Jenkins Remoting Monitoring

Google Summer of Code Program 2021 Project Proposal

Akihiro Kiuchi
aki.develop8128@gmail.com
https://github.com/Aki-7

Project Abstract.

A monitoring system is essential for system admins to manage servers. This project enables Jenkins agent nodes to provide their metrics to monitoring servers even when the agent-controller connection is lost. This project proposes a new plugin to achieve this objective and focuses on managing the threads in the agents and extensibility for various monitoring servers.

Project Description

A monitoring system is indispensable for system admins to minimize the cloud resources and identify the bottlenecks or issues. As for Jenkins, some plugins, like Metrics Plugin or Prometheus metrics Plugin, expose the controller node's metrics. And the Support Core Plugin can collect agent metrics and puts them to support dumps. However, a plugin that allows exposing monitoring endpoints on Jenkins agent or pushing agent metrics to monitoring servers is missing.

This plugin is necessary because agent nodes are likely to consume more CPU and memory resources than controller nodes. Monitoring the agent nodes will help system admins avoid errors like OOM and estimate the proper hardware or cloud resources.

When I saw this project in the ideas list, I thought many people are waiting for it, and I decided to choose this project.

Goal

The goal of this project is to enable agent nodes to provide their metrics (CPU usage, connection status, etc...) to monitoring servers. The agent nodes should be able to provide them even when the connection to the controller is lost.

What not to do

To collect metrics from agent nodes, connections between the agent nodes and monitoring servers are required. In this project, preparing the connections is the system admins' responsibility. And preparing the monitoring server is also what system admins have to do.

Design

Overview

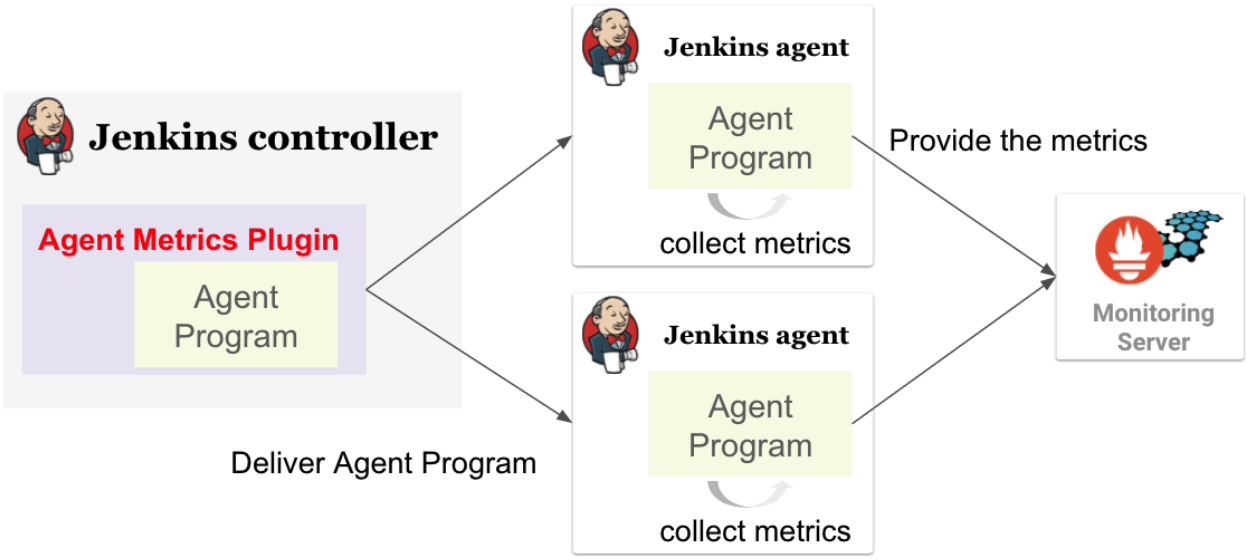

Fig.1 The overview of the proposed plugin.

This project's main deliverable is a plugin that enables agent nodes to provide their metrics to monitoring servers. I name this plugin "Agent Metrics Plugin." When system admins install this plugin, it delivers "Agent Program" to the agent nodes. "Agent Program" is a java program that will be executed in each agent node. It collects metrics of the agent and provides them to monitoring servers.

Deliver and execute Agent Program

Agent Metrics Plugin sends Agent Program using remoting module.

