Autoport in SuperVessel

In this guide, you will learn how to:

- Create an Autoport cluster in SuperVessel
- Access the Autoport service in SuperVessel
- Delete the Autoport cluster in SuperVessel
- Use the Autoport service to search for projects at github.com
- Use the Autoport service to build projects and analyze build and test results
- Use the Autoport service to create, build, and analyze lists of projects

At its most basic, Autoport helps you find and characterize projects hosted at github.com, it deduces how to build and test those projects automatically, and it allows the user to easily submit a build job for those projects over a set of managed Linux build servers (CentOS 7 ppc64le and x86-64, Ubuntu 14.04 ppc64le and x86-64) that are included with the Autoport Service. It also provides a set of reporting tools that help define normal working operation and that help identify and resolve errors.

In a little greater detail, Autoport provides:

- a window into github.com that includes project metrics such as code size, a list of programming languages that are used by the project, the popularity of the project as measured by user granted stars, the number of times that the project has been forked, and last time the project was updated. Collectively, these statistics identify the desirability of porting a project and how difficult that might be. It helps ensure that the right set of skills are applied and ensures that plans are more firmly grounded.
- a Top-N project search capability by programming language (or all programming languages) that produces a list of most popular projects hosted at github.com. This list of projects may be saved to a file and submitted as a single build job, or it may be exported as a file so that it may be shared with research staff members and planners who are working on next generation products. No single individually can be expected to keep track of all open source developments, so this feature provides a way to narrow the focus to the set of most statistically significant projects so that better decisions are made in a wide array of areas related to technology.
- a tightly integrated framework for managing the runtime environment of multiple build servers, each of which is pre-installed with 1000s of packages related to build tools, language runtimes, and most commonly used packages like libsnappy, libblas, and protobuf. Projects written in the following programming languages: C/C++, Java, JavaScript, Python, Perl, PHP, Ruby, and Scala are supported, generally with multiple

build tools per language. For example, Ant, Maven, and Gradle are supported for Java.

- the ability to select the type and version of Java and JavaScript (IBM Java 7, IBM Java 8, OpenJDK 7, OpenJDK 8, Node.js or the IBM SDK for Node.js) to be utilized when building a project. The build environment is dynamically changed to accommodate the user specification on a per project or batch file basis.
- internal apt and yum servers so that users can upload packages and install those packages across the set of managed build servers. Users may also upload Python and Perl modules through the same facility, although internally these are not managed through apt or yum. They are installed through Chef recipes.
- two install services are provided. One is integrated with the managed runtime environment and the other is not. The managed runtime environment provides a well-tested runtime environment that is known to provide a good foundation for open source software development. It can be easily and quickly recreated using special synchronization primitives developed for Autoport. It starts with a clean snapshotted O/S image and then it invokes a set of Chef recipes that install 1000s of packages setting environment variables as required by these subsystems. The non-managed runtime environment gives the user absolute control over the runtime environment to install any package.
- a mechanism to share data -- batch files and project build and test results -- across Autoport installations to promote collaboration and knowledge sharing. In general, Autoport produces data locally and the user decides when and what to copy to shared storage that is user configurable via the Settings Menu. Data is copied using SFTP.
- Historical, Detailed, Comparative Reports, Raw Log Files, and a Smart Console Log Diff Tool that color codes and Google annotates errors to help the user address problems more quickly. These reports work across platform architectures, Linux Distributions, and project versions. Users can quickly set base lines to identify what is a unique problem. Visualization tools are key as log files are large.

That is not to say that the tool will succeed every time or that developers will not have to frequently work around inadequacies in the tool, but the tool should provide significant time savings over time as it encompasses a wide spectrum of development activities.

INTEGRATION OF AUTOPORT WITH DEVELOPMENT

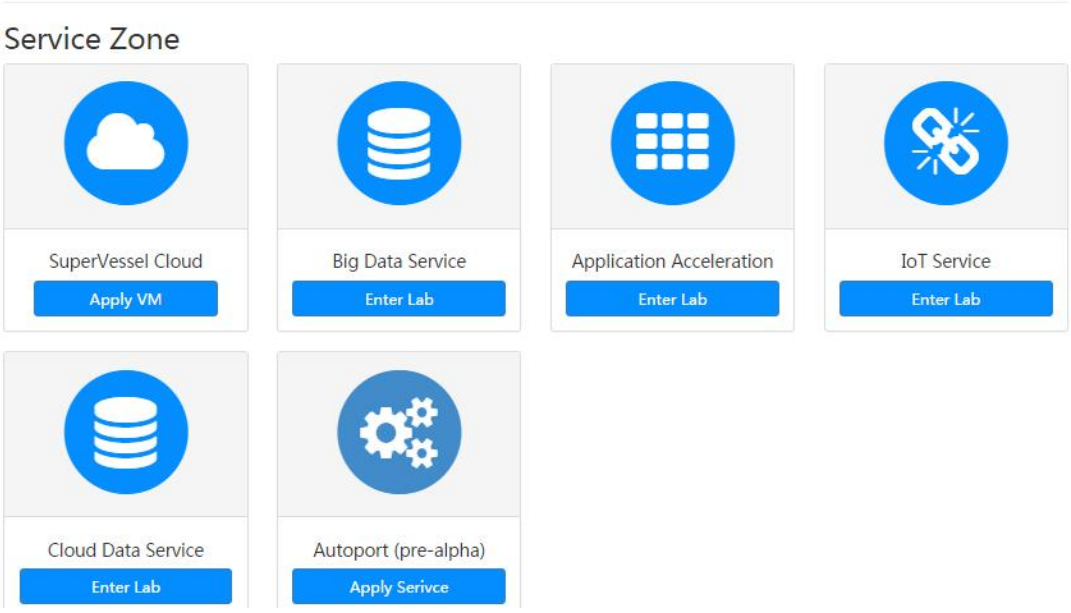
Autoport does not provide a platform for making code changes. Code changes are made outside the tool in a standard development environment. To test your code changes, you will need to make your code visible to Autoport. This can be accomplished by uploading it to github.com. Autoport uses the Github REST APIs to characterize projects and to determine how to build them.

Autoport itself is composed of the following languages and technologies: Python,

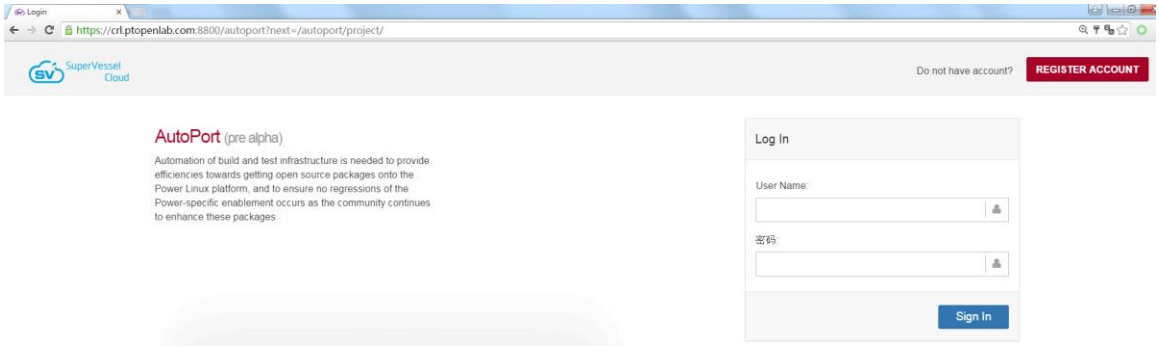
JavaScript, Java, HTML5, CSS, XML, Ruby, Chef, Flask, Rivets, and Bootstrap. It provides internal Yum and Apt servers and uses Apache web server.

