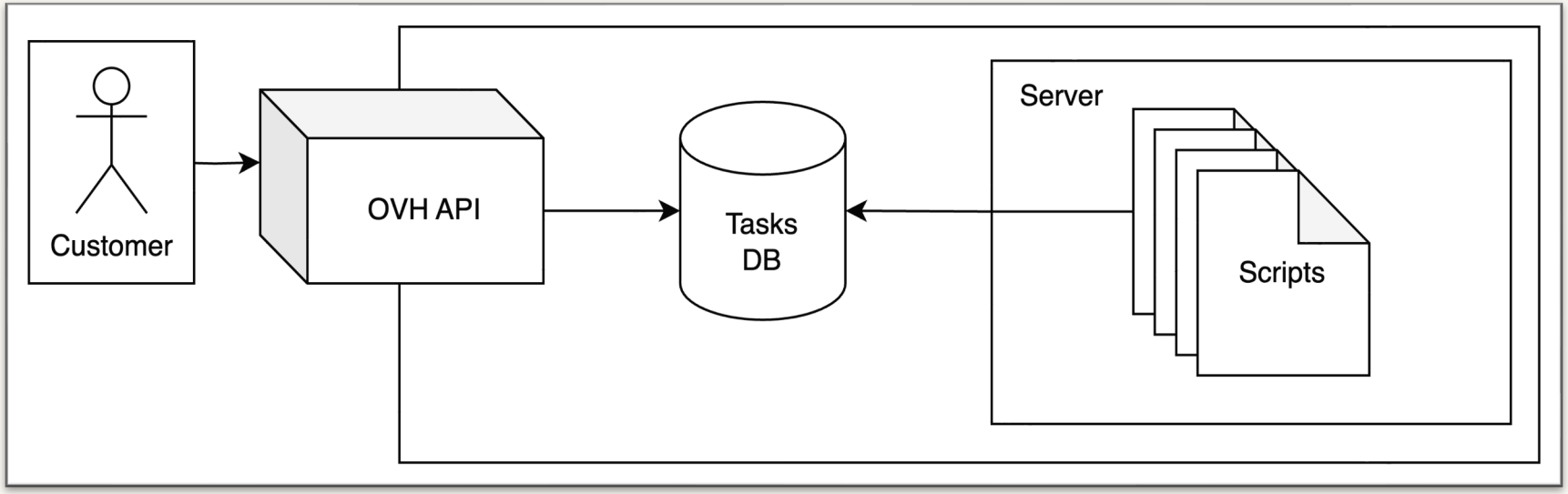
Domain Team @ OVHcloud

- > 5M domain names
- 37 different business workflows
- 2.8M workflow executions per month
  - *Including at least 640K on the 1st of the month*
  - *External providers with rate limits*
- API exposed to customers so they can track their operations
- Customer interaction on domain operations (re-launch, data correction, etc.)
- Potential automatic cancellations with refund

Constraints

- Keep a consistent SRE experience
- Be compatible with the existing tooling for operations
- Be transparent from the user's point of view
- Limit the number of activities performed at the same time
- Handle cancel cases with refund

Architecture



Decisions

- Migrate renew operations (low complexity + significant added value)
- Keep the existing database
- Use temporal only as a workflow engine not the source of truth
- User actions are handled by the tasks service which will interact with the temporal workflow (reset, signals, …)
- Workers synchronize workflow execution with database using interceptors
- Cancelation is handled using interceptors
- Using activity queues to restrict activity concurrency using interceptors

