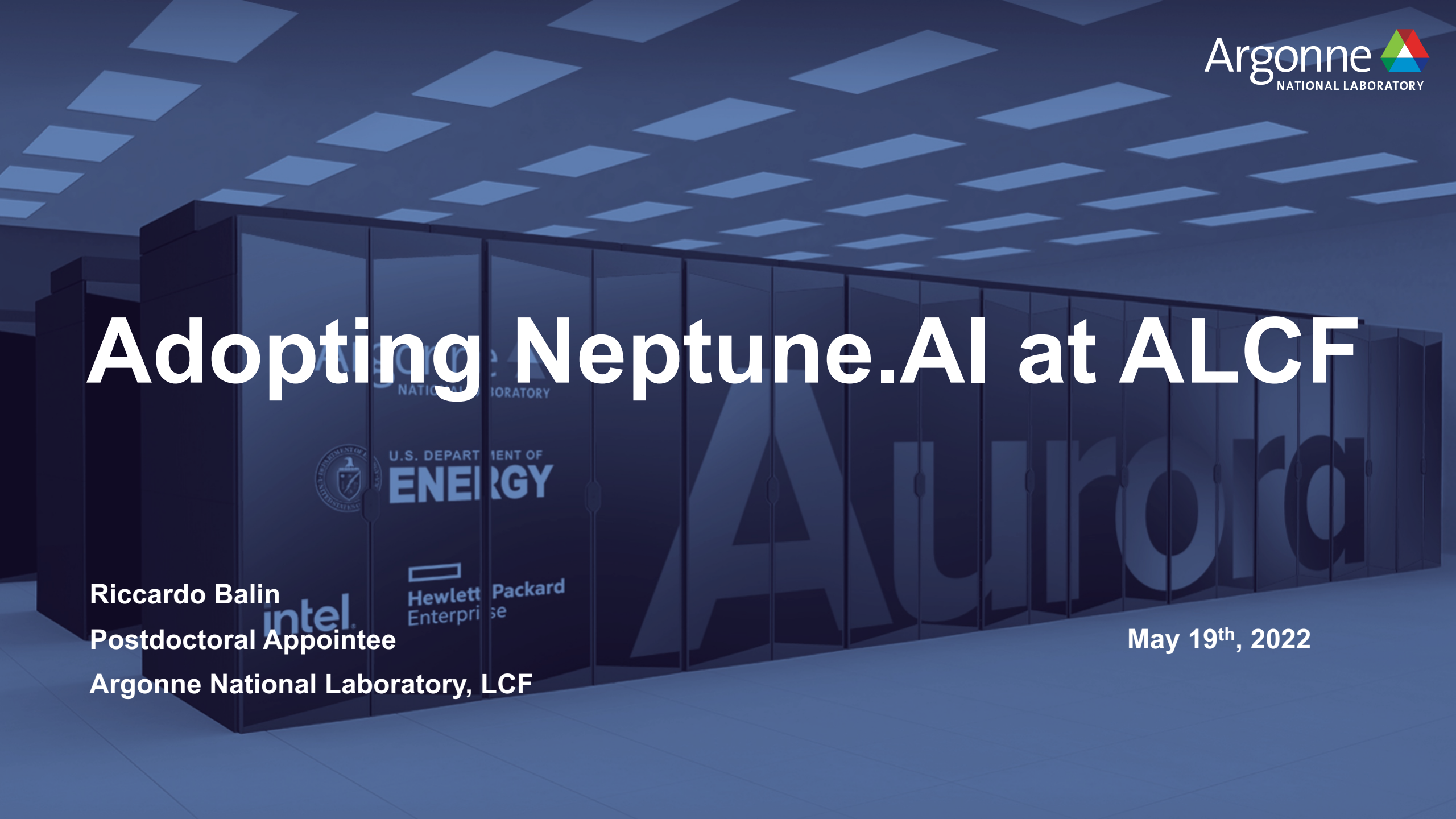


Adopting Neptune.AI at ALCF

Riccardo Balin
Postdoctoral Appointee
Argonne National Laboratory, LCF

May 19th, 2022

Outline

- Introduction
- Account options and payment
- Signing up
- Installation of client library
- Example of Logging a Training Run
- Comparison to Weights & Biases

