[Overview 2](#_Toc381371382)

[What is π 2](#_Toc381371383)

[Monte Carlo method 2](#_Toc381371384)

[Pre-requisites 3](#_Toc381371385)

[Configuration 3](#_Toc381371386)

[Algorithm 3](#_Toc381371387)

[Design 3](#_Toc381371388)

[DAG 4](#_Toc381371389)

[Operators 4](#_Toc381371390)

[RandomEventGenerator : 4](#_Toc381371391)

[SquareCalculus: 4](#_Toc381371392)

[AbstractAggregator: 4](#_Toc381371393)

[Sigma: 5](#_Toc381371394)

[LogicalCompareToConstant : 5](#_Toc381371395)

[Counter: 5](#_Toc381371396)

[Division: 5](#_Toc381371397)

[MultiplyByConstant: 5](#_Toc381371398)

[RunningAverage: 6](#_Toc381371399)

[Console: 6](#_Toc381371400)

[Functionality 6](#_Toc381371401)

[Launching the application 7](#_Toc381371402)

[Monitoring the application 8](#_Toc381371403)

[Operations 13](#_Toc381371404)

[Stopping the application 16](#_Toc381371405)

[Conclusions 17](#_Toc381371406)

Application ‘Pi Calculator’

# Overview

This ‘π’ calculator is one of the simplest applications, to understand the DataTorrent platform. This application computes value of ‘π’ using – ‘Monte Carlo π estimation formula’.

## What is π

‘π’ is a numerical constant that represents the ratio of a circle's circumference to its diameter. It is an irrational number i.e. it has an infinite number of non-repeating decimal digits after the decimal point.

Another way to define ‘π’ is – it is the area of a unit circle. To understand this we need to know that the area of a circle is (π \* r2) and that a unit circle is simply a circle whose radius is equal to 1.

Consider a circle of radius ‘*r*’ inscribed within a square with a side *2r*. The diameter of the circle is equal to the width of the square. The square, hence, has length and width equal to the diameter of the circle (or twice the radius, *2r*).

Area of the circle:



Area of the surrounding square:



If we compare the area of the circle with the area of the square, we can form a ratio:



Rearranging to solve for pi:



In other words, pi is equal to the ratio of the area of an inscribed circle to the area of its outer square, multiplied by four. Our estimate for pi will use the equation above.

## Monte Carlo method

A Monte Carlo method is a mathematical model that relies on chance or repeated random behavior in order to determine a solution to a problem.

Monte Carlo methods can be thought of as statistical simulation methods that utilize sequences of random numbers to perform the simulation.

The name "Monte Carlo'' was coined by [Nicholas Constantine Metropolis](http://scienceworld.wolfram.com/biography/Metropolis.html) (1915-1999) and inspired by [Stanslaw Ulam](http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Ulam.html) (1909-1986), because of the similarity of statistical simulation to games of chance, and because Monte Carlo is a center for gambling and games of chance.

In a typical process one computes the number of points in a circle C that lies inside square S.  The ratio of the number of points that fall inside C to the total number of points tried is an estimation of the ratio of the two areas. The quality of the estimate depends on the number of points used, with more points leading to a more accurate value.

* C is the circle with area Ac
* S is a square with area As
* π = 4 \* (Ac/As)

A simple Monte Carlo simulation to approximate the value of ‘π’ involves randomly selecting points (x, y) in the unit square and determining the ratio m/n, where

* ‘m’ is number of points that satisfy xi2 + yi2 <= 1,
* ‘n’ is the total number of points and
* 1 is the base.

The accuracy tends to increase with the size of sample. The advantage of Monte Carlo simulation is that we do not need any analytic study of the object. This is why Monte Carlo is used in many applied areas.

# Pre-requisites

There are no special pre-requisites needed to run the ‘Pi Calculator’ demo application on the DataTorrent platform.

# Configuration

There is no particular configuration that needs to be done to run the ‘Pi Calculator’ demo application on the DataTorrent platform.

# Algorithm

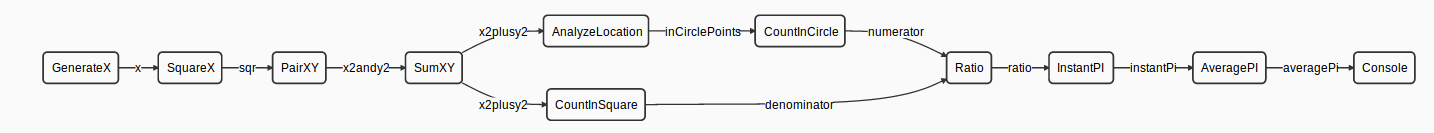
The ‘Pi Calculator’ application written for DataTorrent platform uses this algorithm:

1. Generates random distribution of points.
2. Checks whether the point falls inside the circle or not.
3. Counts the number of points that are being plotted.
4. Gets the ratio of the number of points in the circle to the number of point that fall in the square.
5. Calculates the instantaneous value of ‘π’ after each such point.
6. It then proceeds to calculate the average value of ‘π’
7. The application outputs the final value of ‘π’ on the console.

# Design

The DAG for the application is shown below followed by a description of the operators.

## DAG



This view can be seen in the ‘logicalDAG’ widget in the console.

This DAG is done to break out every computation into its own operator. The aim is to demo a much more complicated application than the standard pi demo.

The Pi-Calculator application described in this document demonstrates the ability of the DataTorrent platform to split the computation into individual operations, each one of which is performed by a separate operator. The platform then figures out the deployment of these operators.

This is very important concept to understand as a typical end user application would involve multiple operations and so has to be designed in the manner shown in the current document.

More distribution enables scale as more operators can be inserted for computations that are bottlenecks. Stream locality like thread local, container local can be used wherever needed. Parallel partitions can be used before streams get unified at "ratio" operators. It is also a demonstration on how low level library operators can be used to construct complicated compute models

Console operator is used for demo purpose. Results can be viewed in any external system. These include databases, web sockets, files, message bus, etc.