Concepts

Interceptor

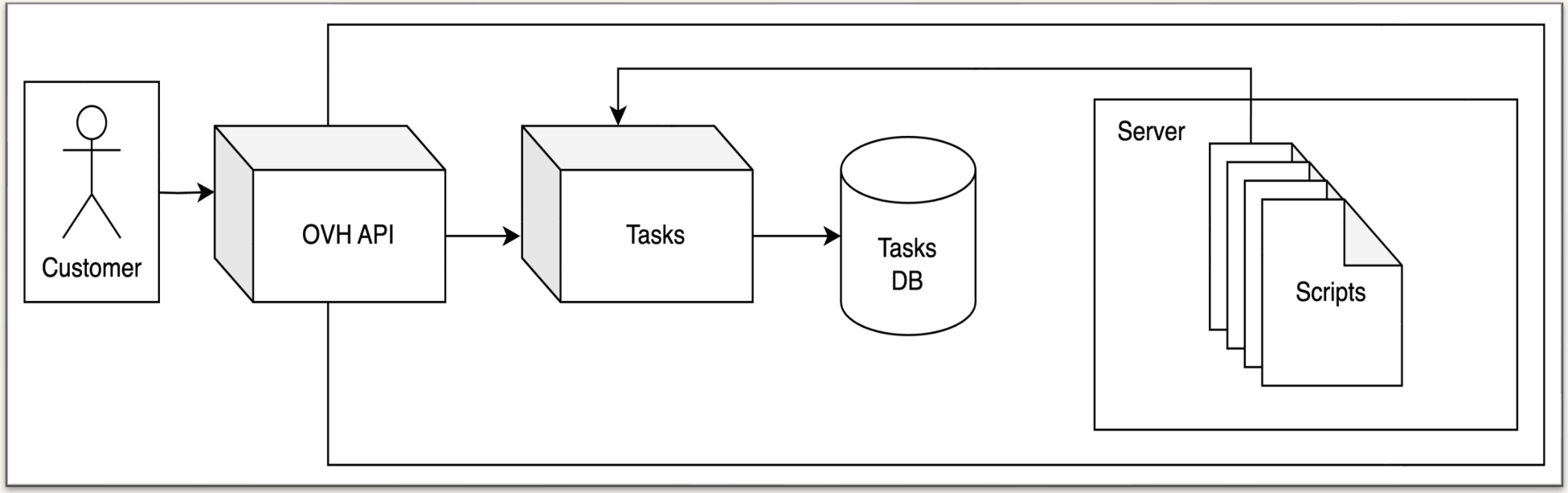
- Looks like middleware in web frameworks
- Allows to modify, augment, or even short-circuit inbound and outbound SDK calls of workflows and activities
- Not well documented yet

Example: Interceptor

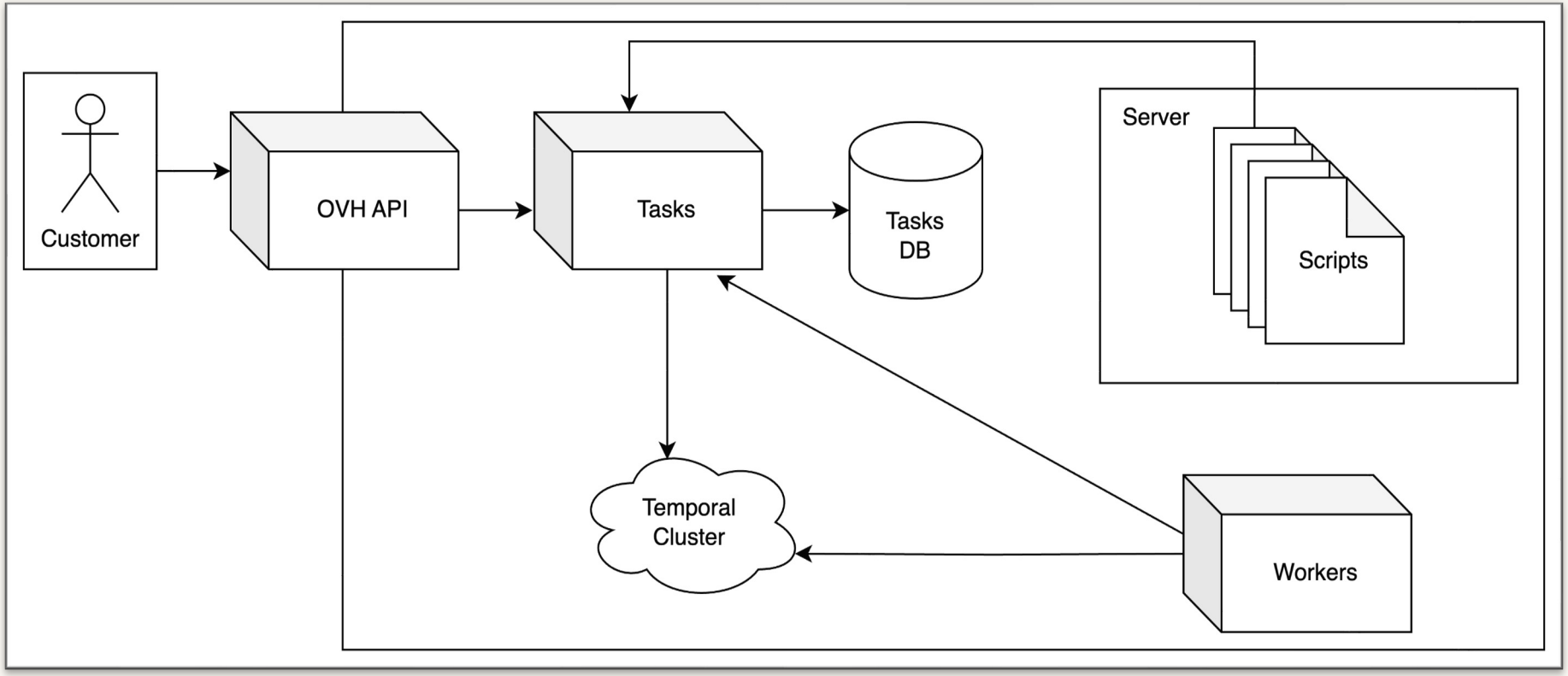
```go
func (i *syncWorkflowInboundInterceptor) ExecuteWorkflow(ctx workflow.Context, input *interceptor.ExecuteWorkflowInput) (any, error) {
    ctx, err := i.root.PreWorkflowExecutionHook(ctx, input.Args...)
    if err != nil {
        return nil, err
    }
    res, err := i.Next.ExecuteWorkflow(ctx, input)
    if err != nil {
        return res, i.root.WorflowExecutionErrorHook(ctx, err, input.Args...)
    }
    return res, i.root.PostWorkflowExecutionHook(ctx, res)
}

