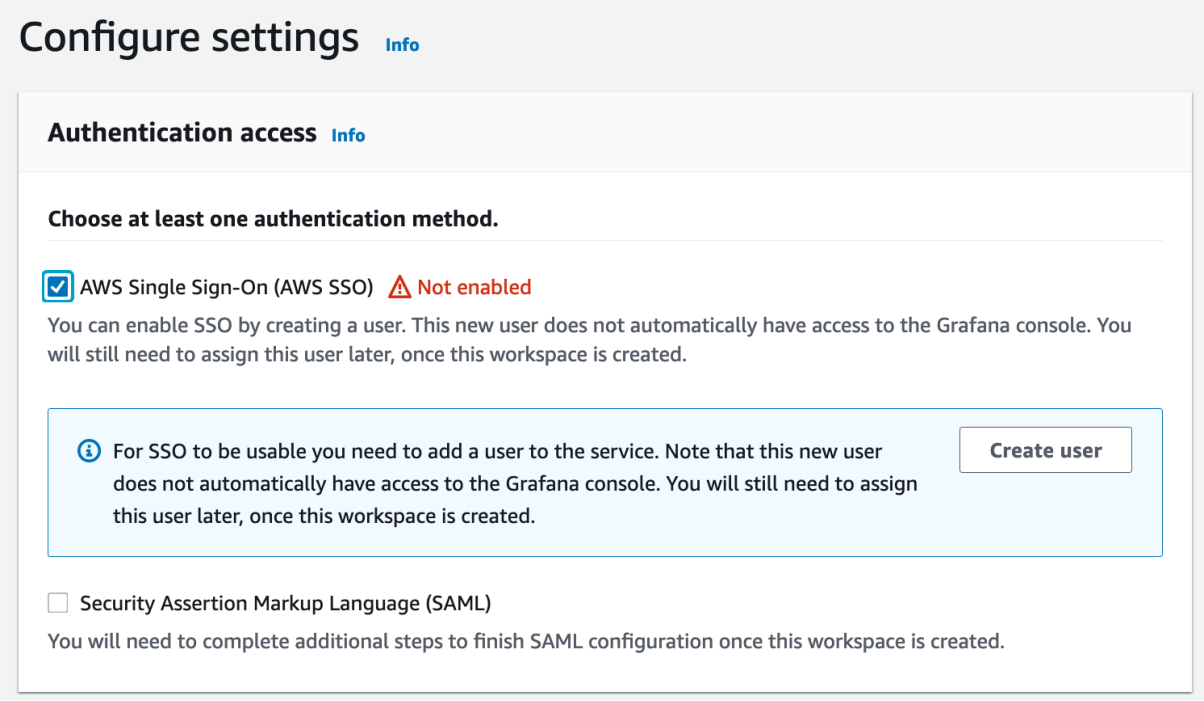
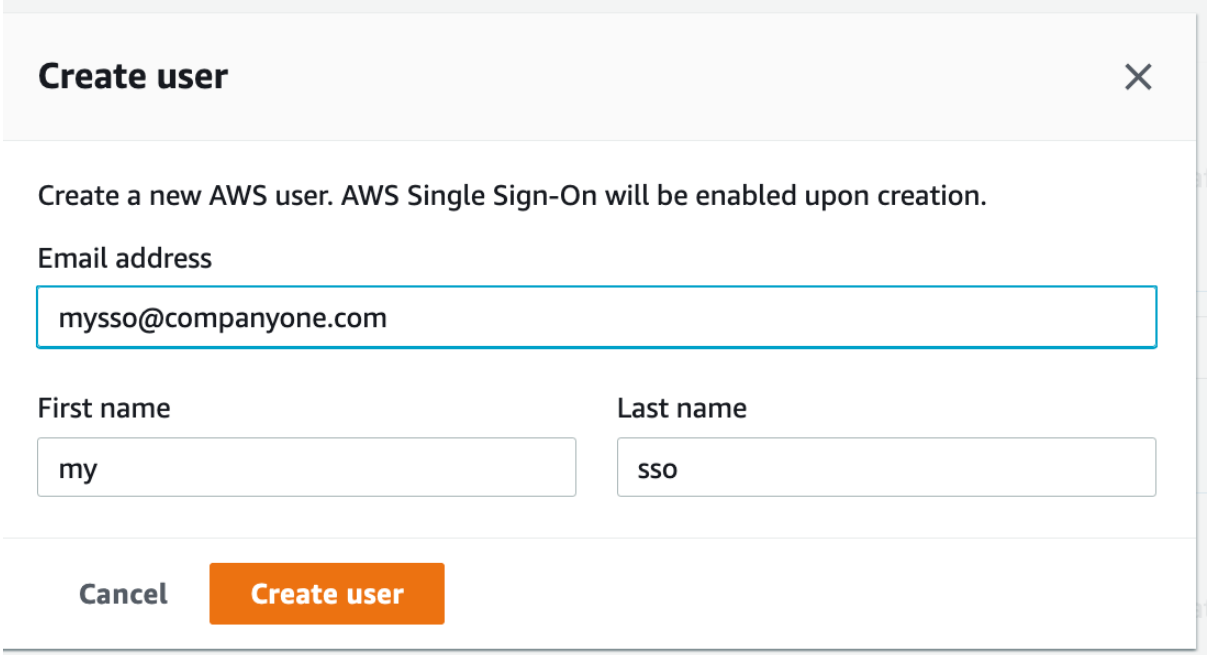
Create a workspace in Amazon Grafana

