# Develop Apache Storm topologies using Python on HDInsight

Apache Storm supports multiple languages, even allowing you to combine components from several languages in one topology. In this document, you will learn how to use Python components in your Java and Clojure-based Storm topologies on HDInsight.

## Prerequisites

* Python 2.7 or higher
* Java JDK 1.7 or higher

## Storm multi-language support

Storm was designed to work with components written using any programming language, however this requires that the components understand how to work with the [Thrift definition for Storm](https://github.com/apache/storm/blob/master/storm-core/src/storm.thrift). For Python, a module is provided as part of the Apache Storm project that allows you to easily interface with Storm. You can find this module at <https://github.com/apache/storm/blob/master/storm-multilang/python/src/main/resources/resources/storm.py>.

Since Apache Storm is a Java process that runs on the Java Virtual Machine (JVM,) components written in other languages are executed as subprocesses. The Storm bits running in the JVM communicates with these subprocesses using JSON messages sent over stdin/stdout. More details on communication between components can be found in the [Multi-lang Protocol](https://storm.apache.org/documentation/Multilang-protocol.html) documentation.

### The Storm module

The storm module (https://github.com/apache/storm/blob/master/storm-multilang/python/src/main/resources/resources/storm.py,) provides the bits needed to create Python components that work with Storm.

This provides things like storm.emit to emit tuples, and storm.logInfo to write to the logs. I would encourage you to read over this file and understand what it provides.

## Challenges

Using the **storm.py** module, you can create Python spouts that consume data, and bolts that process data, however the overall Storm topology definition that wires up communication between components is still written using Java or Clojure. Additionally, if you use Java, you must also create Java components that act as an interface to the Python components.

Also, since Storm clusters run in a distributed fashion, you must ensure that any modules required by your Python components are available on all worker nodes in the cluster. Storm doesn’t provide any easy way to accomplish this for multi-lang resources - you either have to include all dependencies as part of the jar file for the topology, or manually install dependencies on each worker node in the cluster.

### Java vs. Clojure topology definition

Of the two methods of defining a topology, Clojure is by far the easiest/cleanest as you can directly referenc python components in the topology definition. For Java-based topology definitions, you must also define Java components that handle things like declaring the fields in the tuples returned from the Python components.

Both methods are described in this document, along with example projects.

## Python components with a Java topology

[AZURE.NOTE] This example is available at https://github.com/Blackmist/hdinsight-python-storm-wordcount in the **JavaTopology** directory. This is a Maven based project. If you are unfamiliar with Maven, see [Develop Java-based topologies with Apache Storm on HDInsight](/documentation/articles/hdinsight-storm-develop-java-topology) for more information on creating a Maven project for a Storm topology.

A Java-based topology that uses Python (or other JVM language components,) initially appears to use Java components; but if you look in each of the Java spouts/bolts, you’ll see code similar to the following:

public SplitBolt() {  
 super("python", "countbolt.py");  
}

This is where Java invokes Python and runs the script that contains the actual bolt logic. The Java spouts/bolts (for this example,) simply declare the fields in the tuple that will be emitted by the underlying Python component.

The actual Python files are stored in the /multilang/resources directory in this example. The /multilang directory is referenced in the **pom.xml**:

${basedir}/multilang

This includes all the files in the /multilang folder in the jar that will be built from this project.

[AZURE.IMPORTANT] Note that this only specifies the /multilang directory and not /multilang/resources. Storm expects non-JVM resources in a resources directory, so it is looked for internally already. Placing components in this folder allows you to just reference by name in the Java code. For example, super("python", "countbolt.py");. Another way to think of it is that Storm sees the resources directory as the root (/) when accessing multi-lang resources.

For this example project, the storm.py module is included in the /multilang/resources directory.

### Build and run the project

To run this project locally, just use the following Maven command to build and run in local mode:

mvn compile exec:java -Dstorm.topology=com.microsoft.example.WordCount

Use ctrl+c to kill the process.

To deploy the project to an HDInsight cluster running Apache Storm, use the following steps:

1. Build an uber jar:

* mvn package
* This will create a file named **WordCount–1.0-SNAPSHOT.jar** in the /target directory for this project.

1. Upload the jar file to the Hadoop cluster using one of the following methods:
   * For **Windows-based** HDInsight clusters: Connect to the Storm Dashboard by going to HTTPS://CLUSTERNAME.azurehdinsight.cn/ in your browser. Replace CLUSTERNAME with your HDInsight cluster name and provide the admin name and password when prompted.
   * Using the form, perform the following actions:
     + **Jar File**: Select **Browse**, then select the **WordCount-1.0-SNAPSHOT.jar** file
     + **Class Name**: Enter com.microsoft.example.WordCount
     + **Additional Paramters**: Enter a friendly name such as wordcount to identify the topology
   * Finally, select **Submit** to start the topology.

[AZURE.NOTE] Once started, a Storm topology runs until stopped (killed.) To stop the topology, use either the storm kill TOPOLOGYNAME command from the command-line or by using the Storm UI, select the topology, and then select the **Kill** button.

## Python components with a Clojure topology

[AZURE.NOTE] This example is available at https://github.com/Blackmist/hdinsight-python-storm-wordcount in the **ClojureTopology** directory.

