

# 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](https://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](https://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](https://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](https://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,

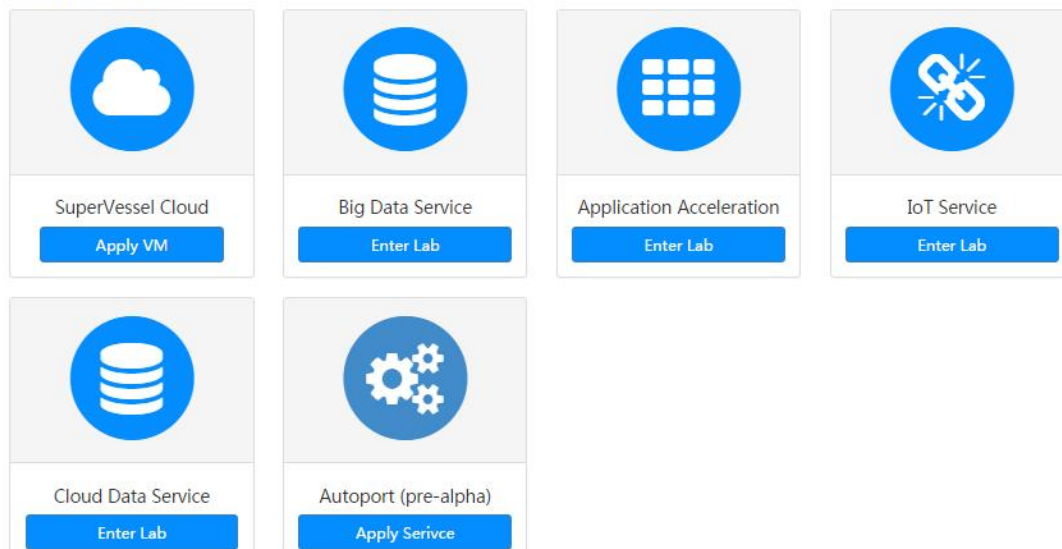
JavaScript, Java, HTML5, CSS, XML, Ruby, Chef, Flask, Rivets, and Bootstrap. It provides internal Yum and Apt servers and uses Apache web server.

## 1. Create an Autoport Cluster

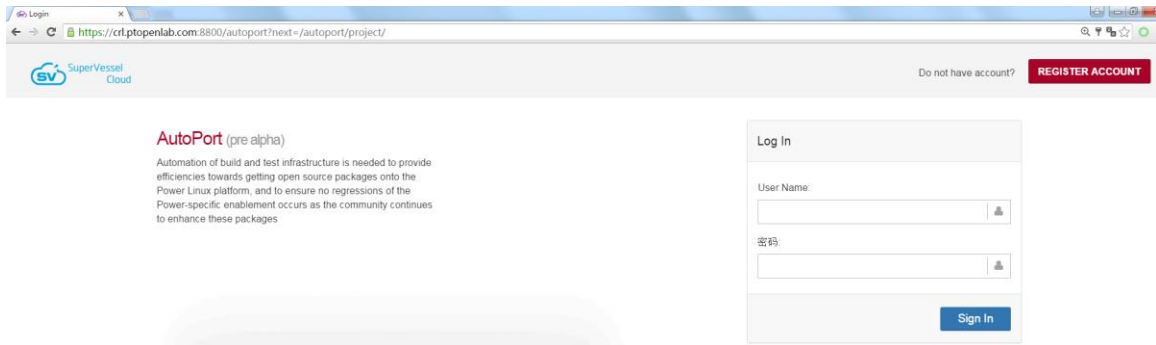
- (1) Access to <http://ptopenlab.com> on your browser, you will see a “Service Zone” on the page. Find “Autoport” icon and click “apply service”.