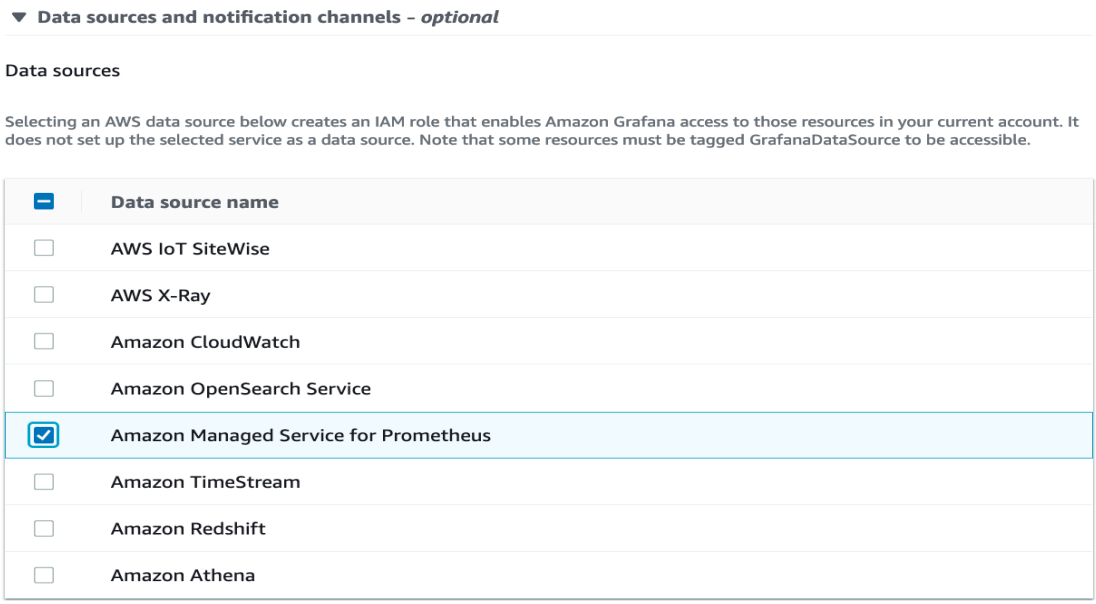
- Go to the Amazon Grafana console, click on **Create workspace**
- Key in the workspace name **emr-on-tfc-summit,** then **Next**
- Use **AWS SSO** as the authentication access type.