I confirmed that we can launch Dropwizard Metrics and a http server on agent nodes using the remoting module (see also Demo Plugin section).

Managing Agent Program

Agent nodes have to keep providing their metrics even when the connection to the controller is down, but at the same time, the Agent Program must be under control. I think we need to pay attention to this point. Primarily, we need special care when uninstalling the plugin while some agent nodes are temporarily disconnected. The Agent Program has to stop and clean up itself when the connection is restored.

Fig. 2 shows the system to handle this. The first time an agent node connects to the controller after installing the Agent Metrics Plugin(①), the plugin sends a request to the agent to check whether the Agent Program has already been loaded on the agent or not (②). At this point, the Agent Program is not loaded yet, so the Agent Metrics Plugin sends the Agent Program via the remoting module(③), and it will be loaded and executed on the agent(④). Then, the Agent Program starts listening the connection status and starts collecting and providing metrics(⑤).

This is the same when the agent JVM restarts. As the Agent Program doesn't persist over JVM restarting, it follows the same workflow path from ① when the agent JVM restarts.

Next, thinking about the situation that the connection was temporarily lost and now restored(①). Agent Program detects the reconnection and schedules to clean up itself in a few seconds(⑥). On the other hand, the Agent Metrics Plugin also sees the reconnection(①) and sends the state check request if the Agent Metrics Plugin still exists on the controller(②). At this point, the Agent Program has already been loaded, and the clean-up schedule will be canceled(⑦). If the Agent Metrics Plugin is uninstalled when reconnecting, the clean-up schedule will be carried out.

To build this system, it is necessary to access the connection status from inside the Agent Program, and I confirmed it is possible (see also Demo Plugin section).