### Service Zone

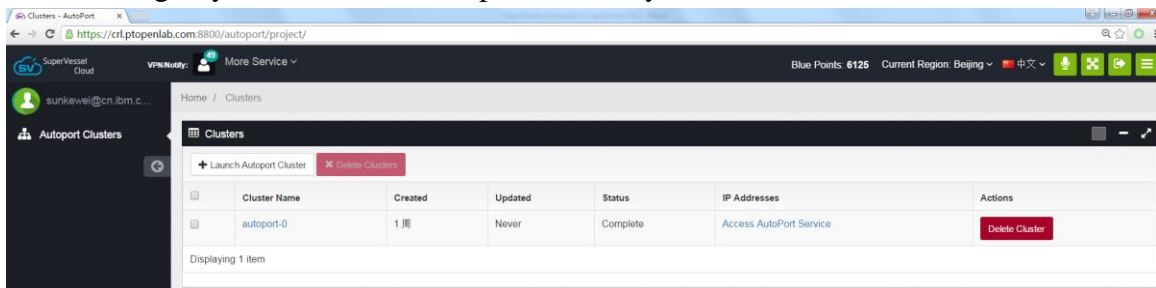


- (2) Once you have opened another page, you will be asked to login Autoport in SuperVessel.

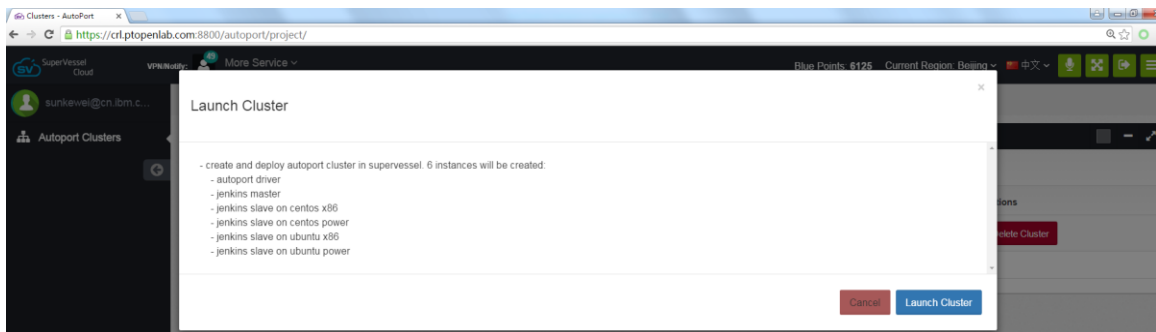


If you don't have an account in SuperVessel, You can register a new one by “REGISTER ACCOUNT” button on the top right of the page.

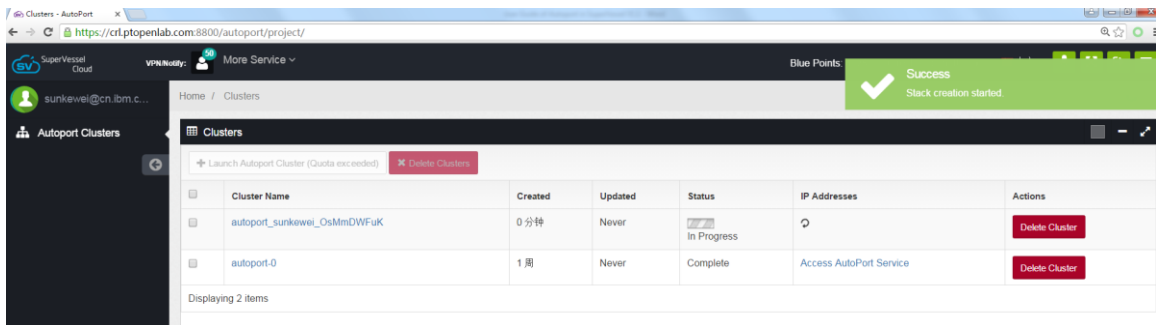
(3) After login, you can see the Autoport clusters you have created.



(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.