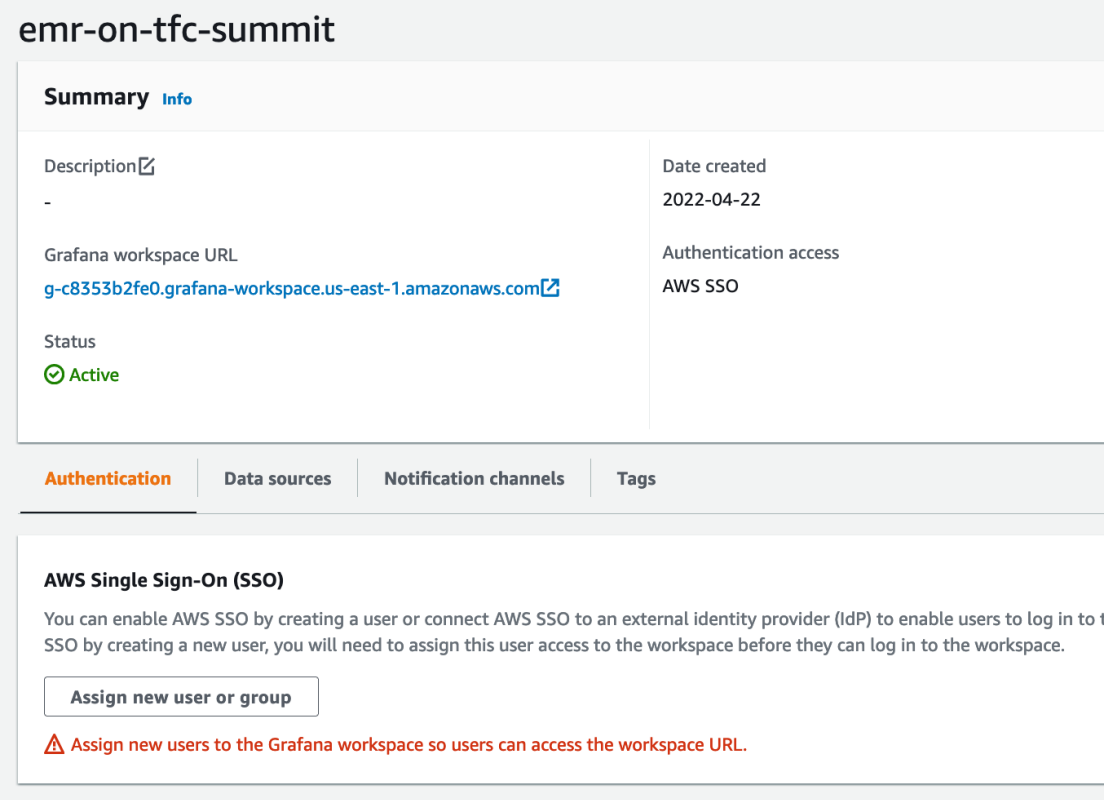
- Create an SSO user with random username and email, if the SSO is not enabled. It's OK to use an existing cross-region SSO user.