```

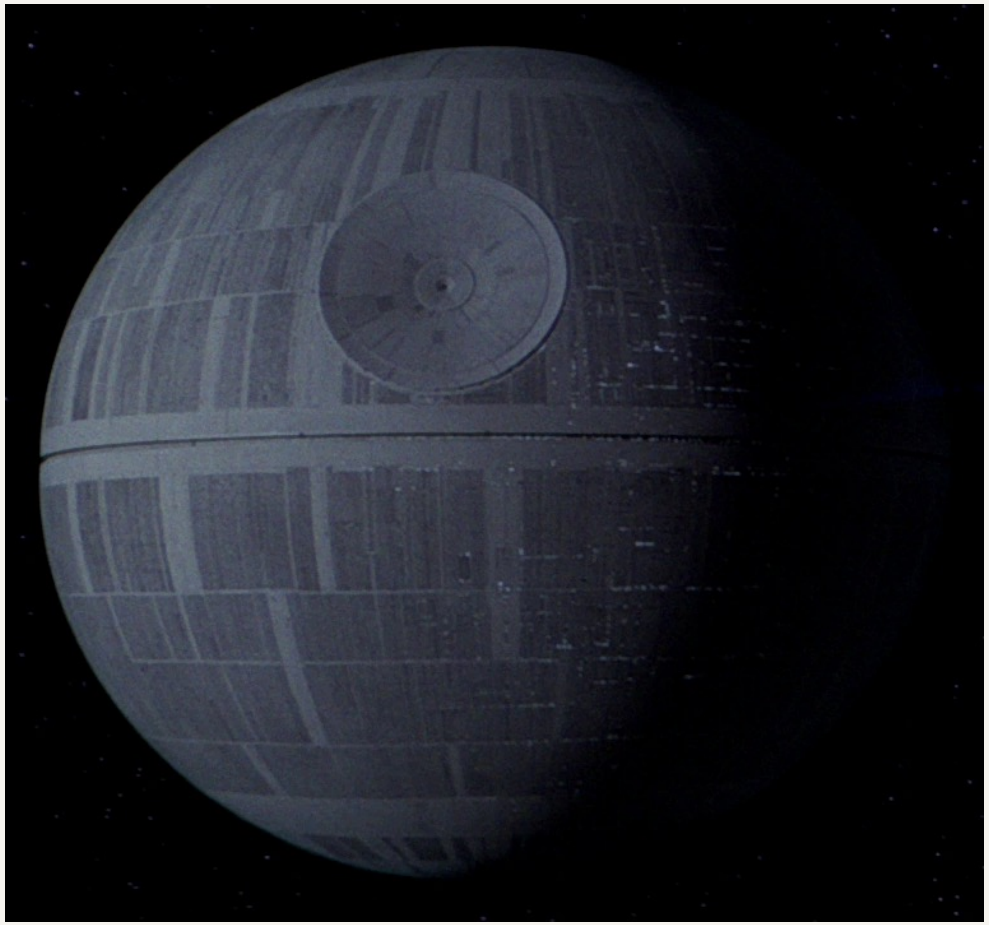
Architecture



Architecture



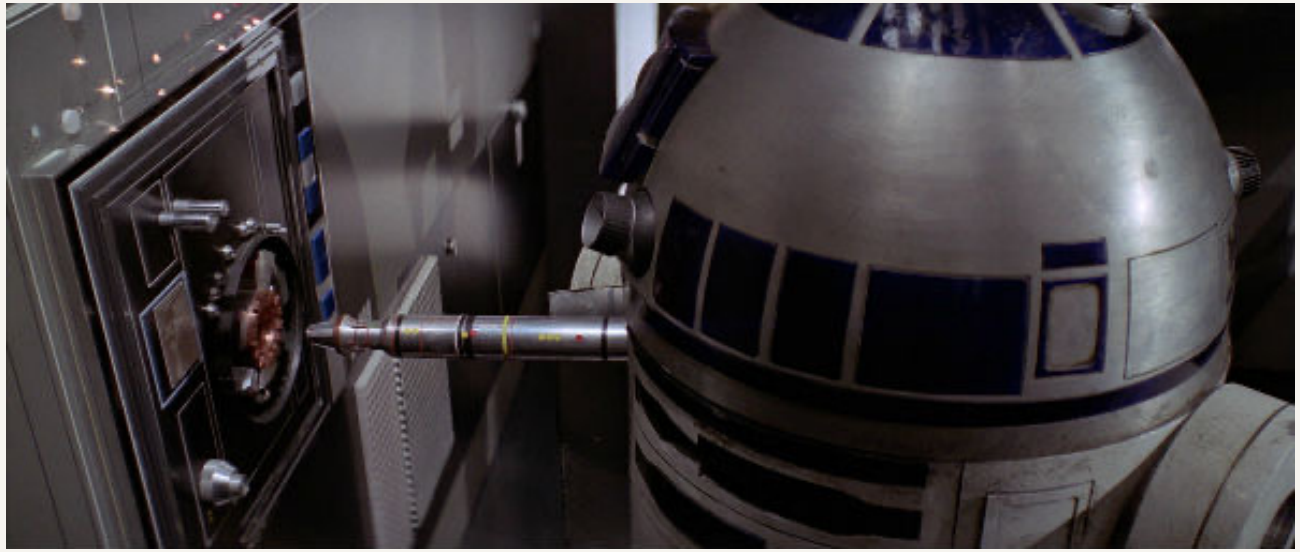
Production



Migration of .info

- Enabled using feature flipping
- Small load (~4 500 op/month)
- No major problems 🤗