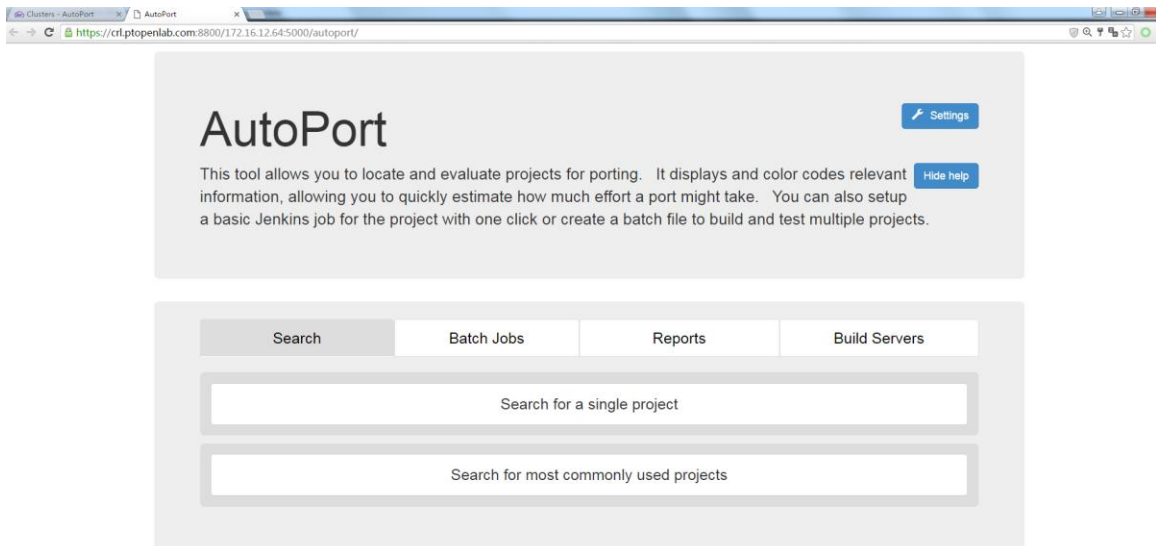


(5) Wait for several minutes until the status of the new created cluster becomes “complete”.



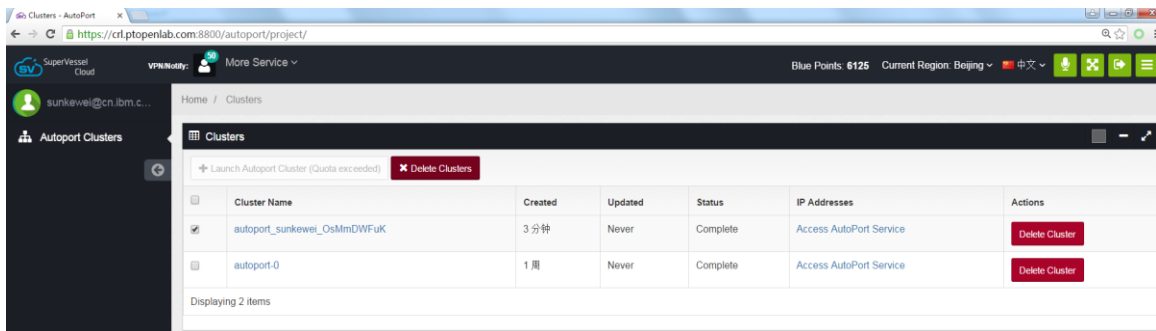
## 2. Access the Autoport Cluster

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

Search for a single project

junit Sort by relevance ▾

Auto-selected this repository because it matches the query exactly, was the top hit, and has 4610 popularity stars. See all results

junit-team / **junit**

★ 4610 🍴 1809 🗂️ Java 📦 27 MB 🕒 updated 7 hours ago

Description

A programmer-oriented testing framework for Java.

Language Composition

HTML - 12 KB  
Java - 1 MB  
JavaScript - 316 bytes  
CSS - 78 bytes

Build steps

The default options were inferred based on the existence of .travis.yml. When modifying build and test options ensure commands are semicolon delimited.

Select build options ▾

mvn verify javadoc:javadoc site:site

Select test options ▾

mvn test -ln

Environment variables separated by blanks or newlines applied as-is.