1. Create an Autoport Cluster

(1) Access to http://ptopenlab.com on your browser, you will see a “Service Zone” on the page. Find “Autoport” icon and click “apply service”.

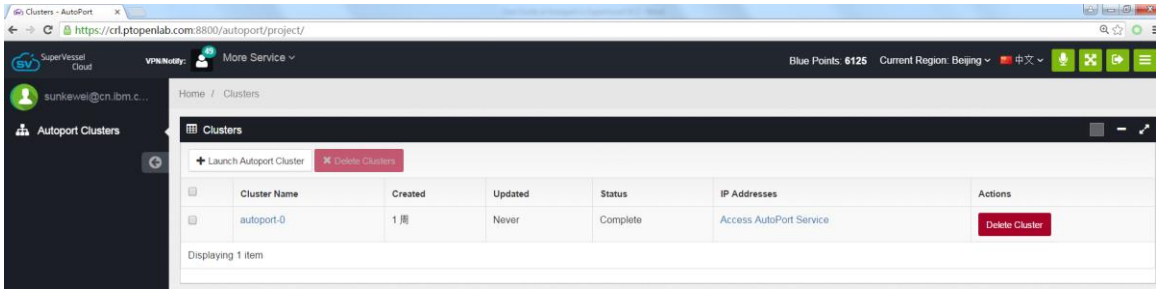


(2) Once you have opened another page, you will be asked to login Autoport in SuperVessel.

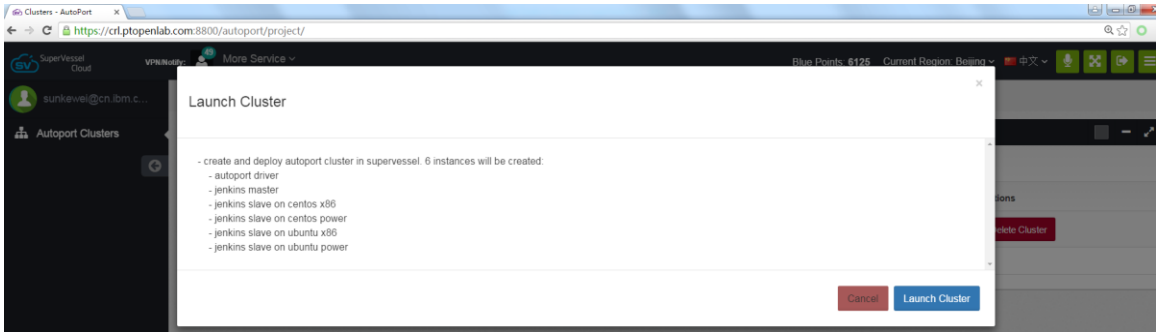


If you don't have an account in SuperVessel, You can register a new one by "REGISTER ACCOUNT" button on the top right of the page.

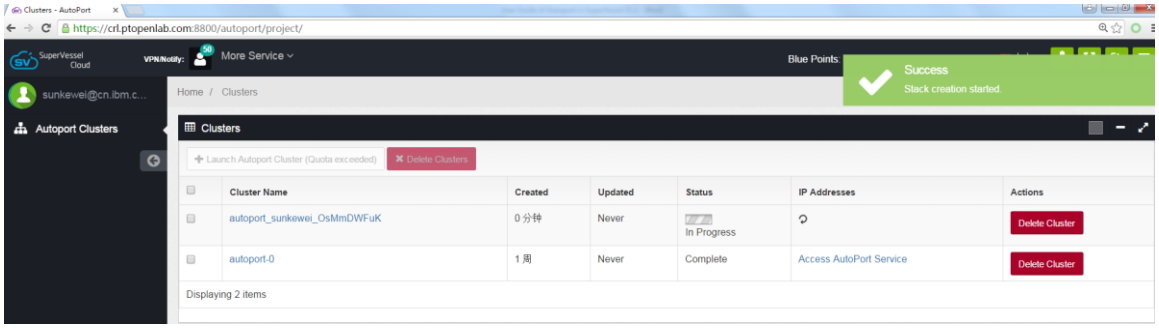
(3) After login, you can see the Autoport clusters you have created.



(4) You can click "Launch Autoport Cluster" button to create a new Autoport cluster. Once you click the button, you will get a pop-up message for Autoport cluster description. Then click the "Launch Cluster" button and a new cluster will be created.

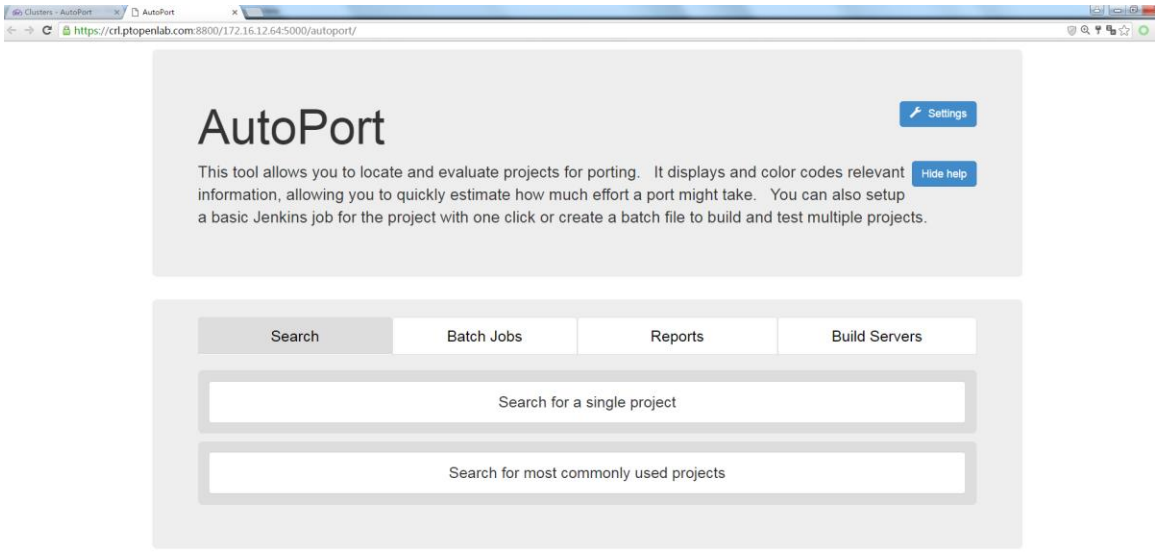


(5) Wait for several minutes until the status of the new created cluster becomes "complete".



