BRING KUBERNETES TO THE SERVERLESS PARTY

Nirvana for application developers is to be able to focus 100% on building the best functionality that will delight users and drive great outcomes for business. Having to worry about compute resources, network configuration and all the other intricacies of infrastructure is, at best, an annoyance. Serverless is one approach that holds out the promise of freeing developers from that drudgery.

However, when serverless is mentioned, typically public cloud vendors and their "Function as a Service" (FaaS) offerings come to mind. In this article, I want to discuss some other options.

Containerization is another technology that has gotten a lot of attention because of its potential to significantly decouple building business logic from worrying about the infrastructure in which that code runs in. And of course, once you get onto the subject of containers, Kubernetes cannot be very far behind.

Kubernetes is one of the most popular open-source projects ever and was the inaugural project of the Cloud Native Computing Foundation (CNCF). In a 2018 blog celebrating Kubernetes' graduation, the CNCF wrote: "Compared to the 1.5 million projects on GitHub, Kubernetes is No. 9 for commits and No. 2 for authors/issues, second only to Linux".

The pace of innovation in the Kubernetes community has been dizzying with over one million contributions from more than 2,000 companies. With all that work, the level of functionality is incredibly rich but accessing that power comes with a "complexity" surcharge. If you have ever built a YAML manifest, you know there are tons of options available for networking and storage, bordering on the very complex; hardly the blissful ignorance hoped for with FaaS.

What you may not know, is that several options bring Function-as-a-Service to Kubernetes including Knative, Kubeless, OpenFaaS, Apache OpenWhisk and others. CNCF has an interesting survey here that identifies Knative as most popular, used by 34% of survey respondents who have installed FaaS on Kubernetes.

Knative consists of two major components; Eventing and Serving. Eventing provides facilities for generating and consuming events. The list of supported event sources that are built in, include Kubernetes object events, messages published to Google PubSub, AWS SQS and Kafka topics and events related to select storage and logging facilities. You can also write your own source.

Knative Serving lets you deploy containers in a serverless fashion with Knative handling auto-scaling, routing and networking, and versioning.

In browsing the Knative documentation, it appears there is a fair amount of work to get to the "less" part but once configured, you are able to launch containers as services in Knative about as easily as the equivalent activity on Google, AWS or Azure.

The choices of "Serving" and "Eventing" as component names may lead you to believe that functions are invoked exclusively as either long-running services or in response to events. While that may be true for much of FaaS workload, there is also a significant requirement for managing workload that runs to completion and that is known as jobs or batch. In fact, the CNCF Working Group on Serverless has launched a subgroup focusing specifically on workflow and their requirements are reflected in this design document.

The importance of workflow orchestration is further demonstrated by offerings such as AWS Step Functions, Azure LogicApps and Google Cloud Composer. Knative Eventing also supports "CloudSchedulerSource" and "CronJobSource" sources but there seem to be several gaps, not only in Knative which still hasn't reached its 1.0 version (v0.13.0 at time of writing) but also tools for orchestrating serverless in general.

There are lots of blogs like this one or this, discussing the importance and value of orchestrating serverless workflows. Many of these requirements go beyond what the CNCF and cloud vendors are working on and include:

- Sequence of functions
- Visibility of relationships (predecessor/successor)
- Relationships with non-FaaS components such as traditional infrastructure and human actions
- Branching depending on success/failure or other indicators
- Parallelism
- Graphical tools for building and authoring flows
- Error recovery such as retry
- Easy access to logs and output
- Business SLAs
- Audit trail

So if you need to orchestrate a combination of serverless and something else, even objects like Kubernetes JOBs, never mind functions in public clouds or traditional applications in on-premises environments, you will have to do a lot of very heavy lifting on your own to gain any measure of integration. In the interest of full disclosure, I work for a commercial vendor that provides very interesting capabilities in this area. You may wish to check us out.