Select environment options ▾

YOUR OWN BUILD COMMAND

Callout boxes:

- Click here to access source code
- Color codes project to show desirability and ease of porting
- Language composition and size
- Build and test commands automatically produced by Autoport

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 screenshot shows the Jenkins web interface for the 'junit' project. The interface includes a search bar at the top, a project header with the 'junit' logo and statistics (4610 stars, 1809 forks, Java language, 27 MB size, updated 7 hours ago), a description 'A programmer-oriented testing framework for Java.', a 'Language Composition' pie chart, and a 'Build steps' section. On the right side, there are five blue buttons: 'Use openjdk 7 -', '+ Add to Batch', 'Use current version -', 'Select Build Servers -', and 'Build + Test'. Callout boxes provide instructions for each button: 'User can select the type and version of Java to be used. Default is shown.' for 'Use openjdk 7 -'; 'Default is current or master branch. You may select releases as well' for '+ Add to Batch'; 'By default, all build servers are selected. You may want to limit this by distro or platform.' for 'Select Build Servers -'; and 'Finally, press this to build and test. A log file is produced for each.' for 'Build + Test'.

Language	Size
HTML	12 KB
Java	1 MB
JavaScript	318 bytes
CSS	78 bytes

The following Jenkins window will be opened in your browser.

Project luke-desktop.485857.ppcle-ubuntu-14.04.N-junit.current

Click the workspace to see the source code.

Autoport log files:  
[.autoport-scratch/build\\_result.arti](#)  
[.autoport-scratch/test\\_result.arti](#)

The logs above are copied to autoport where they are used to generate reports.

Click #1 to see the console log. The blue ball means that the build and test script ran successfully. Red means an error occurred – incorrect build command, test case failure, ....

You don't need to do anything with this window. It is provided for monitoring purposes only, although you can kill a job that is in an infinite loop. Autoport has a very large timeout as some jobs take hours to test.

### 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.



Search

Batch Jobs

Reports

Build Servers

Manage/Compare project results

(e.g. redis, cassandra, mongo)

List local

List archived

List all

This is the list of local project results. Select the project(s) and action you wish to perform.

Test History

Test Detail

Test Compare (2)

Build Log Compare (2)

Test Log Compare (2)

Archive

Remove

<input type="checkbox"/>	Project	Version	Operating System	Build Server	Repository	Date Completed
<input type="checkbox"/>	junit	current	Ubuntu 14.04.4 PPC64LE	ppcle-ubuntu-14.04	local	Fri Mar 4 06:43:38 2016
<input type="checkbox"/>	junit	current	Ubuntu 14.04.4 PPC64LE	ppcle-ubuntu-14.04	local	Fri Mar 4 06:38:29 2016
<input type="checkbox"/>	junit	current	Ubuntu 14.04.4 PPC64LE	ppcle-ubuntu-14.04	local	Tue Mar 8 11:16:14 2016

Showing 1 to 3 of 3 rows

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:

Search

Batch Jobs

Reports

Build Servers

Manage/Compare project results

(e.g. redis, cassandra, mongo)

List local

List archived

List all

Test results for Jenkins Job(s): luke-desktop.719239.ppcle-ubuntu-14.04.N-junit.current.2016-03-04-h06-m43-s38, luke-desktop.468099.ppcle-ubuntu-14.04.N-junit.current.2016-03-04-h06-m38-s29, luke-desktop.485857.ppcle-ubuntu-14.04.N-junit.current.2016-03-08-h11-m16-s14

Test results for project junit-current

Test	T	F	E	S
Platform Date / repository				
Totals				
2016-03-04 06:43:38 / local	948	0	0	0
ppcle-ubuntu-14.04 2016-03-04 06:38:29 / local	948	0	0	0
2016-03-08 11:16:14 / local	948	0	0	0

Back to list

View Build Log

View Test Log

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.