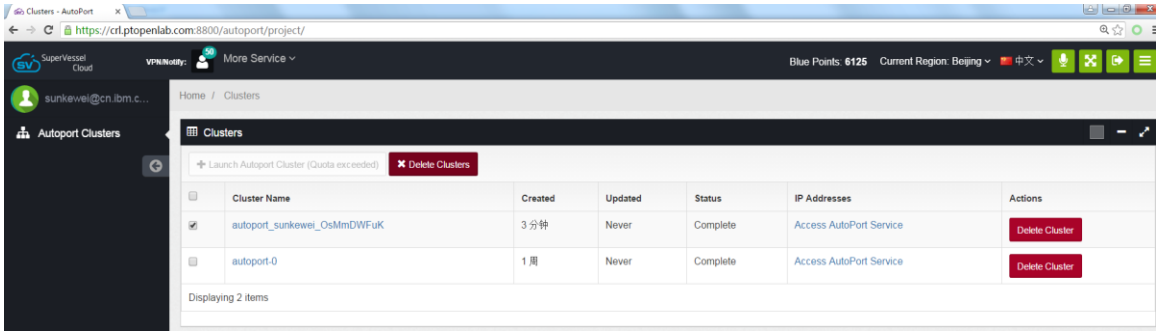
2. Access the Autoport Cluster

For each cluster, click “Access Autoport Service” link, you will access to the Autoport Service in your browser.



3. Delete an Autoport cluster in SuperVessel

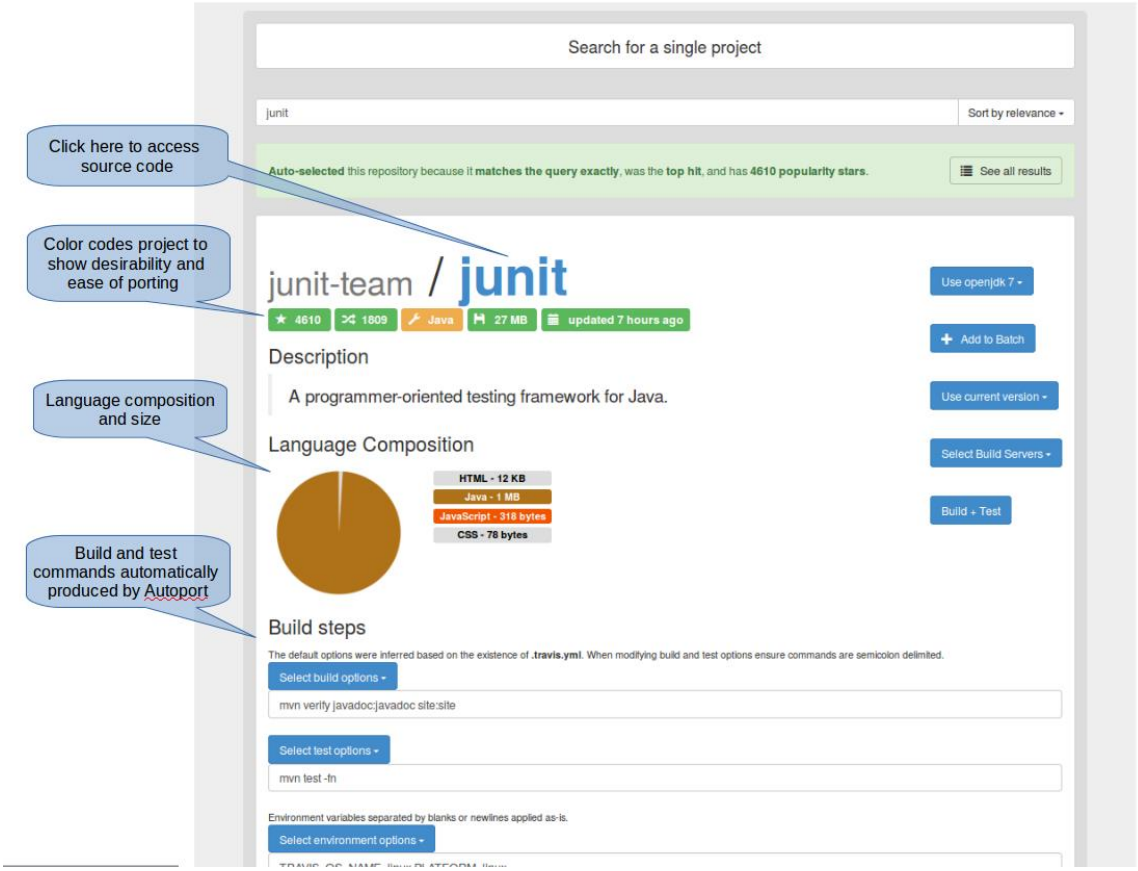
Click the “check box” in the front of the cluster which you want to delete, then click “Delete Clusters ” button.



4. Using Autoport

4.1 Search for a project

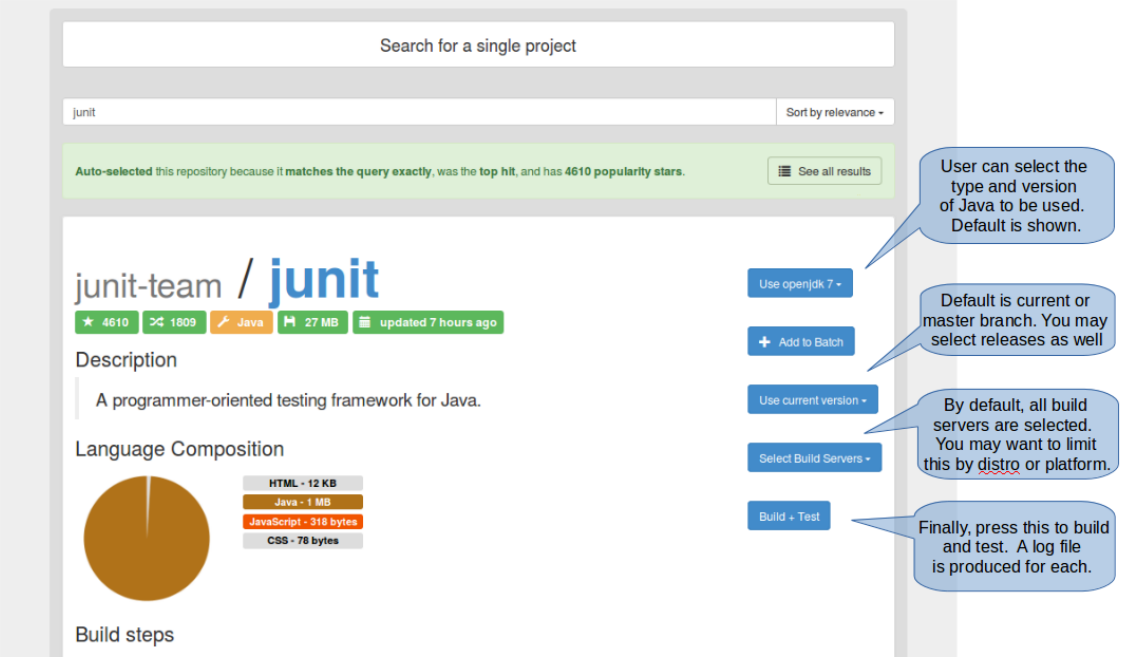
To search for a project simply type the name of the project and press Enter. Ie. junit