Introduction to Neptune.AI

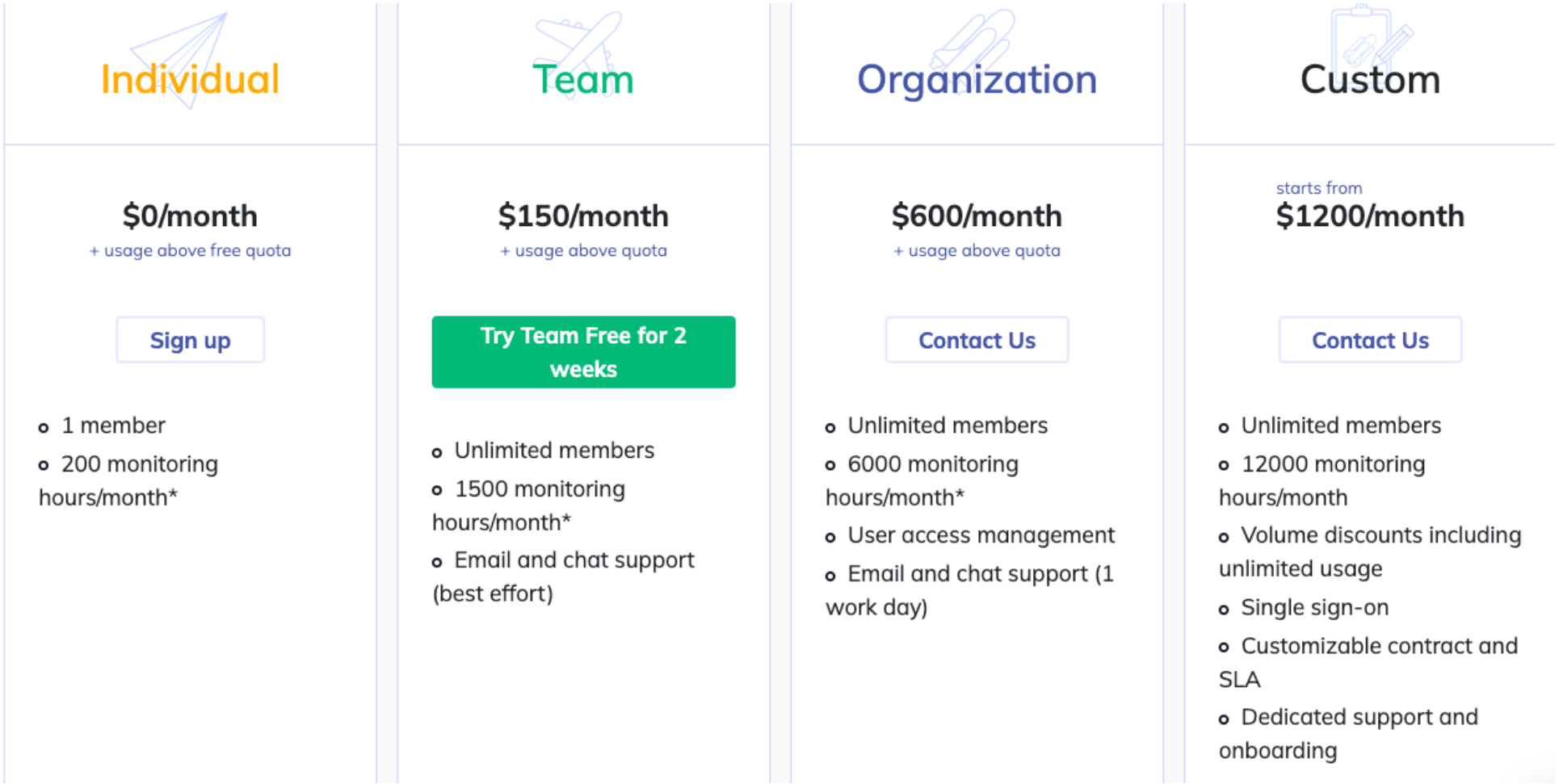
- Neptune is a metadata store for MLOps
- Provides users and teams the ability to log, store, display, organize, compare an query all model-building metadata in a single place
- Website: https://neptune.ai/
- Documentation: https://docs.neptune.ai/
- GitHub page: https://github.com/neptune-ai
- Useful examples to get started: https://github.com/neptune-ai/examples

Introduction to Neptune.AI

General Features

- Log and display ML metadata and training data
  - Code, config files and hyperparameters
  - Performance metrics and loss/accuracy curves
  - Hardware consumption and console log (stdout and stderr)
  - Model weights and model checkpoints (e.g., saved as a .pt file)
  - Any chart or visualization (e.g., plots produced with matplotlib)
  - Training data as an artifact (file path and hash)
- Compare experiments and models
  - Web UI allows autodiff of all metadata between runs
- Monitor ML runs live from web UI
- Query metadata from UI and programmatically
- Integrates with most ML frameworks for automatic logging
- Works as SaaS or with on-premise deployment (keep metadata local)

Account Options for Teams Using Neptune.AI



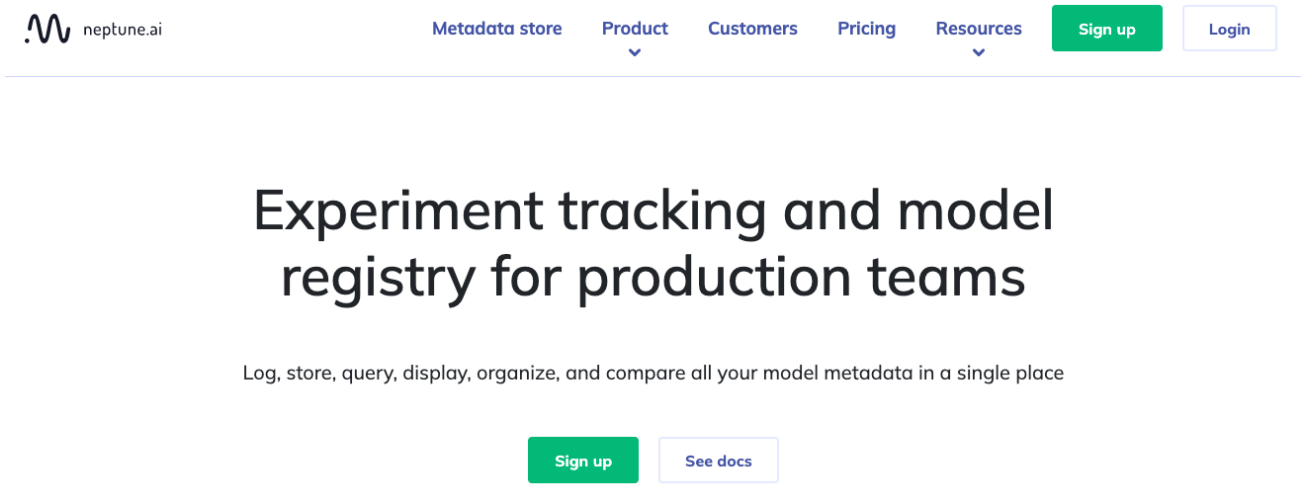
Account Options for Teams Using Neptune.AI