## Operators

The ‘Pi Calculator’ application involves multiple computational steps and the DataTorrent platform provides operators to perform each of these. Specifically, this application consists of the following operators –

|  |  |  |
| --- | --- | --- |
| **Type of Operator** | **Name of the operator in DAG** | **Library** |
| [RandomEventGenerator](#RandomEventGenerator) | GenerateX | lib/testbench/randomeventgenerator |
| [SquareCalculus](#SquareCalculus) | SquareX | lib/match/squarecalculus |
| [AbstractAggregator](#AbstractAggregator) | PairXY | lib/stream/abstractaggregator |
| [Sigma](#Sigma) | SumXY | lib/math/sigma |
| [LogicalCompareToConstant](#LogicalCompareToConstant) | AnalyzeLocation | lib/math/logicalcomparetoconstant |
| [Counter](#Counter) | CountInSquare, CountInCircle | lib/stream/counter |
| [Division](#Division) | Ratio | lib/math/division |
| [MultiplyByConstant](#MultiplyByConstant) | InstantPi | lib/math/multiplybyconstant |
| [RunningAverage](#RunningAverage) | AveragePi | lib/math/runningaverage |
| [Console](#Console) | Console | lib/io/consoleoutputoperator |

### [RandomEventGenerator](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/testbench/RandomEventGenerator.java) :

This is the first operator in the DAG and performs the role of simulating an input received by the DAG.

The name of the instance of this operator is ‘GenerateX’. It generates a random input between a range of 0 and 30,000

### [SquareCalculus](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/SquareCalculus.java):

This operator calculates the square of the input value it receives and outputs this squared value.

‘SquareX’ operator in the DAG performs the function of squaring. It takes integer data coming in from the ‘GenerateX’ and emits the squared value on its ‘integerResult’ port.

### [AbstractAggregator](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/stream/AbstractAggregator.java):

This operator aggregates tuples. It also uses collection size as an input. If size of collection is 0 then all tuples till end window are aggregated, otherwise collection is emitted as soon as collection size reaches to given size

This operator accepts incoming squares from the ‘SquareX’ operator. It is given a colection size of 2 which means that it collects 2 tuples and emits it on its output port. These 2 tuples represent the values of x2 and y2  which are used in determining if the point falls inside the circle or not. The instance of the AbstractAggregator viz. ‘PairXY’ emits these pairs on its output port – ‘pairoperator.output’.

### [Sigma](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/Sigma.java):

‘sumOperator’ is the instance of the Sigma operator and it accepts input on its ‘input’ port. It will add the incoming tuples and emit the result. In this instance, since it is receiving a square of the values as inut, it will in effect perform the operation (x2 + y2) and emits this result on its output port viz. ‘integerResult’.

### [LogicalCompareToConstant](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/LogicalCompareToConstant.java) :

This operator logically compares incoming tuple with a constant and emits the result on the appropriate ports.

Here, the constant is set to (30,000 \* 30,000). If the constant is equal to tuple, then the pair is emitted on equalTo, greaterThanEqualTo, and lessThanEqualTo ports. If the constant is less than tuple, then the pair is emitted on notEqualTo, lessThan and lessThanEqualTo ports. If the constant is greater than tuple, then the pair is emitted on notEqualTo, greaterThan and greaterThanEqualTo ports.

### [Counter](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/stream/Counter.java):

This operator counts the number of tuples delivered to it and emits the count.

There are 2 instances of this type of operator viz. CountInSquare and CountInCircle. The ‘CountInSquare’ takes its input from the ‘Sigma’ operator ‘SunXY’ and counts the number of points that lie inside the square. The count is emitted on its ‘output’ port and will for the denominator for the next step.

The instance ‘CountInCircle’ takes its input from the ‘AnalyzeLocation’ operator’s ‘greaterThan’ output port i.e. those tuples for which the comparison constant is greater than the location of the tuples. This means that the point lies inside the circle and hence it should be counted by the ‘CountInCircle’ operator. The count is emitted on its ‘output’ port and will form the numerator for the next step.

### [Division](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/Division.java):

‘Ratio’ is the instance of the ‘Division’ operator and it has 2 inputs viz. numerator and denominator.

The ‘CountInCircle’ forms the numerator and the ‘CountInSquare’ forms the denominator. A division operation is done on consecutive tuples on ports numerator and denominator. The operator is idempotent as the division is done in order, i.e. the first number on denominator port would divide the first number on the numerator port. The quotient (calculated in the current window) is passed as the output to the next operator’s input.

### [MultiplyByConstant](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/MultiplyByConstant.java):

The instance of this operator viz. ‘InstantPi’ multiplies the incoming tuple by a constant, in this case ‘4’. This constant is arrived at based on the formula being used to calculate ‘π’ in the Monte Carlo estimation technique –



The incoming input from the ‘Ratio’ operator is the ratio (Ac/As). Multiplying it by 4 will give us the value of ‘π’.

However, note that this is the instantaneous value of ‘π’. This instantaneous value of ‘π’ is passed on to the next operator from the output port viz ‘doubleProduct’.

### [RunningAverage](https://github.com/DataTorrent/Malhar/library/src/main/java/com/datatorrent/lib/math/RunningAverage.java):

This operator calculates the running average of the input numbers and emits it at the end of the window. This is an end of window operator. Note that the average is computed over application window.

This way, we get the average value of ‘π’ over the entire application window. This value of ‘π’ is then passed on to the final operator in the DAG which is the ‘Console’ operator.

### [Console](https://github.com/DataTorrent/Malhar/blob/master/library/src/main/java/com/datatorrent/lib/io/ConsoleOutputOperator.java):

The console operator displays the value of ‘π’ received on its console port.

#### Streams

|  |  |
| --- | --- |
| **Stream** | **Description** |
| x | Sends the generated integer value to be squared |
| sqr | Gives the squared value to the next operator |
| X2andy2 | Sends the aggregated tuples to be summed |
| X2plusy2 | Sends the tuples for comparison and to be counted |
| inCirclePoints | Tuples identified as falling inside the circle are sent to be counted |
| Numerator | Count is sent as numerator for ratio calculation |
| Denominator | Count is sent as denominator for ratio calculation |
| Ratio | Calculated ratio is sent to calculate instantaneous value of ‘π’ |
| instantPi | Calculated instantaneous value of ‘π’ is sent for averaging |
| averagePi | Averahe value of ‘π’ is sent to the console to be output |

# Functionality

The detailed functionality of each operator is covered above. Note that in this application, we do use operators which are statefull as well as some which are stateless.

|  |  |  |
| --- | --- | --- |
| **Type of Operator** | **Statefull** | **Partitionable** |
| RandomEventGenerator | - | Y |
| SquareCalculus | N | Y |
| AbstractAggregator | Y | Y |
| Sigma | Y | Y |
| LogicalCompareToConstant | N | Y |
| Counter | Y | Y |
| Division | N | Y |
| MultiplyByConstant | N | Y |
| RunningAverage | Y | N |
| Console | - | Y |

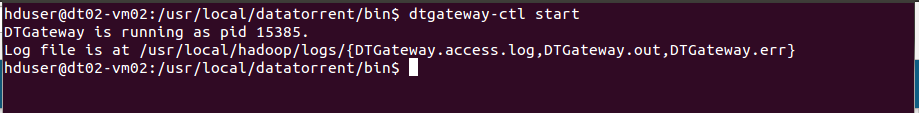
Note: The Console operator is partitionable and can be partitioned based on a key. In the current demo application, it is not being partitioned.

