Building Python Real-Time Applications with Storm

With its ability to process all kinds of data in real-time, Storm is an important addition to your big data bag of tricks. Python is one of the fastest-growing programming languages today. It has become a top choice for both data science and everyday application development. Together, Storm and Python enable you to build and deploy real-time big data applications quickly and easily.

You will begin with some basic command tutorials to set up Storm and learn about its configuration in detail. You will then go through requirement scenarios for creating a Storm cluster. Next, you'll be provided with an overview of Petrel, followed by an example of Twitter topology and persistence using Redis and MongoDB. Finally, you will build a production-quality Storm topology using development best practices.

Who this book is written for
This book is intended for Python developers who want to benefit from Storm's real-time data processing capabilities. If you are new to Python, you'll benefit from the attention paid to key supporting tools and techniques such as automated testing, virtual environments, and logging. If you're an experienced Python developer, you'll appreciate the thorough and detailed examples.

What you will learn from this book
- Install Storm and learn about the prerequisites
- Get to know the components of a Storm topology and how to control the flow of data between them
- Ingest Twitter data directly into Storm
- Use Storm with MongoDB and Redis
- Build topologies and run them in Storm
- Use an interactive graphical debugger to debug your topology as it's running in Storm
- Test your topology components outside of Storm
- Configure your topology using YAML

Learn to process massive real-time data streams using Storm and Python—no Java required!
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'Getting Acquainted with Storm'
- A synopsis of the book’s content
- More information on Building Python Real-Time Applications with Storm
About the Authors

Kartik Bhatnagar loves nature and likes to visit picturesque places. He is a technical architect in the big data analytics unit of Infosys. He is passionate about new technologies. He is leading the development work of Apache Storm and MarkLogic NoSQL for a leading bank. Kartik has a total 10 years of experience in software development for Fortune 500 companies in many countries. His expertise also includes the full Amazon Web Services (AWS) stack and modern open source libraries. He is active on the StackOverflow platform and is always eager to help young developers with new technologies. Kartik has also worked as a reviewer of a book called Elasticsearch Blueprints, Packt Publishing. In the future, he wants to work on predictive analytics.

Barry Hart began using Storm in 2012 at AirSage. He quickly saw the potential of Storm while suffering from the limitations of the basic storm.py that it provides. In response, he developed Petrel, the first open source library for developing Storm applications in pure Python. He also contributed some bug fixes to the core Storm project.

When it comes to development, Barry has worked on a little of everything: Windows printer drivers, logistics planning frameworks, OLAP engines for the retail industry, database engines, and big data workflows.

Barry is currently an architect and senior Python/C++ developer at Pindrop Security, helping fight phone fraud in banking, insurance, investment, and other industries.
Apache Storm is a powerful framework for creating complex workflows that ingest and process huge amounts of data. With its generic concepts of spouts and bolts, along with simple deployment and monitoring tools, it allows developers to focus on the specifics of their workflow without reinventing the wheel.

However, Storm is written in Java. While it supports other programming languages besides Java, the tools are incomplete and there is little documentation and few examples.

One of the authors of this book created Petrel, the first framework that supports the creation of Storm topologies in 100 percent Python. He has firsthand experience with the struggles of building a Python Storm topology on the Java tool set. This book closes this gap, providing a resource to help Python developers of all experience levels in building their own applications using Storm.

What this book covers

Chapter 1, Getting Acquainted with Storm, provides detailed information about Storm's use cases, different installation modes, and configuration in Storm.

Chapter 2, The Storm Anatomy, tells you about Storm-specific terminologies, processes, fault tolerance in Storm, tuning parallelism in Storm, and guaranteed tuple processing, with detailed explanations about each of these.

Chapter 3, Introducing Petrel, introduces a framework called Petrel for building Storm topologies in Python. This chapter walks through the installation of Petrel and includes a simple example.
Chapter 4, Example Topology – Twitter, provides an in-depth example of a topology that computes statistics on Twitter data in real time. The example introduces the use of tick tuples, which are useful for topologies that need to compute statistics or do other things on a schedule. In this chapter, you also see how topologies can access configuration data.

Chapter 5, Persistence Using Redis and MongoDB, updates the sample Twitter topology for the use of Redis, a popular key-value store. It shows you how to simplify the complex Python calculation logic with built-in Redis operations. The chapter concludes with an example of storing Twitter data in MongoDB, a popular NoSQL database, and using its aggregation capabilities to generate reports.