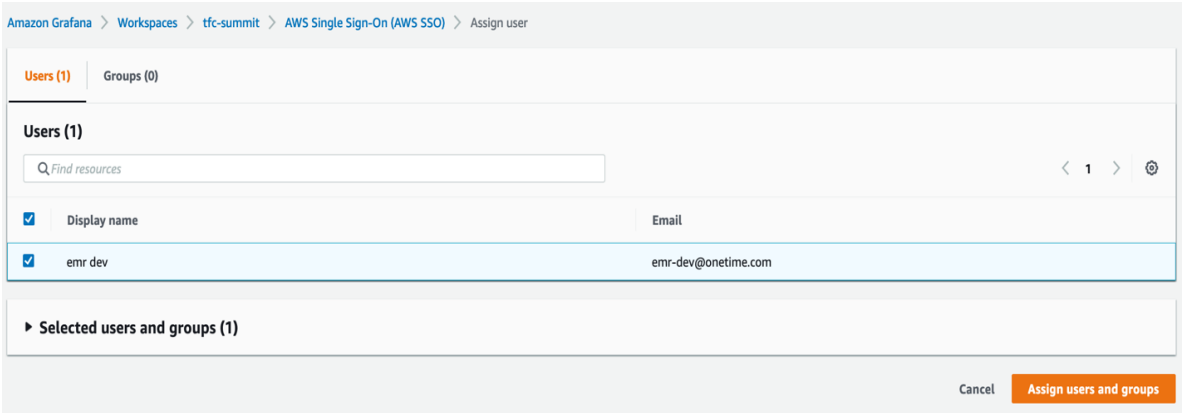
- In the Data sources and notification channels – optional section, select the **Amazon Managed Service for Prometheus.**



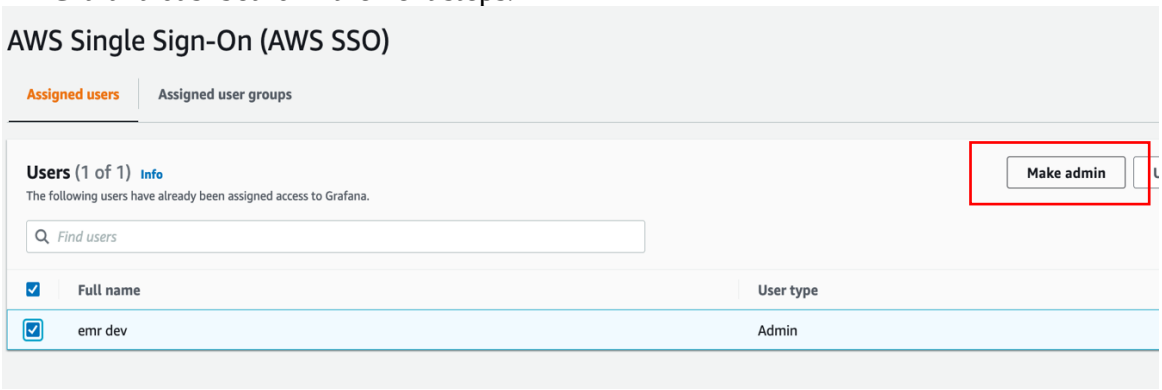
- Proceed to the final Review and create page, then **Create workspace**
- click on the **Assign new user or group** button when you see the warning message in the Authentication tab.



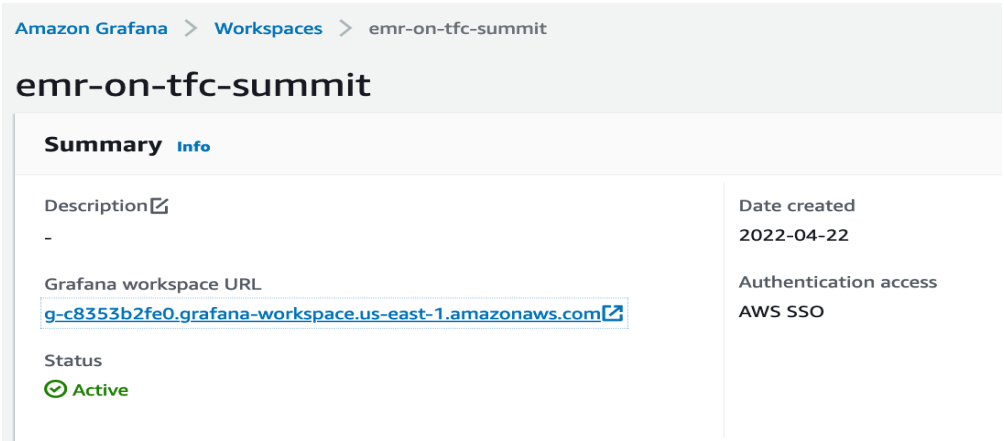
- Assign an SSO user.



