A Guide to Scaling OpenShift Data Science to Hundreds of Users and Notebooks

December 13, 2022 | Kevin Pouget | 5-minute read

Artificial intelligence | Data science

< Back to all posts

Red Hat OpenShift Data Science provides a fully managed cloud service environment for data scientists and developers of intelligent applications. It offers a fully supported environment in which to rapidly develop, train, and test machine learning (ML) models before deploying in production.

In this blog post, you'll see how we stress tested OpenShift Data Science notebook spawner to ensure that it can seamlessly support hundreds of simultaneous users.

The scale testing was performed in a transparent and reproducible manner, under the control of a CI automation. This means that there was no human interaction involved in the process of:

1. Creating the OpenShift cluster on AWS,

2. Installing OpenShift Data Science,
3. Launching the simulated users creating the notebooks and waiting for their completion,
4. Collecting execution artifacts to analyze the system behavior post-mortem.

In this scale test, the users are simulated at the graphical level, meaning that the automation will launch a web browser to access OpenShift Data Science, and it will follow step by step the flow over the different pages. This level of simulation is very close to the end-user experience.

The collage in Figure 1 illustrates some of the steps of a simulated user progressing over its test script:

1. Open a Web browser
2. Login into OpenShift Data Science dashboard
3. Click on the Jupyter tile (figure 1a)
4. Configure the notebook to spawn (figure 1b)
5. Launch a notebook, and wait for its availability (figure 1c)
6. Login into JupyterLab
7. Run a simple validation notebook (figure 1d)