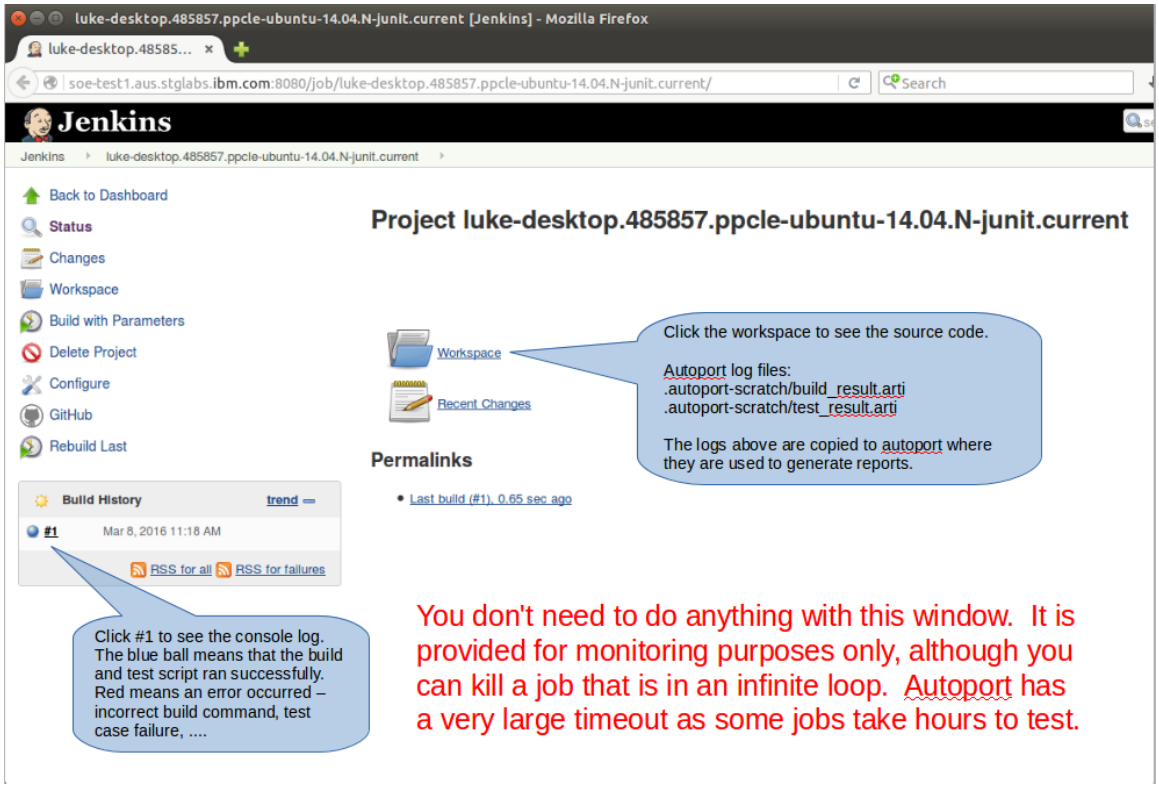
As indicated in the picture above, Autoport attempts to produce a build command automatically. In most cases, it produces several which are visible and maybe selected in a pull-down menu. Or you can specify a command of your own. You should expect to research projects and provide commands about half the time.

4.2 Build a project

When a build job is started, a new window is opened in your browser for each build server that was selected. These windows may be used to monitor the progress of each job as it is being built by Jenkins. This window itself is managed by Jenkins. You can close this window at any time, but it is useful to keep it open to note when Jenkins has completed its processing. When this occurs, you can use Reports Tab to analyze build results. The screen shot below shows how to build and test a project.

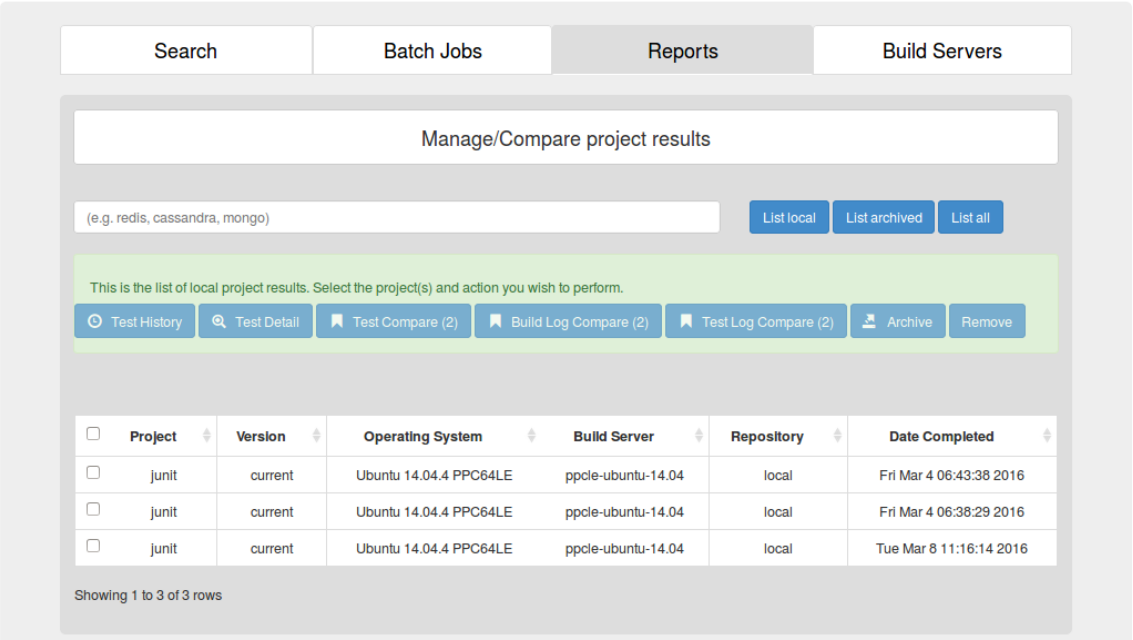


The following Jenkins window will be opened in your browser.



4.3 List build and test results

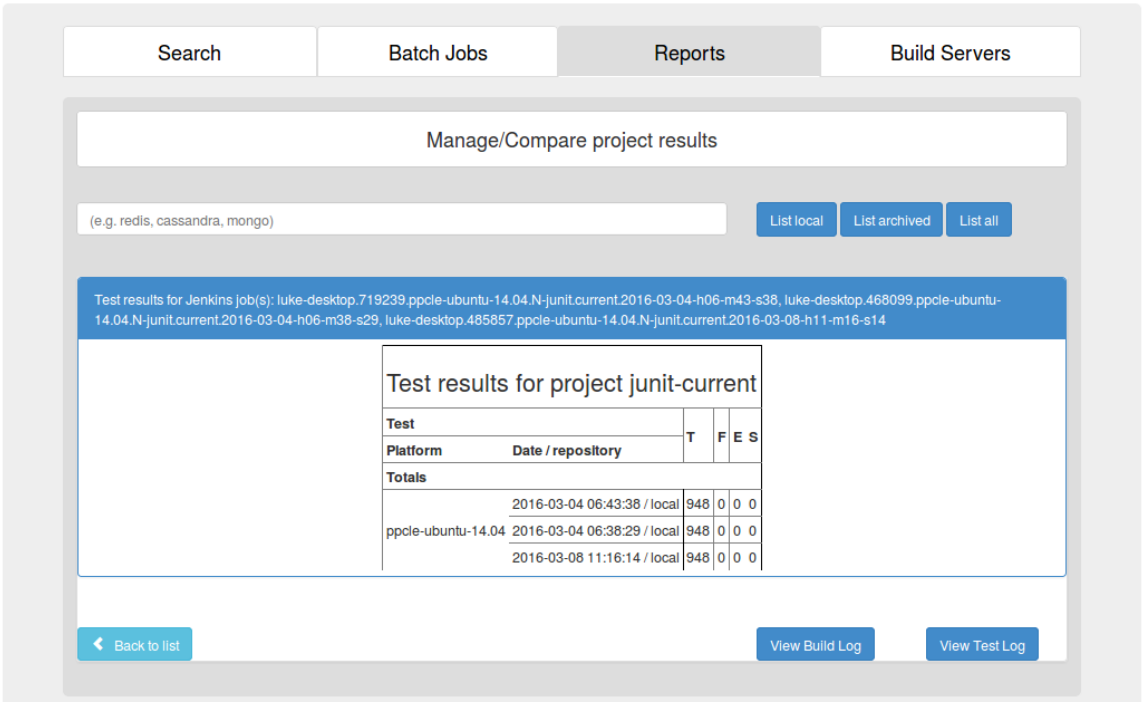
Build results are always presented locally first. If you are happy with the results and wish to share them with other users or simply wish to segregate completed work, copy them to external storage, then select the **Archive** button below. In the SuperVessel cloud, external storage is automatically configured to access swift storage that is associated with your user account, but it should be noted that these parameters can be modified via the Settings Menu to access other storage volumes via SFTP.



