Plugin Health Scoring System

Google Summer of Code Program 2022 Project Proposal

Dheeraj Singh Jodha
jodhadheerajsingh@gmail.com
dheerajodha

Table Of Contents

Plugin Health Scoring System 1

Project Abstract 3

Project Description 3

Project Deliverables 4

High-Level Overview Diagram 5

Implementation 5

- Phase 1: Data Collection 5
- Phase 2: Data Aggregation 11
- Phase 3: Data Presentation 15
- Phase 4: Data Delivery 18

Timeline 20

- Phase 0: Pre-Community Bonding Period (Present - May 20th): 20
- Phase 1: Community Bonding Period (May 20 - June 12): 20
- Phase 2: Coding Phase 1 (June 13 - July 25th): 21
- Phase 3: Coding Phase 2 (July 25 - Sept 4th): 21
- Phase 4: Final Code Submission (Sept 5 - Sept 12th): 22

Future Improvements 23

Continued Involvement 23

Conflict of Interests 24

Major Challenges Foreseen 24

References 25

Relevant Background Experience 25

Personal Information 26

Open source contribution (Jenkins) 27

Language and Other Skill Set 29

Related Research and Work Experience 29

Reference Links and Web URLs 30

Project Abstract

This project aims to introduce a metric system to calculate the health score of each plugin within the Jenkins ecosystem and reflect the final scores on the Plugin Site for the plugin maintainers and users. The health score of each plugin would be shown on the UI with a detailed report of strengths and areas of improvement, making it easier for maintainers to maintain their plugins, and for users to decide if they should install/use a plugin.

Project Description

Plugin maintenance is an extremely important phase of a plugin's lifecycle. Using the analogy of "survival of the fittest", it is those plugins that are currently being widely used, which stood the test of time. Jenkins has always been active in ongoing developments which raises a need for thousands of plugins to catch up with their latest developments, in order to stay useful for a broader set of users. And this process requires sincere efforts from the maintainers' side.

There are a lot of highly useful initiatives going on within Jenkins to assist plugin maintainers. The expertise of the plugin maintainers can range widely, from being highly knowledgeable engineers who created some of the most useful plugins, to novice college graduates who are eager to learn and contribute but don't know where to start.

This project introduces the concept of “Plugin Health Score” which measures the plugin’s development maturity level with the help of accurate numeric figures. This score attempts to provide an accurate picture of how much care and help a plugin needs and makes it easier for every interested potential contributor to have an in-depth look at how they can fulfill those needs based on their expertise. This score also allows users to make a conscious decision before installing and/or using a plugin.

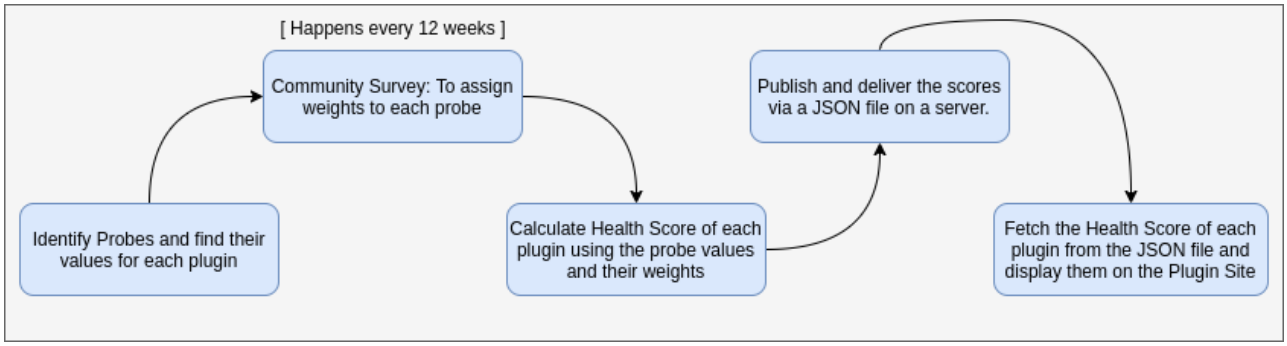
When the users will be able to see the quantitative results (in a form of an increase in plugin health score) due to contribution(s) to the plugin, it instantly makes them happy/proud/encouraged/satisfied. This would pave the way for increased plugin contributions and adoptions, which would imply a healthier plugin ecosystem within Jenkins and more satisfied plugin users.

I strongly believe that this project has the ability to create an impact on Jenkins by simplifying and improving things the way they are currently, which makes me excited for what this project can do in the long run.

Project Deliverables

a) A backend maven project which generates and deploys a JSON file containing the health score details of the plugins.
    i) Scrape each plugin’s repository and run some probes and collect the observations.
    ii) Calculate Plugin Health score with the help of the above observations.
b) Deploy the health scores via a JSON file similar to how Jenkins Update Center does it.
c) A separate section on the Jenkins Plugin Site which displays the detailed report of the health score of each plugin by fetching the JSON data generated above.
d) [Stretch Goal] Display Plugin health score on Plugin Manager.