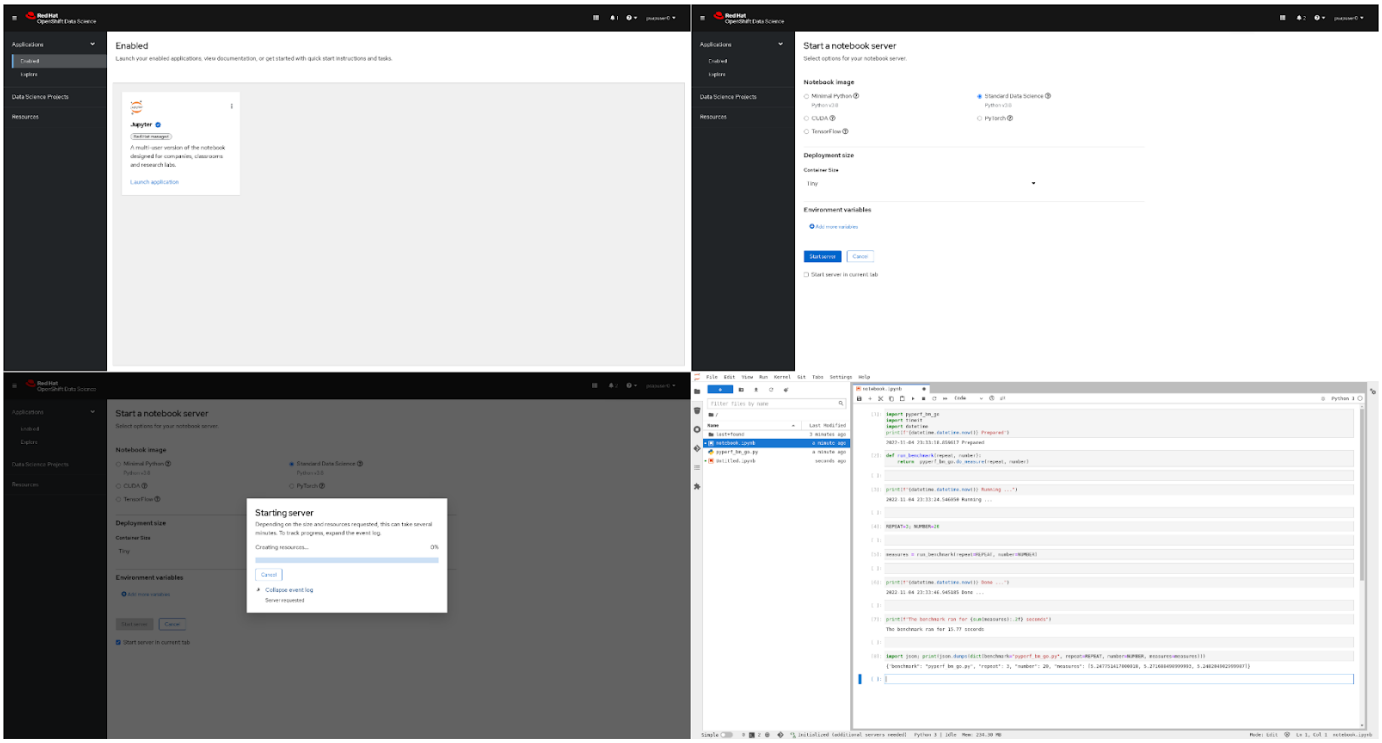

Figure 1: OpenShift Data Science Notebook Spawner

1a: Dashboard Jupyter tile
1b: Dashboard Notebook spawner

1c: Notebook spawner waiting window
1d: JupyterLab validation notebook

Note that the simulated users run in a dedicated *external* environment (another OpenShift cluster), each in a dedicated Pod. This gives the scale test higher accuracy, as it ensures that the *management* of the user simulation does not have any negative effect on the OpenShift Data Science cluster. In addition, the full network stack is involved in this setup, whereas having the users running *inside* the OpenShift Data Science cluster would bypass some of the network components.

In the final part of the test, we collect some execution artifacts to study how the overall system behaved while the notebooks were being created. The main artifacts are the following:

- OpenShift Prometheus database (contains all the metrics about the Pod execution, CPU/memory consumption, ...)
- The Notebook, Pod and Node definitions
- The simulated user logs (contains the timestamps of each of the steps) and screenshots (for troubleshooting and/or visual confirmation of the user behavior)

In the rest of the post, we'll have a look at the main results of the scale testing. The test ran with the following configuration:

The plot in Figure 2 shows the progression of the 300 simulated users:

- At the beginning of the timeline, we see that the 300 user-simulated Pods (outside of the OpenShift Data Science cluster) take around 25s to all start and reach the readiness barrier. This distributed synchronization point, coordinated by the StateSignal package, guarantees that all the users start executing their scenario at the same time, no matter how long OpenShift took to schedule the Pods and start their container.
- Next, the purple triangle is the delay for the users to join the test, between 0s and 1000s (15 minutes). The system is not designed to support 300 users joining at the exact same time (unlikely in OpenShift Data Science real-life use-cases), so we introduced an artificial initial delay. Here, a new user starts every 3 seconds.
- Then, the users run their simulation script, as presented above. In pink, we see the actual notebook spawn time.
- Finally, the user notebooks remain live in the system, although unused. This keeps the resources busy until the end of the overall test.

Note that the last steps (loading and running the notebook) are not time-accurate, due to the challenges of JupyterLab automation. They are not taken into account in the notebook launch time measurements.