This topology was created by using [Leiningen](http://leiningen.org) to [create a new Clojure project](https://github.com/technomancy/leiningen/blob/stable/doc/TUTORIAL.md#creating-a-project). After that, the following modifications to the scaffolded project were made:

* **project.clj**: Added dependencies for Storm, and exclusions for items that may cause a problem when deployed to the HDInsight server.
* **resources/resources**: Leiningen creates a default resources directory, however the files stored here appear to get added to the root of the jar file created from this project, and Storm expects files in a sub-directory named resources. So a sub-directory was added and the Python files are stored in resources/resources. At run-time, this will be treated as the root (/) for accessing Python components.
* **src/wordcount/core.clj**: This file contains the topology definition, and is referenced from the **project.clj** file. For more information on using Clojure to define a Storm topology, see [Clojure DSL](https://storm.apache.org/documentation/Clojure-DSL.html).

### Build and run the project

**To build and run the project locally**, use the following command:

lein do clean, run

To stop the topology, use **Ctrl+C**.

**To build an uberjar and deploy to HDInsight**, use the following steps:

1. Create an uberjar containing the topology and required dependencies:

* lein uberjar
* This will create a new file named wordcount-1.0-SNAPSHOT.jar in the target\uberjar+uberjar directory.

1. Use one of the following methods to deploy and run the topology to an HDInsight cluster:
   * **Windows-based HDInsight**
     1. Connect to the Storm Dashboard by going to HTTPS://CLUSTERNAME.azurehdinsight.cn/ in your browser. Replace CLUSTERNAME with your HDInsight cluster name and provide the admin name and password when prompted.
     2. Using the form, perform the following actions:
        + **Jar File**: Select **Browse**, then select the **wordcount-1.0-SNAPSHOT.jar** file
        + **Class Name**: Enter wordcount.core
        + **Additional Paramters**: Enter a friendly name such as wordcount to identify the topology
     + Finally, select **Submit** to start the topology.

[AZURE.NOTE] Once started, a Storm topology runs until stopped (killed.) To stop the topology, use either the storm kill TOPOLOGYNAME command from the command-line or by using the Storm UI, select the topology, and then select the **Kill** button. <!– deleted by customization

## Pyleus framework

[Pyleus](https://github.com/Yelp/pyleus) is a framework that attempts to make it easier to use Python with Storm by providing the following:

* **YAML-based topology definitions**: This provides an easier way to define the topology, that doesn’t require knowledge of Java or Clojure
* **MessagePack-based serializer**: MessagePack is used as the default serialization, instead of JSON. This can result in faster messaging between components
* **Dependency management**: Virtualenv is used to ensure that Python dependencies are deployed to all worker nodes. This requires Virtualenv to be installed on the worker nodes

[AZURE.IMPORTANT] Pyleus requires Storm on your development environment. Using the base Apache Storm 0.9.3 distribution seems to result in jars that are incompatible with the version of Storm provided with HDInsight. So the following steps use the HDInsight cluster as the development environment.

You can successfuly build the example Pyleus topologies, using the HDInsight head node as the build environment:

1. When provisioning a new Storm on HDInsight cluster, you must ensure that Python Virtualenv is present on the cluster nodes. When creating a new Linux-based HDInsight cluster, use the following Script Action settings with [Cluster customization](/documentation/articles/hdinsight-hadoop-customize-cluster):
   * **Name**: Just provide a friendly name here
   * \_\_ Script URI\_\_: Use https://hditutorialdata.blob.core.windows.net/customizecluster/pythonvirtualenv.sh as the value. This script will install Python Virtualenv on the nodes.
   * [AZURE.NOTE] It will also create some directories that are used by the Streamparse framework later in this document.
   * **Nimbus**: Check this entry so that the script is applied to the Nimbus (head) nodes.
   * **Supervisor**: Check ths entry so that the script is applied to the supervisor (worker) nodes

* Leave other entries blank.

1. Once the cluster has been created, connect using SSH:
   * [Use SSH with Linux-based HDInsight from Linux, Unix, or OS X](/documentation/articles/hdinsight-hadoop-linux-use-ssh-unix)
   * [Use SSH with Linux-based HDInsight from Windows](/documentation/articles/hdinsight-hadoop-linux-use-ssh-windows)
2. From the SSH connect, use the following to create a new virtual environment and install Pyleus:

* virtualenv pyleus\_venv  
  source pyleus\_venv  
  pip install pyleus

1. Next, download the Pyleus git repository and build the WordCount example:

* sudo apt-get install git  
  git clone https://github.com/Yelp/pyleus.git  
  pyleus build pyleus/examples/word\_count/pyleus\_topology.yaml
* Once the build completes, you will have a new file named word\_count.jar in the current directory.

1. To submit the topology to the Storm cluster, use the following command:

* pyleus submit -n localhost word\_count.jar
* The -n parameter specifies the Nimbus host. Since we are on the head node, we can use localhost.
* You can also use the pyleus command to perform other Storm actions. Use the following to list the running topologies, and then kill the word\_count topology:
* pyleus list -n localhost  
  pyleus kill -n localhost word\_count

–>

## Next steps

In this document, you learned how to use Python components from a Storm topology. See the following documents for other ways to use Python with HDInsight:

* [How to use Python User Defined Functions (UDF) in Pig and Hive](/documentation/articles/hdinsight-python)