High-Level Overview Diagram



Implementation

Phase 1: Data Collection

I've referred to the "Contributing to Open Source" document to determine the probes and how to analyze them. There can be 2 categories of probes, **administrative**, and **non-administrative probes**. This list of probes within each category can be easily expanded in the future once the basic proof of concept is ready.

A) Non-administrative probes:

Probe	How is it retrieved?

Check Parent POM Version	**Extract the current POM version:** By traversing the XML structure of the pom.xml file of a plugin and looking for the value of the <version> tag under the <parent> tag. **Compare it with the latest POM version:** TODO: Find a way to get the latest parent POM version and compare how far the plugin's current parent POM version is from it.
Is Jenkinsfile present?	Search the plugin's repo for the presence of Jenkinsfile.
Check base Jenkins version	Traverse pom.xml and find the value of the tag: <jenkins.version>, and check if its value is equal to the recommended Jenkins base version or not.
Check for the discontinued 'divBasedFormLayout' conditionals	Only for the Jenkins version >= 2.277.1, Search for the string "divBasedFormLayout" in the plugin's repo and decide if the plugin follows the currently recommended UI standards or not.
Spotbugs checks	Since newer parent poms have spotbugs enabled by default, we can calculate the score based on: ● Values of `spotbugs.effort` (min, less, more, max) ● Exclusion of Spotbugs checks, ● No. of SpotBugs warnings suppressed (needs more discussion) Find if the pom.xml contains the tags: <spotbugs.effort> and/or <spotbugs.failOnError>. If yes, then that means the plugin has spotbugs checks enabled which is a good thing (in some cases).

SCM URL	Check for <connection> tag in: <scm> <connection> </connection> </scm> And determine if it is still using unauthenticated access protocols (git:// protocol) or using the recommended https:// protocol.
Automatic dependency update checks	Check the presence of the .github/dependabot.yml file.
Check for deprecated JUnit assertThat	Search the plugin repo for "org.junit.Assert.assertThat", if any matches are found then that means the plugin maintainer is yet to replace the deprecated JUnit *assertThat* with Hamcrest *assertThat*.
Plugin BOM	Plugin BOM is used to manage dependencies, and avoid problems like *requireUpperBoundDeps*, *etc*. Search for the following tags in pom.xml, and note the plugin's LTS baseline: <groupId>io.jenkins.tools.bom</groupId> <artifactId>bom-2.289.x</artifactId> <version>...</version> And determine how far is the current BOM from that line.
Contributing Guide	Check for the presence of the CONTRIBUTING.md file in the plugin's repo
Check for wiki references	Since Jenkins wiki was made 'read-only' in Oct 2019, plugin doc. is now maintained in their own GitHub repos. So, search for the presence of "https://wiki*" in the plugin's repo, and decrease the score accordingly.
No. of open bugs	The count of open bugs in Jira for a given plugin, can be found within the link pointed by the "viewUrl" field in the Update Center.

B) Administrative probes:
(All the probes in this section can be done with the help of GitHub APIs)

Probe	How is it retrieved?
No. of open PRs (current), excluding the ones by dependabot	*https://api.github.com/repos/{owner}/{repo}/pulls?state=open*
No. of open PRs by dependabot	Scan through those objects which has "login": "dependabot[bot]" via the route: *https://api.github.com/repos/{owner}/{repo}/pulls?state=open*
No. of open PRs (average)	Keep track of the number of open PRs.
No. of open issues	Count the number of objects in the output of the route: *https://api.github.com/repos/{owner}/{repo}/issues?state=open*
Average time to review PRs	Keep track of when the review comments on a PR were received, via: *https://api.github.com/repos/{owner}/{repo}/pulls/{PR-number}/reviews*
Average time to merge approved PRs	Keep track of the time when the *"state": "APPROVED"* was seen for the PRs via the route: *https://api.github.com/repos/{owner}/{repo}/pulls/{PR-number}/reviews*
Average time to release after PR is merged	Keep track of when the PRs are merged and when the next release happens, via: *https://api.github.com/repos/{owner}/{repo}/releases*

Code Snippets:

Aim: *POC of how a probe value can be calculated.*
For the probe “Check Parent POM Version”, find its current value within a plugin’s pom.xml file. Then update the parent using maven and find out the

latest version of the parent, and compare how far both the values are.

To traverse the pom.xml file, here it is done using ‘DocumentBuilderFactory’ and ‘XPathFactory’, but it can also be done using JDOM.

To run shell commands in Java, we can use ‘Runtime’ and ‘Process’.

Since we’re leveraging the Linux Shell commands, it already makes things like ‘data collection’ a lot easier to do. So, this process can be performed for each of the other probes as well.