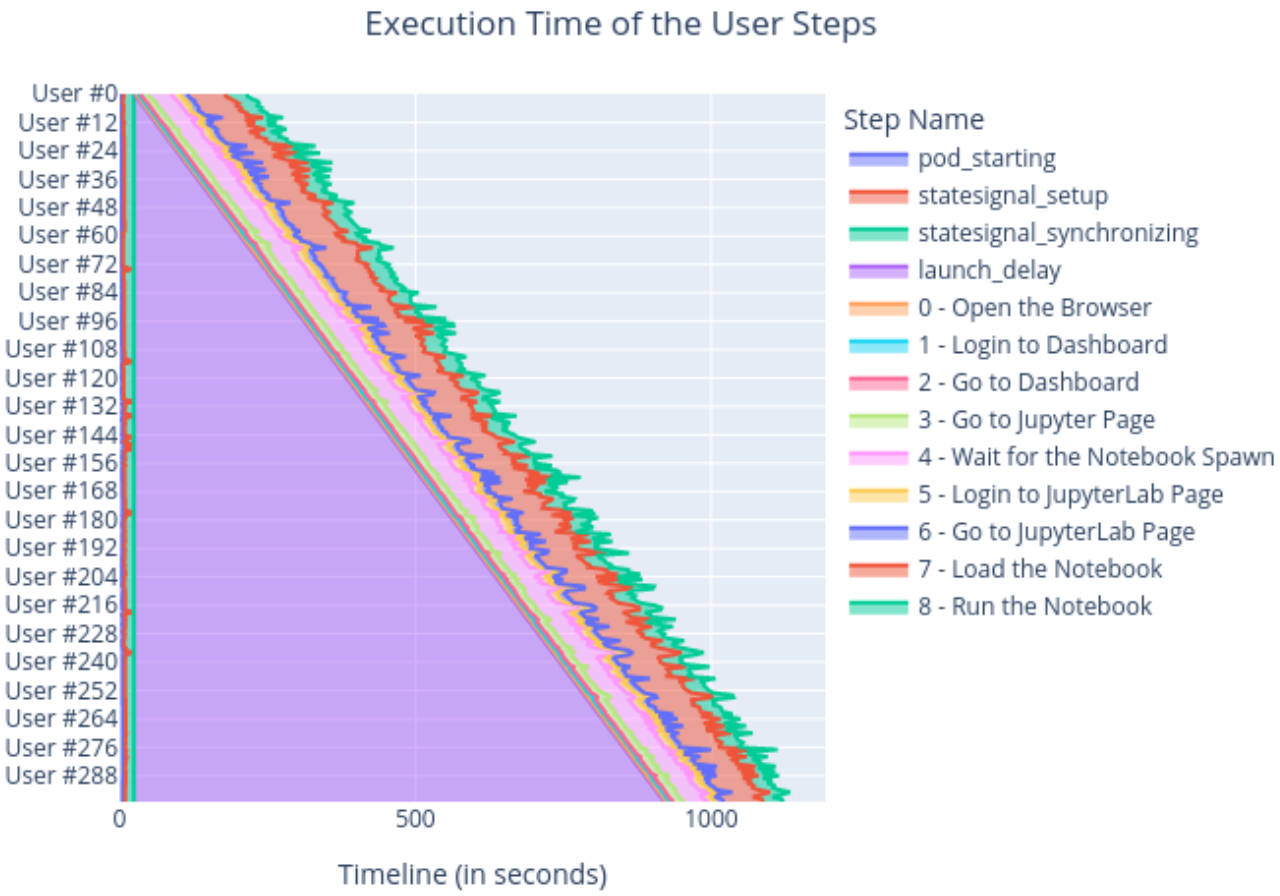

Figure 2a: Progression of the simulated users. Progression with all the steps.

