Understanding Cadence Workflow for Developers and Architects

Overview

Cadence is an open source technology that underpins the architectures of several leading technology organizations. Most notable of these is Uber who originally created Cadence and continue to support ongoing open source development. Cadence drastically simplifies the process of developing and operating multi-step and long running processes at scale. Developers use Cadence to write code more easily while delivering high reliability, scalability, and resilience.

Like any technology, it is vital to have a conceptual understanding of the benefits to determine if Cadence can fit your organization. It is challenging to evaluate a solution without understanding your requirements if you do not know what questions to ask before a potential implementation. To help with that, our industry experts have been digging into what makes Cadence tick and how it is used.

This paper walks you through the basics of Cadence and key learnings to help you understand how it works and the benefits it can provide to your organization. We will give a brief history of Cadence, how workflows work, key features, and how you can apply Cadence to your data infrastructure.

instaclustr
Now part of **Spot by NetApp**

Introduction: A Little Bit of Context and History

Cadence is a backend server application and client library designed to simplify development for the "large number of use cases that span beyond a single request-reply, require tracking of a complex state, respond to asynchronous events, and communicate to external unreliable dependencies".[1] Cadence does this by providing a "fault-oblivious stateful programming model that obscures most of the complexities of building scalable distributed applications". Exactly how it does this and how programmers interact with this model we will explore in the later sections of this paper.

Cadence was initially developed by Uber and open sourced in 2017 under the MIT license. Uber has publicly confirmed that Cadence is a foundation technology within the company, being used for over 500 use cases. In addition, Uber has confirmed that they plan to continue supporting the project through an open community.

In addition to Uber, Cadence is used widely by many leading technology organizations. Publicly documented use cases include DoorDash, StashAway, and Cisco. DoorDash states that Cadence "will bring us massive gains in developer productivity due to its ease of use and abstraction of fussy details."[2]

An Illustrative Example

```
public void manageSubscription(Subscription subscription) {
        activities.sendWelcomeEmail(subscription);

        int billingPeriodNum=0;
        while (billingPeriodNum < subscription.getSubscriptionPeriods()) {
            Workflow.await(subscription.getBillingPeriodDuration(),
                    () -> subscriptionCancelled);

            if (subscriptionCancelled) {
                activities.sendEarlyCancellationEmail(subscription);
                break;
            }

            activities.chargeCustomerForBillingPeriod(subscription);
            billingPeriodNum++;
        }

        if (!subscriptionCancelled) {
            activities.sendSubscriptionOverEmail(subscription);
        }
    }
```

[1] "Overview." Cadence Workflow, cadenceworkflow.io/docs/get-started. Accessed 1 Jan. 2022.
[2] Lin, Alan. "Building Reliable Workflows: Cadence as a Fallback for Event-Driven Processing." Doordash Engineering, 14 Aug. 2020, doordash.engineering/2020/08/14/workflows-cadence-event-driven-processing.

This code is an example of a simple workflow implemented in Cadence. (In Cadence, a workflow is the logic that defines the logical flow of steps. Workflows then execute actions to perform the tasks in the workflow.) Anyone with a basic familiarity with reading Java code can readily understand the business logic and see that the function manages a subscription for a fixed number of periods, charging the customer each period and dealing with the case where the subscription is canceled early. However, thanks to the magic of Cadence, this code will run reliably over periods as long as months or years, through server restarts and failures. It will also scale to handle managing millions of subscriptions concurrently. Let's start peeking behind the scenes to see how the magic works.