# Launching the application

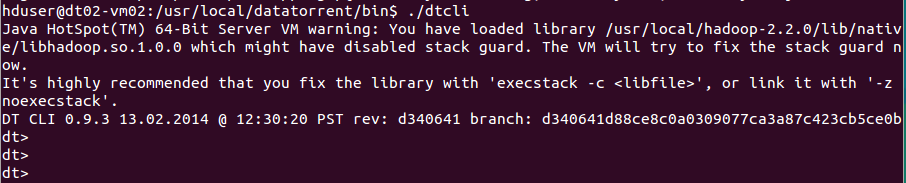
In order to be able to launch sample Demo Applications, please make sure you have completed following steps.

In case you have not started ‘dtGateway’, start the process as follows ...

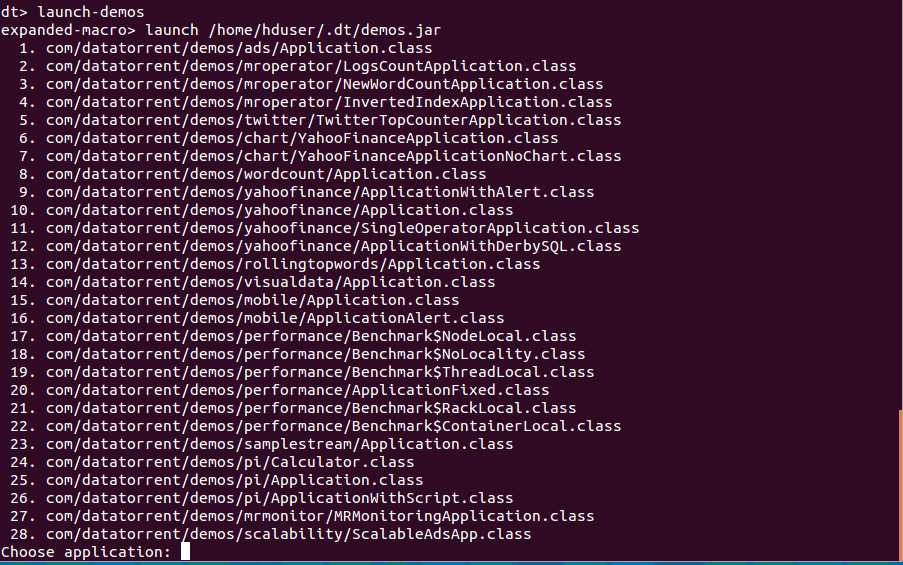
1. Start the DataTorrent Gateway process. This can be done from the command line as –



1. Launch DataTorrent Command Line Interface (dtcli)



1. Launch demo application Jar. (Here, we assume that the Hadoop services are up and running and the demos are being launched in a cluster mode.)



The user can now select and launch the application.

The jar is in ~/.dt/demos.jar and can also be launched as "launch ~/.dt/demos.jar

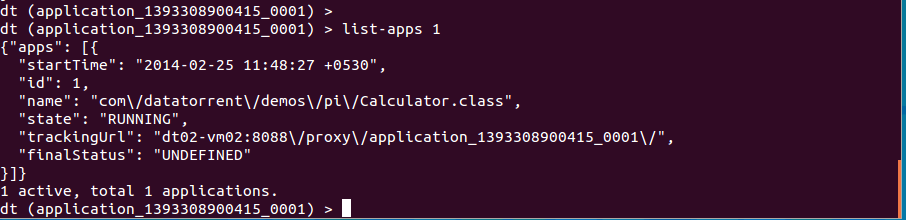
Please refer to DataTorrent Quick Start Guide for additional details.

# Monitoring the application

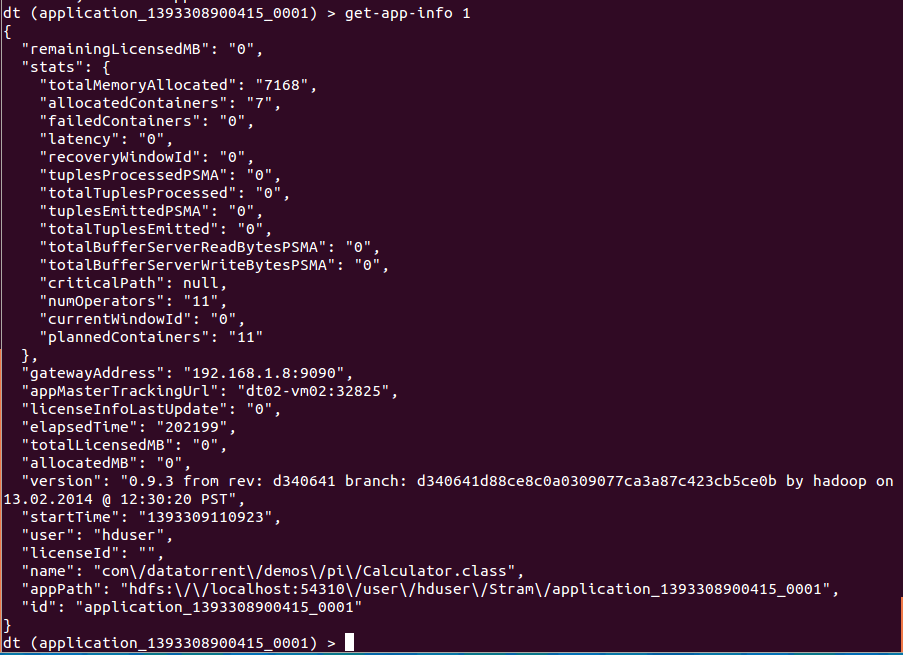
Once launched, the application can be monitored/managed from either the command line viz. ‘dtcli’ or the GUI viz. DataTorrent Gateway Console. The console and cli get data from same webservices layer. Following are some of the sample operations that are typically performed while monitoring the application –