Note you can sort the data in the table above by clicking on the column headers.

4.4 Generate a Test History Report

Referencing the picture above, **Select** a project(s) and Press **Test History.** This yields:



The table above shows:

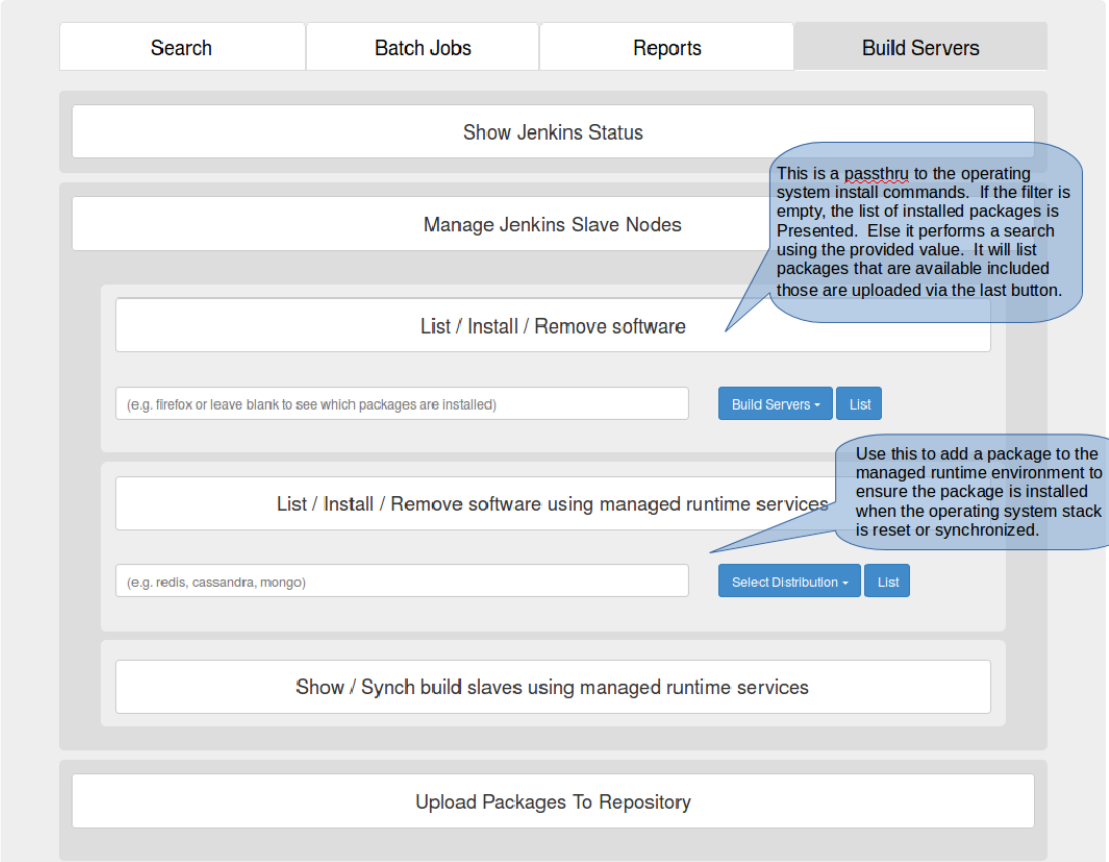
- Test results for three build instances of ‘junit’
- Each time 948 **T**ests were attempted with 0 **F**ailures, 0 **E**rrors, and 0 **S**kipped tests
- All tests were run on the same build server with this label: ppcle-ubuntu-14.04
- Buttons are presented at bottom to view the raw console logs for build and test

Notes:

The test case data for most projects show all zeroes. This is not a bug per se. It cannot be helped as the project does not use a well-known test framework. When this occurs, you should use the log compare tools and raw logs to analyze results.

4.5 Install missing packages

Install services are included on the Build Servers Tab shown below.