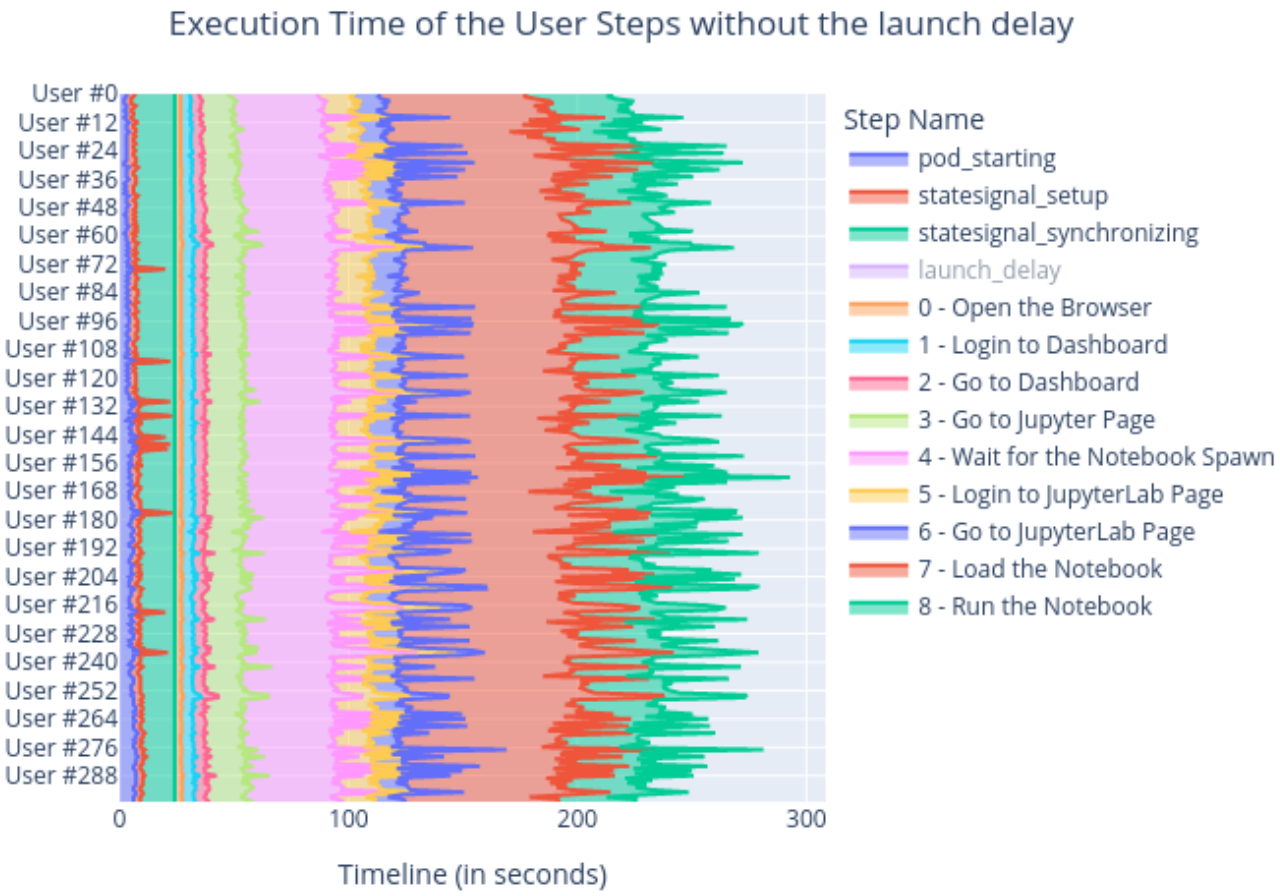

Figure 2b: Progression of the simulated users. Progression with the launch delay hidden.

The chart in Figure 3 is our key performance indicator. It shows the time it took for the simulated users to reach JupyterLab (which is expectedly higher than the notebook creation time, shown in pink in the Figure 2). The initial user startup delay is obviously not included in this timing. We got the following results:

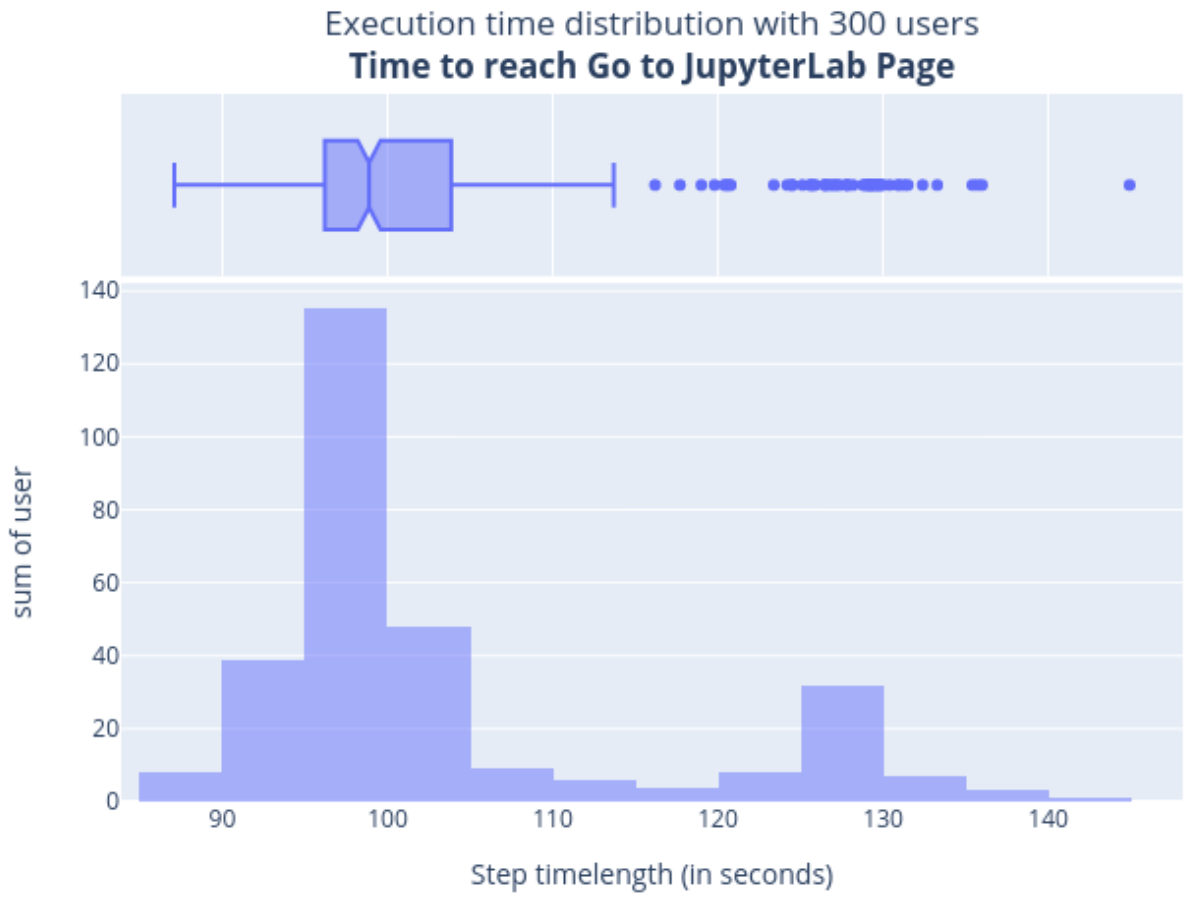

Figure 3: Execution time distribution for getting a usable Notebook

We can observe that there are two peaks starting at 95s and 125s. What is happening behind the scenes is that the frontend of the Jupyter spawner polls the state of the notebook every 30s, after a random initial time (due to execution time of the previous steps). So after 1x30s, the Dashboard first checked if the notebook was ready. It wasn't. And again after 2x30s, still not ready. Then after 3x30s, for 75% of the users, the notebook was ready. And finally at the 4th check, 100% of the notebooks were ready.

At last, we study the OpenShift Data Science cluster system metrics, to ensure that the system is not running too close to its limits.

The plots in Figure 4 show an excerpt of the metrics we analyze, related to the control plane nodes load and OpenShift Data Science components resource usage:

- Figure 4a shows the idle time of the three control plane (aka, master) nodes, in arbitrary units. When the lines go down, the nodes are dealing with a high number of Kubernetes requests (the CPU idle time shrinks). If the CPU idle time reaches zero, the nodes are under high pressure, and they will not be able to serve the requests in due time. The Kubernetes

control plane will crash sooner or later and become unavailable for a bit of time. During development time, this means that the OpenShift Data Science components are sending too many Kubernetes requests, and the components' behavior needs to be improved. In production, this means that the control plane hardware is unable to handle the load, and the instance type should be changed for a stronger one.

- Figure 4b shows the CPU usage of the OpenShift Data Science Dashboard Pods, which handle most of the user-facing Notebooks operations. We see in this plot that the five Pods run well under their requests (orange dashed line) and limits (red dashed line).