1. List the application -

#### Using ‘dtcli’



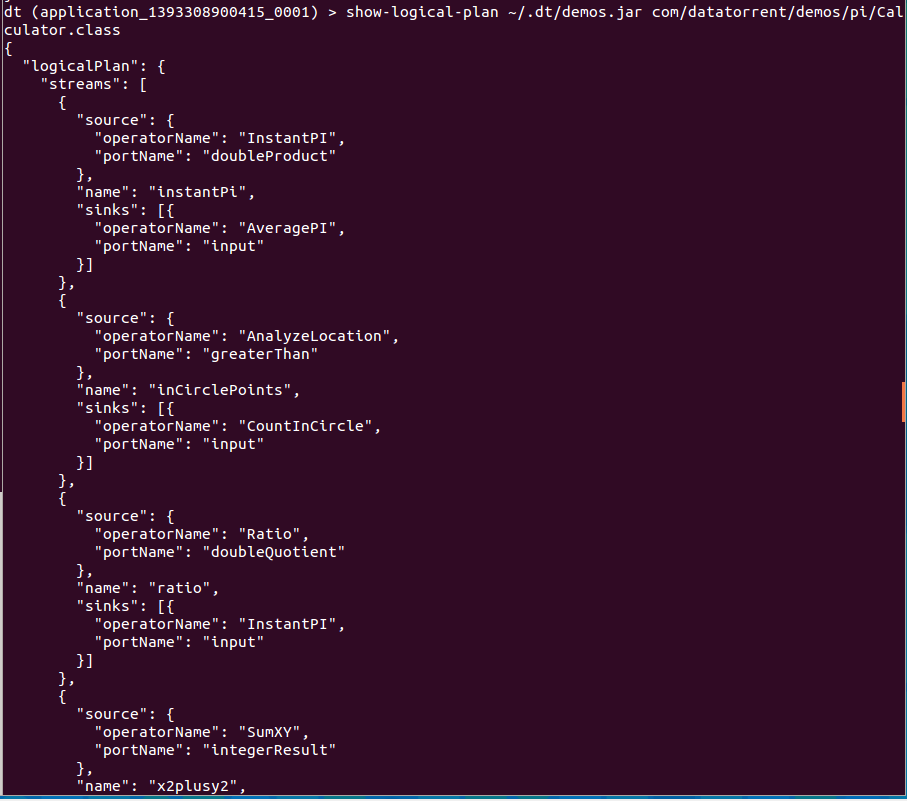
1. Get detailed information about a given application –