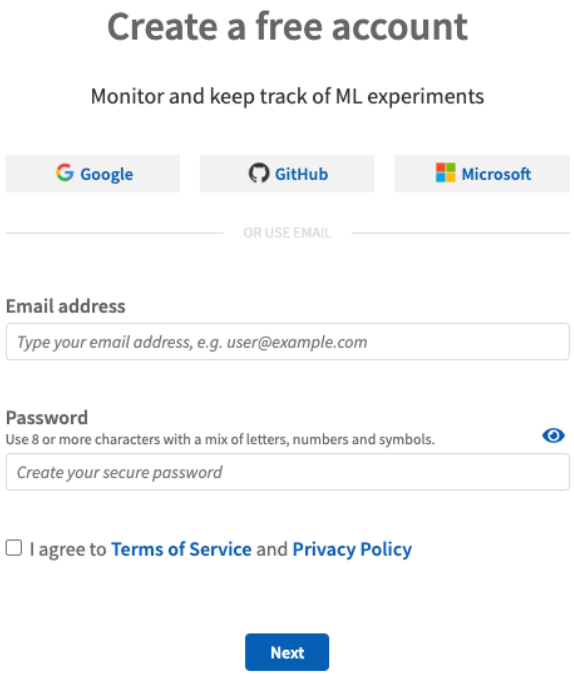
Notes

- Can add storage and monitoring hours for a fee ($10 per 100 hours)
- Unused monitoring hours roll over month-to-month, but get reset every year (for individual account at least)
- Team plans offered for free (with larger base quotas) to academia, research and education groups, and non-profit organizations
- Accounts are charged based on usage, NOT number of users
  - Charged based on monitoring hours consumed by all users as a whole
  - Add infinite users at very small additional cost if usage is negligible
  - Could be attractive option for ALCF, average cost of adding users to account is likely small

Signing Up to Neptune.AI

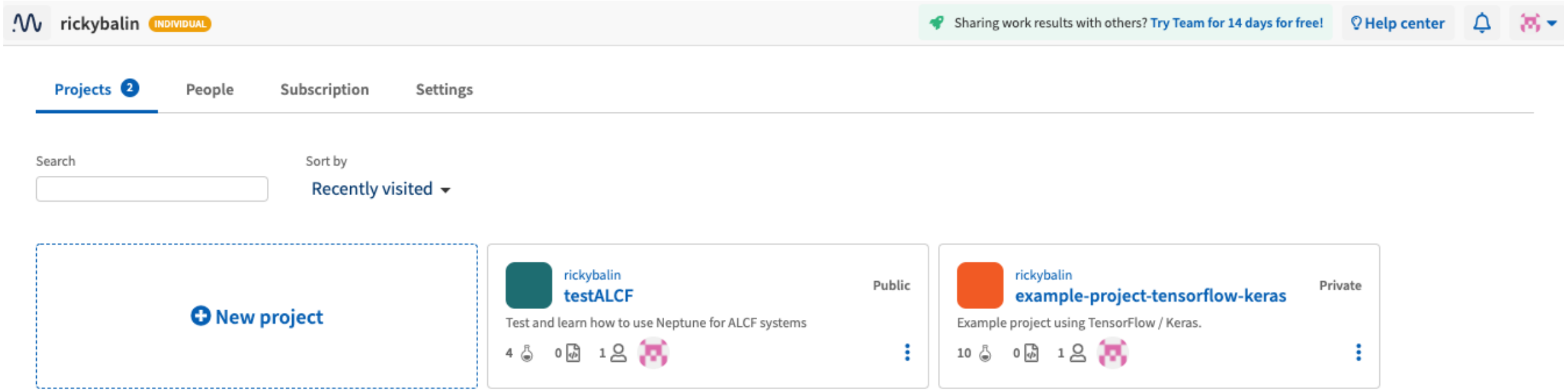
- Click Sign up button on home page
- Select how to sign up (I chose GitHub)
- Select what type of account (individual or team)
- Answer a few questions and you're done!