1. Run Shell Commands

```java
/**
*
* This method provides the functionality to run shell commands
*
* @param   inputCommand  the command which needs to be executed on shell
* @return  the output of the shell command
*
*/

public static StringBuffer runShellCommand(String inputCommand) throws IOException, InterruptedException {
    Runtime run = Runtime.getRuntime();
    Process pr = run.exec(inputCommand);
    pr.waitFor();
    BufferedReader buf = new BufferedReader(new InputStreamReader(pr.getInputStream()));
    String line = "";

    StringBuffer result = new StringBuffer();

    while ((line=buf.readLine())!=null) {
        result.append(line + "\n");
    }

    return result;
}
```

2. Get the value of a tag within the pom.xml file

```java
/**
*
* This method provides the functionality to parse a pom.xml file and get the value of any XML tag.
*
* @param   tagPath hierarchical path of the tag from the root element
* @return  the value within the XML tag
*
*/

public static String getTagValueFromXML(String tagPath) throws IOException, InterruptedException,
ParserConfigurationException, SAXException, XPathExpressionException {
    Document doc;
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(false);
    DocumentBuilder builder;

    builder = factory.newDocumentBuilder();

    doc = builder.parse(rootDir + "/parameterized-trigger-plugin/pom.xml");

    XPathFactory xPathFactory = XPathFactory.newInstance();
    XPath xpath = xPathFactory.newXPath();

    XPathExpression xPathExpression = xpath.compile("//project/name/text() | //" + tagPath + "/text()");

    Object result = xPathExpression.evaluate(doc, XPathConstants.NODESET);

    NodeList nodes = (NodeList) result;

    return nodes.item(0).getNodeValue();
}
```

3. Driver Code

```java
public static void main(String[] {
    runShellCommand("git clone https://github.com/jenkinsci/parameterized-trigger-plugin.git");

    float currentParentVersion = Float.parseFloat(getTagValueFromXML("parent/version"));

    System.out.println("Current Parent POM Version: " + currentParentVersion + "\n");

    System.out.println(runShellCommand("mvn versions:update-parent -f /home/user/eclipse-
workspace/Projects/parameterized-trigger-plugin"));

    float latestParentVersion = Float.parseFloat(getTagValueFromXML("parent/version"));

    System.out.println("\nLatest Parent POM Version: " + latestParentVersion);

    System.out.println("How far is current parent version, from the latest version? Answer: " +
(latestParentVersion-currentParentVersion));

    // Delete the repository after all the operations are done
    runShellCommand("rm -rf parameterized-trigger-plugin");
}
```

4. Output

Phase 2: Data Aggregation

1. **Aim:** This phase of the project involves taking major inputs from the Jenkins Community to determine the weights for each of the probes we identified in the previous phase and use them to calculate the plugin health score.

2. **Community Survey:** We'll conduct a survey that will form community standards, meaning, we'd have mutually decided specific weights for each of the probes.
    a. This survey can be conducted on the Developer Mailing List or via a Google Form or some other medium.
    b. The survey should be small and quick to answer, and each question

should encourage the audience to take a stand and not just stay neutral.
    c. If we don't get enough votes, we'd continue our development process with some default values for the time being.

3. **An Exception:** There will be a few special probes corresponding to some specific plugins that will be "maintainer-centric", meaning, the weights of those will be assigned differently.
    a. For example, let's say based on community standards, the decided weight for "Code Coverage by Automated Tests" is 0.5 (maximum). So, all the maintainers of the plugins would be required to increase their plugin's code coverage in order to improve their health scores.

    b. However, there are some specific library plugins (like okhttp API) that do not need to have high code/test coverage. So, we'd need to excuse some plugins from using the community-decided weights for some specific probes and give preference to a maintainer's say.

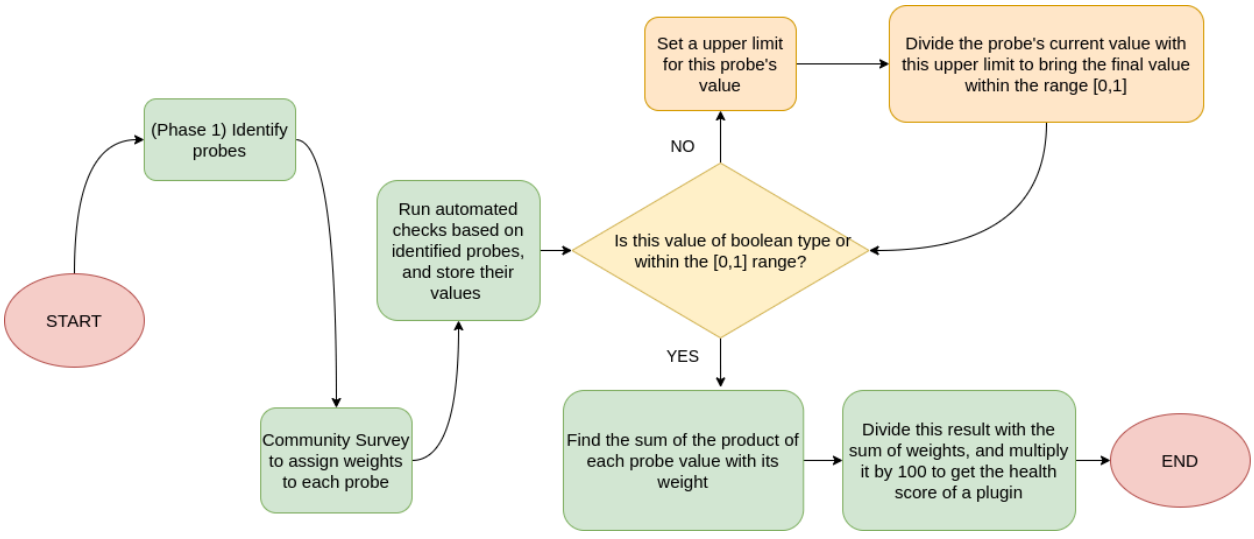
4. **Frequency of the survey:** Weights of the probes would change over time due to multiple reasons, so it's a good idea to conduct this community survey after a regular period of time.
    a. Too high frequency would lead to confusion and maintainer fatigue.
    b. Too low frequency would decrease the value of the scoring system.
    c. Review the weights to make sure that they still align closely to their level of importance.
    d. Whenever needed, adapt/change/re-adjust the weights every LTS (every 12 weeks) based on their (new) importance with respect to plugin maintenance.