1. Connect to the application –

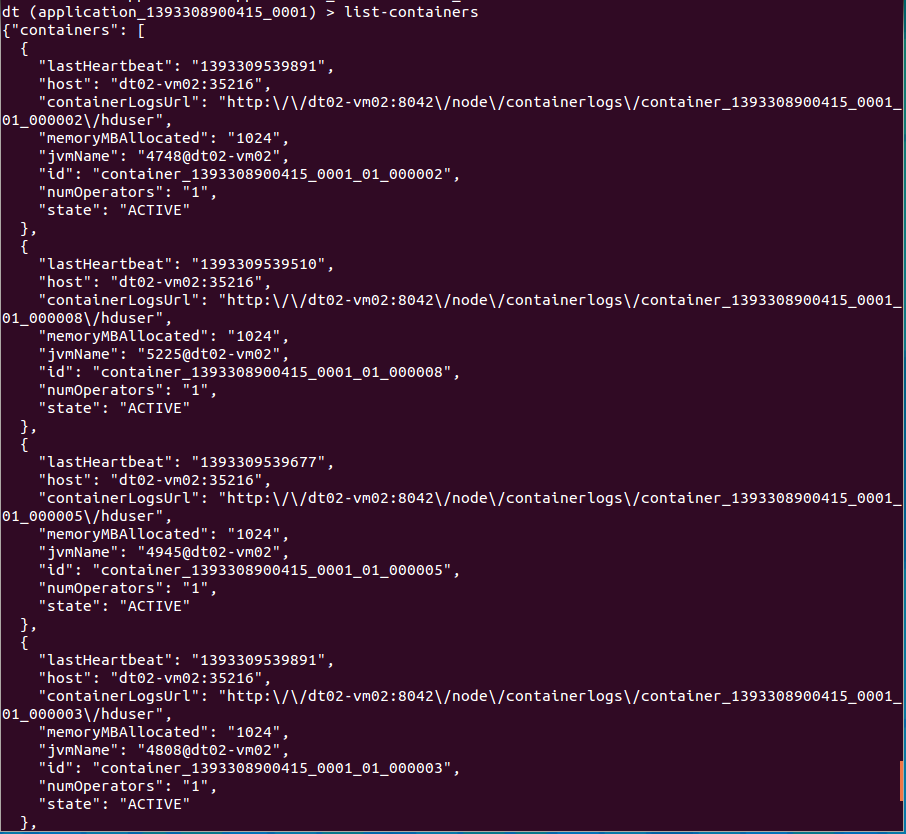


1. Show logical plan



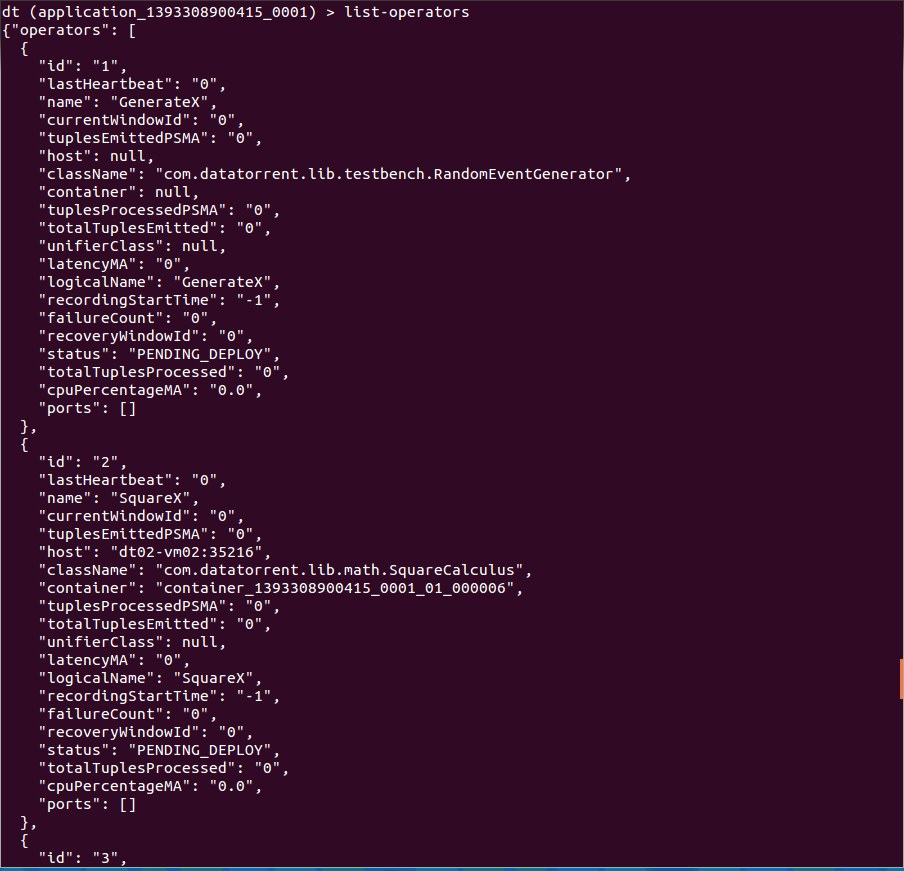
1. List the containers :

This lists the containers for the application that you are connected to, at this point in time.

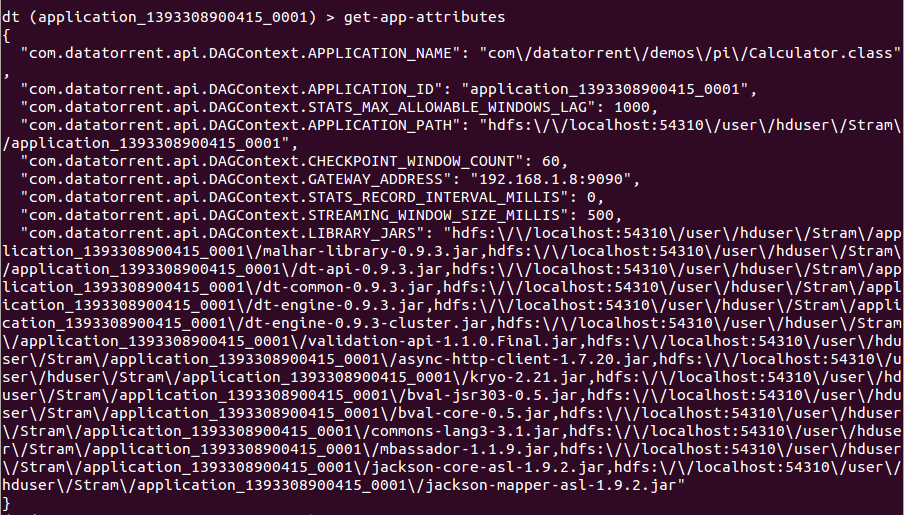


1. List all the operators –

(Note that the screenshot below captures only the first 2. Others are displayed in the same manned in order)