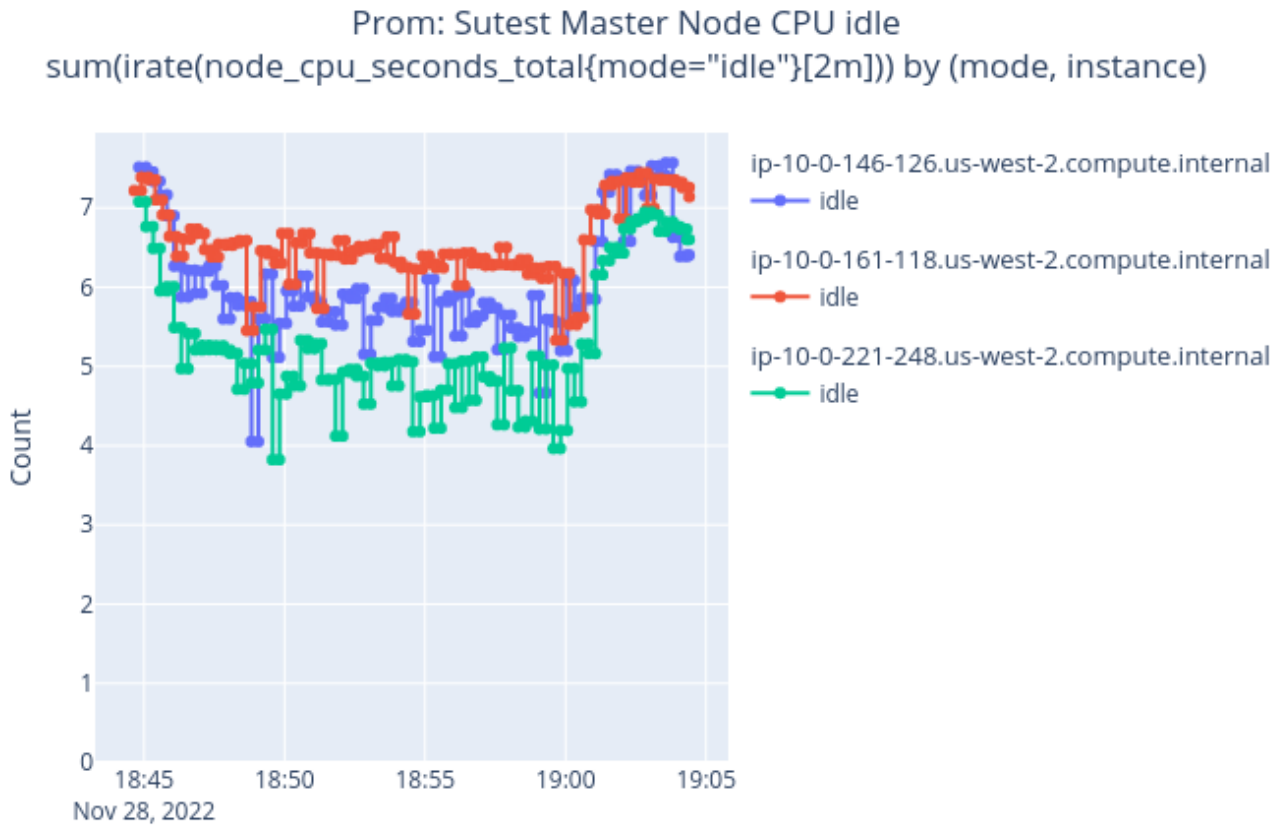

Figure 4a: OpenShift Prometheus metrics visualization. Master nodes CPU idle time (higher is better).

