Volt Active Data Active(SP)

Stream Data Processing

Volt Active Data Active(SP): Stream Data Processing

Table of Contents

List of Figures

Chapter 1. Active(SP) Stream Data Processing

Volt Active(SP) is a cloud-native framework for building streaming data pipelines. There are lots of streaming data products out there. What makes Active(SP) stand out is its balance of flexibility, scalability, and reliability. Seamless integration with Apache Kafka and Volt Active Data — an ACID database for high performance applications requiring maximum throughput and durability — allow Active(SP) to support complex processing of streams at maximum speed while minimizing your total cost of ownership.

The best way to understand what Active(SP) can do is to try it yourself. There are two sample pipelines you can build and run on your own infrastructure to see how easily and flexibly it integrates with Kafka and Volt Active Data. See the instructions now.

Or if you want to learn more, read about how Active(SP) works, the architecture that allows it to support complex business requirements while remaining both flexible and robust at runtime.

Finally, if you like diving in and getting to the heart of the matter, see the sections on designing, building, and running your own pipeline from scratch.

- Chapter 2, *See Active(SP) in Action*
- Chapter 3, *How Active(SP) Stream Processing Works*
- Chapter 4, *Designing Your Active(SP) Pipeline*

Chapter 2. See Active(SP) in Action

The best way to understand what Active(SP) does is to see it in action. The Active(SP) quick start implements two pipelines that demonstrate:

- Streaming data to Kafka
- Streaming data from Kafka to Volt Active Data

The source code for the sample is simple, easy to read, and useful both as a demonstration and as a template for building your own pipelines.

But let's get started. The steps for running the sample pipelines are:

1. Make sure you have the necessary environment setup
2. Download the sample sources
3. Build the sample application and post it to a Docker repository
4. Run the pipelines in the cloud

2.1. What You Will Need

To run the quick start pipelines you will need an environment to build the sample from Java source files into a Docker image, a cloud environment (such as Kubernetes) with access to a Docker repository, an Apache Kafka server and a Volt Active Data database cluster. The build process requires access to the Volt Active Data software repositories (see your Volt sales representative for more information) and the following software:

- Java SDK version 17 or greater
- Maven
- Docker

The recommended runtime environment includes:

- Docker
- Kubernetes
- Helm
- Kafka
- Volt Active Data V14.0 or later
- A Volt Active Data license including Active(SP)

2.2. Downloading the Sample Application

The quick start is available from the Volt Active Data repositories, including a Maven file for downloading and structuring the destination folders on your local system. First, set default to the directory where you want to install the sample source files, then issue the following shell command:

```
$ mvn archetype:generate \
 -DarchetypeGroupId=org.voltdb \
 -DarchetypeArtifactId=volt-stream-maven-quickstart \
 -DarchetypeVersion=1.0.0
```

The maven script will first ask you for the group ID and artifact ID. These represent the package prefix (such as *org.acme*) and the name for the sample directory, respectively. In the following examples we will use *org.acme* as the package prefix and *sample* as the sample name. The script then asks a series of questions where you can take the default answer. For example:

```
$ mvn archetype:generate \
>  -DarchetypeGroupId=org.voltdb \
>  -DarchetypeArtifactId=volt-stream-maven-quickstart \
>  -DarchetypeVersion=1.0.0

     [ . . . ]

Define value for property 'groupId': org.acme
Define value for property 'artifactId': sample
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.acme: :
Confirm properties configuration:
kafka-bootstrap-servers: REPLACE-ME-IN-PIPELINE-YAML
voltdb-servers: REPLACE-ME-IN-PIPELINE-YAML
voltsp-api-version: 1.0.0
groupId: org.acme
artifactId: sample
version: 1.0-SNAPSHOT
package: org.acme
 Y: :

     [ . . . ]

[INFO] ------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------
```

What the script does is create a subdirectory in the current folder named after the artifact ID. Within that directory tree are the Java source files for building the pipeline template and resources needed to run the pipelines. For example, if you chose *sample* as your artifact ID and *org.acme* as the group ID::

- `sample/` — contains a README and the Maven pom.xml file for building the sample pipelines
- `sample/src/main/java/org/acme/` — contains the Java source files defining the pipelines
- `sample/src/main/resources` — contains assets, including Helm YAML files and SQL schema, needed to run the pipelines