1. Get application attributes :

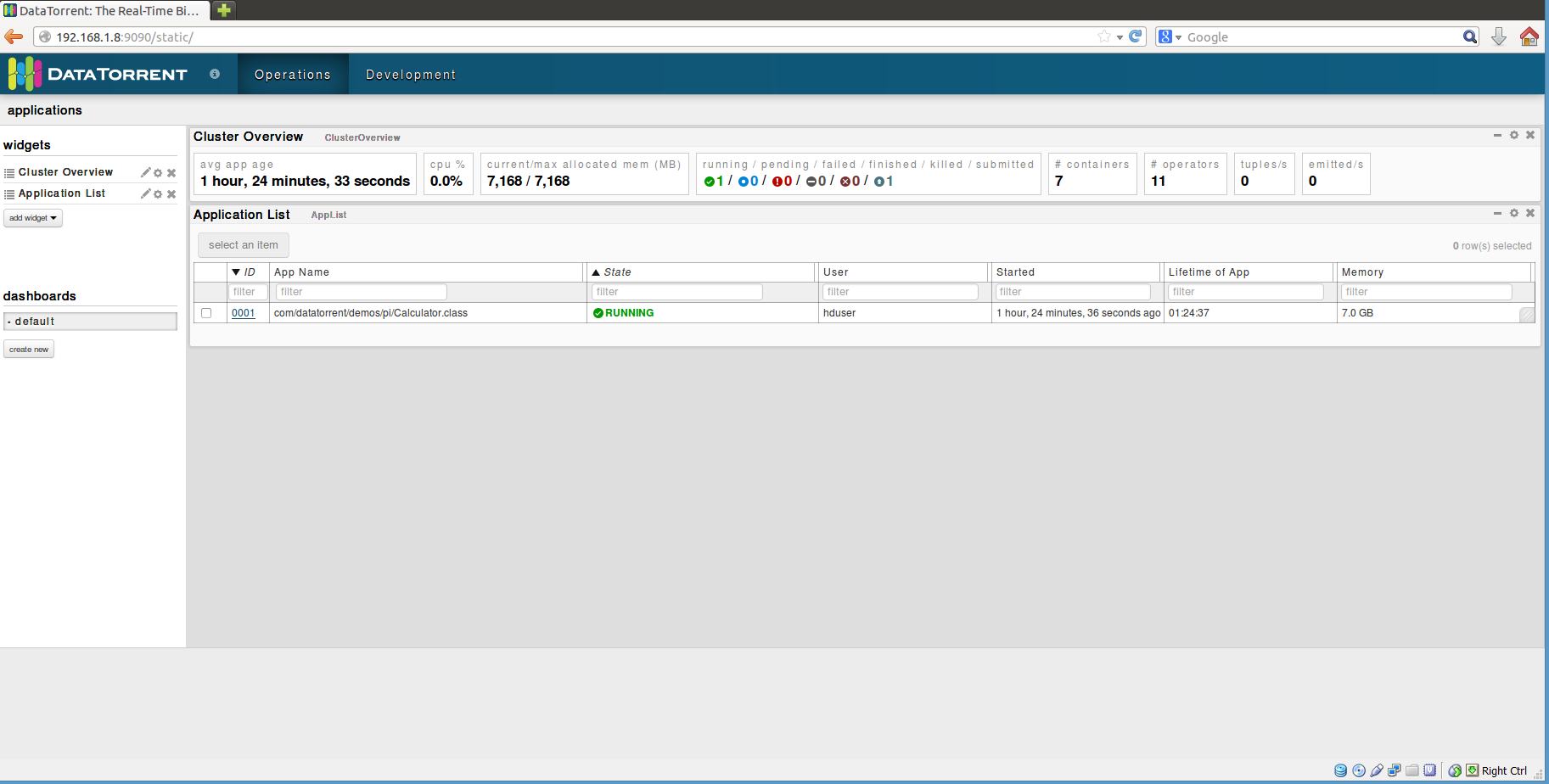


1. Similarly the user can use other dtcli to monitor/manage the application.

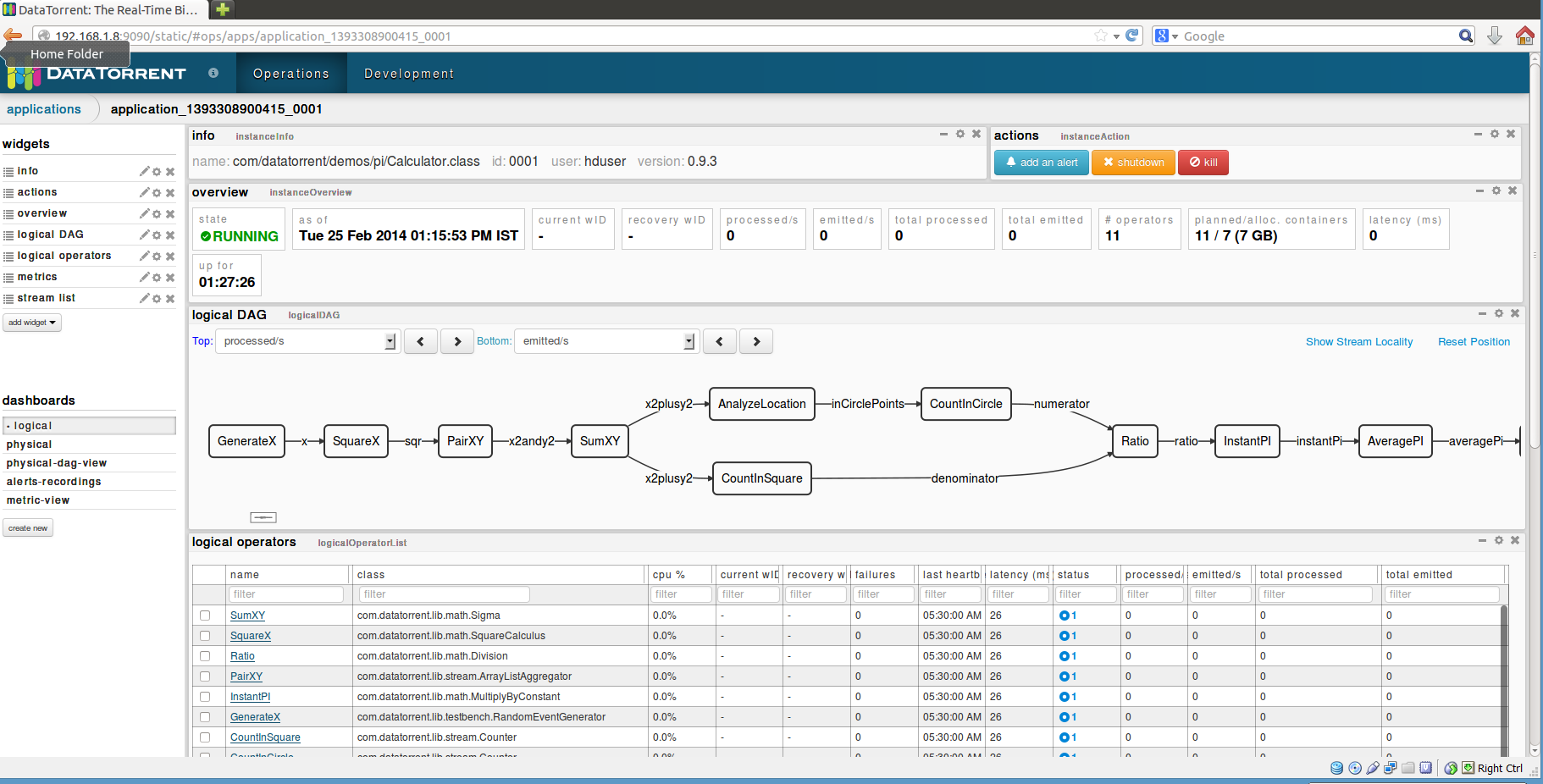
#### Using ‘Gateway Console’

# Operations

1. On the Gateway console, you will see the apps as below –



1. In order to get detailed about the application, click on ‘ID’ for the application and you will get to the screen below –