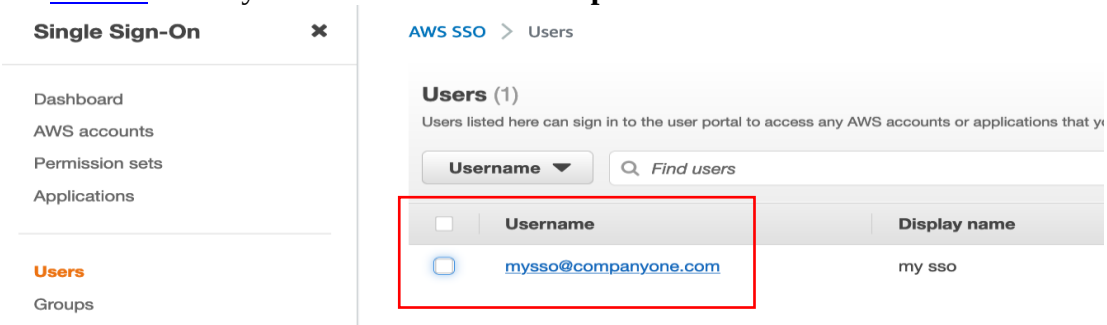
- Re-select the user and **set it as an admin.** This option lets the user add data sources to the Grafana dashboard in the next steps.



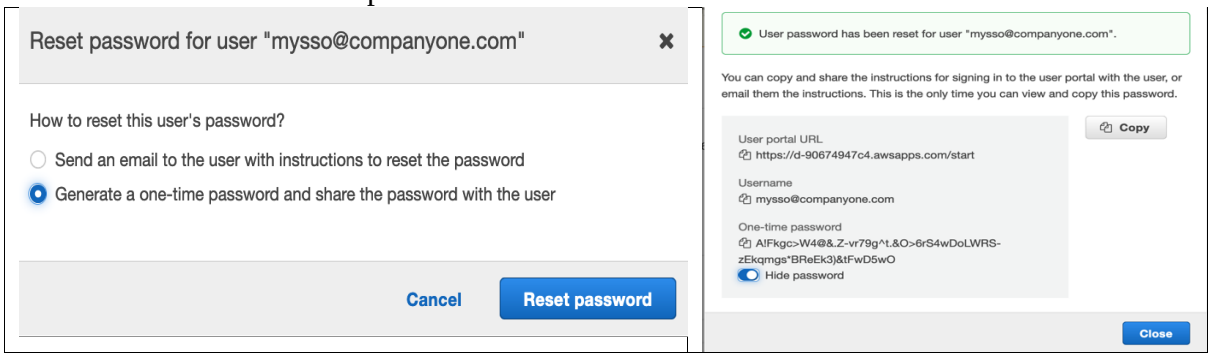
- Go back to the workspace console and click the **Grafana workspace URL**. Login via AWS SSO.



- If you don't have the username or password, can get them from the [AWS SSO User console](). Click your Username then **Reset password**.