5. **Communication Medium:** Our aim is to make sure to notify the maintainers well in advance that the rules have changed. We can make use of:
    a. Developer Mailing list
    b. UI where we display the health scores
    c. LTS Changelogs

Calculating Health Score

- Scores can be calculated by using a weighted decision matrix that will have all the probes (from Phase 1) along the rows and all the plugins along the columns.
- There will be an additional column called '**Weight**' which stores some fixed values, say, in the range of 0 to 0.5 (both inclusive).
- These weight values per probe tell us how important a probe is with respect to plugin management/maintenance.
- For example, the **weight value of 0.5** for the probe 'Automate dependency update checks' means that it is **highly important** for a plugin to have Automated Dependency update checks enabled in order to have smoother plugin maintenance for the maintainer and an easy plugin usage experience for the users.

Flow Chart of the Algorithm:



Weighted Decision Matrix

	Weight	**Plugin 1:** parameterized-trigger	**Plugin 2:** git-plugin	**Plugin 3:** gitlab-plugin
Parent POM	0.4	0	1	1
Jenkinsfile	0.3	1	1	1
Jenkins base version	0.2	0	0	0
'divBasedFormLayout'	0.1	1	0	0
Spotbugs	0.3	0	1	0
SCM URL	0.1	1	1	1
Automated dep. checks	0.5	1	1	1
Plugin BOM	0.4	1	1	1
Contributing guide	0.2	0	1	0
No. of open PRs [max=500]	-0.1	10/500 = 0.02	17/500 = 0.034	202/500 = 0.404
Average time to review PRs (in days) [max=50]	-0.1	10/50 = 0.2	3/50 = 0.06	15/50 = 0.3
Average time to merge approved PRs (in days) [max=75]	-0.1	5/75 = 0.06	2/75 = 0.026	50/75 = 0.66
Sum of Values with weight	2.4	0.3+0.1+0.1+0.5+0.4-0.1*0.02-0.1*0.2-0.1*0.06 = 1.486	0.4+0.3+0.3+0.1+0.5+0.4+0.2-0.1*0.034-0.1*0.06-0.1*0.026 = 2.384	0.4+0.3+0.1+0.5+0.4-0.1*0.4404-0.1*0.3-0.1*0.66 = 1.601
Total Score (out of 100**)**		(**1.486***100)/2.4 = 61	(**2.384***100)/2.4 = 99	(**1.601***100)/2.4 = 66

NOTE:

- If we add more probes, the **algorithm can automatically readjust itself** to always calculate the score out of 100.
- However, whenever we introduce a new probe, make sure to have already figured out its weight and its max value limit (if the probe value is of non-boolean type).
- The negative weight of a probe signifies that with the increase in value of this probe, the health of the plugin will decrease, and vice versa.

Health Percentage categories:

- Between 0 - 40%: Low health.
- Between 41 - 75%: Medium health.
- Between 76 - 100%: High health.

