Step 1:

Continuous integration tool:

Travis CI-

Getting started was no issue, the initial sign-up process was like CircleCI. There is a clear and abundant amount of documentation available to assist the user experience. Travis CI, like CircleCI, asks to connect to your repositories through a specific service (GitHub being one of them). Once linked, the interface is much like CircleCI, with the biggest difference being that the speed and accuracy of connecting your repo to this CI are much slower and more clunky than CircleCI Travis CI has a simple interface, this simplicity could help newer DevOps who would normally get lost with other CI tools. The pros far outweigh the cons, though this tool is a little slower, the clear documentation can help almost anyone to get started with CI tools. There are no sandboxes to test out the tool, but there are many resources outside of documentation to help the user (The Help page being one, this provides documentation AND a community to reach out to for questions).

Travis CI has been around since 2011, it ranks 7/10 in the most popular CI tools category, the most recent site travis-ci.org was shut down and accounts were migrated to travis-ci.com. This service remains free and mostly community run/supported. Currently TravisCI has many repositories on Github that have continued to be worked on to this day (10/1/2021). The most recent commit was on the 29th of September 2021, with over 900 contributors on one specific repo. TravisCI supports over 30 coding languages and is trusted by many great companies, one being Heroku!

Real Time Error Monitoring

Bugsnag-

Initiating Bugsnag was rather simple, after signing up and reading documentation, there is a vast quantity of resources available for setup and bug detection. The documentation provides information for each language that is available to the user and provides the steps for using the tool. Provides a tool for diagnosing and evaluating mobile, web, and desktop applications. UI is similar to Rollbar; projects can be found relatively easy and can be linked to repositories like Rollbar can.

Bugsnag was founded in 2013 and ranks 3/10 in the most popular Real Time Error Monitoring category. Bugsnag is a trusted resource used by many popular companies such as: lyft, tinder, Mercedes benz, pandora, and 6,000 more. Bugsnag has over 450 repositories on GitHub with each repository focusing on a specific platform or language. Each repository has over 30 contributors and has a commit with as little as 9 hours ago (10/1/2021, 11AM CST). Bugsnag is trusted by many of the best engineering teams and is built to handle projects at an enterprise-scale.

Step 2:

- extraLargeArray timing: The insert time is much slower than the append, the time compared to the append is roughly 1 second: 4 milliseconds.
- largeArray timing: The insert time is once again slower. ~ 9ms insert: ~600 µs append (append is faster, microseconds).
- mediumArray timing: ~ 190 µs insert: ~ 150 µs append.
- smallArray timing: ~ 50 µs insert: ~110 µs append.
- tinyArray timing: ~ 40 µs insert: ~ 90 µs append.

| **Array** | **Insert** | **Append** | **Winner** |
|---|---|---|---|
| ***extraLargeArray*** | 1 sec | 4 millisec | Append |
| ***largeArray*** | 9 millisec | 600 µs | Append |
| ***mediumArray*** | 190 µs | 150 µs | Append |
| ***smallArray*** | 50 µs | 110 µs | Insert |
| ***tinyArray*** | 40 µs | 90 µs | Insert |

Observation-

It appeared that the push() would prevail and remain the fastest option during the first three tests. The larger tests contained arrays that ranged from 100,000-1,000; the results changed once the array became smaller. Once the array size became 100 or less, the unshift() action became the victor. One of the greatest differences between unshift() and push() is that the latter places the new item at the end of the array and does not need to manipulate the array. Unshift on the other hand, places the item at the beginning of the array and moves the complete array. Push is much quicker during larger arrays because it only needs to add an item at the end. Unshift needs to push larger quantities and therefore takes more time. Towards the last few tests, smaller arrays allowed unshift to overshadow push, with less items to work with, unshift can perform its job much quicker, whereas push still needs to get to the end of the array. An example of how this would look would be like a strongman vs. a man with great speed (let’s call them Bolt). Bolt needs to take a weight to the far end of a bar where there is plenty of space, he runs and gets to the end before the strongman can push the other weights and place the new one at the bottom. As the weight decreases, the strongman can just push all of the weights with greater ease and can place the item before Bolt can make it to the end.