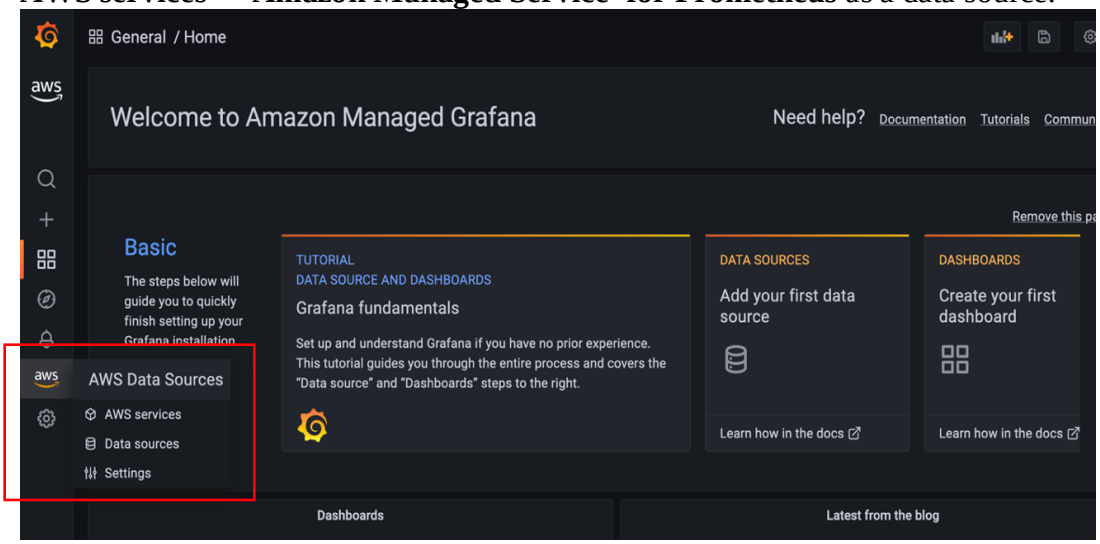


- Generate a one-time password

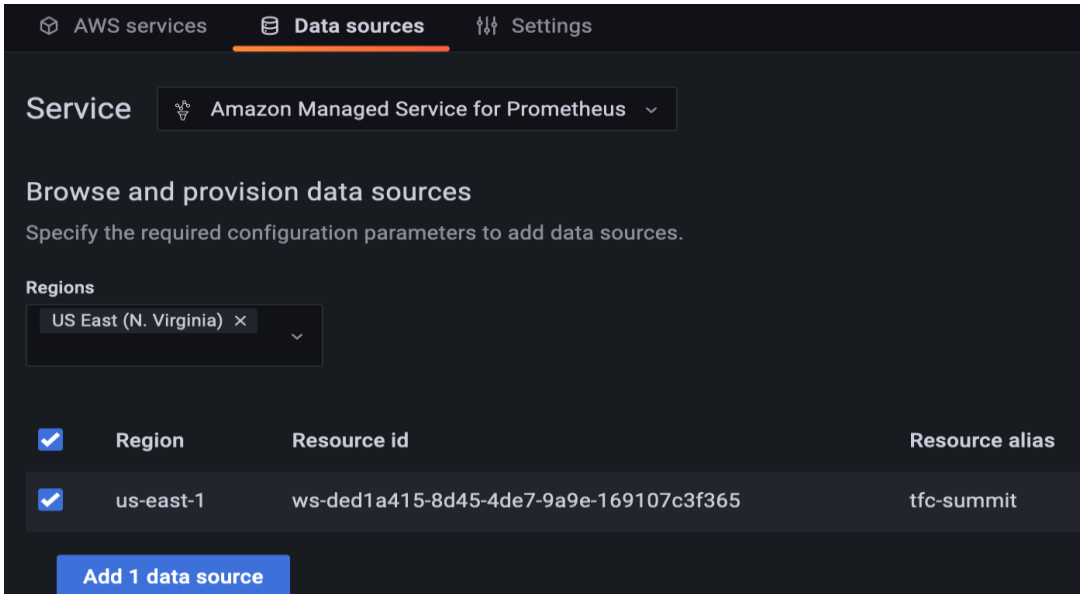


Add Prometheus as data source

- After login successfully, select the smaller **AWS logo** on the left ribbon, then choose **AWS services -> Amazon Managed Service for Prometheus** as a data source.



- Choose your **region** and the **data source**, then click **Add 1 data source.**
  Select the data source with a correct Resource alias, if you have multiple Prometheus data sources.