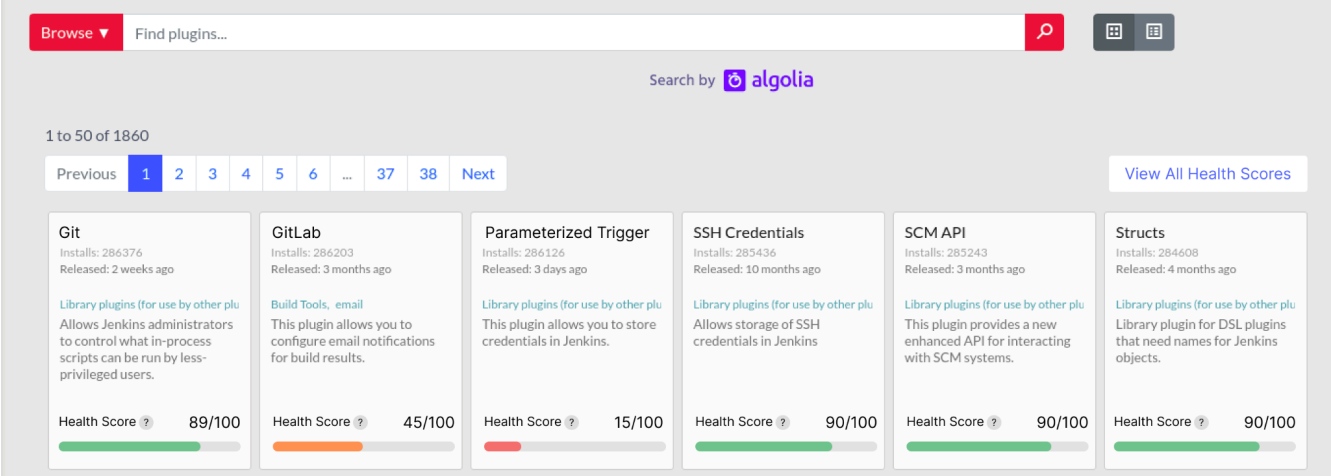
Phase 3: Data Presentation

☐ Health Score: Quick Peek

- **WHAT?** Proposing a way to present health scores on plugin cards on the Plugin site @ https://plugins.jenkins.io/ where everyone naturally goes to find out about Jenkins plugins.

- **WHY?** To give visitors of the plugin site a way to quickly glance over the health score of all the plugins in view, as shown below.

- **HOW?** This plugin card component's code can be found here. We'd just need to delete the code which renders the *<Icon>* and *<Developers>* components, and instead, create a new component something like *<HealthScoreBar>* which renders the required information as shown below. Making a progress bar is not difficult, a good article to refer to would be this.



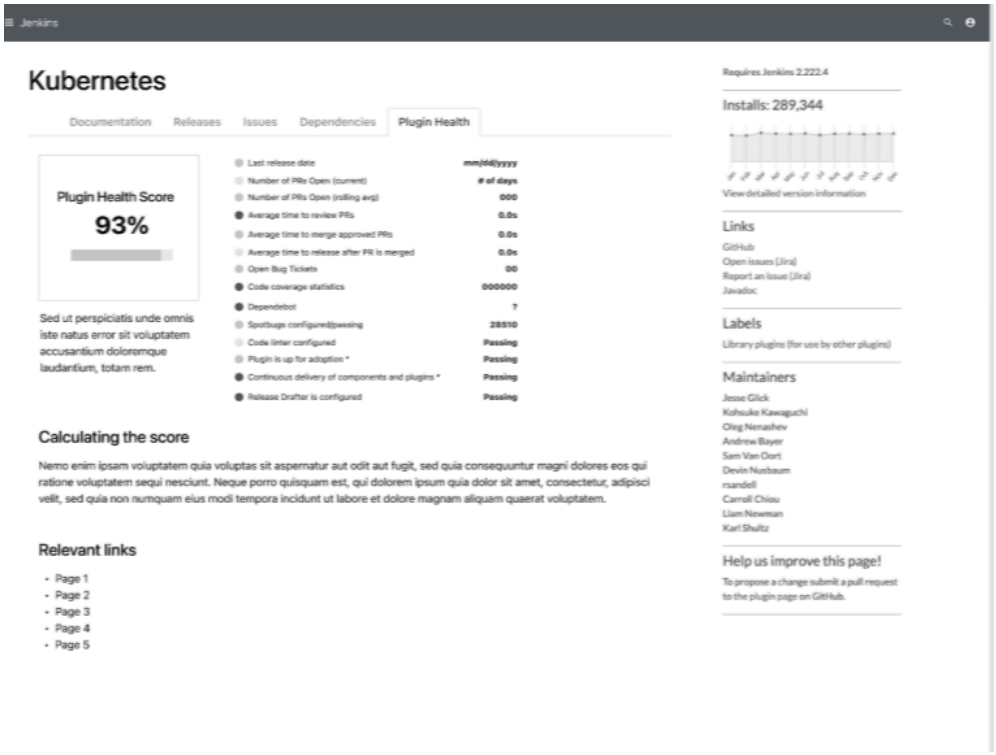
☐ Individual Health Score: Closer Look

- **WHAT?** This page covers a detailed report on the health score of an individual plugin.

- **WHY?** After looking at the health score of a plugin, the plugin maintainers or (potential) users would want to know how it was calculated, what were the parameters taken into consideration, how well the plugin performed on those criteria, and how exactly the plugin can be improved. All these questions would be answered on this section of a plugin's page.

- **HOW?**
    - This can be achieved by, first of all, creating a component, say, 'HealthScore' representing this new page at the location: *plugins/plugin-site/src/components* within the plugin-site repo.
    - Then add a new ID (say 'score') inside the template file of the plugin, followed by calling the component whenever the current page's ID matches the ID of the health score page.
    - There will also be related links from the "Adopt a Plugin Tutorial"

for the data points, guiding the maintainers (or contributors) on how to improve the health score easily.