The screenshot shows the Jenkins 'Build Servers' tab. At the top is a navigation bar with 'Search', 'Batch Jobs', 'Reports', and 'Build Servers'. Below this is a 'Show Jenkins Status' button. The main section is titled 'Manage Jenkins Slave Nodes'. It contains two primary sections for software management. The first section, 'List / Install / Remove software', has a search input field with the placeholder '(e.g. firefox or leave blank to see which packages are installed)', a 'Build Servers' dropdown menu, and a 'List' button. A callout bubble points to this section, stating: 'This is a passthru to the operating system install commands. If the filter is empty, the list of installed packages is Presented. Else it performs a search using the provided value. It will list packages that are available included those are uploaded via the last button.' The second section, 'List / Install / Remove software using managed runtime services', has a search input field with the placeholder '(e.g. redis, cassandra, mongo)', a 'Select Distribution' dropdown menu, and a 'List' button. A callout bubble points to this section, stating: 'Use this to add a package to the managed runtime environment to ensure the package is installed when the operating system stack is reset or synchronized.' At the bottom of the main section is a 'Show / Synch build slaves using managed runtime services' button. The very bottom of the interface has an 'Upload Packages To Repository' button.

## 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](https://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.

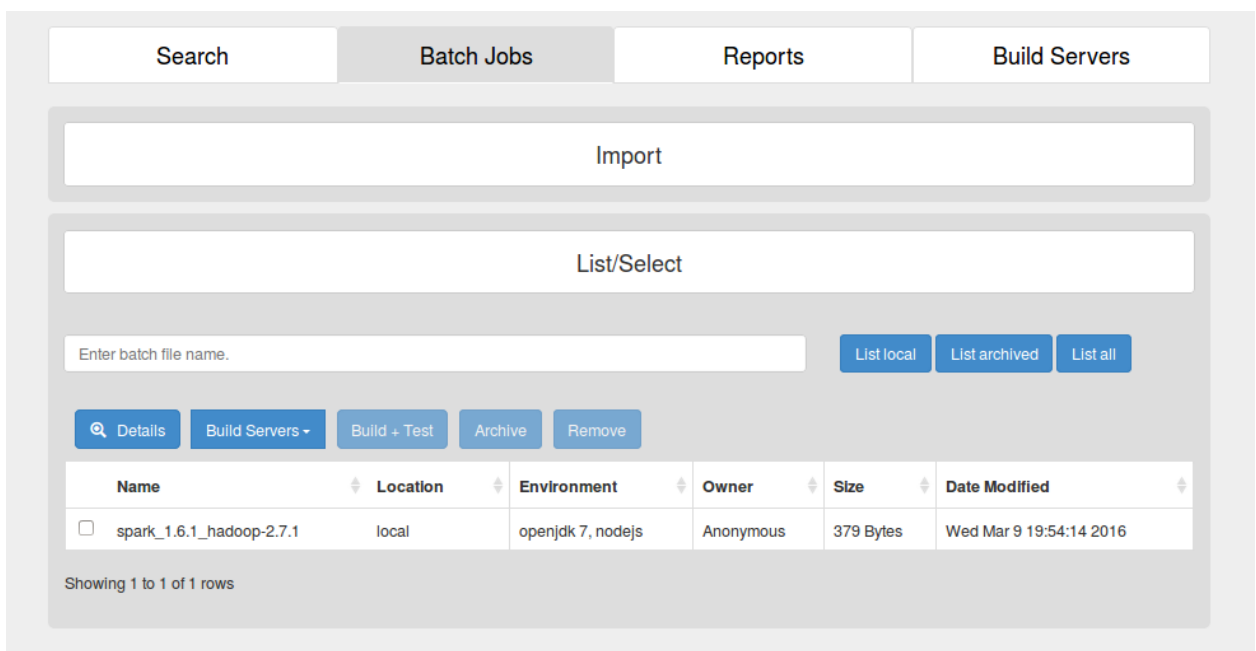
The screenshot displays the 'Batch Jobs' tab in a web application. At the top, there are navigation tabs: 'Search', 'Batch Jobs', 'Reports', and 'Build Servers'. Below these is a search bar labeled 'Search for a single project' containing the text 'spark' (circled with a red circle and labeled '1'). To the right of the search bar is a dropdown menu 'Sort by relevance'. Below the search bar is a green informational box with instructions on saving, exporting, and clearing the batch file repository list. Inside this box is a text input field containing 'spark\_1.6.1\_hadoop-2.7.1' (circled with a red circle and labeled '3') and three buttons: 'Save' (circled with a red circle and labeled '4'), 'Export', and 'Clear'. Below the green box is a section titled 'Batch File Repositories' showing a list of repositories. The first entry is 'apache / hadoop' with various statistics. Below this is a green box stating 'Auto-selected this repository because it matches the query exactly, was the top hit, and has 7735 popularity stars.' with a 'See all results' button. At the bottom, the 'apache / spark' repository is shown with its statistics and a 'Description' section containing the text 'Mirror of Apache Spark'. To the right of the repository details are three buttons: 'Use openjdk 7', '+ Add to Batch', and 'Use v1.6.1 version' (circled with a red circle and labeled '2').