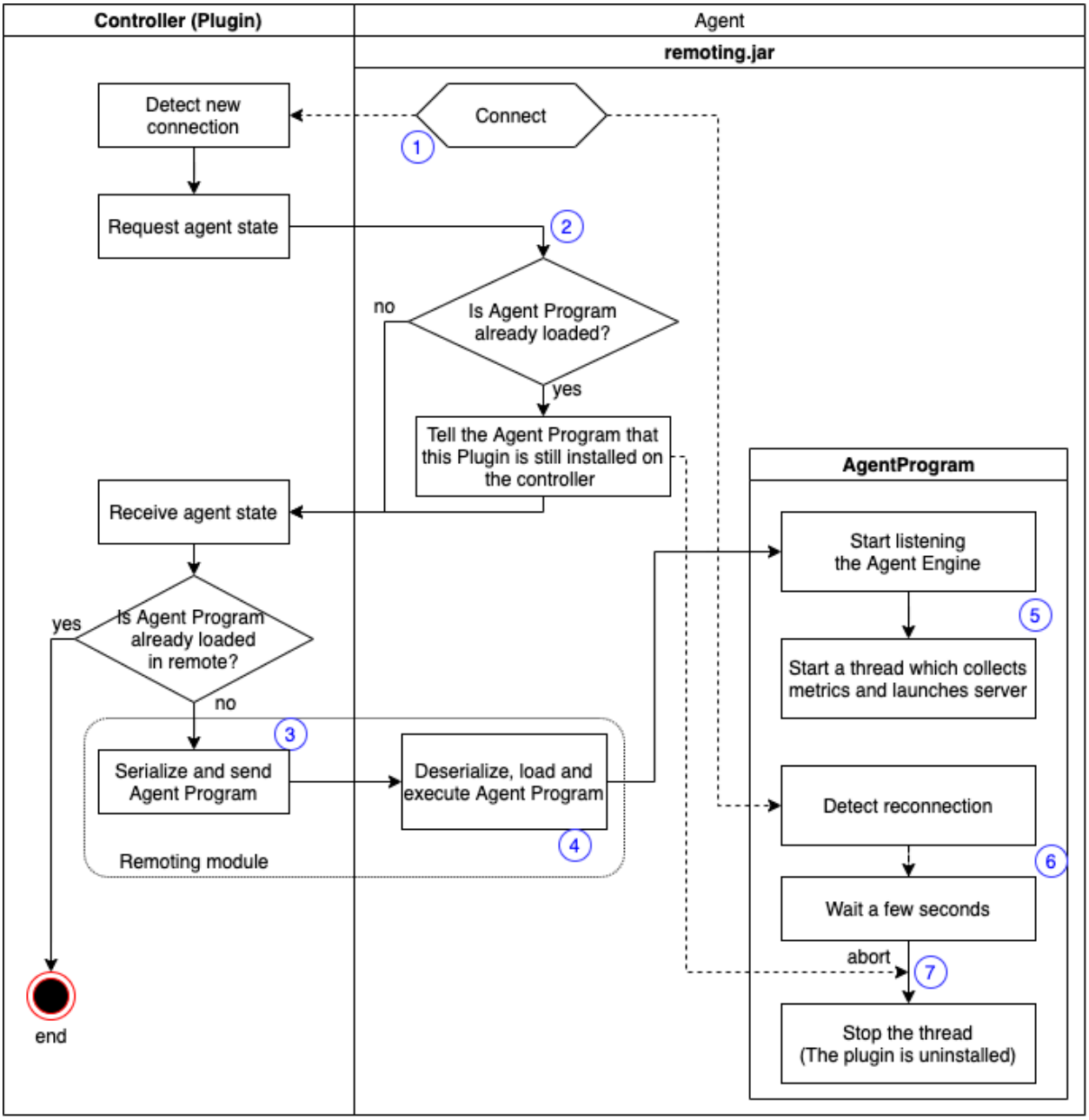

Fig. 2 The activity chart of Agent Program management.

How to expose metrics

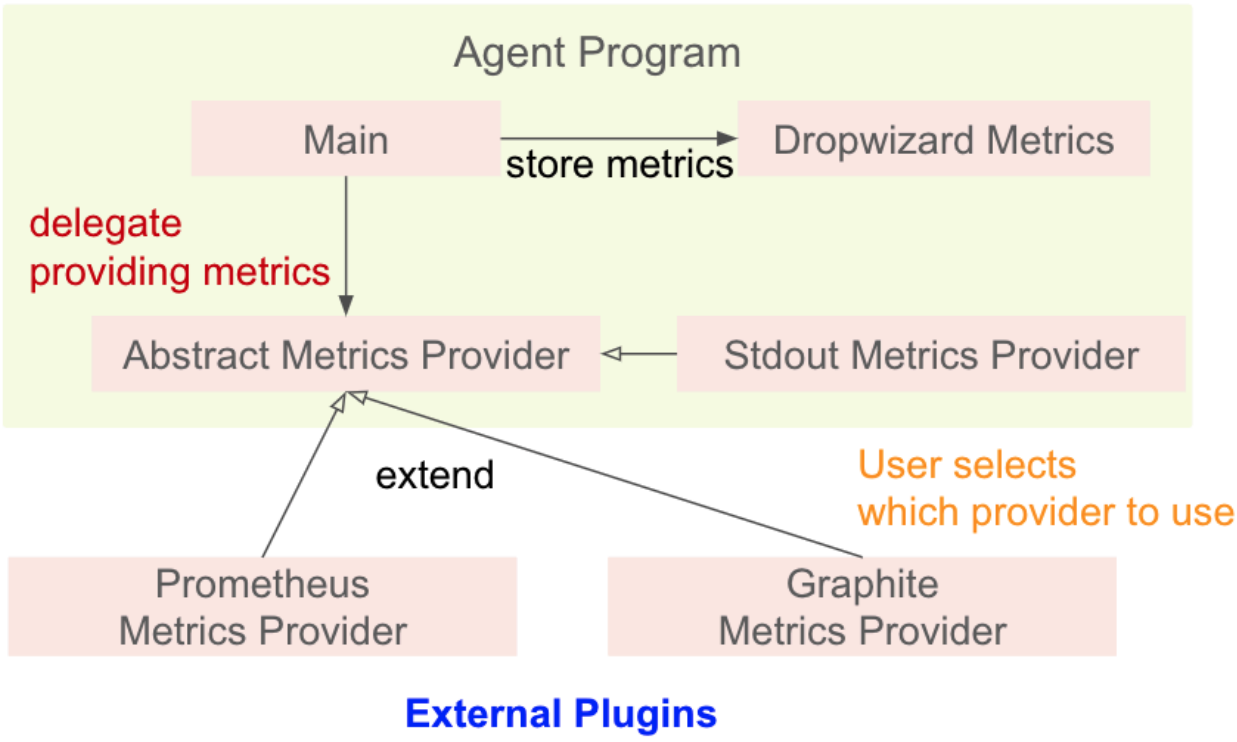

Fig.3 Agent Program can delegate metrics providing to external plugins

Today, there are so many monitoring servers, and they collect metrics in different formats and different methods. To provide agent node metrics to these monitoring servers, Agent Program delegates metrics providing to external plugins. Users can install the suitable plugin for their monitoring environment. This project's deliverables include the external plugin for Prometheus.