Chapter 6, Petrel in Practice, teaches practical skills that will make developers more productive using Storm. You learn how to use Petrel to create automated tests for your spout and bolt components that run outside of Storm. You also see how to use a graphical debugger to debug a topology running inside Storm.

Appendix, Managing Storm Using Supervisord, is a practical demonstration of monitoring and control of Storm using a supervisor over the cluster.
In this chapter, you will get acquainted with the following topics:

- An overview of Storm
- The "before Storm" era and key features of Storm
- Storm cluster modes
- Storm installation
- Starting various daemons
- Playing with Storm configurations

Over the complete course of the chapter, you will learn why Storm is creating a buzz in the industry and why it is relevant in present-day scenarios. What is this real-time computation? We will also explain the different types of Storm's cluster modes, the installation, and the approach to configuration.

**Overview of Storm**

Storm is a distributed, fault-tolerant, and highly scalable platform for processing streaming data in a real-time manner. It became an Apache top-level project in September 2014, and was previously an Apache Incubator project since September 2013.
Real-time processing on a massive scale has become a requirement of businesses. Apache Storm provides the capability to process data (a.k.a tuples or stream) as and when it arrives in a real-time manner with distributed computing options. The ability to add more machines to the Storm cluster makes Storm scalable. Then, the third most important thing that comes with Storm is fault tolerance. If the Storm program (also known as topology) is equipped with reliable spout, it can reprocess the failed tuples lost due to machine failure and also give fault tolerance. It is based on XOR magic, which will be explained in Chapter 2, The Storm Anatomy.

Storm was originally created by Nathan Marz and his team at BackType. The project was made open source after it was acquired by Twitter. Interestingly, Storm received a tag as Real Time Hadoop.

Storm is best suited for many real-time use cases. A few of its interesting use cases are explained here:

- **ETL pipeline**: ETL stands for Extraction, Transformation, and Load. It is a very common use case of Storm. Data can be extracted or read from any source. Here, the data can be complex XML, a JDBC result set row, or simply a few key-value records. Data (also known as tuples in Storm) can be enriched on the fly with more information, transformed into the required storage format, and stored in a NoSQL/RDBMS data store. All of these things can be achieved at a very high throughput in a real-time manner with simple storm programs. Using the Storm ETL pipeline, you can ingest into a big data warehouse at high speed.

- **Trending topic analysis**: Twitter uses such use cases to know the trending topics within a given time frame or at present. There are numerous use cases, and finding the top trends in a real-time manner is required. Storm can fit well in such use cases. You can also perform running aggregation of values with the help of any database.

- **Regulatory check engine**: Real-time event data can pass through a business-specific regulatory algorithm, which can perform a compliance check in a real-time manner. Banks use these for trade data checks in real time.

Storm can ideally fit into any use case where there is a need to process data in a fast and reliable manner, at a rate of more than 10,000 messages processing per second, as soon as data arrives. Actually, 10,000+ is a small number. Twitter is able to process millions of tweets per second on a large cluster. It depends on how well the Storm topology is written, how well it is tuned, and the cluster size.
Storm program (a.k.a topologies) are designed to run 24x7 and will not stop until someone stops them explicitly.

Storm is written using both Clojure as well as Java. Clojure is a Lisp, functional programming language that runs on JVM and is best for concurrency and parallel programming. Storm leverages the mature Java library, which was built over the last 10 years. All of these can be found inside the storm/lib folder.

**Before the Storm era**

Before Storm became popular, real-time or near-real-time processing problems were solved using intermediate brokers and with the help of message queues. Listener or worker processes run using the Python or Java languages. For parallel processing, code was dependent on the threading model supplied using the programming language itself. Many times, the old style of working did not utilize CPU and memory very well. In some cases, mainframes were used as well, but they also became outdated over time. Distributed computing was not so easy. There were either many intermediate outputs or hops in this old style of working. There was no way to perform a fail replay automatically. Storm addressed all of these pain areas very well. It is one of the best real-time computation frameworks available for use.

**Key features of Storm**

Here are Storm's key features; they address the aforementioned problems:

- **Simple to program**: It’s easy to learn the Storm framework. You can write code in the programming language of your choice and can also use the existing libraries of that programming language. There is no compromise.

- **Storm already supports most programming languages**: However, even if something is not supported, it can be done by supplying code and configuration using the JSON protocol defined in the Storm Data Specification Language (DSL).

- **Horizontal scalability or distributed computing is possible**: Computation can be multiplied by adding more machines to the Storm cluster without stopping running programs, also known as topologies.