Signing Up to Neptune.AI

- Logging in gets to to your home space in the user interface



Installation of Client Library

- Using Pip on ThetaGPU
  - Install Neptune client simply with *pip install neptune-client*
  - Can load my env with *conda activate /grand/datascience/balin/NeptuneAI/testPipInstall/env/nept*
- Neptune also allows an on-premise deployment
  - Metadata is stored on local machines or private cloud
  - Can be deployed as Kubernetes Cluster or VM running Kubernetes under the hood
  - There is a free 30 day on-premise trial
  - More information and instructions at https://docs.neptune.ai/administration/on-premises-deployment

Example Training Run

- Source code for example found at /grand/datascience/balin/NeptuneAI/testPipInstall/src
- Simple serial training of 1 layer ANN on random data
- Modifying code to log metadata with Neptune API is simple
  - Import Neptune module

```
# Neptune
import neptune.new as neptune
```

  - Initialize Neptune logging, and include source code files

```
    # Initialize Neptune logging
    run = neptune.init(
              project="rickybalin/testALCF",
              api_token="eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vYXBwLm5lcHR1bmUuYWkiLCJhcGlfdXJsIjoiaHR0
HVuZS5haSIsImFwaV9rZXkiOiJhMjllYmRkMS1lMzI2LTQ0NzctOWE2MS05M2MwNzE2YzhhYzkifQ==",
              source_files=["*.py"],
              )  # my credentials taken from the project I created on my Neptune account
```

Example Training Run

- Modifying code to log metadata with Neptune API is simple
  - Log some training hyperparameters

```
# Log training parameters
train_params = {'n_epochs': args.Nepochs, 'mini_batch': args.batch,
                'learning_rate': args.learning_rate, 'n_samples': args.nSamples}
model_params = {'n_neurons': args.nNeurons, 'n_inputs': args.nInputs, 'n_outputs': args.nOutputs}
system_params = {'device': args.device}
run['train_params'] = train_params
run['model_params'] = model_params
run['system_params'] = system_params
```

  - Pass Neptune run object to training function

```
model, timeStats = trainNN(inputs, outputs, args, logger_conv, run)
```

  - Log loss and accuracy at each epoch

```
        # Log loss and accuracy to Neptune
        run["training/loss"].log(loss)
        run["training/acc"].log(acc)
```

Example Training Run

- Modifying code to log metadata with Neptune API is simple
  - Upload model checkpoint

```
# Upload model to Neptune
run["model"].upload('./NNmodel.pt')
```

  - Stop Neptune logging at end of program

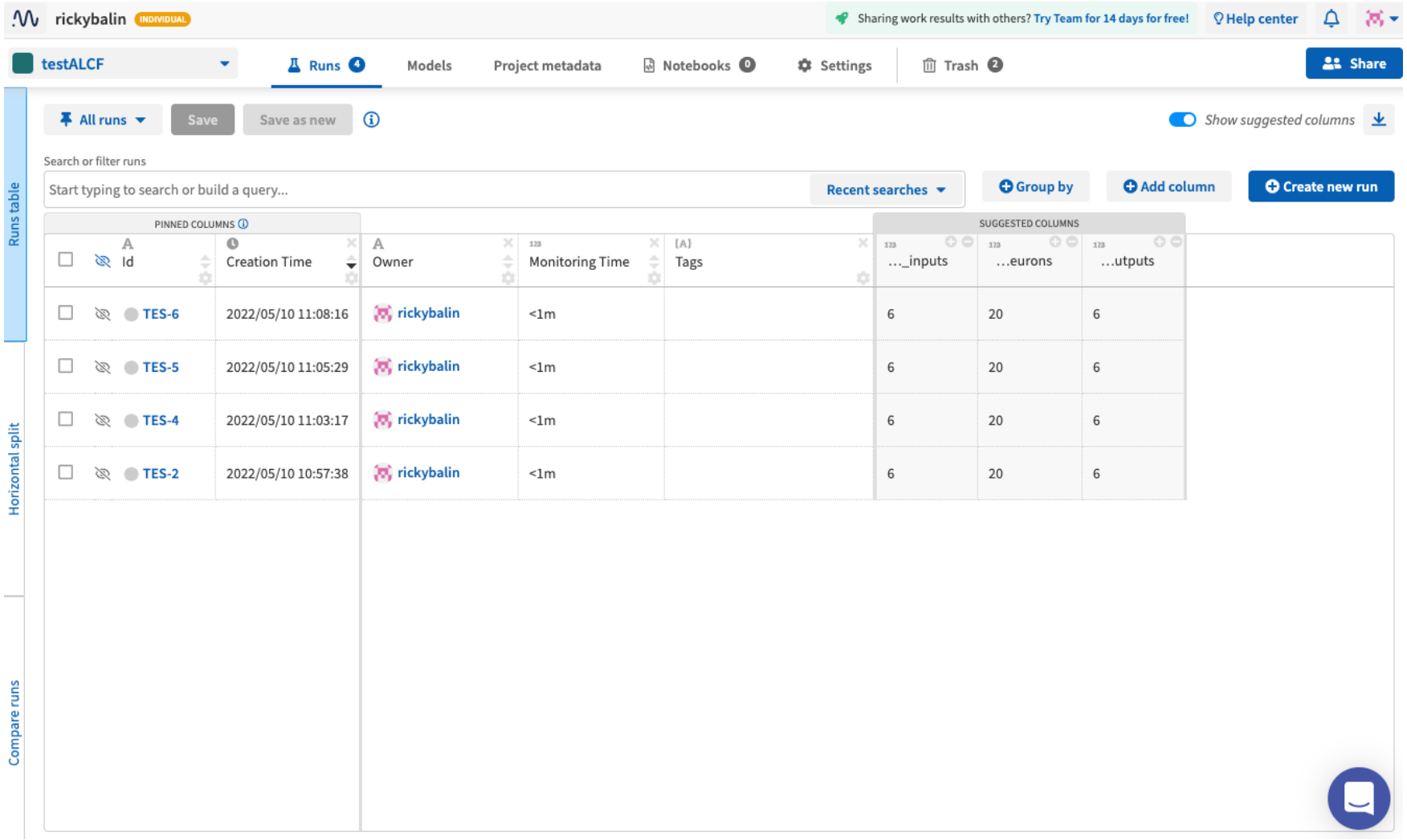
```
# Stop Neptune logging
run.stop()
```