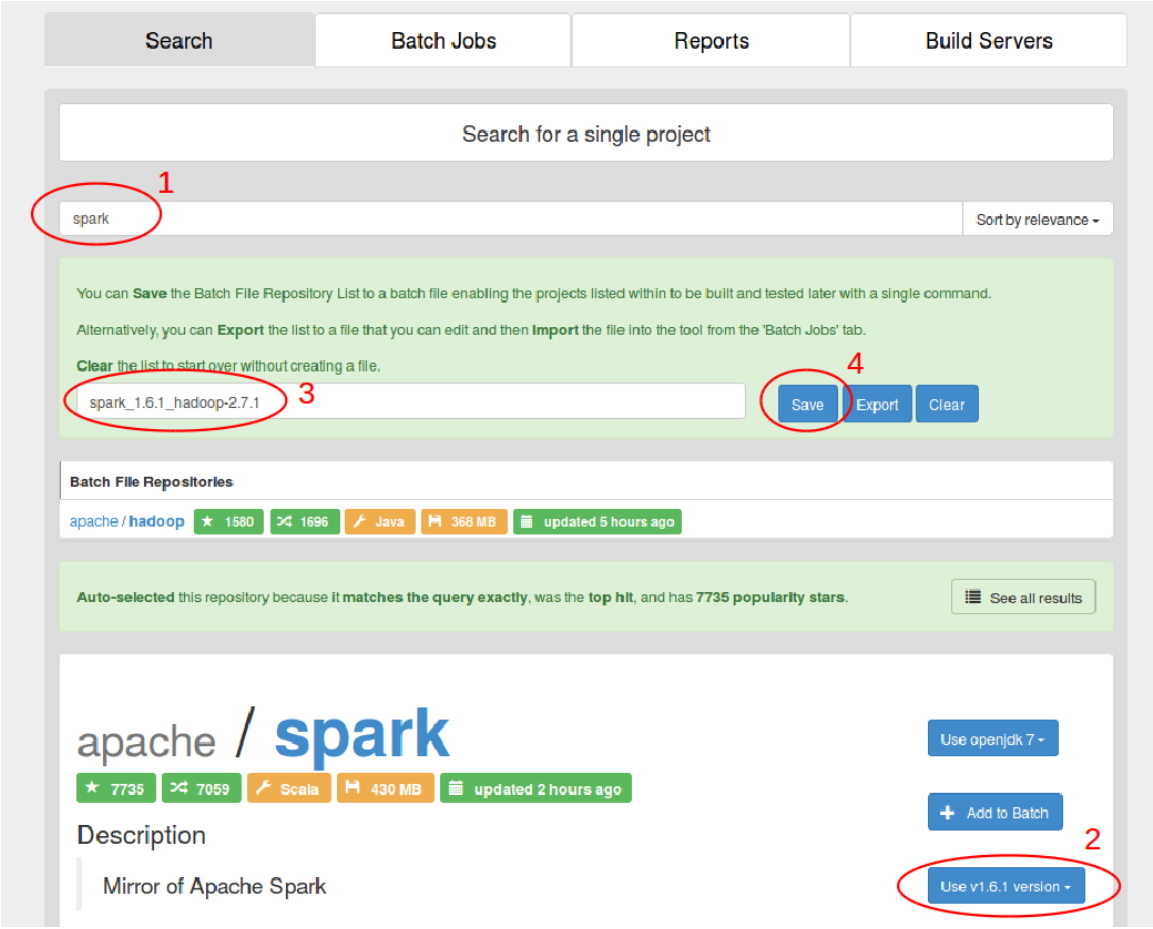


4.6 Create a batch file to test a solution

The default implementation of the batch file is to process all of the projects in parallel on the named build server. This type of batch file is created from the *Search Tab* → *Search for most commonly used projects* → *Search.* In this scenario, there is no inherent relationship between the projects, so they are built concurrently to save time. The default implementation is aligned towards ecosystem testing and is not discussed further in this document. Just note that care must be taken to exclude projects that are intended for Windows, OS X, and Android. These projects are also hosted at github.com.

This scenario is oriented towards a particular solution, so it is important to only include projects that are needed by the solution, to identify specific versions of each project, and to change the processing mode of the batch file to be synchronous.

The first step is to name the projects and versions. A screen shot is provided below. It shows the second project being added to a batch file, but the basic process is the same for each project. First, search for project x, select the desired version for project x, and then press **+Add to Batch** button. Note you can customize build and test commands for each project as they are saved to the batch file or you can modify them later as we will.

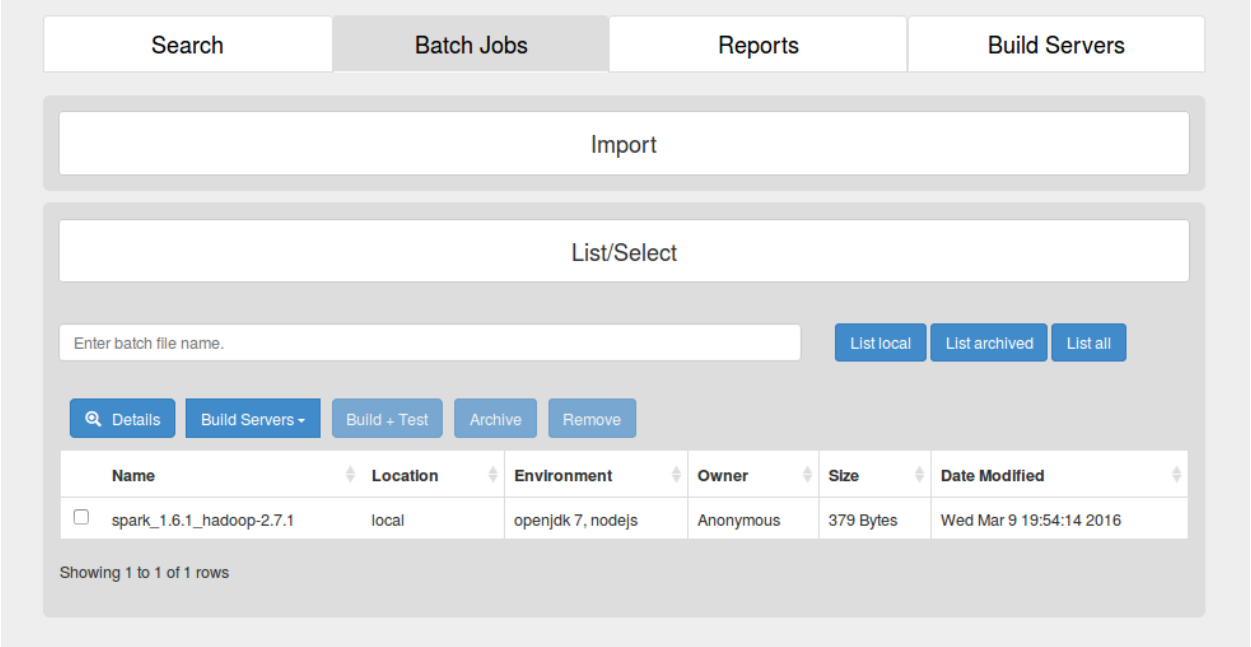


4.7 List batch files

Like build and test results, batch files are created locally initially. Once created, they can be archived but the general practice is to test them extensively first to ensure that they work as expected before archiving them as that makes them visible to other Autoport installations.

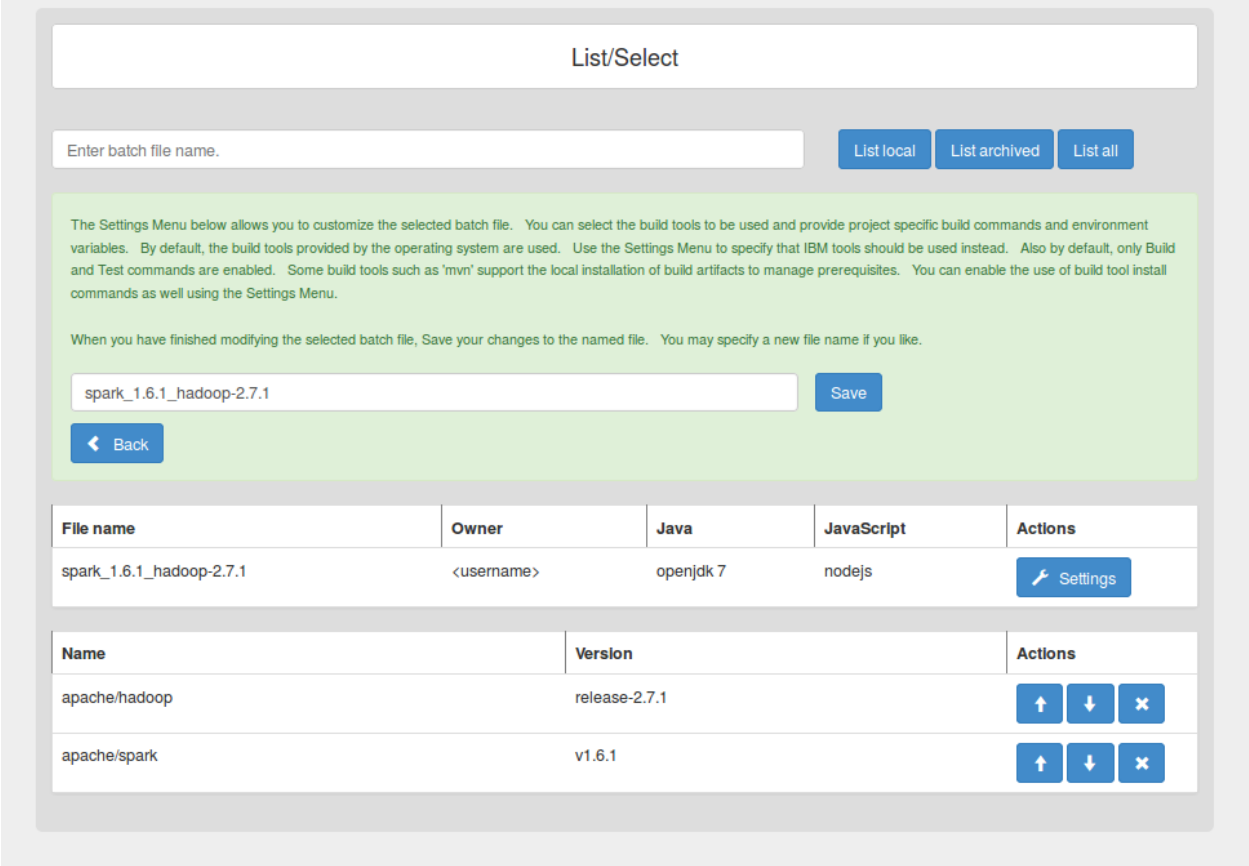
Here we pressed **List local** to find the batch file we just created.