➢ Until 1st of the month…



Migration of .info

- Duplicate operations 🙀

➢ Detected quickly

➢ Not caused by temporal

➢ Rescued by the idempotence of the new workflows 😅

Migration of .fr

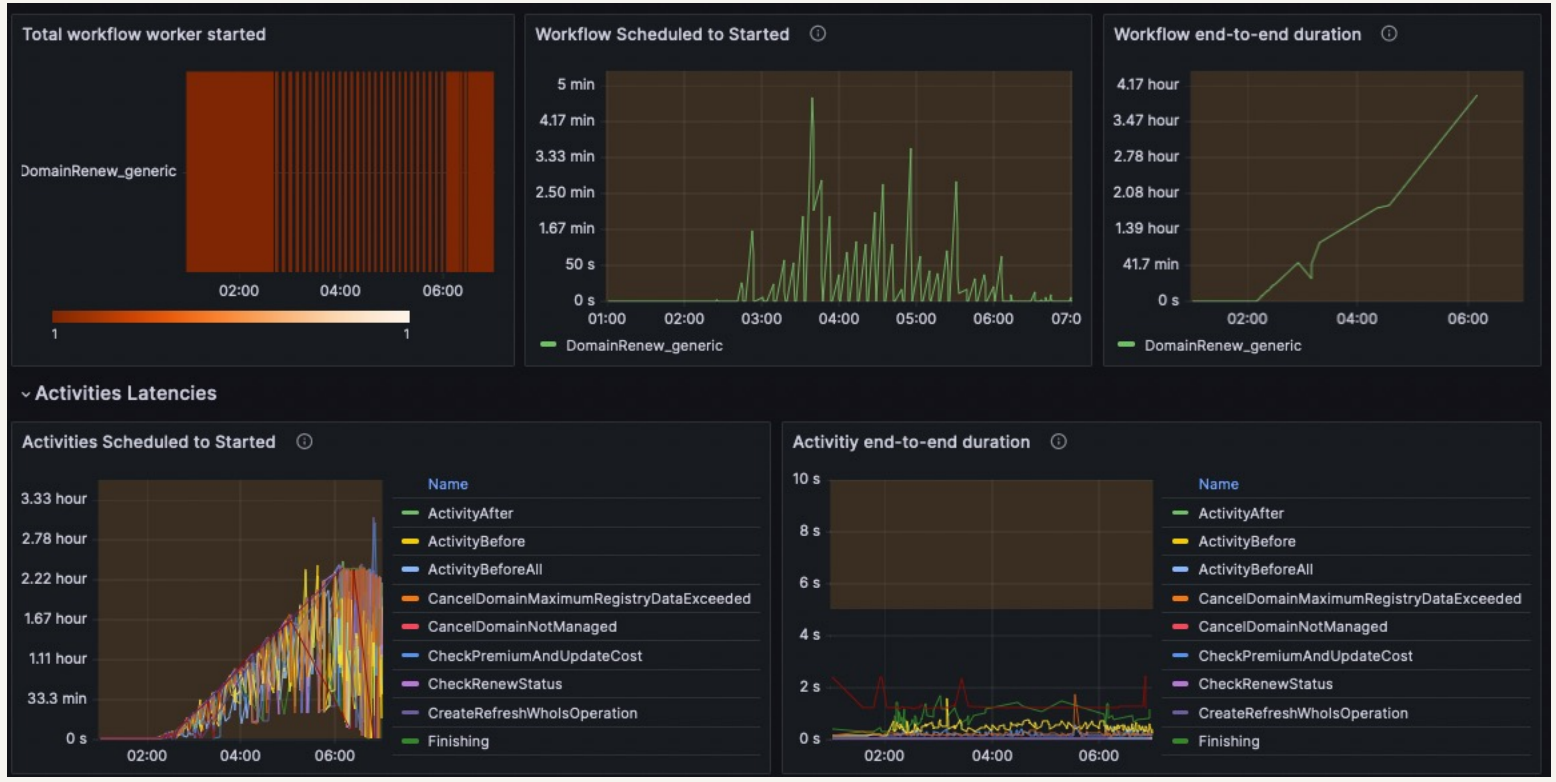
- Same workflow
- Bigger load (~125 000 op/month)
- Rate limits on workers
  - *WorkerActivitiesPerSecond*
  - *MaxConcurrentWorkflowTaskExecutionSize*
- No major problems 🤗

➢ **Until 1st of the month…**

Migration of .fr

Migration of .fr

- Workflows take a long time to be processed
- Workers don't seem overloaded
- Scaling up workers doesn't fix the problem
- Change the rate limits doesn’t fix the problem



Migration of .fr

- Temporal history service overloaded
- Too many idle connections on DB

➢ Scaling up the temporal services does fix the problem but…

Migration of .fr

Migration of .fr

- 1M queued tasks processed in less than 30 min 🚀
- Services called by workflows were saturated
- As we use a connection pool to manage registry calls, quite a few of calls timed out
- Workflows were failing du to that

➢ Reconfiguration of rate limits + bulk relaunch of workflows (reset before the last activity) and everything were good ✅

Results

- We configured auto scaling on temporal cluster
- Everything is stable since then
- We process 600K operations each month
- 80% of renew operations are migrated
- Average renewal time reduced from 5 minutes to 5 seconds
- We plan to use temporal for other types of asynchronous workflows

Useful links

- Website: https://temporal.io/
- Github: https://github.com/temporalio
- Courses: https://learn.temporal.io/
- Documentation: https://docs.temporal.io/
- Blog: https://temporal.io/blog
- Slack: https://temporal.io/slack
- Examples:
  - Go: https://github.com/temporalio/samples-go
  - Java: https://github.com/temporalio/samples-java
  - Typescript: https://github.com/temporalio/samples-typescript
  - Python: https://github.com/temporalio/samples-python
  - PHP: https://github.com/temporalio/samples-php
  - .Net: https://github.com/temporalio/samples-dotnet

Thank You

If you want to give some feedbacks: