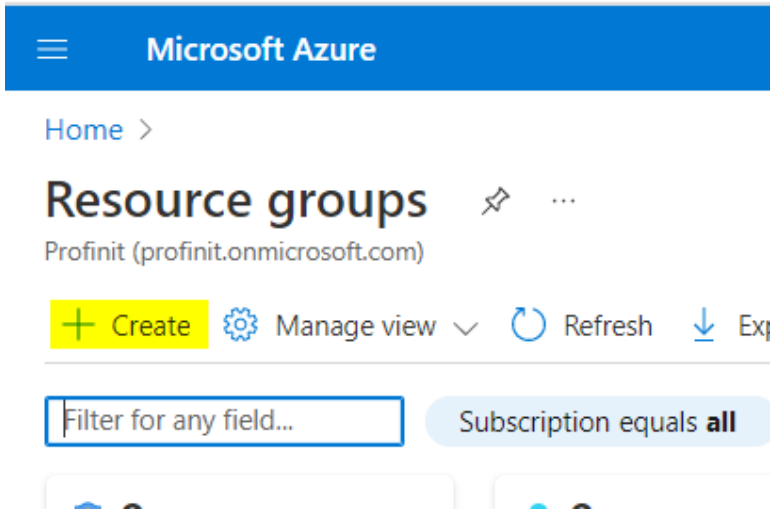
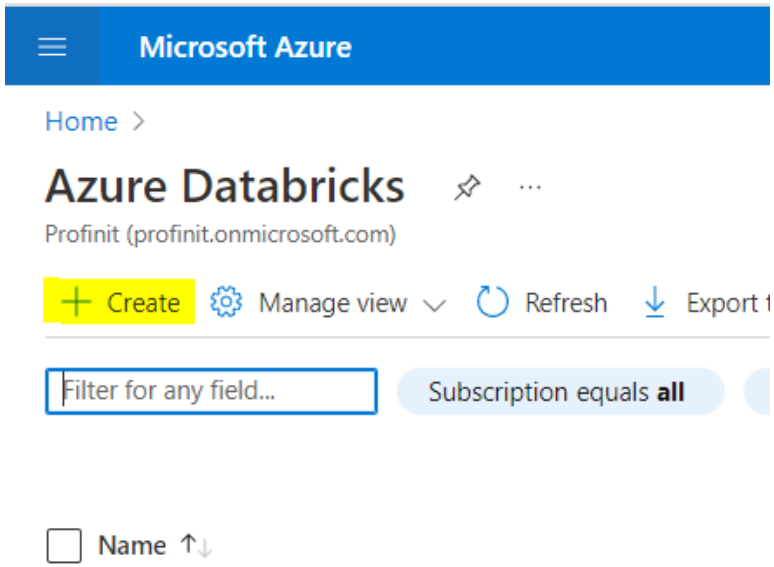
Initial steps in Azure

Login to your azure students account https://azure.microsoft.com/cs-cz/free/students/ and create a resource, where you will host your services.



Create Databricks Workspace



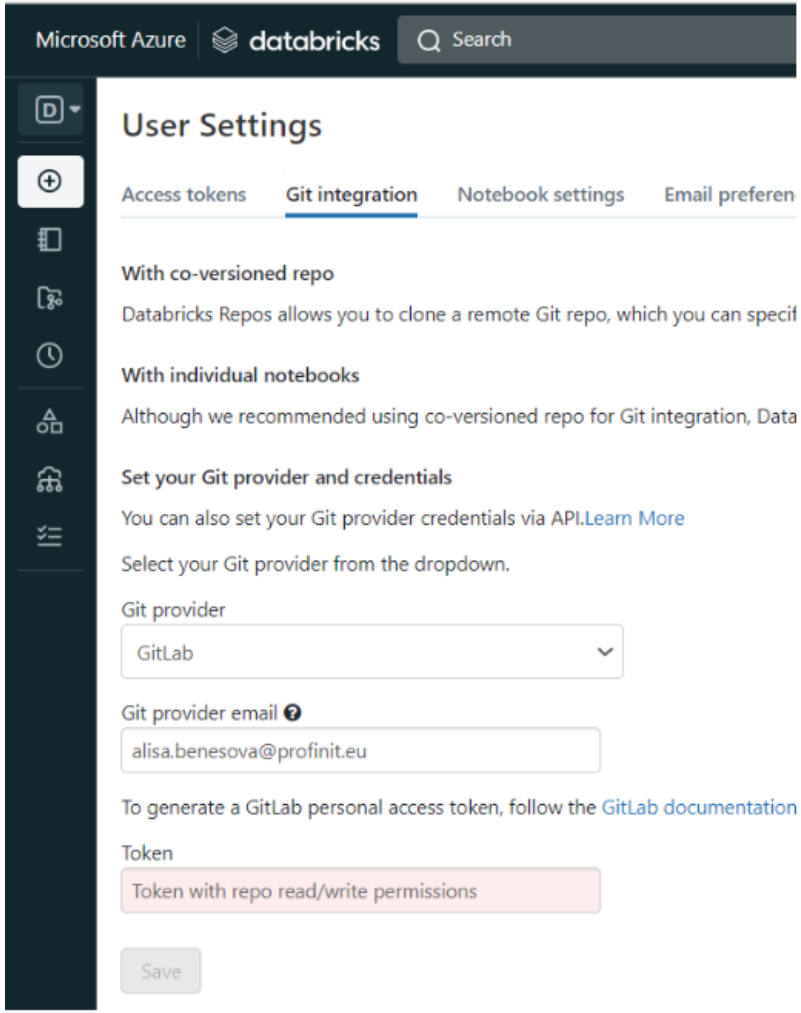
Select your newly created Dbx workspace, Launch, Sign in and voila.