Notice in the picture below that the environment indicates that openjdk 7 will be used. We want will want to change this to openjdk 8 and provide our own build commands.



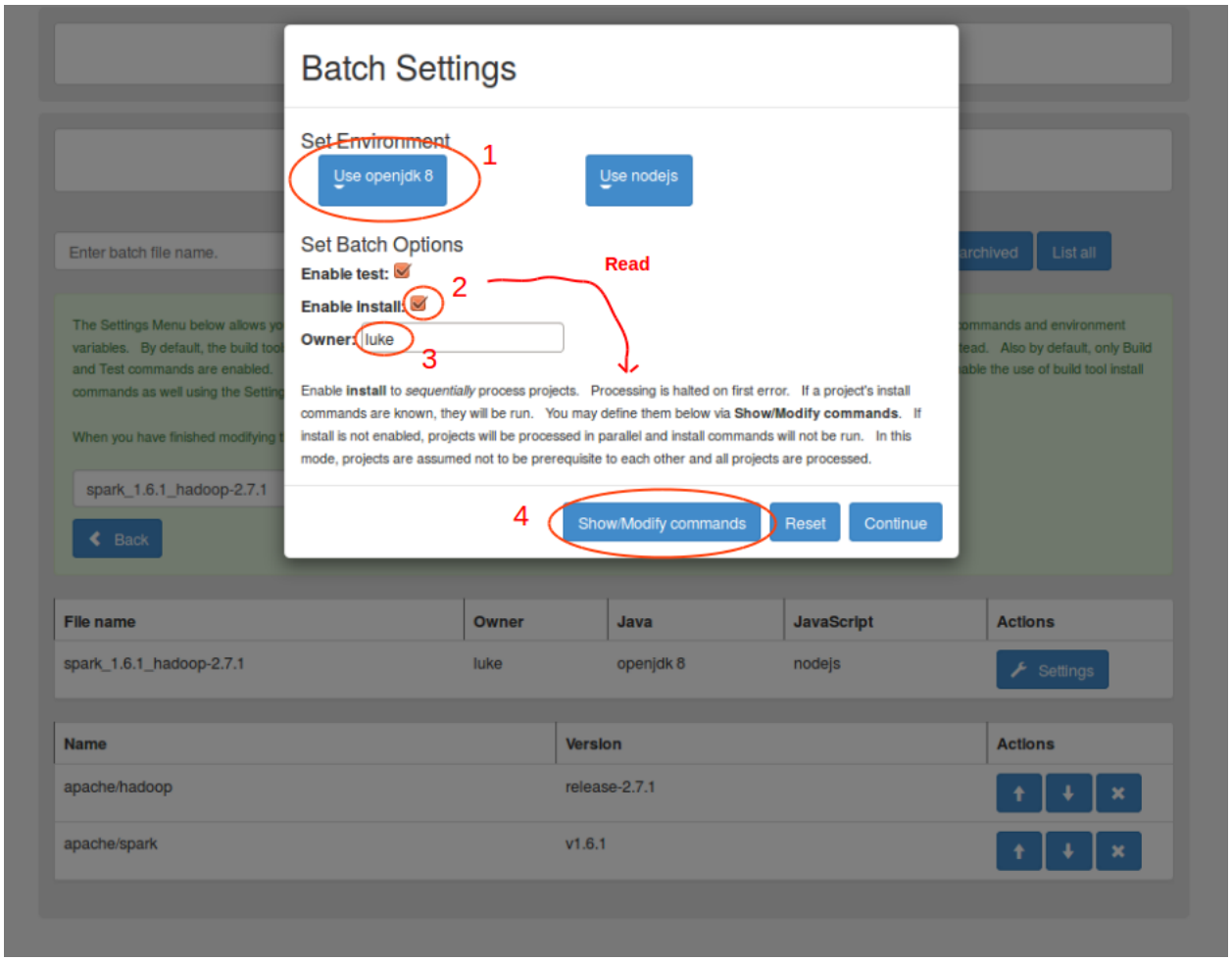
4.8 Customize a batch file

Given the picture above, one would **Select** the batch file and Press **Details** to show:



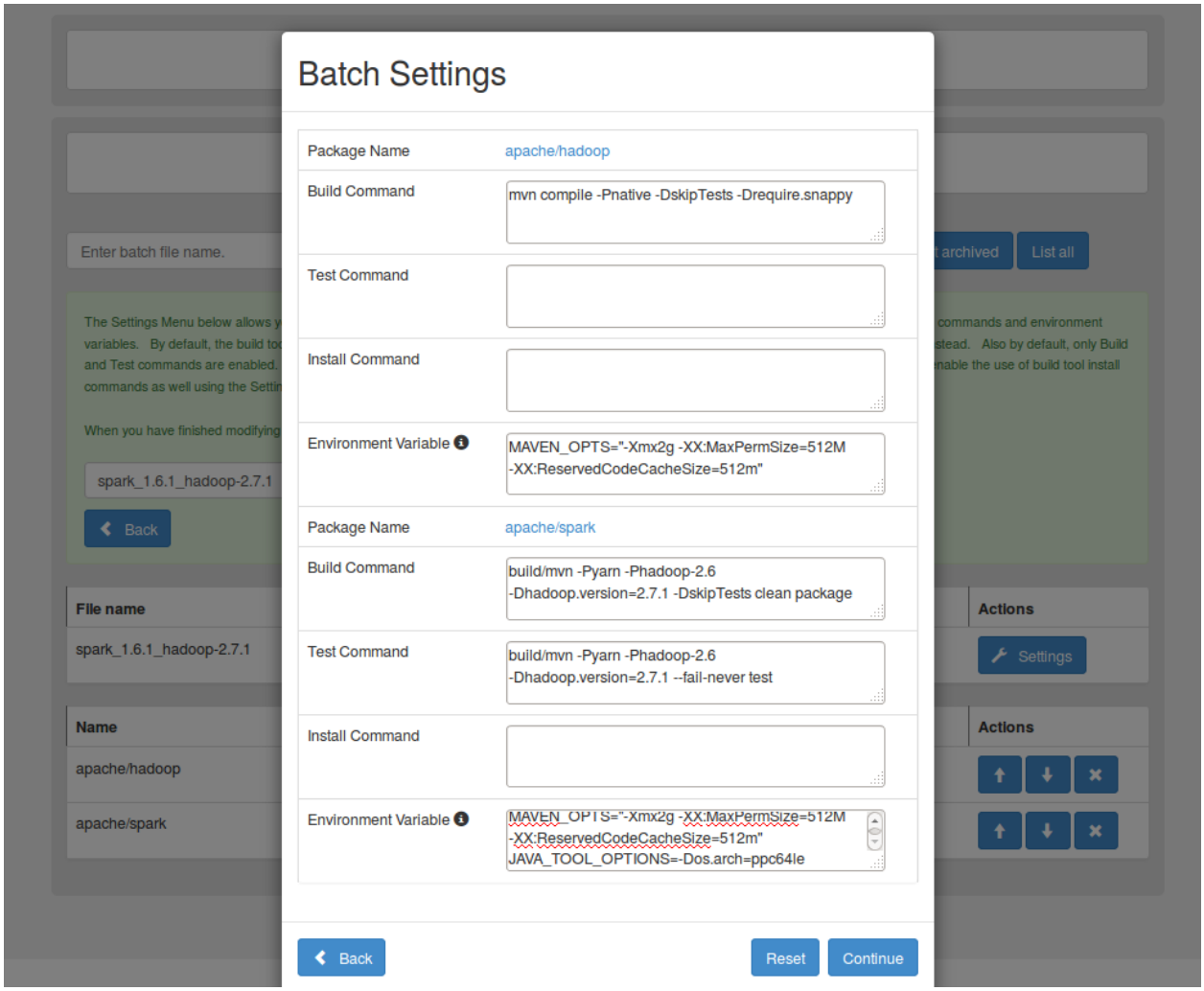
The projects in a batch file are processed in order. In this case, the order is correct, Hadoop will be built before Spark, so next we select **Settings** button to the right of batch file name. This will allow us to modify the Owner, Java, and JavaScript fields, which are included in the batch file header. They are common parameters that apply to all of the projects in the batch file.