The first thing you need to know is that you won't be calling the method above directly from your code, instead, it is invoked using something like this:

```
SubscriptionWorkflow workflow =
workflowClient.newWorkflowStub(SubscriptionWorkflow.class);

workflow.manageSubscription(subscription);
```

In this example, `workflowClient` is a Cadence client class that has been instantiated with a connection to a Cadence cluster. We use that to get a stub matching an interface that contains the workflow method defined above. That stub is what we then used to invoke the method. When we invoke the method via the stub, we are asking the Cadence server to invoke the method. It will then find a registered worker service to run that workflow (there is some simple code to do that registration that we haven't shown) and ask that worker to run the function. As Cadence is now between the code that kicks off the workflow and the execution of the workflow, it can perform a few essential functions:

- load balancing across multiple worker servers and easy addition of more workers to the pool if required (the process just starts up and registers itself with Cadence)
- reliably queuing workflow requests until a worker is available and throttling of request to avoid overwhelming workers (very helpful if you need a buffer to allow auto-scaling of workers to occur to react to a peak load)
- in the event of worker failures, it automatically restarts the workflow processes on different worker instances.

Performing all of these functions requires Cadence itself to be highly reliable and scalable, and so it is itself a distributed, horizontally scalable system with high availability built-in. Cadence also supports highly available and scalable data stores such as Apache Cassandra®. Building this reliability and scalability into Cadence allows worker programs to be simple, single-server applications while operating as part of a highly reliable and scalable system.

Now looking at workflow methods and how they interact with Cadence, the next key

concept to look at is activities. You might have noticed in the sample code that there is an `activities` variable being used that is not defined in the function scope. `activities` is defined in the containing class as

```
private final SubscriptionActivities activities =
    Workflow.newActivityStub(SubscriptionActivities.class);
```

`WorkFlow` is one of the Cadence client library classes and the `newActivityStub` method is very similar to the `newWorkflowStub` method we used to invoke our workflow in the section above. It is giving us a stub that matches the SubscriptionActivities interface that defines `sendWelcomeEmail` and the other functions called by the main workflow function (this code is not shown but it is just whatever your business logic needs). Just like with workflows, using the stub actually asks Cadence to execute the activity rather than executing it directly. This provides all the same reliability and scalability benefits that we described for workflows. For activities, it also provides the benefit that Cadence is able to reliably save the result of activity after it is executed. This is very important when it comes to restarting a workflow that has failed or been suspended part way through execution. We'll explain how restarting a workflow works in the next section.

Restarting Workflows

For this explanation we'll use close to the simplest Cadence workflow example you could imagine. It executes one activity to concatenate a string and then sleeps for 10 seconds (note: there is no actual need to put string concatenation in an activity, it's a trivial example to simplify the explanation). Print statements have been added to help track what's going on. Here is the core of the code:

```
public static class GreetingWorkflowImpl implements GreetingWorkflow {

 private final GreetingActivities activities =
     Workflow.newActivityStub(GreetingActivities.class);

 @Override
 public String getGreeting(String name) {
   String greetingActivityResult = "Not set";
   System.out.println(

       "About to run the composeGreeting activity. Activity result = " +
        greetingActivityResult);
```

```
greetingActivityResult = activities.composeGreeting("Hello", name);

   System.out.println("About to sleep. Activity result = " +
        greetingActivityResult);
   Workflow.sleep(Duration.ofSeconds(10));

   System.out.println(
       "Finished running workflow method. Activity result = " +
        greetingActivityResult);
   return greetingActivityResult;
 }
}

static class GreetingActivitiesImpl implements GreetingActivities {
 @Override
 public String composeGreeting(String greeting, String name) {
   System.out.println("Inside the composeGreeting activity");
   return greeting + " " + name + "!";
 }
}
```

The workflow is kicked off with some code like the following:

```
GreetingWorkflow workflow =
     workflowClient.newWorkflowStub(GreetingWorkflow.class);
String greeting = workflow.getGreeting("World");
System.out.println("Workflow returned: " + greeting);
```

The printed output that this produces looks like:

```
About to run the composeGreeting activity. Activity result = Not set
Inside the composeGreeting activity
About to sleep. Activity result = Hello World!
Finished running workflow method. Activity result = Hello World!
Workflow returned: Hello World!
```

Pretty boring! Any programming 101 student could probably trace through the logic in those methods and guess that that was the output that you would expect to be printed. The reason that this is boring is that Cadence has used a feature called Sticky Execution to keep a single instance of the `GreetingWorkflowImpl` around and largely executed the code as would be expected if Cadence wasn't involved at all.