- **Fault tolerant**: Storm manages worker and machine-level failure. Heartbeats of each process are tracked to manage different types of failure, such as task failure on one machine or an entire machine's failure.

- **Guaranteed message processing**: There is a provision of performing auto and explicit ACK within storm processes on messages (tuples). If ACK is not received, storm can do a reply of a message.
• **Free, open source, and lots of open source community support**: Being an Apache project, Storm has free distribution and modifying rights without any worry about the legal aspect. Storm gets a lot of attention from the open source community and is attracting a large number of good developers to contribute to the code.

**Storm cluster modes**
The Storm cluster can be set up in four flavors based on the requirement. If you want to set up a large cluster, go for distributed installation. If you want to learn Storm, then go for a single machine installation. If you want to connect to an existing Storm cluster, use client mode. Finally, if you want to perform development on an IDE, simply unzip the storm TAR and point to all dependencies of the storm library. At the initial learning phase, a single-machine storm installation is actually what you need.

**Developer mode**
A developer can download storm from the distribution site, unzip it somewhere in $HOME, and simply submit the Storm topology as local mode. Once the topology is successfully tested locally, it can be submitted to run over the cluster.

**Single-machine Storm cluster**
This flavor is best for students and medium-scale computation. Here, everything runs on a single machine, including **Zookeeper**, **Nimbus**, and **Supervisor**. Storm/bin is used to run all commands. Also, no extra Storm client is required. You can do everything from the same machine. This case is well demonstrated in the following figure:
Multimachine Storm cluster

This option is required when you have a large-scale computation requirement. It is a horizontal scaling option. The following figure explains this case in detail. In this figure, we have five physical machines, and to increase fault tolerance in the systems, we are running Zookeeper on two machines. As shown in the diagram, Machine 1 and Machine 2 are a group of Zookeeper machines; one of them is the leader at any point of time, and when it dies, the other becomes the leader. Nimbus is a lightweight process, so it can run on either machine, 1 or 2. We also have Machine 3, Machine 4, and Machine 5 dedicated for performing actual processing. Each one of these machines (3, 4, and 5) requires a supervisor daemon to run over there. Machines 3, 4, and 5 should know where the Nimbus/Zookeeper daemon is running and that entry should be present in their storm.yaml.

So, each physical machine (3, 4, and 5) runs one supervisor daemon, and each machine's storm.yaml points to the IP address of the machine where Nimbus is running (this can be 1 or 2). All Supervisor machines must add the Zookeeper IP addresses (1 and 2) to storm.yaml. The Storm UI daemon should run on the Nimbus machine (this can be 1 or 2).

The Storm client

The Storm client is required only when you have a Storm cluster of multiple machines. To start the client, unzip the Storm distribution and add the Nimbus IP address to the storm.yaml file. The Storm client can be used to submit Storm topologies and check the status of running topologies from command-line options. Storm versions older than 0.9 should put the yaml file inside $STORM_HOME/.storm/storm.yaml (not required for newer versions).
The jps command is a very useful Unix command for seeing the Java process ID of Zookeeper, Nimbus, and Supervisor. The kill -9 <pid> option can stop a running process. The jps command will work only when JAVA_HOME is set in the PATH environment variable.

Prerequisites for a Storm installation

Installing Java and Python is easy. Let's assume our Linux machine is ready with Java and Python:

- A Linux machine (Storm version 0.9 and later can also run on Windows machines)
- Java 6 (set export PATH=$PATH:$JAVA_HOME/bin)
- Python 2.6 (required to run Storm daemons and management commands)

We will be making lots of changes in the storm configuration file (that is, storm.yaml), which is actually present under $STORM_HOME/config. First, we start the Zookeeper process, which carries out coordination between Nimbus and the Supervisors. Then, we start the Nimbus master daemon, which distributes code in the Storm cluster. Next, the Supervisor daemon listens for work assigned (by Nimbus) to the node it runs on and starts and stops the worker processes as necessary.

ZeroMQ/JZMQ and Netty are inter-JVM communication libraries that permit two machines or two JVMs to send and receive process data (tuples) between each other. JZMQ is a Java binding of ZeroMQ. The latest versions of Storm (0.9+) have now been moved to Netty. If you download an old version of Storm, installing ZeroMQ and JZMQ is required. In this book, we will be considering only the latest versions of Storm, so you don't really require ZeroMQ/JZMQ.