Pic Credits: Jake Leon.

☐ Filter Plugins

- As per the discussions, we'd move forward with a percentage-based rating system where the plugin's health scores will be out of 100.
- On the left side of the site are some filtering options already.
- We plan to provide an add-on functionality that would allow visitors to view the plugins sorted (asc or desc) by their health scores.
- This would be helpful for contributors who're looking to contribute to plugins that need the most help, or to the users who are curious about which plugins are most well-maintained.

Phase 4: Data Delivery

Create a new Update Center-like project

- **WHAT?** Idea is to create a maven project similar to the current Update Center which will offer similar JSON object files as an output but would solely focus on the Health Score of the plugins.

- **WHY?** Current Update Center is already processing such a huge data of 1000s of plugins and adding more data points, related to health scores, into each plugin might make this service a little bit more difficult to maintain by the maintainers. Making use of a separate service would eliminate the overhead of maintaining more data for the current Update center and make this new service lightweight, focused on a specific area, and hence easier to maintain.

- **HOW?**
    - In the future, when we decide to display health scores on Plugin Manager in each controller, we'd need to make sure that the JSON file from where we're fetching the scores, needs to be as small as possible.
    - At the same time, we'd need to make sure that there's another JSON file containing the detailed breakdown of the health score, which will be used by the Plugin Site
    - So, we'd need to deliver by deploying 2 separate JSON files for 2 separate use-cases.
    - **NOTE:** The actual implementation needs to be reviewed or further explored later. But this would be one of the prototypes which would serve our purpose for the time being.

JSON object used by Plugin Manager to render health score:

```
"git":{

  "name":"git",
  "title":"Git",
  "doc":"https://plugins.jenkins.io/git",

  “healthScore”:38,

}
```

JSON object used by Plugin Site to render detailed health score:

```
"git":{

  "name":"git",
  "title":"Git",
  "doc":"https://plugins.jenkins.io/git",

  “healthScore”:38,
  “healthScoreParameters”: [
    {isParentPomLatest : 0.4},
    {isJenkinsfilePresent : 0.3},
    {checkJenkinsBaseVersion : -0.2},
    {checkDeprecatedLayouts : -0.1},
    {isSpotbugsEnabled : 0.3},
    {isScmUrlUpdated : 0.1},
    {isAutomatedDepCheckEnabled : 0.5},
    {isUsingPluginBom : 0.4},
    {isContributingGuidePresent : 0.2}
  ]
}
```

NOTE: The naming convention of the objects of “healthScoreParameters” can be improved to use less character space.

Timeline

Phase 0: Pre-Community Bonding Period (Present - May 20th):

- Discuss with the project mentors the necessities and requirements.
- Continue the discussions and attend meets/office hours.
- Contribute to Jenkins to strengthen my knowledge of the Jenkins ecosystem.
- Experience firsthand the perception of an important target persona of this GSoC project: the plugin maintainer, by continuing contributing to the “Parameterized Trigger” plugin under the “Adopt a Plugin” initiative.

Phase 1: Community Bonding Period (May 20 - June 12):

- Get acquainted with the mentors and the other community members by scheduling meetings and having discussions related to the project.
- Discuss approaches for the project in more detail with the mentor/s. Work more on **creating** the **design document** for the entire project to make the workflow of the entire development process clean and documented so that the development process is smooth.
- Prepare the Community Survey: Identify the set of probes to get the inputs from the community on how much weight we should assign to each one of them.
- Work on optimizing the health-score-calculation algorithm as per the use case.
- Go through the site layout of the Update Center to understand how it all comes together.

Phase 2: Coding Phase 1 (June 13 - July 25th):

Week #	Tasks	Deliverables
Week 1 & 2 (June 13th - June 26th)	• Develop a maven project which collects the data related to each of the pre-decided probes from a handful of plugins' repo.	• Data Collection POC
Week 3 & 4 (June 27th - July 10th)	• Code the logic of the Health Scoring Algorithm. • Run this algorithm over the data collected for each plugin to find out their health score.	• A working Health scoring algorithm. • Health Scores of plugins.
Week 5 & 6 (July 11th - July 25th)	• Testing and Buffer week use (for further analysis and Documentation). • Any unexpected barriers. • Blog Post for Phase 1.	• Document new features. • Test Suite for features developed in the previous week. • Present an unofficial report on the features • Blog Post for Phase 1 • Alpha release and community feedback

Phase 3: Coding Phase 2 (July 25 - Sept 4th):

Week #	Tasks	Deliverables

Week 7 & 8 (July 25th - Aug 7th)	• Deploy the health scores via a JSON file similar to how Jenkins Update Center does it.
	  ○ The scores would have already been generated in the previous phase.
	  ○ Publish the JSON file over a
		• Plugin Health scores are delivered via a hosted JSON file.