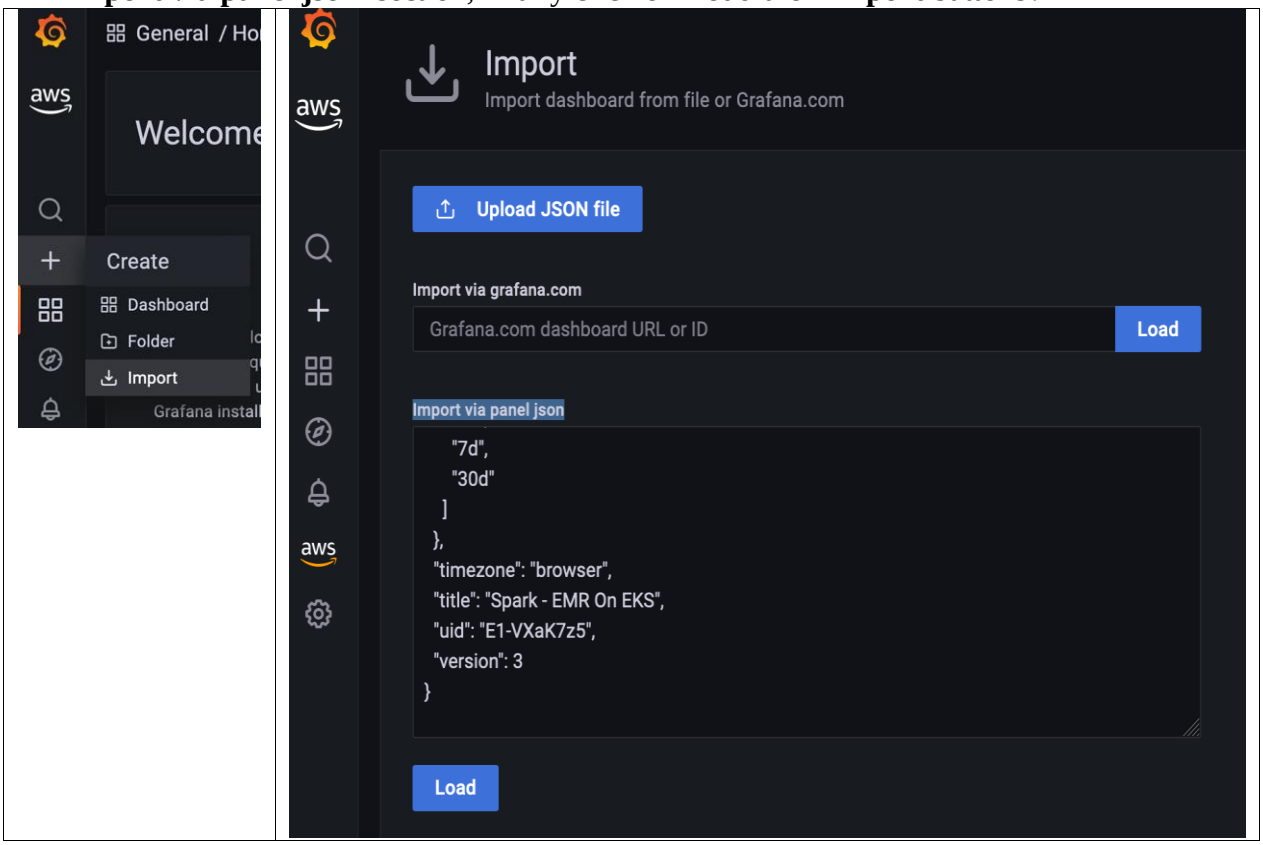


Create a dashboard for Spark

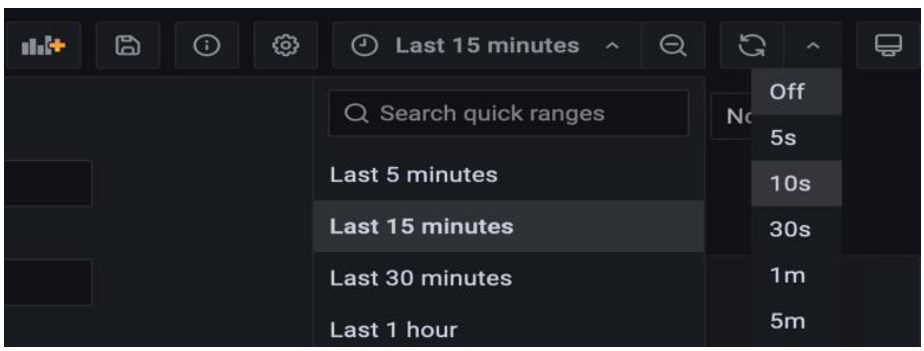
- A pre-defined Spark dashboard template is created already. Open the following link and copy the file content.