2.3. Building the Sample Application

Once you download the sample source files, you can build the pipeline templates using Maven. Set default to the *sample* directory created in the previous step and issue the `mvn clean package` command:

```
$ cd sample
```

```
$ mvn clean package
```

Next you can load the completed pipeline templates into a Docker repository so they are available for use in your cloud environment. The easiest way to do this, since they are resources you will need to reference both now and when running the pipelines, is to define a few helpful environment variables for assets that are unique to you. These include the name of the Docker repository you will use and the Volt license file required to run the pipelines. For example:

```
$ export MY_DOCKER_REPO=johnqpublic/projects
$ export MY_VOLT_LICENSE=$HOME/licenses/volt-license.xml
```

Now you can issue the docker commands to build an image and push it to your repository:

```
$ docker build                                          \
      --platform="linux/amd64"                          \
      -t ${MY_DOCKER_REPO}:voltsp-quickstart--latest \
      -f src/main/resources/Dockerfile .
$ docker push ${MY_DOCKER_REPO}:activesp-quickstart--latest
```

2.4. Running the Sample Pipelines

You are almost ready to run the sample pipelines. The last step before you can run the pipelines is to set up the infrastructure they need as input and output. That is, identify an available Kafka server and/or a Volt Active Data database, depending on which pipeline you run. For Kafka, if the server does not allow automatic creation of topics, you may need to create the *greetings* topic manually. For VoltDB you will need a server that has the necessary table defined. The easiest way to to that is initialize and start the database and apply the DDL in the `src/main/resources` folder:

```
$ voltdb init -f -D ~/db/sample
$ voltdb start -D ~/db/sample &
$ sqlcmd < src/main/resources/ddl.sql
```

Once you've identified the data source and destination, you can update the Helm properties files for the two pipelines to match your selections. For example, if you have a Kafka broker running at *kafka.acme.org* and a VoltDB database running on *volt.acme.org*, you can insert those addresses into the YAML files `kafka-to-volt-pipeline.yaml` and `random-to-kafka-pipeline.yaml` in `src/main/resources`. For example, `kafka-to-volt-pipeline.yaml` might look like this (changes highlighted):

```
replicaCount: 1

resources:
  limits:
    cpu: 2
    memory: 2G
  requests:
    cpu: 2
    memory: 2G

streaming:
  javaProperties: >
    -Dvoltsd.pipeline=org.acme.KafkaToVoltPipeline
    -Dvoltdb.server=volt.acme.org
    -Dkafka.consumer.group=1
```

```
    -Dkafka.topic=greetings
    -Dkafka.bootstrap.servers=kafka.acme.org
```

Once you have set up the necessary infrastructure and edited the YAML files, you are ready to start the pipelines. You start the pipelines using Helm and specifying a name for the pipeline, the Active(SP) chart (*voltdb/voltsp*), your license, your Docker registry, your image name, and a pointer to the YAML properties file. If you have not defined environment variables for the Docker repository and license file yet, now is a good time to do that. For example:

```
$ export MY_DOCKER_REPO=johnqpublic/projects
$ export MY_VOLT_LICENSE=$HOME/licenses/volt-license.xml

$ helm install pipeline1 voltdb/voltsp                \
  --set-file streaming.licenseXMLFile=${MY_VOLT_LICENSE}  \
  --set image.repository=${MY_DOCKER_REPO}                \
  --set image.tag=activesp-quickstart--latest             \
  --values test/src/main/resources/random-to-kafka-pipeline.yaml
```

The Helm command starts the Kubernetes pod and starts pushing random hello statements into the Kafka topic. You can then start the second pipeline, which pulls the statements from the topic and inserts them into the GREETINGS table in the database:

```
$ helm install pipeline2 voltdb/voltsp                \
  --set-file streaming.licenseXMLFile=${MY_VOLT_LICENSE}  \
  --set image.repository=${MY_DOCKER_REPO}                \
  --set image.tag=activesp-quickstart--latest             \
  --values test/src/main/resources/kafka-to-volt-pipeline.yaml
```

Once the pipelines are running you can see the results by monitoring the greetings topic in Kafka or querying the GREETINGS table in VoltDB:

```
$ sqlcmd --servers=volt.acme.org
> select count(*) from greetings;
```

You can also use Prometheus and Grafana to monitor your pipelines, including a custom Grafana dashboard. See Section 4.6, “Monitoring Your Pipeline” for more information.