  - Log training data as an artefact (I did not do this since I generated training data within program)

```
# Log training data on Neptune
# If loaded data from file: run["dataset/train_data"].upload("./data/train_data.csv")
# Can also ave dataset versions as Neptune artifacts with the track_files() method
run["dataset/train_data"].track_files('data/train_data.csv')
```

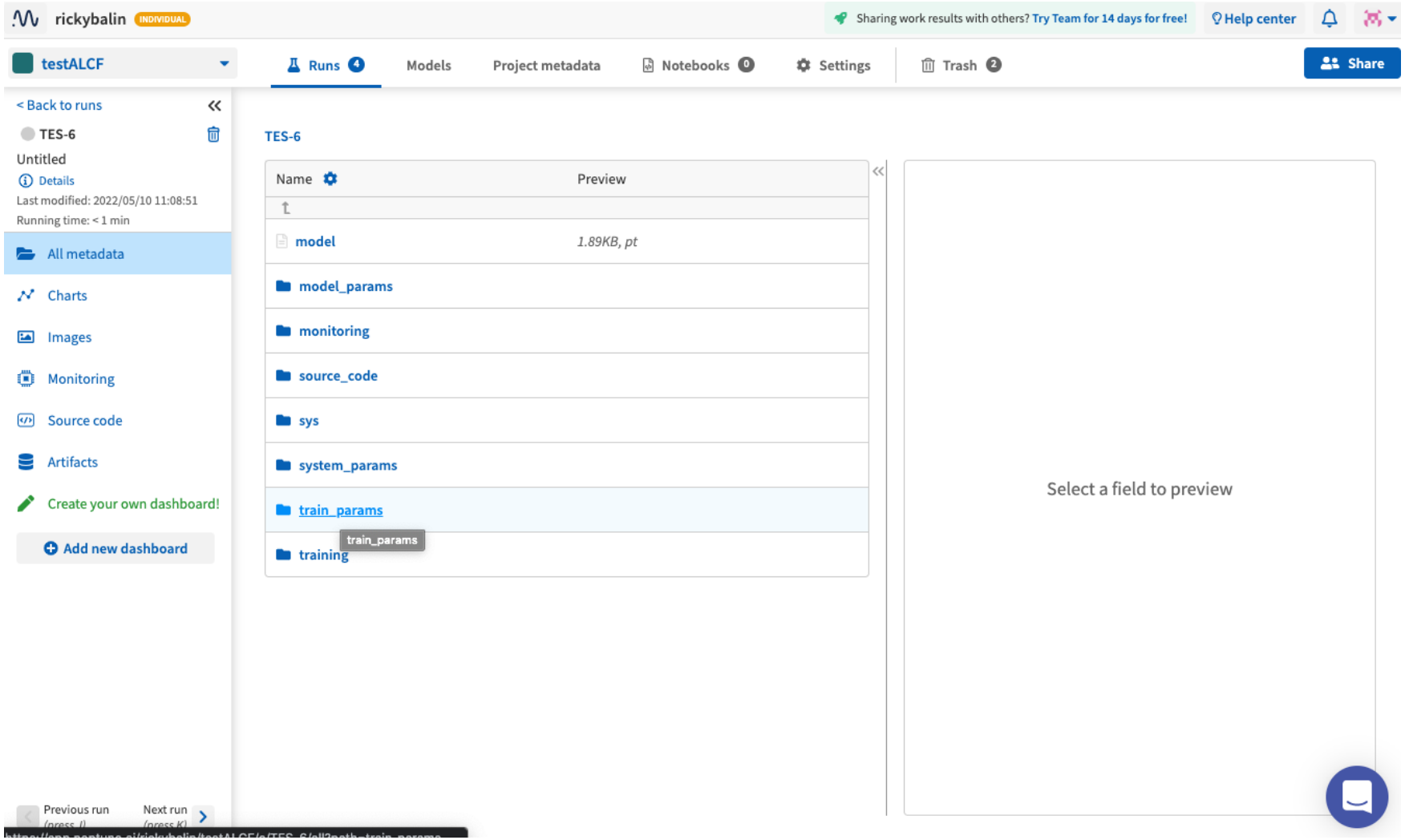
Example Training Run

- Visualize and compare experiments on web UI



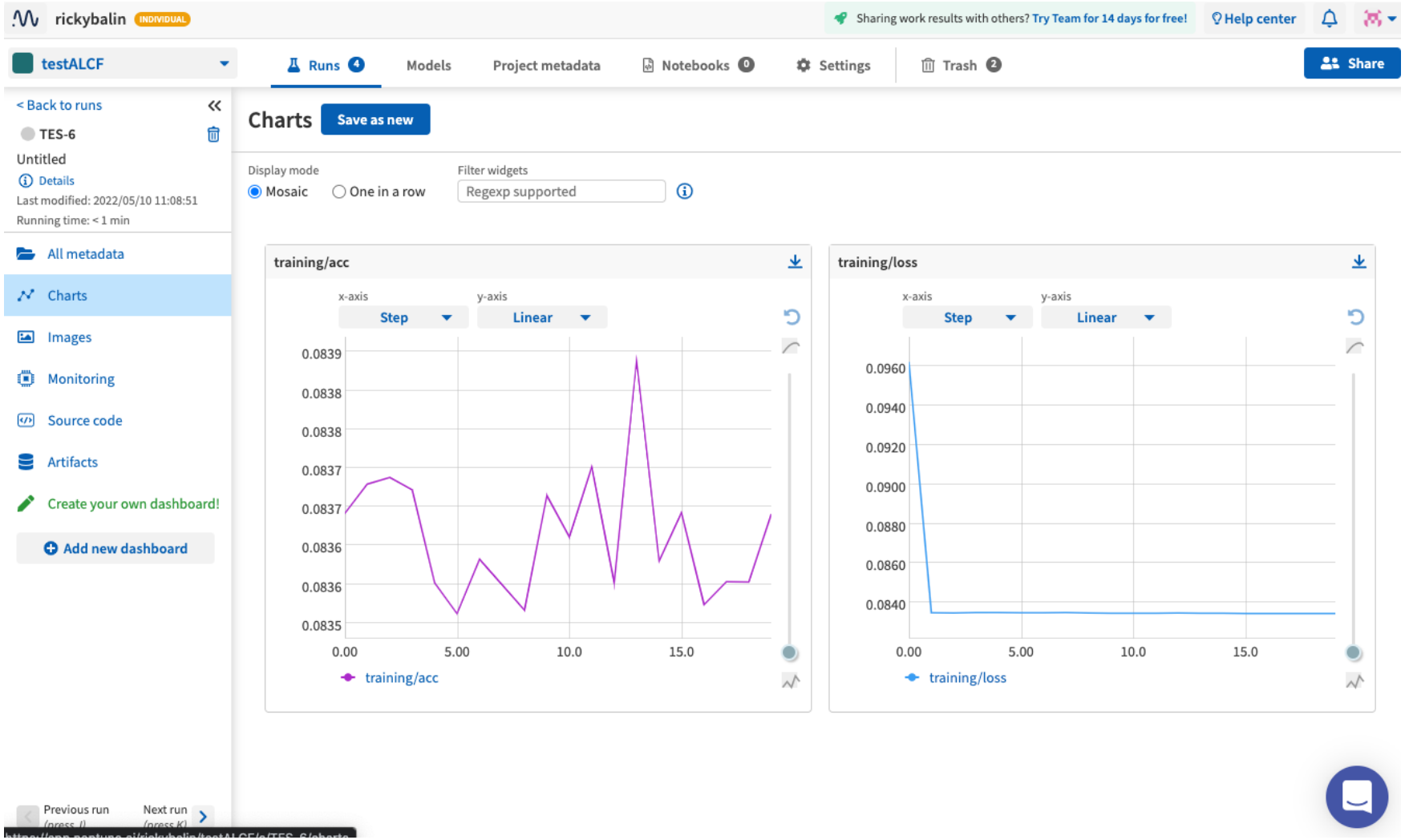