  https://raw.githubusercontent.com/aws-ia/terraform-aws-eks-blueprints/main/examples/analytics/emr-on-eks/examples/grafana-dashboard-for-spark/emr-eks-grafana-dashboard.json

- Click the + icon and choose the **Import** option. Paste the template file content to the **Import via panel json** section, finally click on **Load** then **Import** buttons.



- On the dashboard, set the time range to **15 minutes** and change the refresh frequency to **10 seconds**.



- Congratulations! You have successfully setup a Grafana dashboard for EMR on EKS.

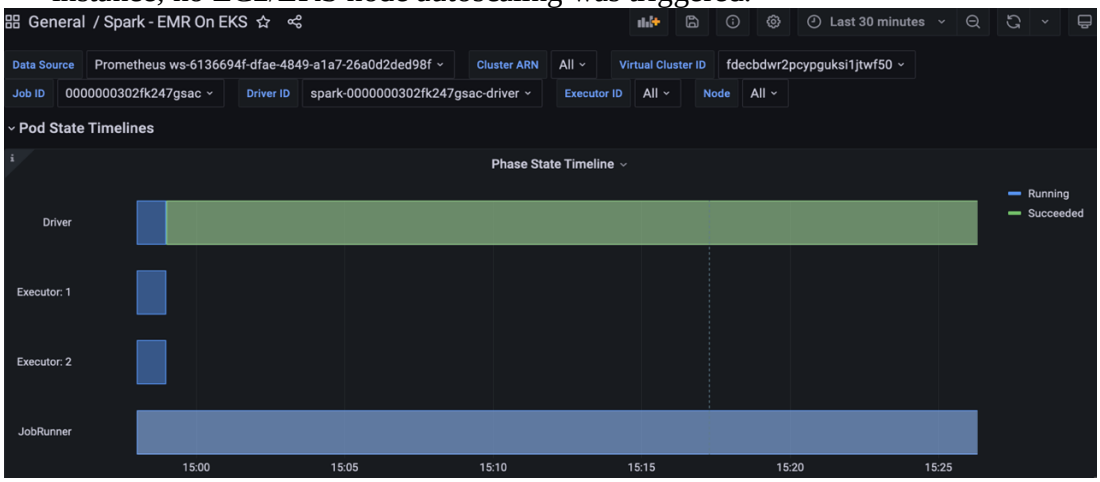
Follow the [workshop instruction] to submit a Spark job and monitoring its autoscaling performance on Grafana.

Appendix

- **Understand the dashboard**

  Pod State Timelines section - A graph that tracks a job autoscaling performance when firing up a Spark application with EMR on EKS. It collects the time & pod status information, then visualize it. It displays when a pod status is changed from pending to running, from running to succeeded.

  - The following example shows a 2-executor job was run on an existing EC2 instance, no EC2/EKS node autoscaling was triggered.



  - The following case is for a 47-executor job - a medium size Spark application that was still waiting for the compute resources. Because it took approx.3 minutes before starting to scheduling all the executors at once. Unfortunately, we have reached the max number of instance quota after the 3-minute startup time. The job was not running at all. **The autoscaling was managed by Cluster Autoscaler (CA).**

- On the other hand, the same Spark job was submitted to the AZ at the same time, where nodes are scheduled **by Karpenter**. We can see the job was managed to run, simply because of the instant scale-up reaction within 1minute. Though some pods are still waiting for the available compute resources, over 50% pods were managed to get to the running state.