Sticky Execution is an important performance optimization. However, for this exploratory example, it is stopping some interesting stuff from happening that will help us understand how Cadence restarts workflows after failures or long waits. When sticky execution is turned off (with a small change in the initialization code) the output is different:

```
About to run the composeGreeting activity. Activity result = Not set
Inside the composeGreeting activity
About to run the composeGreeting activity. Activity result = Not set
About to sleep. Activity result = Hello World!

About to run the composeGreeting activity. Activity result = Not set
About to sleep. Activity result = Hello World!
Finished running workflow method. Activity result = Hello World!
Workflow returned: Hello World!
```

It's clear from this that there is a lot more going on than the previous run. What has happened is that, as a result of turning off sticky execution, Cadence is discarding the running GreetingWorkflowImpl object each time the thread of execution leaves the workflow to run an activity or sleep and then creating a new object after that external activity is complete and it's time to execute the next step in the workflow object.

One thing that might be noticed in the output above is that the "Inside the composeGreetingActivity" message appears only once, implying that the activity method has only been executed once. This is because, after running an activity, Cadence saves the output in the workflow history and the next time that same step in the workflow is hit, simply returns the saved result from the previous execution without executing the actual activity code. Similarly, when the code is being watched and executed in real-time, there would be a ten-second pause after the first "about to sleep" but no pause the second time it comes around.

This saving of activity output to be replayed when workflow logic is re-executed is what enables Cadence to execute activities in the sequence defined by the logic of the workflow code while still being able to restart workflow execution whenever needed.

For this, to work, there are a couple of key rules to be aware of:

- Code within the workflow itself has to be entirely deterministic. To take a trivial example, you wouldn't want to use a standard random function as this would return a different value each time it was executed. More importantly, you shouldn't call any external services directly from workflow code that could fail including operations like reading and writing to files—anything like that should be done in activities.
- Code within a workflow should not modify a state external to the workflow class or, if

that is unavoidable, must do so in a manner that is idempotent. It is pretty easy to see from the example above that incrementing an external counter, for example, would result in different results depending on how Cadence chose to execute the workflow. Again, activities can be used for this type of requirement.

- If you want Cadence to retry failed activities, then relevant activity implementations should be idempotent as well (and have a retry policy specified).

Other Key Features

So far, only the bare fundamentals of Cadence have been covered. In addition to these fundamentals, Cadence offers many powerful features that are beyond the scope of this paper to cover in detail. Some key features to be aware of include:

- **Workflow signals:** allow external processes to send signals or messages to a running workflow. Workflows can wait for a specific signal to be received before proceeding.
- **Workflow queries:** allow external processes to query state from a running workflow.
- **Child workflows:** allows workflows to spawn other workflows. Among other uses, this is important for managing very long-running workflows with many steps as once a child workflow is complete there is no need to replay its full history when restarting the parent workflow.
- **Async activities:** these allow workflow execution to continue (and execute more activities) while a long-running activity completes. The workflow can later wait to confirm the completion of the activity.

The details of these and many more features can be found in the Cadence documentation.

Applying Cadence in Your Application Architecture

Hopefully, the explanations above have given you a conceptual understanding of the function and usefulness of Cadence. However, like any technology, it will be most successful and valuable when applied to the right problems with the right approach.

Next, you need to ask what indications you have of a business requirement where Cadence can be a valuable part of the technical solution? In general, your requirement should:

- Consist of **multiple steps**. If all you need to do is invoke a single service or even multiple services where the ordering is not important, then an approach such as putting a message on an Apache Kafka® topic and implementing Kafka consumers with your service logic is probably a simpler solution.