	server.	
Week 9 & 10 (Aug 8th - Aug 21st)	• Build new Health Score-specific React components. • Integrate them with the Plugin Site. • Fetch the health scores of plugins from the hosted JSON file into the React components. • Attach the relevant reference links and follow the design document.	• Health scores are visible on the plugin cards at Plugin Site. • A detailed breakdown of the health score is visible for each plugin.
Week 11 (Aug 22nd - Aug 28th)	• Scale-up the project by including more data probes in the health scores of all the available plugins.	• Ability to see health scores of each of the plugins on the Plugin Site.
Week 12 (Aug 29th - Sept 4th)	• Buffer time for further analysis and Documentation. • Any unexpected barriers. • Blog Post for Phase 2 • Work on stretch goals if all the above tasks are done.	• Document new features. • Present an unofficial report on the features. • Blog Post for Phase 2

Phase 4: Final Code Submission (Sept 5 - Sept 12th):

I expect by this time the project would have been completed satisfactorily. In this period, I plan to complete formalities (if any) from the GSOC side or Jenkins side. Code would already be on GitHub and ready to use. The following things can also be worked upon:

- Bug fixes.
- **Feature requests** from **community feedback**.
- Stretch goals.

Future Improvements

- Work on the monitoring aspect of this project.
- Work on the ability for the health score algorithm to be run every day to update the scores and reflect them on the hosted JSON files.
- Continue to support and release new versions of the project.
- Add community-requested features.
- Extend this project by also presenting the Plugin Health score on Jenkins Plugin Manager.
- Add blog posts to the jenkins.io website with regards to this project or regarding a topic that benefits the community.
- Create some beginner-friendly issues which can be taken up by contributors during Hacktoberfest 2022.
- I would like to contribute to Jenkins in general, developing additional features for this project and continuing to work on stretch goals in correspondence to this project.
- Present my work at online meetups like Jenkins Contributor Summit etc.
- Reach out to some of my juniors at college to introduce them to the Jenkins community.

Continued Involvement

- My aim is simple, I want to learn Jenkins.
- I would love to explore the UI/UX side of Jenkins because I've been very much impressed with the recent developments that are being done by the UI/UX SIG.
- I will also adopt a plugin (at least one) in the near future which would help me learn more about Jenkins and the overall plugin development process. After getting experience in plugin maintenance, I'll start contributing to the main (core, or highly-used) plugins.
- I also want to write the LTS Changelogs, it will be like a level up for me as

I'm used to writing the weekly changelogs. And I've got the idea that the LTS Changelog process requires a great deal of careful planning, and knowledge of some terminologies related to changelogs, all of which interests me to a point of exploring this in the future.

- At one point in time, I would also like to participate with Jenkins Advocacy and Outreach SIG and explore the idea of bringing Jenkins to the final year projects of college students. But that would require me to become more confident with Jenkins first, which brings me to my 1st point ;-)

Conflict of Interests

I do not have any other significant commitments than my job during the GSoC phase. I can assure you that the code that I'll be contributing to Jenkins will not be claimed by my employer. I'll be able to devote 20 hours/week to work on this project, and since I usually stay at home during weekends I'll make use of this time to work on this project as well.

Major Challenges Foreseen

- Collecting responses to our community survey to determine weights for probes, can become challenging. As people aren't always enthusiastic to participate in surveys.
- The frequency of when we change the rules must not be too slow or too fast, we need to strike a near-perfect balance and make sure that the proper communication channels must be followed while declaring any changes in rules, and maintainers do not see this as just-another-chore.
- Improve the score calculation algorithm by making it dynamic in nature and be able to deal with boolean as well as unbounded continuous values as input types.
- Dealing with bias in the survey would be a good challenge. For example,

    - A situation when there's a tie for a data probe to have either of the weights 0.3 or 0.5.
    - People taking up the survey might vary every time, making the algorithm independent of the number of participants.

References

- Jenkins Plugin Site
- Jenkins Update Center
- React References for the rendering of the User interface
- Execute Shell commands in Java
- GitHub API
- Contributing to Open Source

Relevant Background Experience

- I have been actively contributing to open source for over a year now. I have good knowledge of git and GitHub and practice strict git discipline.
- I have been contributing to Jenkins since Feb 2021, and have touched upon several areas briefly, like
    - setting up a jenkins.io dev environment locally to deliver Weekly Changelogs,
    - learning about JCasC Plugin and writing a Blogpost + YouTube tutorial on it,
    - contributing to an entirely new Jenkins project called Custom Distribution Build Service, and bringing in some new and useful features to it by reading the ReactJS code via reverse engineering,
    - contributing to the Adopt a Plugin initiative by Mark Waite and Darin Pope, by setting up tutorial structure, which will eventually be delivered on jenkins.io

To sum it up I have contributed to a couple of open-source organizations even while having a full-time commitment side by side, all done remotely. Therefore, I believe that I have the necessary skillsets to deliver a really good quality

production-ready code in the time frame allotted to me. My experience from my recent internship will definitely help me drive this project to a conclusion. Moreover, I am ready to learn everything and improve myself wherever I lack and face all the challenges that come before me, and emerge victoriously.

