Mistral workflow service

Mistral is a workflow service that reduces the cost of operations and time to market and increases the reliability of systems built on top of it.

Mistral is seamlessly integrated with OpenStack and helps automate complex, multi-step processes involving interactions with multiple systems in any distributed environment, including OpenStack cloud. Mistral allows us to represent these processes as a set of discrete states, and provides an execution mechanism and life-cycle management for them. We use the term “workflow” to denote this process.

What is a workflow?

A workflow is a defined sequence of tasks performed to achieve a desired result. For example, if you want to watch a movie in the theater, you need to open a website, pick a show time, purchase a ticket online, pick up friends, go to the theater, buy food and beverages, and watch the movie.

Some steps of this workflow may assume different outcomes. For example, if you picked a show time, you may learn that there are no tickets left for this time (see Figure 1). In this case, you can either pick a different time or decide to not watch the movie and instead, for example, go to a restaurant. In this case you’ll need to start a workflow that leads to having dinner in a restaurant.
Workflows in computing

When a computer process (such as booking an airline flight or recovering system infrastructure from a disaster) is automated, the process is also decomposed into individual smaller tasks. In addition, the process defines how to move from one task to another depending on a task’s result.

Unlike regular programming languages (Java, Python, etc.), workflow engines enable natively describing such a process as a set of tasks where each task has its own state (running, failed, successful or cancelled) and result; this is referred to as developing stateful processes.

The state of a workflow is durable (or persistent), meaning that it’s stored in a database or another type of storage that keeps data on a hard drive. State management is what makes a difference when choosing what to use for process automation: a regular programming language and its libraries or a workflow technology.

Why use a workflow?

Resizing images (JPG, PNG, TIFF, etc.) using services of a computing cloud such as OpenStack, Amazon Web Services (AWS) or Microsoft Azure is an example where a workflow technology such as Mistral works well. Let’s see why.

As shown in Figure 2, a process that resizes images may consist of the following tasks:

1. Selecting the images you want to resize using your chosen criteria
2. Uploading images to cloud storage
3. Creating and booting up a number of virtual machines (VMs)
4. Assigning each VM with a subset of images
5. Signaling VMs to resize images assigned to them and store the results where needed
6. After all images are resized, deleting the original images from cloud storage
7. Terminating the VMs.
The entire process can take days and should include:

- A mechanism to track the progress of the whole process at each stage
- Reliability, so that the process is not limited by a single machine’s lifespan
- A recovery mechanism in case something fails in the middle of the process
- Parallel processing using multiple VMs to reduce overall execution time
- A mechanism to synchronize parallel activities so that cleanup can be started only after all the images are resized
- The capability to stop the whole process and clean up all the resources
- A mechanism to send notifications upon completion via email or other communication channels.

How a workflow helps meets these needs

A workflow helps meet these needs through:

- State management
- Reliability
- Conditional transitions between tasks and parallelism
- Synchronization
- Rerunning a workflow
- Life-cycle management.

Each of these topics is discussed in the following sections.

**State management**

When a process is represented as a workflow, we get a network-like structure, essentially a graph, consisting of individual tasks with state and result. State can take one of the following values: RUNNING, ERROR, SUCCESS, PAUSE or CANCELLED. When the workflow is initiated, responsibility to maintain the state of individual tasks and the entire workflow is offloaded to the workflow engine, in our case Mistral. After that, it is simple at any time to see how the workflow is progressing: whether it’s still running, has finished successfully or has finished with an error.

**Reliability**

Unlike a regular program or a script, a workflow in Mistral can be run by a distributed, scalable and reliable execution mechanism. It is possible to run a cluster of workflow engines so that if some of the engines crash or are stopped manually, the rest of the engines will continue to process workflows. As a result, it is possible to run processes for days or even weeks without needing to worry about a single machine lifespan.
Conditional transitions and parallelism
When describing a workflow, transitions between tasks can be flexibly defined, and the situations in which events will be activated can also be defined. For example, it’s possible to define a transition that will run a task sending a notification about a failure to a person if some of the VMs do their job successfully and some fail. If one task in the workflow has transitions to two or more other tasks, they will effectively be running in parallel.

Synchronization
If a workflow has multiple parallel activities, or branches, a synchronization point can be used to indicate that all of the branches must be complete before continuing. In Mistral this is easy to do using a “join” operation. For example, a task can be marked as “join: all”. The task will then wait for all inbound tasks to complete before it runs itself.

