Securing YARN Containers with the Java Security Manager

Overview

Standard deployments of YARN allow applications to perform a variety of privileged actions on every node of the cluster. Actions such as spawning a subprocess, opening arbitrary network connections, or file creation can be performed within a YARN container; each of these actions has the potential to destabilize a host if used incorrectly. This risk is most relevant on clusters which provide massive scale data processing on YARN as a service to very large user bases. The following proposal details the design and considerations for enabling the Java Security Manager on YARN to mitigate the risks of user applications destabilizing the cluster.

There are two possible models for wrapping YARN containers in a Security Manager, client side security and server side security. These models are mutually exclusive and they have different sets of dependencies and implementation details. Effort has primarily been focused on the server side implementation since it is not dependent on projects outside of the Apache Hadoop ecosystem.

Server side security

The server side implementation requires every YARN container to launch with the java security manager enabled. Any applications which do not run within a JVM will need to be disabled. This can be accomplished by parsing the command sent to the NodeManager to ensure a single java command is provided. If the command attempts to run a process outside of a JVM an exception must be thrown, and the container launch process aborted. Additionally the Nodemanager will be responsible for modifying the container command to include JVM options to enable the security manager and provide a complete policy implementation.

For example the ContainerLauncher may receive the following command from the ApplicationMaster:

```
${JAVA_HOME}/bin/java -cp ${HADOOP_HOME}/hadoop-common.jar:……
```

The application master would modify this command to enable the security manager:

```
${JAVA_HOME}/bin/java -Djava.security.manager \
        -Djava.policy.file==${application_dir)/java.policy  -cp ${HADOOP_HOME}/…
```

The policy file will be generated dynamically to allow the minimum level of permissions for user code, while providing all permissions to Java and Hadoop libraries.

Client side security

Implementing this feature outside of the YARN Nodemanager will require additional controls on submitted jobs. The Java security manager will need to be enabled within the YARN application prior to running any user provided code, such as within the YarnChild class for MapReduce. Cluster administrators will need to restrict what actions users can perform to prevent them from submitting applications which do not enable the security manager. This can be accomplished by restricting the ways users can interact with the cluster. For example, a job submission portal like Hue can be configured to ensure all applications submitted will use the security manager.

Limiting user permissions

Using the Java security manager you can prevent user code from opening network connections while allowing standard YARN operations (block retrieval, Application Master communication, etc.). In an example Mapper we attempt to open an arbitrary socket with the Java security manager enabled using the sample policy file. This action is denied by the security manager which provides the following stack trace:

```
16/06/10 09:58:03 INFO mapreduce.Job: Task Id : attempt_1465501163456_0011_m_000001_0, Status :
FAILED
Error: java.security.AccessControlException: access denied ("java.net.SocketPermission" "localhost:0"
  "listen,resolve")
   at java.security.AccessControlContext.checkPermission(AccessControlContext.java:372)
   at java.security.AccessController.checkPermission(AccessController.java:559)
   at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
   at java.lang.SecurityManager.checkListen(SecurityManager.java:1134)
   at java.net.ServerSocket.bind(ServerSocket.java:375)
   at java.net.ServerSocket.<init>(ServerSocket.java:237)
   at java.net.ServerSocket.<init>(ServerSocket.java:128)
   at NetworkTest$TokenizerMapper.map(NetworkTest.java:39)
   at NetworkTest$TokenizerMapper.map(NetworkTest.java:25)
   at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:145)
   at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:787)
   at org.apache.hadoop.mapred.MapTask.run(MapTask.java:341)
   at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
   at java.security.AccessController.doPrivileged(Native Method)
   at javax.security.auth.Subject.doAs(Subject.java:415)
   at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1693)
   at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
```

Attempting to access files which are not necessary for the client code (such as raw blocks, the contents of /tmp, etc.) must also be limited by the policy file. If client code attempts to access any files which are not whitelisted in the policy file an AccessControlException similar to the excerpt above will be thrown. The core Hadoop codebase must be provided additional permissions to operate. In the example policy file all of the jars in '/opt/hadoop/jars' are provided all permissions, which means any code in that directory succeeds any security check provided it isn't called by untrusted code.

