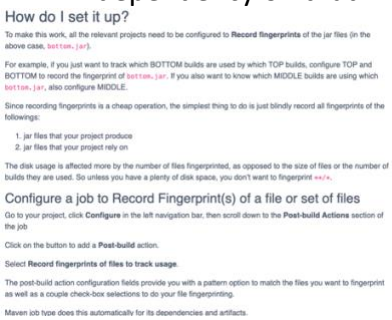
Jayden Banks

Week7-assesment
Part 1

Continuous Integration Jenkins

Features:

- Over 1400 plugins
- Supports entire dev life cycle
- Fingerprints allows you to track which version of a file is used by which version of a dependency on that file



How do I set it up?

To make this work, all the relevant projects need to be configured to **Record fingerprints** of the jar files (in the above case, `bottom.jar`).

For example, if you just want to track which BOTTOM builds are used by which TOP builds, configure TOP and BOTTOM to record the fingerprint of `bottom.jar`. If you also want to know which MIDDLE builds are using which `bottom.jar`, also configure MIDDLE.

Since recording fingerprints is a cheap operation, the simplest thing to do is just blindly record all fingerprints of the followings:

1. jar files that your project produce
2. jar files that your project rely on

The disk usage is affected more by the number of files fingerprinted, as opposed to the size of files or the number of builds they are used. So unless you have a plenty of disk space, you don't want to fingerprint `**/*`.

Configure a job to Record Fingerprint(s) of a file or set of files

Go to your project, click **Configure** in the left navigation bar, then scroll down to the **Post-build Actions** section of the job

Click on the button to add a **Post-build** action.

Select **Record fingerprints of files to track usage**.

The post-build action configuration fields provide you with a pattern option to match the files you want to fingerprint as well as a couple check-box selections to do your file fingerprinting.

Maven job type does this automatically for its dependencies and artifacts.

The directions are clear and easy to follow
I don't see a sandbox, but it is very detailed
https://www.jenkins.io/doc/book/using/fingerprints/

- Remote access API

Remote Access API

Jenkins provides machine-consumable remote access API to its functionalities. Currently it comes in three flavors:

1. XML
2. JSON with JSONP support
3. Python

Remote access API is offered in a REST-like style. That is, there is no single entry point for all features, and instea they are available under the `".../api/"` URL where `"..."` portion is the data that it acts on.

For example, if your Jenkins installation sits at https://ci.jenkins.io, visiting https://ci.jenkins.io/api/ will show just the top-level API features available – primarily a listing of the configured jobs for this Jenkins instance.
Or if you want to access information about a particular build, e.g.
https://ci.jenkins.io/job/Infra/job/jenkins.io/job/master/lastSuccessfulBuild/, then go to
https://ci.jenkins.io/job/Infra/job/jenkins.io/job/master/lastSuccessfulBuild/api/ and you'll see the list of functionalitie for that build.

What can you do with it?

Remote API can be used to do things like these:

1. retrieve information from Jenkins for programmatic consumption.
2. trigger a new build
3. create/copy jobs

The directions are great here as well and while I don't completely understand what this is doing, I could still implement it easily.
https://www.jenkins.io/doc/book/using/remote-access-api/

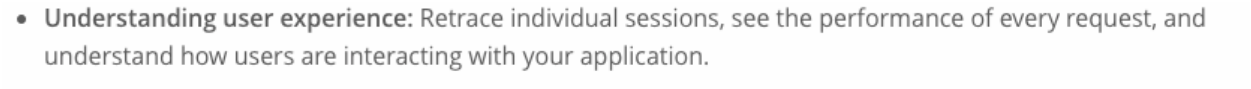
This software has been around since 2004, but has grown a lot in the past decade and changed from what it had been. It is popular, but some feel like it has many plugins that are obsolete, redundant or are not maintained properly. Many feel there are better more current options.

Real Time Error Monitoring Raygun

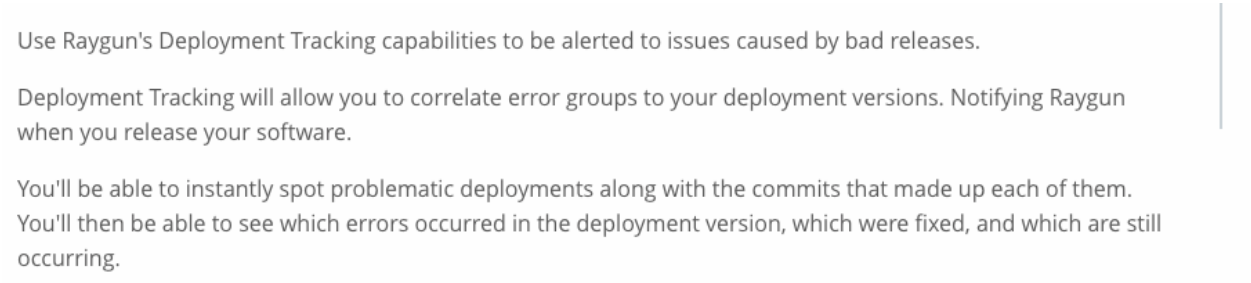
Features:

Jayden Banks

- Real User Monitoring automatically identify front end performance issues that cause slow pages



- Supports Javascript, React, Vue and Angular



- Track errors and issues with a bad release

The documentation was very well laid out and I had no issues going through it and gained a basic understanding of the software. Each language gives there unique set-up instructions and I bet I could have this set up in 30 minutes or less.

Part 2

Node runtime.js times
tinyArray
Insert: 37.726 µs
Append: 90.497 µs

smallArray
Insert: 50.357 µs
Append: 136.253 µs

mediumArray
Insert: 154.965 µs
Append: 144.091 µs

largeArray
Insert: 6.213ms
Append: 552.303 µs

extraLargeArray
Insert: 831ms
Append: 4.00ms

Jayden Banks

At first when using the smaller arrays, it looks like insert is quicker and more efficient (both are quick). After getting to the bigger arrays, you start to see insert fall behind append until its ratio is 200 times slower. Clearly append scales much better than insert because of linear like growth pattern as opposed to inserts exponential looking pattern.

Extra Credit: “Unshift is slower than push because it also needs to unshift all the elements to the left once the first element is added.” This means that as the array gets longer and longer the work to unshift 1 element into the array exponentially grows. Append on the other hand only has to focus on getting to the end of the array and adding a new number. Append never worries about what is already in the array and therefore grows in a linear, efficient, and scalable pattern.