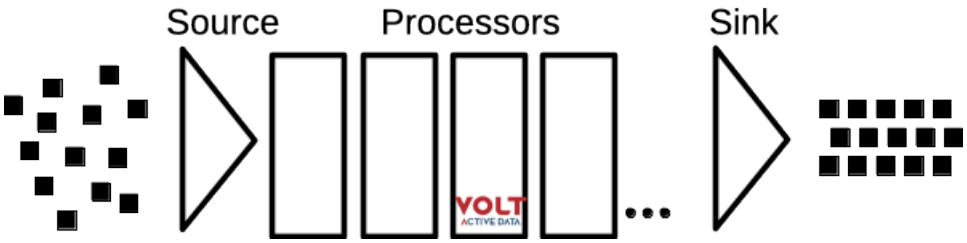
Chapter 3. How Active(SP) Stream Processing Works

Stream data processing has become a critical component of business operations. The exponential growth of available data and the pressure to act on information in real time have made traditional computing approaches obsolete. It is no longer sufficient to gather data and post process it to determine what actions to take. Now businesses need to operate on the data *in flight* to filter, format, validate, measure, and respond to events in a timely manner.

And for simple operations this works. Many operations, like filtering data based on fixed rules or converting from one format to another, can be performed at speed. However, the Achilles heel of stream data processing is the fact that many operations still require access to up-to-date and entrusted information such as customer accounts, inventory levels, and resource availability. These stateful operations, if performed against a traditional SQL database, incur the same latency to which previous centralized operations were susceptible. Which is where Active(SP) comes in.

By integrating stream data processing with Volt Active Data — an ACID database designed to maximize throughput without sacrificing consistency, durability, or availability — Active(SP) makes it possible to combine both stateless and stateful processing in flight and at speed.

Figure 3.1. Active(SP) Architecture



The Active(SP) architecture consists of three primary parts: sources, sinks, and processors. And the Domain Specific Language you use to define Active(SP) pipelines mirror the exact same structure, letting you define the source, one or more processors, and a sink:

```
stream
   source
       processor
       processor
       processor
        [ ... ]
   sink
```

Where the processors can be any combination of stateless or stateful operations, with Volt Active Data providing real time access to reference data that can be used to verify, authenticate, authorize, or in other ways validate and enhance the data as it passes.

The advantages the Active(SP) architecture offers are:

- **Cloud Native** — Active(SP) pipelines are designed from the ground up to run in the cloud. It is also self contained and does not require any additional infrastructure (such as resource managers, schedulers, or the like). This allows for easy setup, scaling, and management.

- **Apache Kafka and Volt Active Data integration** — Kafka is supported out of the box as a data source and both Kafka and Volt Active Data are supported as sinks for the pipeline, so that setting up the initial pipeline template is trivial.
- **Complex business logic** — Partitioned procedures in Volt Active Data can be used to incorporate complex, stateful operations on the data without sacrificing latency.
- **Flexibility** — The pipelines are designed as templates, using placeholders for key resources such as server addresses and topic names, so that different pipelines can be created from the same template by identifying different resources in the properties at runtime.
- **Scalability** — The pipelines themselves can be scaled at runtime completely separately from the resources, such as Kafka servers or Volt Active Data cluster nodes allowing you to optimize computing resources to match actual needs.

Chapter 4. Designing Your Active(SP) Pipeline

Writing your own Active(SP) pipeline is simple. Each pipeline consists of a data *source*, one or more *processors* that operate on the data, and ends by sending the resulting record to a data target or *sink*. You describe the structure of your pipeline using a Domain Specific Language (DSL), written in Java. The DSL includes classes and methods that define the structure of your pipeline and can be compiled into the actual runtime code.

For Active(SP) the DSL describes the three primary components of the pipeline: the source, the processors, and the sink. Like so:

```
stream
   .consumeFromSource( [ . . . ] )
       .processWith( [ . . . ] )
       .processWith( [ . . . ] )
       .processWith( [ . . . ] )
           [ . . . ]
   .terminateWithSink( [ . . . ] )
```

The following sections describe:

- How to start your pipeline project
- Defining the source and destination for the pipeline
- Defining the business operations on the data (the processors)
- Building and running the pipeline