Rerunning a workflow
If a workflow fails in the middle, for example due to a problem with an environment, Mistral enables fixing the problem manually and continuing the workflow from the exact point where it failed. This capability makes Mistral very different from various homegrown solutions that need to start the process from the very beginning, which can be very expensive if the duration of the workflow is long: days or weeks.

Rerunning a workflow with Mistral from the point it failed is possible because of Mistral state management. The workflow engine always keeps the state of the process in durable storage so that we know from where to continue.

Life-cycle management
Mistral provides full life-cycle management for workflows. You can start a workflow, see how it’s running, and then decide to continue, pause the workflow temporarily and then resume, or completely cancel it.

Workflows can be represented graphically
Workflows can be represented graphically because they are, in fact, graphs of states (or tasks). As shown in Figure 3, graphical representation makes monitoring and operational control much easier than with other solutions that don’t use state management.
Mistral key features

- Simple YAML-based language for writing workflows
- Scalable service with REST API
- Suitable for designing distributed processes
- Helps make a stateful process that we can easily monitor
- Helps transfer data between different systems/services
- Allows accumulation of data during workflow execution
- Enables easy building of any event-based processing pipelines
- Allows injection of arbitrary code snippets into workflows
- Native integration with OpenStack
- Can be used as a standalone service without OpenStack

Mistral use cases

- Automation of administrative tasks
- Automatic scaling of distributed applications
- Deployment and life-cycle management of applications
- Self-healing of applications
- Application rolling upgrades
- Reconfiguration of applications
- Disaster recovery
- Big Data analysis and reporting

The concept of action in Mistral

“action” is a language keyword to define what concrete job a task needs to perform. Action is an important concept in Mistral. Also, users are not limited by only the actions provided out of the box. You can also write new actions in Python and easily plug them into Mistral.

Mistral architecture

Mistral consists of three main components: the API server, the engine and the executor (see Figure 4). The API server handles HTTP requests to use various Mistral functionality. There can be multiple co-existing API servers. In this case they should be put behind a load balancer. The engine is the main component responsible for parsing and running workflows. The executor is essentially a worker that runs task actions. There can be multiple engines and multiple executors.

Mistral architecture was designed with asynchronous DNA. This means that it’s natural for Mistral to orchestrate external systems where it needs to send signals to them to start running a job, and then wake up and continue the workflow when an external system responds back with the result of the job.

Architecturally, it makes no difference to the Mistral engine what handled a particular job (action), the Mistral executor or an external system; this makes interacting with any external systems very natural. Also, the engine does not actively wait for a result, meaning that it’s not holding any resources (such as objects in memory, sockets, file descriptors, etc.) specifically to wait for the result of a job. As a result, Mistral is fully asynchronous and can be scaled easily.
Using Mistral in Nokia CloudBand network service deployment

To deploy a network service, CloudBand NFV uses the TOSCA (Topology and Orchestration Specification for Cloud Applications) standard to describe the topology of the network service. When it's time to deploy the service, CloudBand creates an Operation Execution Plan based on the service topology and user configuration. This plan describes operations required to deploy the service. The operations compose a graph, meaning that some operations depend on other operations.

There are two ways to execute the plan:

- Using plain Java code (or code in a different language)
- Using a workflow engine such as Mistral

The option with Java code includes all the challenges of process automation described earlier. Based on our experience, these challenges are not trivial.
Instead, CloudBand utilizes the Mistral workflow service, which is a mature, open-source project proven to be useful for process automation. After building the Operation Execution Plan, CloudBand converts it into a Mistral workflow and offloads all the burden of executing and managing its state to Mistral, thereby gaining all the advantages of using Mistral.

**Other use cases for Mistral in CloudBand**
- Life-cycle management workflows
- Healing
- Auto-scaling
- Rolling upgrades
- Enabling a plug-in system (i.e., adding an OpenStack Virtual Infrastructure Manager node runs workflows).

**Conclusion**

Mistral is a workflow service that is seamlessly integrated with OpenStack and helps automate complex, multi-step processes involving interactions with multiple systems in any distributed environment, including OpenStack cloud. With multiple use cases and a rich feature set, Mistral reduces the cost of operations and time to market and increases the reliability of systems built on top of it.

For more information about Mistral, please visit the [OpenStack Mistral website](#).