Personal Information

Hi! I'm Dheeraj, I'm based in Navi Mumbai, India. I've been contributing to Jenkins for a year now, and all of my contributions so far have been related to documentation and some frontend fixes, so this will be my first time contributing to Jenkins Java code. The things that interest me about Jenkins are the community and the scope of learning that this project has to offer to the new contributors. On a professional front, I have benefitted immensely by contributing to this project and I felt a strong desire and deep interest in giving back to this community through my small contributions throughout some of the really busy times of my early career phase (post-college job search phase, then giving sincere time to my first internship phase, exploring other career fields during the post-internship phase). And I think GSoC is a perfect opportunity for me to do things on a larger scale and leave a lasting impact with the choice of my project.

I learned about the Jenkins Project from my college senior: Sladyn Nunes, who was a GSoC contributor in 2020 for the "Custom Distribution Build Service" project.

What makes me special is my passion for volunteering and improving things. I see GSoC as one of the stations in my Open Source journey, and NOT as a destination. And no matter if I get selected this year or not, I'd still continue contributing here (as long as my job doesn't get crazy) because I genuinely want the best people to work on the projects. I just got lucky this year that Google made GSoC available to working professionals as well. I found out about this, after graduating last year, during one of the Docs Office Hours meetings, probably around October, and was very happy to hear this.

Open source contribution (Jenkins)

- **Custom Distribution Build Service, 2021**

  Description/Motivation: Interested to build my knowledge on ReactJS and contribute to this project as a potential GSoC 2021 candidate.

JavaScript Feature	#171
JavaScript Feature	#172
Styling Fix	#173

- **Jenkins Configuration as Code (JCasC), 2021**

  Description/Motivation: Learned a new feature related to JCasC, so I wanted to share it with everyone in the form of a video tutorial + blog.

Blogpost + YouTube Tutorial	#4361

- **Jenkins Changelogs, 2021 - Present**

  Description/Motivation: Saw this as an opportunity to learn about changelogs and how the jenkins.io website works. Also a member of Jenkins Infra GitHub Org.

Changelog 2.296	#4389
Changelog 2.297	#4403
Changelog 2.297 (Updated)	#4409
Changelog 2.298	#4416
Changelog 2.299	#4437

Changelog 2.300	#4447
Changelog 2.301	#4454
Changelog 2.302	#4460
Changelog 2.303	#4470
Changelog 2.304	#4478
Changelog 2.305	#4487
Changelog 2.306	#4499
Changelog 2.308	#4522
Changelog 2.318 (Updated)	#4660
Changelog 2.326 (Updated)	#4780

NOTE: Since Tim Jacomb automated the process of Changelog Generation, I've been reviewing the automated PR for the weekly changelogs with others during Docs Office Hours and also during personal time quite frequently (not much for the past few weeks though).

- **Jenkins @ Hacktoberfest, 2021**

  Description/Motivation: Improve the documenta... Ummm... main motivation was the Hacktoberfest TShirt ;-)

Documentation	#44

- **Jenkins' Adopt a Plugin Initiative, 2021**

  Description/Motivation: Learn about Jenkins Plugin development and improve plugins to create a better experience for both the maintainers + users.

Update	#235 (Open)

Update	#239
Feature	#240 (Open)
Documentation	#Commits on Nov 6 & 7

Language and Other Skill Set

- Java (expert)
- Python (intermediate)
- JavaScript (intermediate)
- Linux (intermediate)
- Selenium (expert)
- Git (intermediate)

- ReactJS (expert)
- Ansible (intermediate)
- Spring (beginner)
- Maven (beginner)
- Hibernate (beginner)

Related Research and Work Experience

- Publication:

  Published a research paper titled "**Foreign Money Transfer using Blockchain**" in the IInd International Conference on Automation, Computing, and Communication. [Certificate]

- Internship Experience:

Software Quality Engineering Intern @ Red Hat

- ➢ Worked on Red Hat Certification System (RHCS)
- ➢ Ensured the quality of the Red Hat Certificate System by manually testing its subsystems, and automating the Test cases of one of its subsystems: Token Processing System.
- ➢ Designed and Developed an Automation Framework to replace manual efforts by 100%, by using Selenium WebDriver and Python, and integrated it with GitLab's CI/CD pipelines.
- ➢ Performed routine manual regression testing to identify, diagnose, and report bugs to the Development team.

- Job Experience:

Associate Software Quality Engineer @ Red Hat

- ➢ Working with Red Hat's OpenShift Quality Engineering Team.
- ➢ I just started recently, so honestly, I'm still figuring things out myself :-)

Reference Links and Web URLs

- **LinkedIn**: dheeraj-sj
- **Instant Messaging**: @dheerajodha (Gitter)
- **Timezone**: Indian Standard Time (IST), UST+5:30
- **Location**: Navi Mumbai, India
- **Alternate email ID**: dheerajsinghjodha@yahoo.com.au