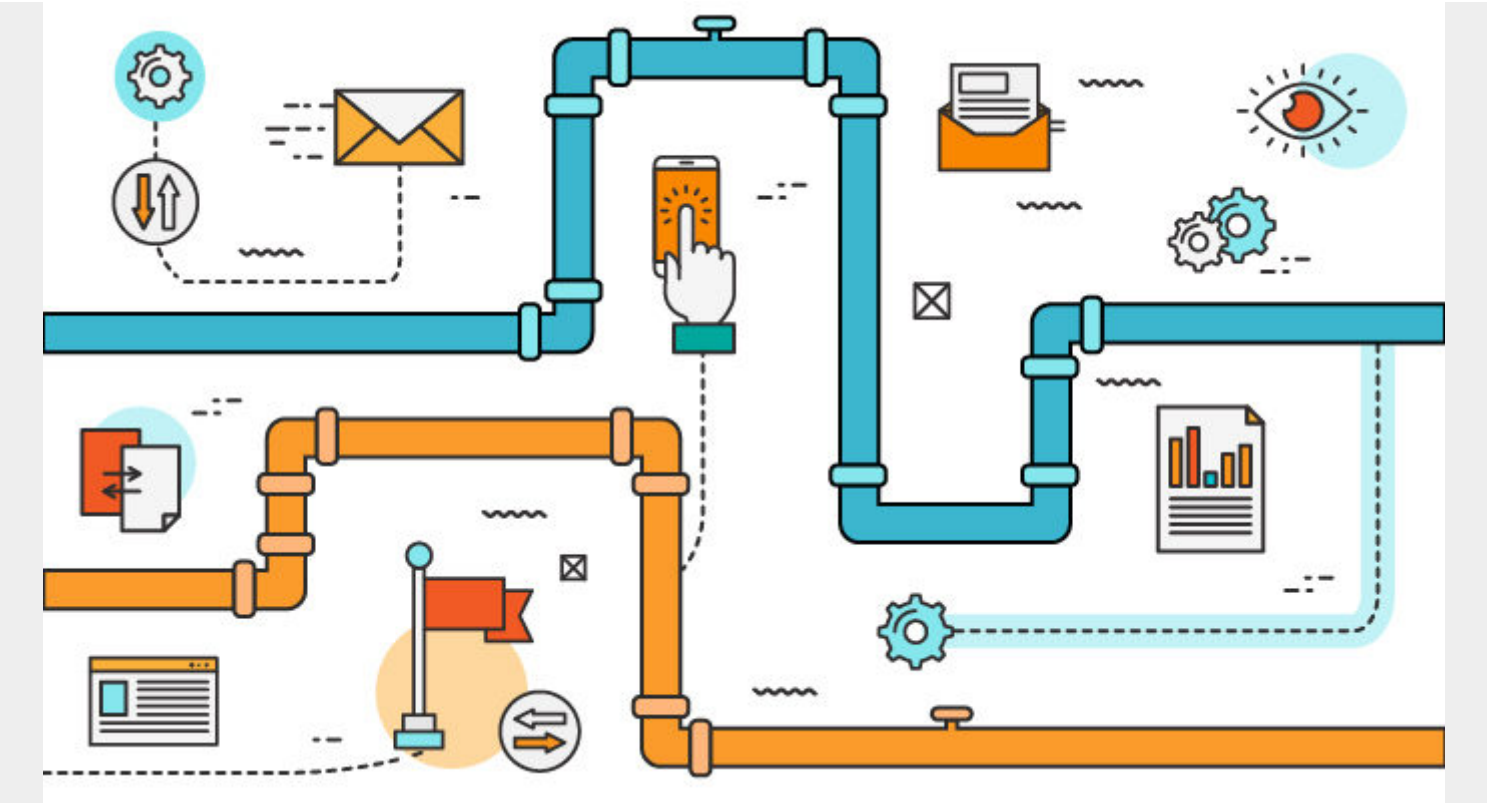
WORKFLOW ORCHESTRATION: AN INTRODUCTION



The Wikipedia definition of “workflow” contains the mysterious warning “This article may be too technical for most readers to understand”. So, I’m avoiding the technique of quoting from that source. It’s mysterious because it seems pretty simple what workflow is:

Some series of activities (work) performed in a reasonable sequence (flow).

What may account for the perceived complexity is the number of wildly different industries and segments the term is used in. When it comes to IT, both workflow and orchestration, individually and as a phrase, are used in disciplines including:

- Cloud and infrastructure management
- Service management
- Release management
- Application management

This article will focus on using workflow orchestration to run stateful business applications in production, including:

- Why that is now called application workflow orchestration
- How it’s unique and differs from other disciplines
- What to think about when choosing such a tool

One final note of introduction; there is some discussion regarding the differences between orchestration and automation. In this discussion, the distinction is negligible and so the terms are

used interchangeably, mainly to avoid excessive repetition.