## 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.



The screenshot shows the 'Batch Jobs' tab in the Autoport interface. At the top, there are four tabs: 'Search', 'Batch Jobs' (selected), 'Reports', and 'Build Servers'. Below the tabs is an 'Import' button. Underneath is a 'List/Select' section containing a text input field labeled 'Enter batch file name.' and three buttons: 'List local', 'List archived', and 'List all'. Below this is a row of action buttons: 'Details' (with a magnifying glass icon), 'Build Servers' (with a dropdown arrow), 'Build + Test', 'Archive', and 'Remove'. The main part of the interface is a table with the following columns: Name, Location, Environment, Owner, Size, and Date Modified. There is one row in the table with the following data: Name: spark\_1.6.1\_hadoop-2.7.1, Location: local, Environment: openjdk 7, nodejs, Owner: Anonymous, Size: 379 Bytes, Date Modified: Wed Mar 9 19:54:14 2016. At the bottom of the table, it says 'Showing 1 to 1 of 1 rows'.

Name	Location	Environment	Owner	Size	Date Modified
<input type="checkbox"/> spark_1.6.1_hadoop-2.7.1	local	openjdk 7, nodejs	Anonymous	379 Bytes	Wed Mar 9 19:54:14 2016

Showing 1 to 1 of 1 rows

## 4.8 Customize a batch file

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

List/Select

List localList archivedList all

The Settings Menu below allows you to customize the selected batch file. You can select the build tools to be used and provide project specific build commands and environment variables. By default, the build tools provided by the operating system are used. Use the Settings Menu to specify that IBM tools should be used instead. Also by default, only Build and Test commands are enabled. Some build tools such as 'mvn' support the local installation of build artifacts to manage prerequisites. You can enable the use of build tool install commands as well using the Settings Menu.

When you have finished modifying the selected batch file, Save your changes to the named file. You may specify a new file name if you like.