Example Training Run

- Click on run name to visualize metadata logged



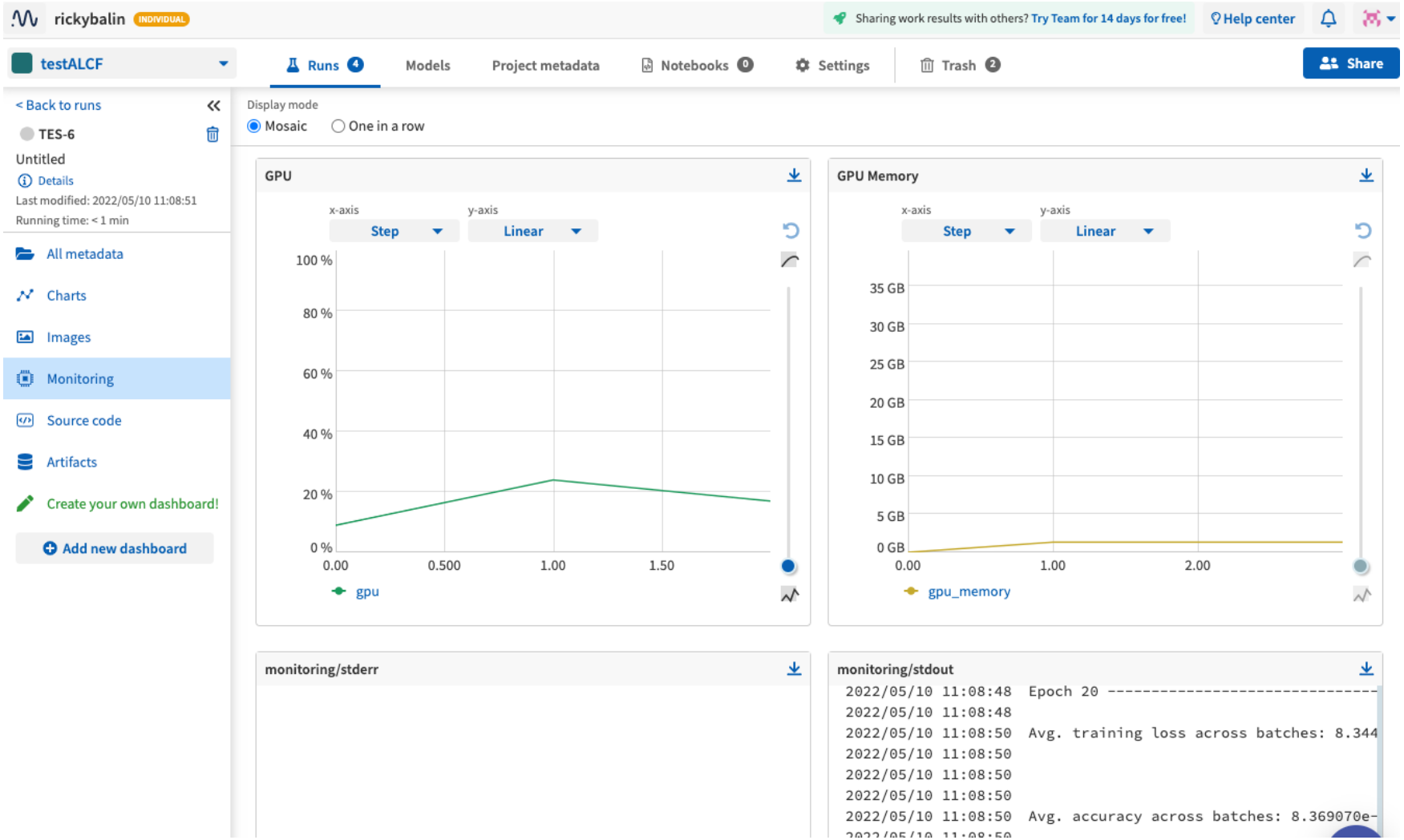
Example Training Run

- Click on run name to visualize metadata logged



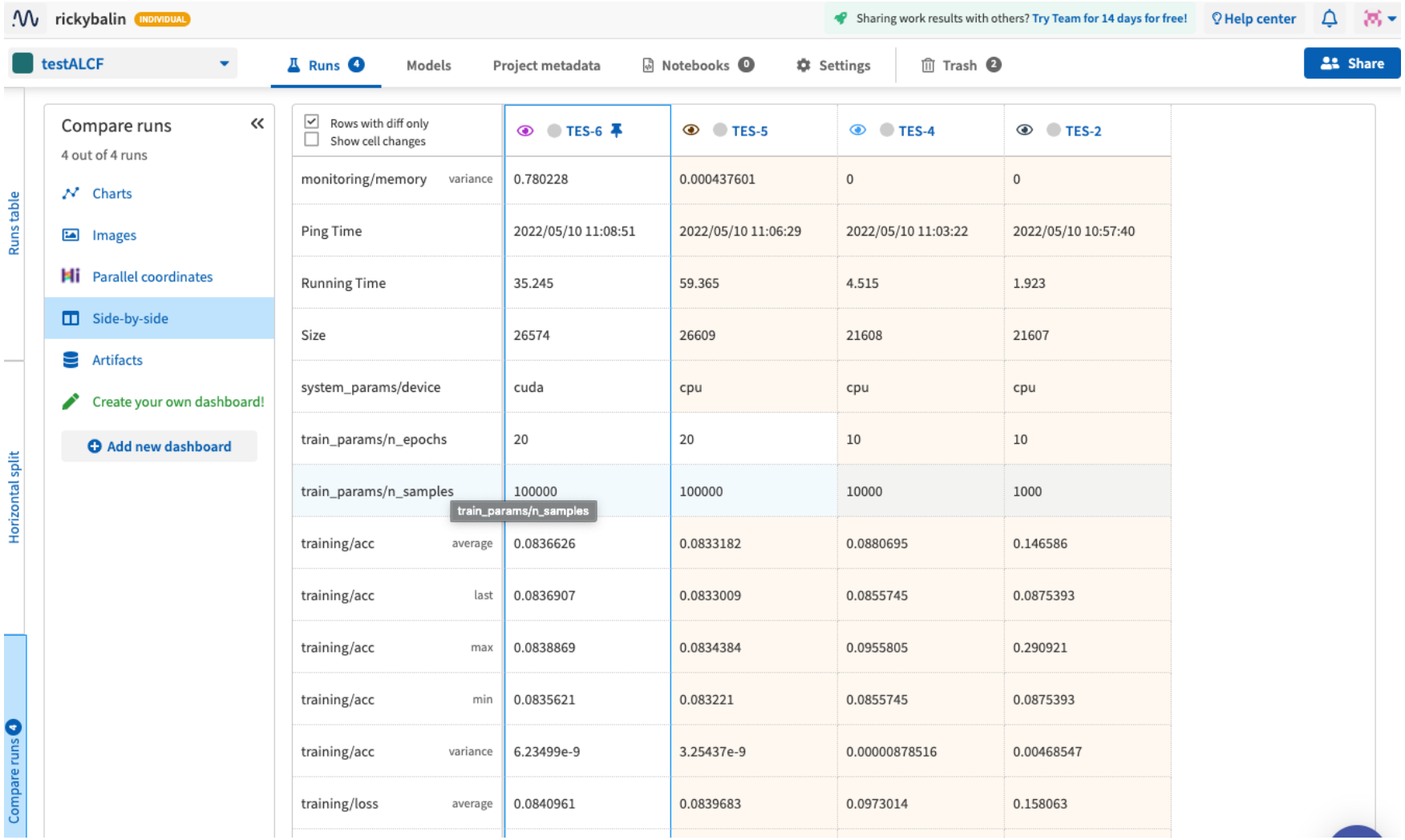
Example Training Run

- Click on run name to visualize metadata logged



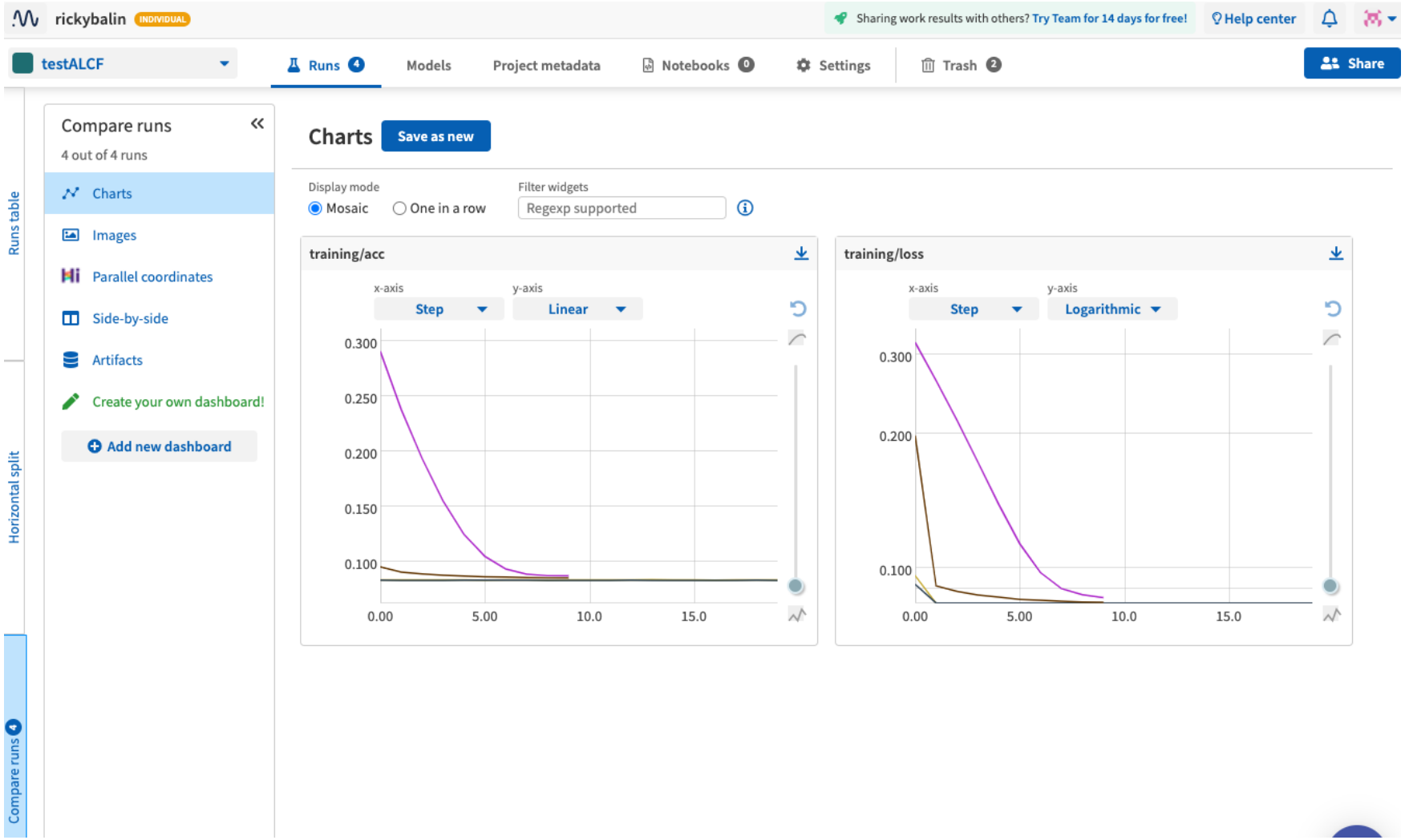
Example Training Run

- Select what runs to compare clicking on the eye next to the run name



Example Training Run

- Select what runs to compare clicking on the eye next to the run name



Comparison with W&B

Notable similarities

- Allow on-premise deployment
- Minimal changes to code for logging
- Distributed training support
- Web UI to navigate through experiments and metadata

Notable differences

- Neptune has limited TensorBoard support
- Neptune does not have interactive project level reports
- Neptune has more features to organize and search experiment metadata

Backup Slides

Charging Monitoring Hours

- Monitoring hours are the time spent actually logging metadata via the Neptune API
- Browsing, exploring, querying the metadata with the web UI or programmatically does not accumulate monitoring hours
- If logging instances are separated > 10 minutes in clock time, instances logged individually with 1 sec cost
- If logging instances are separated < 10 minutes in clock time, logging time is the entire between two instances
- Within first and last logging instance separated < 10 minutes, can log infinitely many times at no additional cost