What are some types of orchestration?

Here are some common orchestration types:

Cloud orchestration

Cloud orchestration is typically used to:

- Provision, start, or decommission servers
- Allocate storage
- Configure networking
- Enable applications to use cloud services like databases

This set of activities is the modern equivalent of infrastructure setup initially performed manually by system administrators and later with provisioning tools like Chef, Puppet, and others.

Service orchestration

Service orchestration takes a broader approach and seeks to provide a complete end-to-end solution for delivering a "service".

In an ideal world, this set of activities would include everything from designing an application according to the business requirements, all the way through to running it in production. However, frequently, the tasks included in a service are more a function of the capabilities provided by a particular set of tools from a specific supplier rather than being a holistic approach encompassing all the required tasks.

When used by providers of Service Desk and similar tools, the orchestration may be limited to tracking the status of tasks and approvals and less on the actual execution of the tasks themselves.

Release orchestration

This is the definition according to Gartner: "Release Orchestration tools provide a combination of deployment automation, pipeline and environment management, and release orchestration capabilities to simultaneously improve the quality, velocity and governance of application releases.

ARO tools enable enterprises to scale release activities across multiple, diverse and multigenerational teams (e.g., DevOps), technologies, development methodologies (agile, etc.), delivery patterns (e.g., continuous), pipelines, processes and their supporting toolchains".

Application workflow orchestration

Now let's focus on our topic by looking at an example of a connected vehicle application. One of its goals is to reduce vehicle downtime by collecting and monitoring the telematics data collected from sensors on the vehicle.

Extensive data is generated about every aspect of vehicle operation. That data is ingested and analyzed by machine learning algorithms to predict potential failure. If a problem is anticipated, the application correlates vehicle location to service depots with parts availability, directing drivers to

complete the preventative repair in route versus a roadside repair.

The workflow includes:

- Watch for telematics data arriving from a third-party provider
- Move the data on a regular basis from its cloud-based landing location to a Hadoop cluster
- Enrich the sensor data with vehicle history, fleet ownership and warranty data by pulling those data sets from internal systems of record
- Run the analytics
- Select a service depot based on the vehicle location and parts availability
- Order a part if none are available and replenish inventory if this repair reduces the on-hand amount below a threshold
- Book a service appointment
- Notify the driver and other interested parties

Note that in this example, the orchestration tool is moving data and invoking application components to accomplish the desired business outcome. The responsibility of the orchestrator is to:

- Invoke the right process at the right time
- Ensure that one process completes successfully before the next one starts
- Provide visualization and management of the workflow

Additional requirements are less obvious but just as critical.

We need a way to build the flow, and if errors occur, we need notification to be sent to interested parties. There must be a way to examine the details of each workflow step to determine what is being done, what was the result, view any messages that the processes may have generated, monitor status and progress and display that information, and of course we need logging and tracking for auditing and governance purposes.

Last but certainly not least, we need a way to assign business priorities to these tasks and some completion/service level rules to ensure the workflow operates within some agreed-to quality of service definition.

Why application workflow orchestration?

If you examine the steps described above, you may recognize similarities to other processes and tools called by different names, like data pipelines or schedules or even batch jobs.

A major reason for this new term is to make it very clear that new platforms, technologies, and data sources are very much a part of orchestration. Whether it is cloud, containerization, data and analytics, streaming and microservices architectures, they are all very integral part of application workflow orchestration.

Workflow orchestration best practices

Some of the many uses cases in which application workflow orchestration play a significant role include:

- Orchestrating data pipelines

- Training ML models
- Detecting fraud in AML flows
- Applying preventive maintenance analytics to maximize oil well production

Generally, IT focuses on the code being run by application workflow orchestration tools but rarely allocates the same level of attention to the design and maintenance of the workflows themselves. Since workflows are code, whether written in JSON, XML, Python, Perl or Bash, they should be treated like code.

Here are some recommendations for standard practices to consider adopting and capabilities to look for in selecting a application workflow orchestration tool.

Support an "as-code" approach

Whether the workflows are authored via some graphical interface or written directly in code, version control is mandatory. Of course, in order to enable modern deployment pipelines, it should be possible to store and manage workflows in some text or code-like format.