Zookeeper installation

Zookeeper is a coordinator for the Storm cluster. The interaction between Nimbus and worker nodes is done through Zookeeper. The installation of Zookeeper is well explained on the official website at http://zookeeper.apache.org/doc/trunk/zookeeperStarted.html#sc_InstallingSingleMode.
The setup can be downloaded from:

The following are the Zookeeper commands that are used:

- Starting the zookeeper process:
  ```
  ../zookeeper/bin/./zkServer.sh start
  ```
- Checking the running status of the zookeeper service:
  ```
  ../zookeeper/bin/./zkServer.sh status
  ```
- Stopping the zookeeper service:
  ```
  ../zookeeper/bin/./zkServer.sh stop
  ```

Alternatively, use `jps` to find `<pid>` and then use `kill -9 <pid>` to kill the processes.

**Storm installation**

Storm can be installed in either of these two ways:

1. Fetch a Storm release from this location using Git:
   
   ```
   https://github.com/nathanmarz/storm.git
   ```
2. Download directly from the following link:
   ```
   https://storm.apache.org/downloads.html
   ```

Storm configurations can be done using `storm.yaml`, which is present in the conf folder.

The following are the configurations for a single-machine Storm cluster installation.

Port # 2181 is the default port of Zookeeper. To add more than one zookeeper, keep entry – separated:

```
storm.zookeeper.servers:
  - "localhost"

# you must change 2181 to another value if zookeeper running on another port.
storm.zookeeper.port: 2181
```

# In single machine mode nimbus run locally so we are keeping it localhost.
Getting Acquainted with Storm

# In distributed mode change localhost to machine name where nimbus daemon is running.

nimbus.host: "localhost"

# Here storm will generate logs of workers, nimbus and supervisor.

storm.local.dir: "/var/stormtmp"

java.library.path: "/usr/local/lib"

# Allocating 4 ports for workers. More numbers can also be added.

supervisor.slots.ports:
  - 6700
  - 6701
  - 6702
  - 6703

# Memory is allocated to each worker. In below case we are allocating 768 mb per worker.

worker.childopts: "-Xmx768m"

# Memory to nimbus daemon- Here we are giving 512 mb to nimbus.

nimbus.childopts: "-Xmx512m"

# Memory to supervisor daemon- Here we are giving 256 mb to supervisor.

supervisor.childopts: "-Xmx256m". In this setting, we reserved four supervisor ports, which means that a maximum of four worker processes can run on this machine.

Notice supervisor.childopts: "-Xmx256m". In this setting, we reserved four supervisor ports, which means that a maximum of four worker processes can run on this machine.

storm.local.dir: This directory location should be cleaned if there is a problem with starting Nimbus and Supervisor. In the case of running a topology on the local IDE on a Windows machine, C:\Users\<User-Name>\AppData\Local\Temp should be cleaned.

Enabling native (Netty only) dependency

Netty enables inter JVM communication and it is very simple to use

Netty configuration

You don't really need to install anything extra for Netty. This is because it's a pure Java-based communication library. All new versions of Storm support Netty.
Add the following lines to your storm.yaml file. Configure and adjust the values to best suit your use case:

```yaml
storm.messaging.transport: "backtype.storm.messaging.netty.Context"
storm.messaging.netty.server_worker_threads: 1
storm.messaging.netty.client_worker_threads: 1
storm.messaging.netty.buffer_size: 5242880
storm.messaging.netty.max_retries: 100
storm.messaging.netty.max_wait_ms: 1000
storm.messaging.netty.min_wait_ms: 100
```

### Starting daemons

Storm daemons are the processes that are needed to pre-run before you submit your program to the cluster. When you run a topology program on a local IDE, these daemons auto-start on predefined ports, but over the cluster, they must run at all times:

1. Start the master daemon, nimbus. Go to the bin directory of the Storm installation and execute the following command (assuming that zookeeper is running):
   ```bash
   ./storm nimbus
   ```
   Alternatively, to run in the background, use the same command with nohup, like this:
   ```bash
   Run in background
   nohup ./storm nimbus &
   ```

2. Now we have to start the supervisor daemon. Go to the bin directory of the Storm installation and execute this command:
   ```bash
   ./storm supervisor
   ```
   To run in the background, use the following command:
   ```bash
   nohup ./storm supervisor &
   ```

If Nimbus or the Supervisors restart, the running topologies are unaffected as both are stateless.
3. Let's start the storm UI. The Storm UI is an optional process. It helps us to see the Storm statistics of a running topology. You can see how many executors and workers are assigned to a particular topology. The command needed to run the storm UI is as follows:

```
./storm ui
```

Alternatively, to run in the background, use this line with `nohup`:

```
nohup ./storm ui &
```

To access the Storm UI, visit `http://localhost:8080`.