4.1. Setting Up Your Development Environment

You could start your Active(SP) pipeline project from scratch, setting up the necessary folder structure, creating Java source files and defining the Maven dependencies and Helm properties by hand. But it is much easier to start with a template, and the quick start example described in Chapter 2, *See Active(SP) in Action* can be used for just that. Follow the instructions for downloading the quick start, specifying your organization's ID as the group ID and your pipeline name as the artifact ID to create your template. Let's say you are creating a pipeline called *mydatapipe*, the resulting template might have the following folder structure:

```
mydatapipe
- src
  - main
    - java
      - org
        - acme
    - resources
  - test
       ...
```

The following are the key files you will use for creating your own pipeline from the quick start sample:

- `mydatapipe/pom.xml` — The Maven project file for building the pipeline
- `mydatapipe/src/main/java/{your-org}/*.java` — Pipeline definition files you can revise and reuse to match your pipeline's source, sink, and processors.
- `mydatapipe/src/main/resources/*.yaml` — Helm property files you can use to describe the data resources the pipeline requires, such as Kafka streams, Volt Active Data databases, and their properties.

4.2. Choosing the Source and Destination for the Pipeline

The first thing the pipeline needs is a data source. This is defined in the .consumeFromSource method. Active(SP) supports Apache Kafka as a data source out of the box. You specify the type of source with the Sources.kafka() class, which has methods for setting the associated properties. For example, the following code sample creates a data source from a Kafka topic:

```
.consumeFromSource(
    Sources.kafka()
       .withBootstrapServers("${kafka.bootstrap.servers}")
       .withTopicNames("${kafka.topic}")
       .withGroupId("${kafka.consumer.group}")
       .withStartingOffset(KafkaStartingOffset.EARLIEST)
       .withKeyDeserializer(LongDeserializer.class)
       .withValueDeserializer(StringDeserializer.class)
```

Note that the source definition uses placeholders (property names enclosed in braces and prefixed with a dollar sign) in place of actual servers addresses, topic names, and group IDs. This way the pipeline acts a template, defining the actions to take, while the actual sources and destinations can be defined at runtime by setting property values in the Helm YAML files (see Section 4.5, “Running Your Pipeline”).

Similarly, Active(SP) supports two standard data destinations, or sinks: Kafka and Volt Active Data. You define the sink in much the same way you define the source. The following code samples define a Kafka topic and a Volt stored procedure as the final destination for the stream:

```
.terminateWithSink(
    Sinks.kafka()
       .withBootstrapServers("${kafka.bootstrap.servers}")
       .withTopicName("${kafka.topic}")
       .withValueSerializer(StringSerializer.class)
       .withKeyExtractor(String::hashCode, IntegerSerializer.class)

.terminateWithSink(
    Sinks.volt().procedureCall()
        .withHostAndStandardPort("${voltdb.host}")
        .withProcedureName("${voltdb.procedure}")
```

4.3. Defining the Business Operations (Processors)

The source and sink define where the stream starts and ends. But it is the processors in the middle that do the real work of transforming the data into actionable business decisions. The processors are executed sequentially as defined in the pipeline definition and can be any function or method you choose. For example, the quick start random-too-kafka-pipeline uses an inline function to convert the text to uppercase:

```
.processWith(
        string -> string.toUpperCase()
)
```

For more complex processing, you can include the processor source code separately elsewhere in the pipeline definition file or define and build it as a separate class. But the key advantage of Active(SP) is that it provides built-in methods for integrating common processors using Kafka and Volt Active Data. For example, invoking a Volt stored procedure as a processor is a simple matter of identifying the procedure name and database server:

```
.processWith(VoltFunctions.<Object[],
    VoltTable>procedureCall("CountByUser")
       .withHostAndPort("${voltdb.host}", "${voltdb.port}"))
```

4.4. Building and Loading Your Pipeline

Once you complete the pipeline definition you are ready to build and load the pipeline into your Docker repository. You can use the Maven build file created when you downloaded the sample as a template without any modifications:

```
$ mvn clean package
```

Similarly, you can build and load the Docker file using the same commands as before. Be sure you have the defined the environment variable pointing to your Docker repository. Give the docker image a meaningful name. The following example uses *mypipe--latest* as the image name:

```
$ export MY_DOCKER_REPO=johnqpublic/projects
$ export MY_VOLT_LICENSE=$HOME/licenses/volt-license.xml
$ docker build                                          \
      --platform="linux/amd64"                          \
      -t ${MY_DOCKER_REPO}:mypipe--latest \
      -f src/main/resources/Dockerfile .
$ docker push ${MY_DOCKER_REPO}:mypipe--latest
```

4.5. Running Your Pipeline

Once you have prepared and loaded your pipeline template, you are ready to wrap up the final details before running the pipeline. This includes specifying the runtime value for any placeholders you use in the pipeline definition. For example, if you are using a Kafka topic as a source and Kafka or Volt Active Data as the sink, you will need to identify the servers, topics, and/or table names to use.