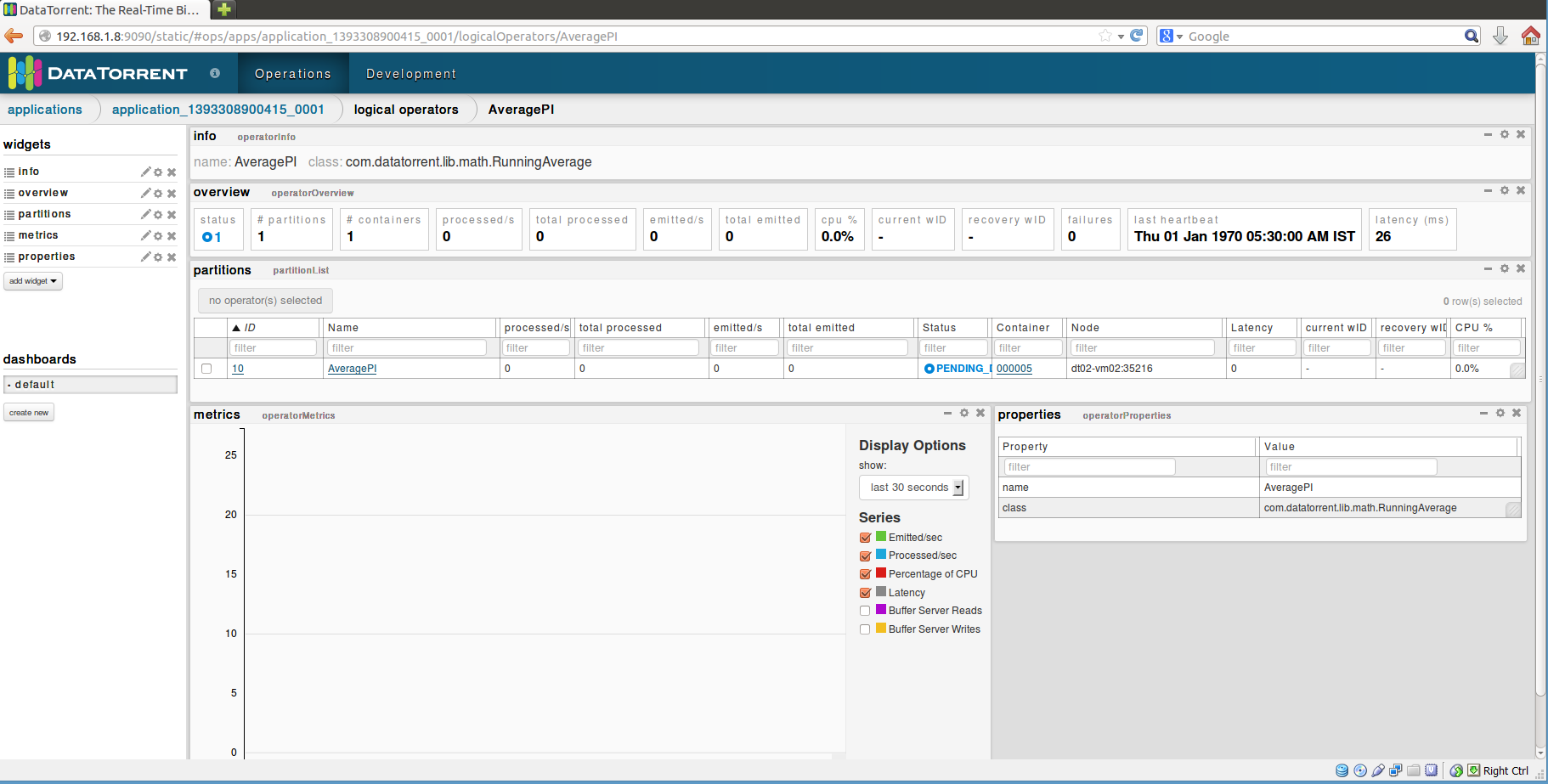


As you can see, it shows you a lot of details about the application, such as –

* Overview
* Logical DAG
* Logical operators
* Streams (not captured in the screenshot, but can be seen whe you scroll downwards)
* Various mertices, etc.

Additionally, you can select additional widgets as well as rearrange the widgets as per your wish.

1. Clicking on any of the operators will give you the details for that particular operator. For example, the screenshot below is for the AveragePI operator.