Jar Signing

It will be necessary to sign all of the core Hadoop libraries for either the client or server implementation of this feature. The java policy file allows you to provide permissions based on a library's signature. Signing the Hadoop codebase will remove the requirement for tracking the file locations of every Hadoop jar. This is a more extensible solution which does not require co-locating all of the jars, and supports advanced security policy configurations. Using this solution a Hadoop administrator could provide different permission rules for every codebase. For example an administrator could configure a three tiered approach:

1. Core Hadoop libraries are provided full filesystem and network access
2. User contributed code is limited to specific work directories and network ports
3. Unsigned code is provided minimum configurations to perform basic processing

Implementing jar signing will require integrations into every project associated with YARN. Further work is required to evaluate the best approach to sign projects such as Pig and Hive.

Applications on MapReduce

Currently both Pig and Hive build uber jars prior to job submission. The core libraries for each of the projects are packaged alongside the generated job plans. This presents a challenge for wrapping the container in a security manager since we have no way to distinguish the trusted core libraries from user code. One possible solution would entail modifying the jar creation process for each of these components to separate any generated content from the library jars. This would allow the original libraries to maintain any signature metadata provided. Alternatively if the security manager is applied within YARN application (i.e. Client side security) only a few modifications need to be made to the applications. Ultimately the only portion of MapReduce applications we want to restrict is the user provided code. As such these applications should be able to run mostly outside of a security manager, with the exception of user defined functions.

Performance

Cursory testing of the performance impact of the Java security manager on MapReduce jobs has yielded positive results. Across a series of teragen/terasort jobs running with and without the security manager enabled only a ~5% overhead in runtime was encountered. Further testing is required to determine the impact of larger and more varied workloads.

Appendix

Generated Policy File

```
/* AUTOMATICALLY GENERATED ON Fri Jun 10 06:10:31 PDT 2016*/
/* DO NOT EDIT */

grant codeBase "file:/usr/java/jdk1.7.0_67/jre/lib/ext/*" {
  permission java.security.AllPermission;
};

grant codeBase "file:/usr/java/packages/lib/ext/*" {
  permission java.security.AllPermission;
};

grant codeBase "file:/opt/hadoop/jars/-" {
  permission java.security.AllPermission;
};

grant {
  permission java.util.PropertyPermission "java.version", "read";
  permission java.util.PropertyPermission "java.vendor", "read";
  permission java.util.PropertyPermission "java.vendor.url", "read";
  permission java.util.PropertyPermission "java.class.version", "read";
  permission java.util.PropertyPermission "os.name", "read";
  permission java.util.PropertyPermission "os.version", "read";
  permission java.util.PropertyPermission "os.arch", "read";
  permission java.util.PropertyPermission "file.separator", "read";
  permission java.util.PropertyPermission "path.separator", "read";
  permission java.util.PropertyPermission "line.separator", "read";
  permission java.util.PropertyPermission "java.specification.version", "read";
  permission java.util.PropertyPermission "java.specification.vendor", "read";
  permission java.util.PropertyPermission "java.specification.name", "read";
  permission java.util.PropertyPermission "java.vm.specification.version", "read";
  permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
  permission java.util.PropertyPermission "java.vm.specification.name", "read";
  permission java.util.PropertyPermission "java.vm.version", "read";
  permission java.util.PropertyPermission "java.vm.vendor", "read";
  permission java.util.PropertyPermission "java.vm.name", "read";
  permission java.util.PropertyPermission "awt.Toolkit", "read";
  permission java.util.PropertyPermission "file.encoding", "read";
  permission java.util.PropertyPermission "file.encoding.pkg", "read";
  permission java.util.PropertyPermission "hadoop.metrics.log.level", "read";
  permission java.util.PropertyPermission "hadoop.root.logger", "read";
  permission java.util.PropertyPermission "java.awt.graphicsenv", "read";
  permission java.util.PropertyPermission "java.awt.printerjob", "read";
  permission java.util.PropertyPermission "java.class.path", "read";
  permission java.util.PropertyPermission "yarn.app.container.log.dir", "read";
  permission java.util.PropertyPermission "yarn.app.container.log.filesize", "read";
  permission java.util.RuntimePermission "loadLibrary.gplcompression";
  permission javax.security.auth.AuthPermission "getSubject";
};

grant {
  permission java.io.FilePermission "/data/yarn/nm/filecache/10/resourceTest-1.0.jar", "read";
  permission java.io.FilePermission
"/data/yarn/nm/usercache/hdfs/appcache/application_1465501163456_0006/-", "read";
};
```