Think in microservices

Avoid monoliths. This applies to workflows just as it does to application code. Identify functional components or services. Use an "API-like" approach for workflow components to make it easy to connect, re-use and combine them, like this:

Service (Flow) A: Do something1, emit "something1 done" Emit "something2 running", Do something 2, Emit "something2 done" Emit "Service A" done	Service B: Wait for "Service A" done Do B thing, emit BThing done	Service C: DO NOT run while something2 is running Wait for BThing done Etc.

Don't reinvent the wheel

If you have a common function, create a single workflow "class" that can be "instantiated" as frequently as required, yet maintained only once.

Instead of creating multiple versions of a service, use variables or parameters that can accommodate the variety.

Process lineage

Data lineage is frequently cited as a major requirement in complex flows to support problem analysis.

Process lineage is just as important and a mandatory requirement for effective data lineage. Without the ability to track the sequence of processing that brought a flow to a specific point, it is very difficult to analyze problems.

The need for process lineage arises quickly when a problem occurs in a pub/sub or "launch-and-forget" approach used in triggering workflows.

Make the work visible

Process relationships should be visible.

One scenario where such visualization is particularly valuable is when everything appears perfectly normal, but nothing is running. Having a clear line of sight between a watcher or sensor that is waiting for an event and the downstream process that wasn't triggered because the event did not occur can be extremely valuable.

Codify SLAs

The best way to define a non-event as an error is by defining an "expectation", commonly called a service level.

At its most basic, an unmet service level agreement (SLA) is identified as an error. For example, we expect a file to arrive between 4:00 PM and 6:00 PM. It takes approximately 15 minutes to cleanse and enrich the file and another 30 minutes to process that file. So, we can set the SLA to be 6:45 PM. If the processing hasn't completed by then, whether it's running late or hasn't even started yet, the error can be recognized at 6:45 PM if the flow hasn't completed.

A more sophisticated approach is to use trending data to predict an SLA error as early as possible. We know the cleanse step runs approximately 15 minutes because we collect the actual execution time for the last 'n' occurrences. The same is true for the processing step. If the cleanse step hasn't finished by 6:15, or if the processing step hasn't started by 6:15, we know we'll be late. We can generate alerts and notifications as soon as we know, so that we have the maximum time to react and possibly rectify the problem.

A final enhancement is providing "slack time" to inform humans how much time remains for course correction. In the above scenario, if the cleanse step doesn't start on time, at 6:00 PM, there are 45 minutes available to fix the problem before the SLA is breached.

Categorize

As you are designing your workflow "microservices" and connecting tasks into process flows, make sure you tag objects with meaningful values that will help you identify relationships, ownerships and other attributes that are important to your organization.

Use coding conventions

Imagine creating an API for credit card authorization and calling it "Validate". If your response is "sounds good", this blog probably isn't for you. I'm hoping most will think the name should be more like "CreditCardValidation" or something similarly meaningful.

This point is simply to think about the workflows you create in a similar way. It may be great to call a workflow "MyDataPipeLine" when you are experimenting on your own machine but that gets pretty confusing even for yourself, never mind the dozens or hundreds of other folks, once you start running in a multi-user environment.

Think of others

You may be in the relatively unique position of being the only person running your workflow. More likely, that won't be the case.

But even if it is, you may have a bunch of workflows and you don't want to have to re-learn every time you need to analyze a problem or when you modify or enhance it.

Include comments or descriptions or if it's really complicated, some documentation. And remember to rev that together with the workflow.

Keep track

Inquiring minds want to know… everything. Who built the workflow, who ran it, if it was killed or paused, who did it and why? Did it run successfully, or did it fail? If so, when and why? How was it fixed?

And so on, and so on.

Basically, when it comes to workflows for important applications, you can never have too much information. Make sure your tool can collect everything you need.

Prepare for the worst

You know tasks will fail. Make sure you collect the data that will be needed to fix the problem and that you keep it around for a while. That way, not only can you meet the "Keep Track" requirement, but when problems occur, you can compare this failure to past failures or to previous successes to help determine the problem.

Harness intelligent self-healing

Finally, look for flexibility in determining what is success and what is failure. It's correct and proper to expect good code but we have all seen code that issues catastrophic error messages, but the task completes with an exit code of zero.

You should be able to define what is an error and what is not and accordingly you should be able to define automated recovery actions for each specific situation.

What do you think?

Application workflow orchestration, in one way or another is almost universally used, but rarely discussed. It would be great to add lots of voices to this discussion.

Do you have some practices or requirements you would add or remove? Is there another point of view you would like to put forward?

Related reading

- BMC Workload Automation Blog
- Workflow Orchestration vs. Continuous Integration: What's the Difference?
- How Workflow Orchestration Improves Application Development & Monitoring

- Beginner's Guide To Workplace Automation