- Have **separate external interactions at the start and end of the process** (i.e. is asynchronous from the point of view of the application or user invoking the process). If all of your steps need to be implemented in a single, synchronous call, then it's probably simpler just to implement them directly in your code. While there are use cases where it is sensible to execute a Cadence workflow synchronously, these are more likely to occur in a batch process or child workflow execution rather than directly from user interaction (so the overall process is in effect asynchronous from a user point of view).
- **Need a high-level scale** and reliability or have **many suitable requirements** to implement. If you only have a single requirement that it is suitable for Cadence then it's unlikely to be worth the overhead of learning Cadence unless you have requirements for high level of scale and reliability. Alternative approaches such as managing your process state in the database you are already using are likely to be simpler to build and operate. However, build-your-own approaches are unlikely to achieve Cadence's ability to scale to millions of concurrent workflows and tens of thousands of events per second while delivering very high levels of reliability without substantial effort. In addition, even if you don't have high scale requirements, if you have many smaller-scale requirements that otherwise fit Cadence then it is likely to be worth the learning curve to gain the developer productivity benefits from the structured approach that Cadence provides.
- **Not require you to maintain workflow definitions in a business-friendly language.** Many workflows or decision engines allow you to directly maintain workflow definitions in a business-friendly format, even plain English. While it is possible to build execution engines for these languages on top of Cadence, Cadence does not directly provide the ability to use such languages to specify workflows. If direct maintenance of workflow logic by business analysts or business users is a requirement, then Cadence will be, at best, part of your solution.

Once you've determined that Cadence is a fit for your use case, be prepared that, like any sophisticated new technology, you will need to be deliberate in your planning to maximize your chances of success. Some key factors that we recommend considering are:

- **Be prepared for a new programming model:** Using Cadence does require you to structure your application logic in a specific manner and adhere to certain rules. While these rules are not overly restrictive once you are used to them, you need to get familiar with concepts like what logic needs to be split out into an activity and avoid introducing non-determinism into your workflow code. While adapting to these concepts may be an overhead initially, once you are familiar with them, they give you a solid conceptual framework that can improve your productivity in many circumstances.
- **Start small and grow:** it should be clear even from the overview in this paper that there is quite a lot to learn when approaching your first task with Cadence. As a result, organizations looking to adopt Cadence are often best off undertaking this learning through a single project initially and then using that project to seed the learning of other teams that have a use for Cadence.

- **Be part of the community:** Cadence is a true open source project committed to the community. Whether you are learning to implement your first workflow or dealing with an issue that only appears at the highest levels of scale, there are people and resources to assist you in the Cadence community. Don't be afraid to reach out and ask questions, and once you start to find your feet, join in and help someone else or contribute back to the project when you see the opportunity.
- **Don't forget operations:** This article has covered the dev side of things (and barely scratched the surface of that). Operations is another whole perspective to understand before successfully running Cadence in production. For Cadence, this includes specific topics such as sharding and global domains as well as the standard topics you'd expect for any system such as deployment topology and configuration, sizing, monitoring, backup and restore, etc. Of course, Instaclustr's managed platform is designed to take care of many of these considerations for you, delivering you a fully production-ready and supported infrastructure in minutes rather than months.

Conclusion

Cadence is proving to be a powerful tool for reducing development and operations effort for many requirements. Its value and reliability at scale have been proven in several of the world's leading technology companies. Developers need to understand Cadence's capabilities so they can easily configure, deploy, and run Cadence with ease.

By understanding the benefits outlined in this whitepaper, developers and architects alike can determine if Cadence is the right solution to enable their applications.

About Instaclustr

Instaclustr helps organizations deliver applications at scale through its managed platform for open source technologies such as **Apache Cassandra®**, **Apache Kafka®**, **Apache Spark™**, **Redis™**, **OpenSearch®**, **PostgreSQL®**, and **Cadence®**.

Instaclustr combines a complete data infrastructure environment with hands-on technology expertise to ensure ongoing performance and optimization. By removing the infrastructure complexity, we enable companies to focus internal development and operational resources on building cutting edge customer-facing applications at lower cost. Instaclustr customers include some of the largest and most innovative Fortune 500 companies.