After pressing the Settings Button, you should see something like this:



If you read the fine print in the picture above, you will see that the **Enable Install** option if selected changes the default processing to synchronous. The rationale for this is if you are going to build a project and install it, then this action must be completed before the next project is processed, but note you don't actually have to specify any install commands.

Next press the **Show/Modify commands** button. You should see a menu that looks like:



However, please note that we have updated the commands. Now we need to make sure that our changes are preserved in the batch file. This is accomplished by pressing **Continue** (or Back) and then **Save** in the underlying menu in the green area.

Notes:

The Hadoop test command is omitted as it takes hours to run. Only build commands are required. If the build command is not specified, then the project is skipped.

Install commands are not needed for Java projects built by Maven, because it maintains a local repository of jar files in the users home directory (~/.m2).

4.9 Running a batch file

First, generate the list of batch files so that you can select the specific batch file that you would like to run. Use **List local** as in section 4.7. Then, **select** the batch file. Choose one or more **Build Servers** to run the batch file. Finally, press **Build + Test**. Then, wait for a pop-up menu indicating that the batch job has been submitted. At this point, the batch job should be visible in the Reports tab where you can monitor and analyze batch job results.

Notes:

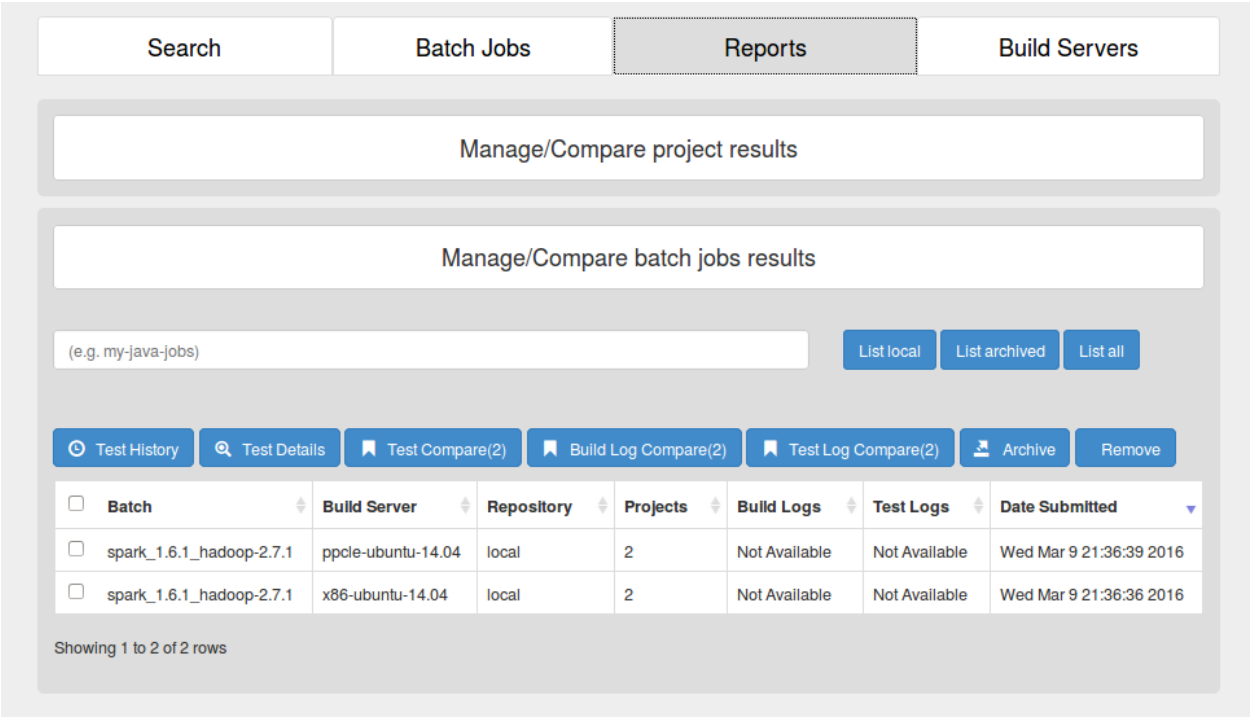
Jenkins windows are not automatically opened in your browser as there may be a very large number of projects in a batch file. The report tab shows the availability of build and test logs, which is the primary consideration for the generation of reports, but it is possible to monitor the progress of a batch file in a Jenkins window from the perspective of the build server where the batch job is being run. This may be accomplished by navigating as follows:

Build Servers Tab→ Manage Jenkins Slave Nodes → Show / [Clean or Sync] build slaves using managed runtime services → "click here" link of the relevant jenkins build server

4.10 List batch job test results

Press **List local** on the Reports Tab under Manage/Compare batch job results.

You should see something like:



You should see log statistics advance from “Not Available” to “1” to “2” and so on as project are built, tested, installed, and log files are transferred from Jenkins to Autoport. Log files are always copied even on failure, but it is worth noting that the execution of commands is halted on first failure. There is no way to ignore the failure of a test command even if the failure is owing to 1 test case out of 1000. Command processing is strictly predicated on command exit status. Finally, note that the screen is not refreshed automatically. You have to periodically press the **List Local** button to see the numbers advance. Again note that you can open a window into Jenkins to follow its processing as described in section 4.9.

4.11 Analyze batch job test results

This is similar to section 4.4.