Save

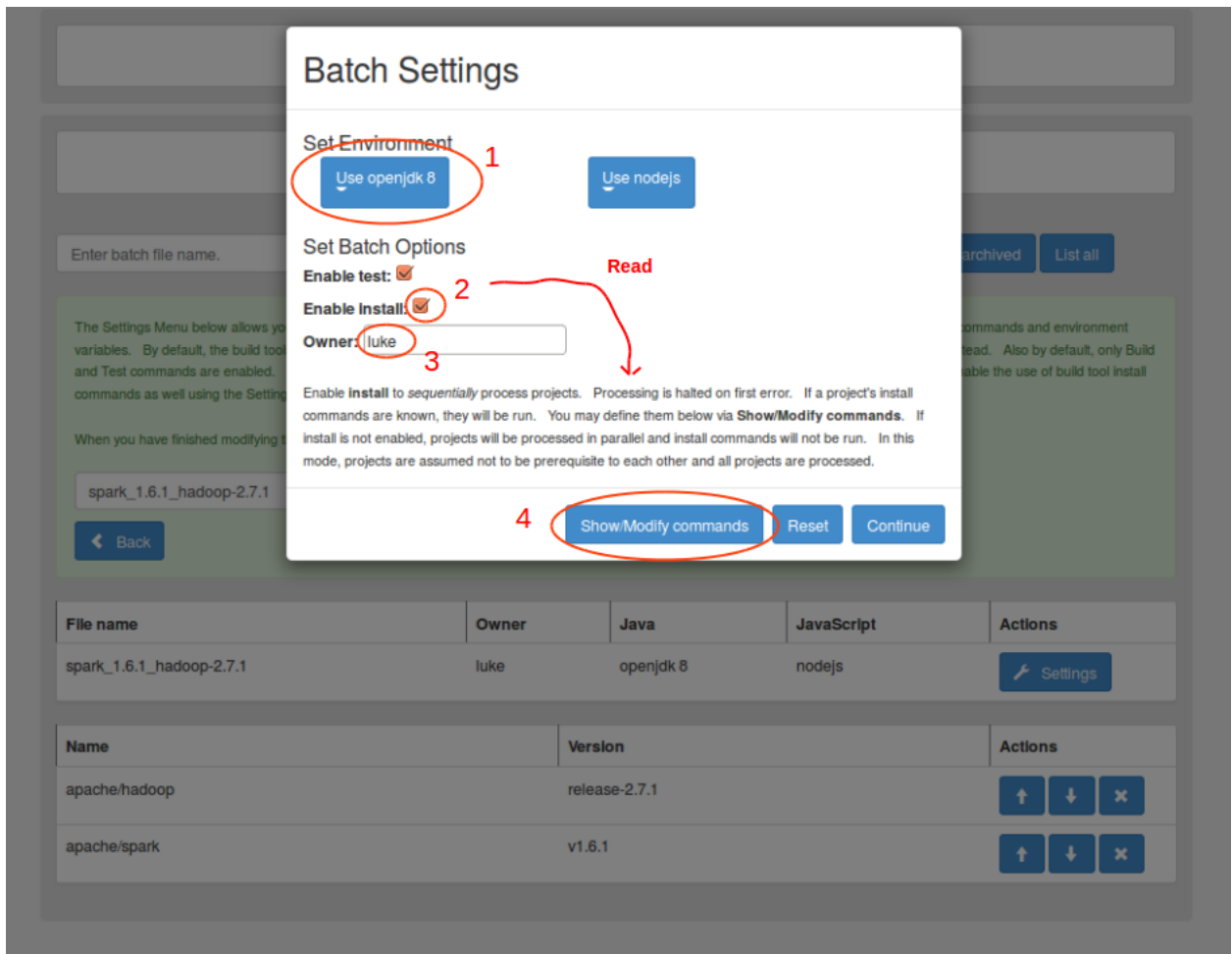
Back

File name	Owner	Java	JavaScript	Actions
spark_1.6.1_hadoop-2.7.1	<username>	openjdk 7	nodejs	<div>Settings</div>

Name	Version	Actions
apache/hadoop	release-2.7.1	<div>↑↓×</div>
apache/spark	v1.6.1	<div>↑↓×</div>

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:

**Batch Settings**

Package Name	apache/hadoop
Build Command	mvn compile -Pnative -DskipTests -Drequire.snappy
Test Command	
Install Command	
Environment Variable ⓘ	MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"
Package Name	apache/spark
Build Command	build/mvn -Pyarn -Phadoop-2.6 -Dhadoop.version=2.7.1 -DskipTests clean package
Test Command	build/mvn -Pyarn -Phadoop-2.6 -Dhadoop.version=2.7.1 --fail-never test
Install Command	
Environment Variable ⓘ	MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m" JAVA_TOOL_OPTIONS=-Dos.arch=ppc64le

Back Reset Continue

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:

<input type="checkbox"/> Batch	Build Server	Repository	Projects	Build Logs	Test Logs	Date Submitted
<input type="checkbox"/> spark_1.6.1_hadoop-2.7.1	ppc64-ubuntu-14.04	local	2	Not Available	Not Available	Wed Mar 9 21:36:39 2016
<input type="checkbox"/> spark_1.6.1_hadoop-2.7.1	x86-ubuntu-14.04	local	2	Not Available	Not Available	Wed Mar 9 21:36:36 2016

Showing 1 to 2 of 2 rows

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.