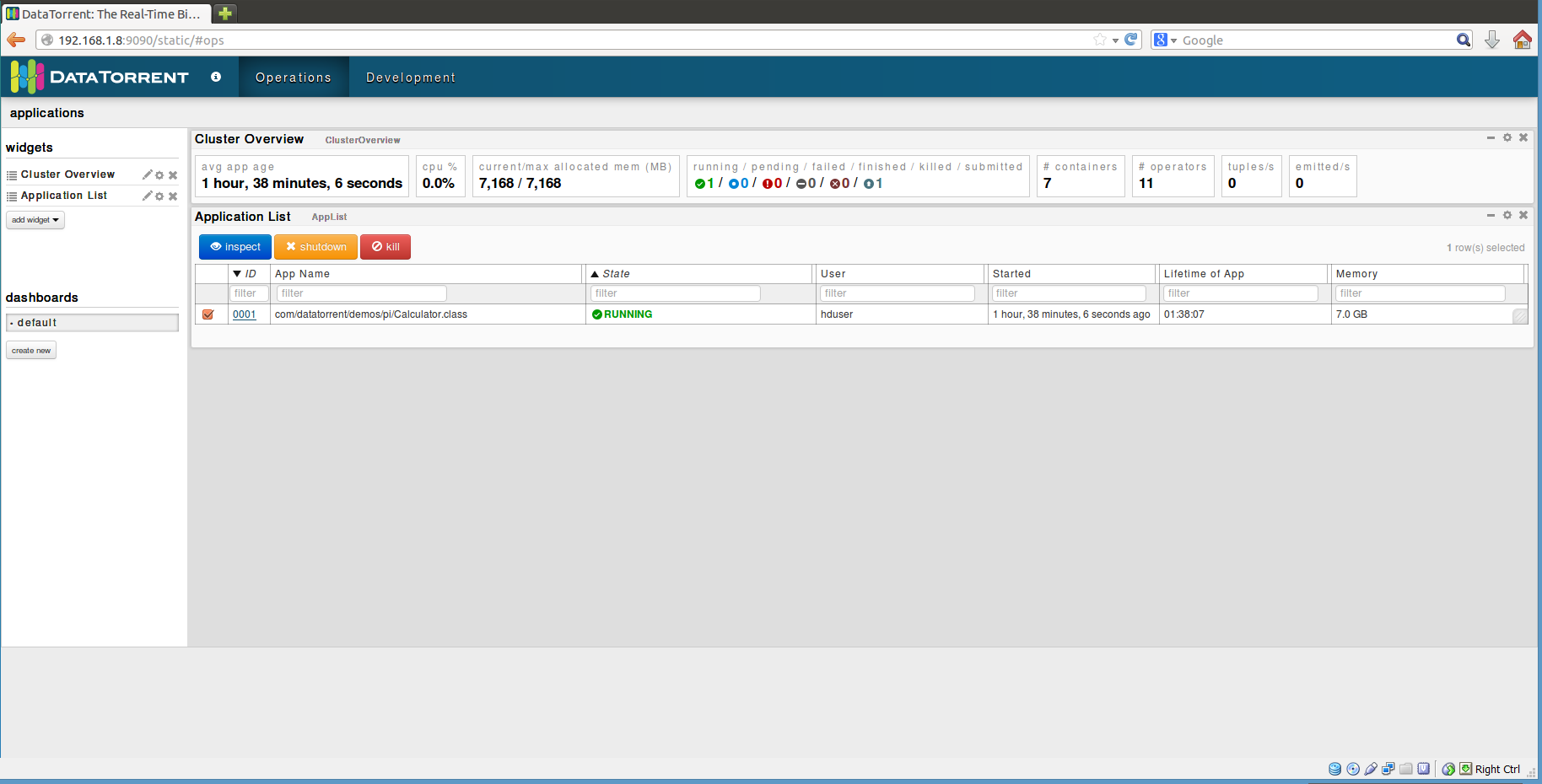


You can see detailed information about –

* partitions
* Metrices
* Properties

You can add more widgets here as well

1. Selecting an application from the list of applications also enables the operations to control the application viz. inspect, shutdown, kill. The user can click on any of the buttons to perform the corresponding operation.



# Stopping the application

From dtcli

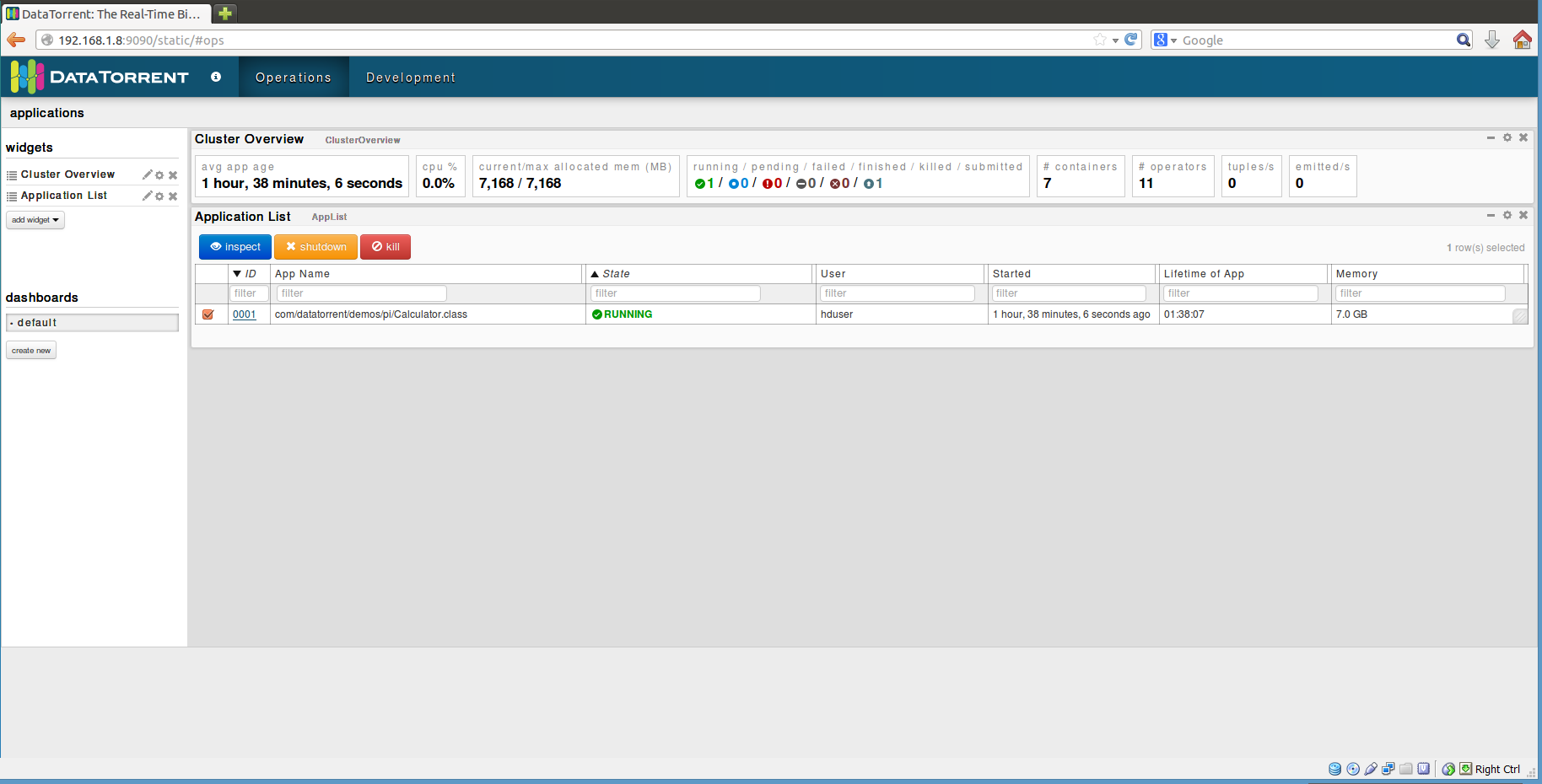
The user can use the ‘shutdown-app’ command to shut down the application from the dtcli.

Since this is a platform for streaming applications, in most of the cases, the application would not be shutting down The user will have to kill the application. In order to achieve this, a kill-app command is provided which can be used.



From DT Gateway Console

An application can be either shutdown or killed from the DT Gateway console. Selecting an application from the list of applications causes the ‘shutdown’ and ‘kill buttons are activated along with the ‘inspect’ button. The user can then shutdown or kill the application using these buttons.



# Conclusions

Each of the operations which form a part of the Monte Carlo estimation technique is performed by the various DataTorrent operators. The ‘Pi Calculator’ application computes the value of ‘π’ using the Monte Carlo estimation technique.

This application demonstrates how a logical computation is broken down into granular operations. Each operation is performed by an individual operator. Multiple of the same operations can be performed on the platform simultaneously by using partitioning. When there is an increase in incoming data, new partitions can be dynamically created at run time. The platform can handle this and scale in and out as needed. The platform follows fundamental tenet of achieving scalability by distributing resource utilization across a commodity cluster. The design of an application must be scalable by construction. Platform has various knobs that users can avail of, and to do so the applications must be designed as such. This ability is one of the most crucial expertises needed to design scalable DataTorrent applications.