What to expose as metrics

Agent Program collects the basic metrics about JVM and OS. I will use Metrics Plugin as a reference. Agent Program also collects the status of controller-agent connection as Jenkins-agent-specific metrics. Collecting other Jenkins-agent-specific metrics is a future work.

Configure the Agent Program

How to configure

This plugin can define three-level configuration, Global level, Label group level, and individual level. Label-group-level configuration is supposed to be used for autoscaling services like Kubernetes. These autoscaling service plugins have a function to attach the same label to their managing nodes. But this label-group-level configuration feature is optional in this GSoC because development cost is high, and it seems to work well for many users with only the other two levels.

How to apply configuration

Because we don't change the configuration of monitoring agents so often, we just stop the existing Agent Program and start a new Agent Program to apply new configuration. See also the pseudocode below. Configuration of each external plugin will be stored within an ExternalPlugin instance.

```
/**
 * Called when user changes configuration
 */
onConfigurationChange(Config config) {
  Computer[] computers = getComputers();
  ExternalPlugin[] plugins = getPlugins(); // external plugins to apply
  AgentProgram agentProgram = new AgentProgram(config, plugins);
  for (Computer computer : computers) {
    computer.getChannel().callAsync(new RestartAgentProgramCommand(agentProgram));
  }
}

/**
 * Stop the existing Agent program and start a new agent program on the agent side.
 */
private final class RestartAgentProgramCommand extends MasterToSlaveCallable {
  public RestartAgentProgramCommand(AgentProgram newAgentProgram) {
    // ...
  }
}
```

Demo Plugin

https://github.com/Aki-7/jenkins-demo-plugin

To check the essential functions, such as listening the connection status from inside the Agent Program, I create a demo plugin that launches an HTTP server on every connected agent, collects the connection status as metrics, and exposes them in a Dropwizard Metircs's JSON format using the HTTP server. This doesn't include perfect management of the Agent Program.

Consideration of using Java agent

We have a discussion about how to launch the monitoring program on agent nodes. There are two possible ways under consideration.

1. Deliver Agent Program to agent nodes via remoting module and launch it in a thread (as described above).

Pros:

- Very easy to introduce. System admins can use this feature only by installing the plugin (and optional configurations).
- Introducing external plugins is also easy.

Cons:

- Agent nodes cannot collect metrics before the first connection to the controller.

2. Launch as a java agent together with remoting.jar. And add the JAR to remoting packaging, e.g., for Docker image.

Pros:

- Super flexible to collect metrics (also see demo code below).
- We don't need to handle dynamic configuration changes.

Cons:

- To launch the Java agent, we need to add -javaagent option to the launch command like

*java -javaagent:<path to Java agent jar> *
-jar agent.jar -jnlpUrl <url> -secret <secret> -workDir <dir>

  This can be an obstacle for some system admins.

- We can collect metrics super flexibly by using Javassist, but this is something like metaprogramming, so this Java agent can be broken even by changing a private method in remoting.jar (also see demo code below). Of course, this won't happen if we don't use Javassist, but the flexibility will be lost.

I love the Jenkins plugin system, and I want many users to try this feature, so I choose the former approach in this proposal. But I don't persist in it. I'll seek a better approach through the GSoC.

Demo code

https://github.com/Aki-7/jenkins-demo-jave-agent

This demo code is a Java agent which launches an HTTP server and exposes the connection status in Dropwizard Metrics's JSON format (same as the demo plugin). But the way to acquire the connection status is less maintainable.

Project Deliverables

- June 8 (coding begins) - the environment to develop and test.
- July 13-17 (Evaluation)

  - Agent Metrics Plugin with only connection status metrics and the stdout provider.
  - Agent Metrics Plugin with basic metrics.
  - Tests
- August 24-31 (Final Evaluation)
  - Global and individual configuration.
  - External plugin to provide the metrics to Prometheus servers.
  - Tests and documents.
- Sept 8 (Final Results Announced)
  - Improve documents. The documents include a tutorial document for Agent Metrics Plugin and a development guide for an external metrics provider plugin.
- Post GSoC - What will you do?
  - Implement external metrics provider plugin for Graphite and so on.
  - Implement label-level configuration.