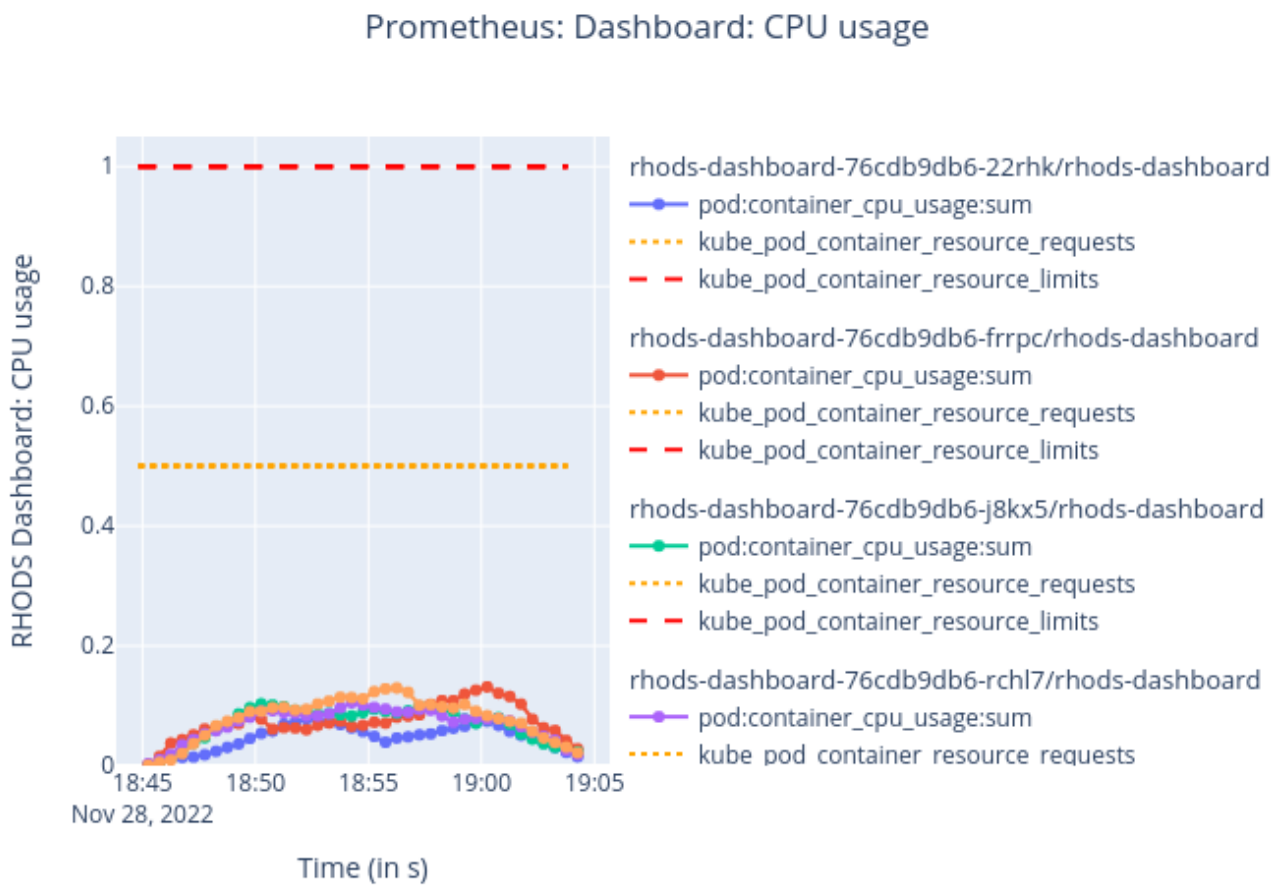

Figure 4b: OpenShift Prometheus metrics visualization. OpenShift Data Science Dashboard CPU usage (lower is better).

To conclude this blog post, let us introduce the future steps of the performance and scale testing of OpenShift Data Science:

- Benchmark the notebook images shipped with the default OpenShift Data Science notebook images to ensure that the performance does not regress from one version to the next one (for Python and the key packages installed, for CUDA GPU computing, ...),
- Quantify the effects of auto-scaling on performance.
- Lower the impact on the control plane (master nodes) without adversely affecting notebook performance.
- Scale test OpenShift Data Science model-serving and pipeline upcoming capabilities, as well as on-prem (self-managed) environments.
- Run the scale tests on-demand, on customer clusters, to validate the performance or troubleshoot issues.
- Scale test the notebook spawner with up to 3000 users (long term goal).