In the preceding code examples, the pipeline definition used the placeholders *voltdb.host*, *voltdb.port*, and *voltdb.procedure* for Volt Active Data assets. To fill in these placeholders, you edit the YAML properties file for your pipeline. If you used the quick start sample as a template, this means you can rename one of the YAML files (in `src/main/resources/`) with a meaningful name and edit it to fill in the appropriate values for the placeholders. You put these placeholder assignments in the `streaming.javaProperties` property. For example:

```
streaming:
  javaProperties: >
    -Dvoltdb.host=volt.acme.org
    -Dvoltdb.port=21212
    -Dvoltdb.procedure=MYDATA.insert
```

You use the same process for assigning values to any Kafka or application-specific placeholders your pipeline definition uses. For example:

```
streaming:
  javaProperties: >
    -Dvoltdb.host=volt.acme.org
    -Dvoltdb.port=21212
    -Dvoltdb.procedure=MYDATA.insert
    -Dkafka.bootstrap.servers=kafa.acme.org
    -Dkafka.topic=mydata
    -Dkafka.consumer.group=42
```

Now you are ready to run your pipeline. Use the **helm install** command to start the pipeline, specifying *voltdb/voltsp* as the chart and your edited YAML as the properties file. (If this is your first time running a pipeline, it is a good idea to issue a **helm repo update** command first to make sure you have access to the latest charts.) You will also need to include your volt license file:

```
$ export MY_DOCKER_REPO=johnqpublic/projects
$ export MY_VOLT_LICENSE=$HOME/licenses/volt-license.xml

$ helm install mydatapipe voltdb/voltsp                    \
  --set-file streaming.licenseXMLFile=${MY_VOLT_LICENSE}   \
  --set image.repository=${MY_DOCKER_REPO}                 \
  --set image.tag=mypipe--latest                           \
  --values test/src/main/resources/mydatapipe.yaml
```

Once you start the pipeline you can use the **kubectl get pods** to verify the processes have started. If there are any issues you can use **kubectl logs {*pod-id}*** to get details on what is happening.

4.6. Monitoring Your Pipeline

Once your pipeline is running, the natural next question is "how is it doing?" The nature of streaming pipelines is that they quietly just do their work; problems only come to light if they start to slow down the flow of data. Ideally, you can detect issues before they impact the stream itself. To help you do this, Active(SP) has metrics reporting built into the framework. All you need to do it turn it on.

If you set the Helm property `monitoring.prometheus.enabled` to *true* when starting the pipeline, it becomes a Prometheus client, reporting metrics that integrate with your existing Prometheus and Grafana infrastructure. For example:

```
$ helm install mydatapipe voltdb/voltsp                    \
  --set-file streaming.licenseXMLFile=${MY_VOLT_LICENSE}   \
  --set image.repository=${MY_DOCKER_REPO}                 \
  --set image.tag=mypipe--latest                           \
  --values test/src/main/resources/mydatapipe.yaml         \
  --set monitoring.prometheus.enabled=true
```

If you do not have an existing Prometheus infrastructure or if you simply want to quickly evaluate the pipeline performance without having to design your own dashboard, Volt provides a package containing Prometheus, Grafana, and a custom Grafana dashboard for reporting on Active(SP) pipelines. To start the management console with your pipeline, set `management-console.enabled` to *true* when you start the pipeline:

```
$ helm install mydatapipe voltdb/voltsp                    \
  --set-file streaming.licenseXMLFile=${MY_VOLT_LICENSE}   \
```

```
  --set image.repository=${MY_DOCKER_REPO}                \
  --set image.tag=mypipe--latest                          \
  --values test/src/main/resources/mydatapipe.yaml        \
  --set monitoring.prometheus.enabled=true                \
  --set management-console.enabled=true
```

Note that one copy of the management console can report on multiple pipelines. So you only need to start it with one of the pipelines. Alternatively you can leave it out of the pipeline startup entirely and start the management console separately with its own **helm install** command:

```
$ helm install mydataconsole voltdb/volt-stream/charts/management-console
```