Proposed Schedule

March 30 - April 14 (Application Period)

- Discuss with mentors for a better understanding of this project.
- Create the demo plugin to know what the remoting module can do.

April 14 - May 18 (Acceptance Waiting Period)

- Learn more about multithreading and class loaders in Java.
- Create PRs related to this project.

May 18 - June 8 (Community Bonding Period)

- Discuss with mentors to refine the design of this project.
- Setup the environment to develop and test this plugins.

June 8 - July 17 (Coding period 1)

Week 1,2 (June 8 - 18)
- Create a controller component that delivers the program to the agent, launches it in a thread, and manages it.

Week 3 (June 21 - 25)
- Create the Agent Program and collect the connection status as metrics in the agents.

Week 4 (June 28 - July 2)
- Define AbstractMetricsProvider Class, which is the abstract class to provide collected metrics to monitoring servers. And implement the StdoutMetricsProvider, which is the concrete class of AbstractMetricsProvider and prints collected metrics data to the standard out.

Week 5 (July 5 - 9)
- Collect the basic metrics like CPU usage, using Metrics Plugin as a great reference.

Week 6 (July 12 - 16)
- Prepare a presentation / Buffer

I'll write tests in parallel.

July 17 - Aug 24 (Coding period 2)

Week 7 (July 19 - 23)
- Enable to set global configurations and apply them to each agent node. The configurations define what data to collect and which metrics provider to use.

Week 8 (July 26 - 30)
- Create an external plugin that includes the concrete class of AbstractMetricsProvider for Prometheus.

Week 9 (Aug 2 - 6)
- Enable to set individual configuration and apply them to each agent.

Week 10 (Aug 9 - 17)
- Prepare a presentation. Write documents.

I'll write tests in parallel.

Future Improvements

- Plugins that provide metrics for Graphite and other monitoring servers.
- More Jenkins-agent-specific metrics.
- Label-level configuration.

Continued Involvement

I respect the engineers who commit to the big OSS project, and I'm so happy if I can be such an engineer at Jenkins taking this opportunity.

Conflict of Interests or Commitment

No conflicts.

Major Challenges foreseen

Managing the Agent Program properly seems to be a complex task in this project. For example, creating and destroying the Agent Program thread must be thread-safe to ensure only one Agent Program thread runs. I guess handling this kind of multithreading stuff will be one of the main challenges.

References

- Metrics Plugin
- Dropwizard Metrics
- Prometheus
- Libraries to bridge Dropwizard Metrics and other monitoring systems.
  - Prometheus
  - Graphite
- OpenTelemetry
  - Client libraries guideline

Relevant Background Experience

I worked as a backend engineer for 18 months at Cookpad inc, a food tech company, which operates Japan's largest recipe sharing service called "Cookpad." Cookpad uses Jenkins to build and test, and uses Grafana to check server load. I was sometimes given time to improve the development environment and tools freely, and I shorten the build time of a certain project from about 20 minutes to 10 minutes by optimizing the build process.
The laboratory I belong to manages part of the servers and routers of its building, and we collect their metrics using Grafana via SNMP.

Personal

I'm a college student in the Department of information and communication engineering at the University of Tokyo. I have worked as a web developer for more than three years. For the first year, I was working as a frontend engineer using React and Redux. After that, I switched to work as a backend engineer, and I was getting attracted to CI/CD and DevOps. The first time I used Jenkins during the work, I was so surprised to hear that volunteers created it, and I realized the OSS community's power. I want to contribute to such a splendid community and enhance my ability through the GSoC.

Availability and commitments

I have no commitments or conflicts during this GSoC.

Experience

Free Software Experience/Contributions (no relation to Jenkins...):

Fix type definition of express-mysql-session (Typescript).
DefinitelyTyped/DefinitelyTyped#49543

Skill Set

- Java - Elementary
- Javascript / Typescript - High
- Python - Medium
- Monitoring System - Middle High
- Web knowledge - High
- Network - High
- Linux - High

Related Research and Work Experience:

Working experience at Cookpad - Backend engineer (1.5 years)
- Usage experience of Jenkins / Grafana.

Lab - Managing servers and routers (1 year)
- Usage experience of collectd / Grafana to collect and visualize router metrics.