4. We will now start `storm logviewer`. Storm UI is another optional process for seeing the log from the browser. You can also see the storm log using the command-line option in the `$STORM_HOME/logs` folder. To start logviewer, use this command:

```
./storm logviewer
```

To run in the background, use the following line with `nohup`:

```
nohup ./storm logviewer &
```

To access Storm's log, visit `http://localhost:8000/logviewer`.

5. DRPC daemon: DRPC is another optional service. DRPC stands for Distributed Remote Procedure Call. You will require the DRPC daemon if you want to supply to the storm topology an argument externally through the DRPC client. Note that an argument can be supplied only once, and the DRPC client can wait for long until storm topology does the processing and the return. DRPC is not a popular option to use in projects, as firstly, it is blocking to the client, and secondly, you can supply only one argument at a time. DRPC is not supported by Python and Petrel.

Summarizing, the steps for starting processes are as follows:

1. First, all the Zookeeper daemons.
2. Nimbus daemons.
3. Supervisor daemon on one or more machine.
4. The UI daemon where Nimbus is running (optional).
5. The Logviewer daemon (optional).
6. Submitting the topology.

You can restart the Nimbus daemon anytime without any impact on existing processes or topologies. You can restart the supervisor daemon and can also add more supervisor machines to the Storm cluster anytime.

To submit jar to the Storm cluster, go to the bin directory of the Storm installation and execute the following command:

```
./storm jar <path-to-topology-jar> <class-with-the-main> <arg1> ... <argN>
```

Playing with optional configurations

All the previous settings are required to start the cluster, but there are many other settings that are optional and can be tuned based on the topology's requirement. A prefix can help find the nature of a configuration. The complete list of default yaml configuration is available at https://github.com/apache/storm/blob/master/conf/defaults.yaml.

Configurations can be identified by how the prefix starts. For example, all UI configurations start with ui*.

<table>
<thead>
<tr>
<th>Nature of the configuration</th>
<th>Prefix to look into</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>storm.*</td>
</tr>
<tr>
<td>Nimbus</td>
<td>nimbus.*</td>
</tr>
<tr>
<td>UI</td>
<td>ui.*</td>
</tr>
<tr>
<td>Log viewer</td>
<td>logviewer.*</td>
</tr>
<tr>
<td>DRPC</td>
<td>drpc.*</td>
</tr>
<tr>
<td>Supervisor</td>
<td>supervisor.*</td>
</tr>
<tr>
<td>Topology</td>
<td>topology.*</td>
</tr>
</tbody>
</table>
Getting Acquainted with Storm

All of these optional configurations can be added to $STORM_HOME/conf/storm.yaml$ for any change other than the default values. All settings that start with `topology.*` can either be set programmatically from the topology or from $storm.yaml$. All other settings can be set only from the $storm.yaml$ file. For example, the following table shows three different ways to play with these parameters. However, all of these three do the same thing:

<table>
<thead>
<tr>
<th>/conf/storm.yaml</th>
<th>Topology builder</th>
<th>Custom yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changing $storm.yaml$ (impacts all the topologies of the cluster)</td>
<td>Changing the topology builder while writing code (impacts only the current topology)</td>
<td>Supplying topology.yaml as a command-line option (impacts only the current topology)</td>
</tr>
<tr>
<td><code>topology.workers: 1</code></td>
<td><code>conf. setNumberOfWorker(1);</code> This is supplied through Python code</td>
<td>Create topology.yaml with the entry made into it similar to $storm.yaml$, and supply it when running the topology Python: <code>petrel submit --config topology.yaml</code></td>
</tr>
</tbody>
</table>

Any configuration change in $storm.yaml$ will affect all running topologies, but when using the `conf.setXXX` option in code, different topologies can overwrite that option, what is best suited for each of them.

**Summary**

Here comes the conclusion of the first chapter. This chapter gave an overview of how applications were developed before Storm came into existence. A brief knowledge of what real-time computations are and how Storm, as a programming framework, is becoming so popular was also acquired as we went through the chapter and approached the conclusion. This chapter taught you to perform Storm configurations. It also gave you details about the daemons of Storm, Storm clusters, and their step up. In the next chapter, we will be exploring the details of Storm's anatomy.
Where to buy this book

You can buy Building Python Real-Time Applications with Storm from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.