Launch Workspace

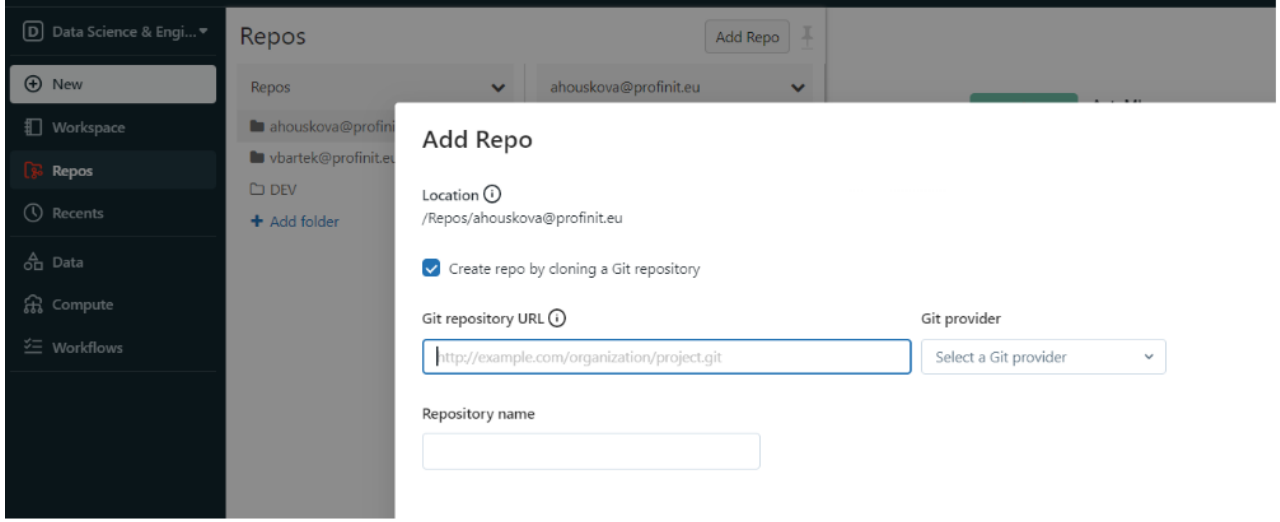
Initial steps in Databricks

Connect your GIT repo

Generate personal access token from your git provider, see github, gitlab



Navigate to Repos and add your repository; pull if anything present

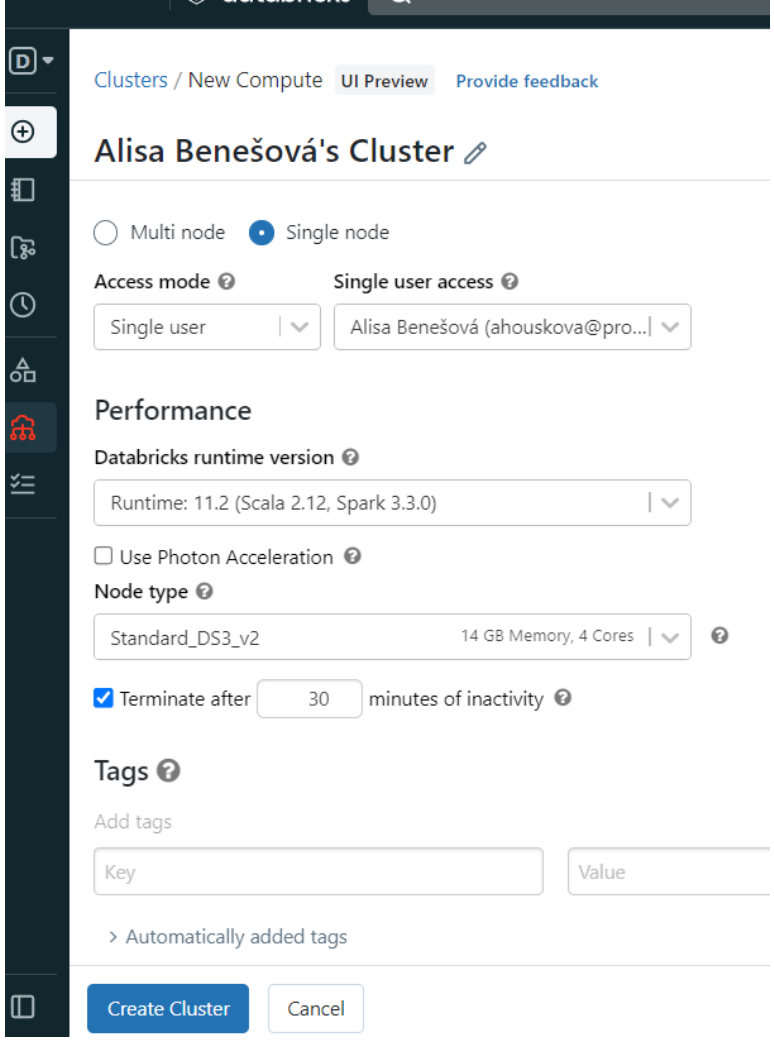


Create a cluster

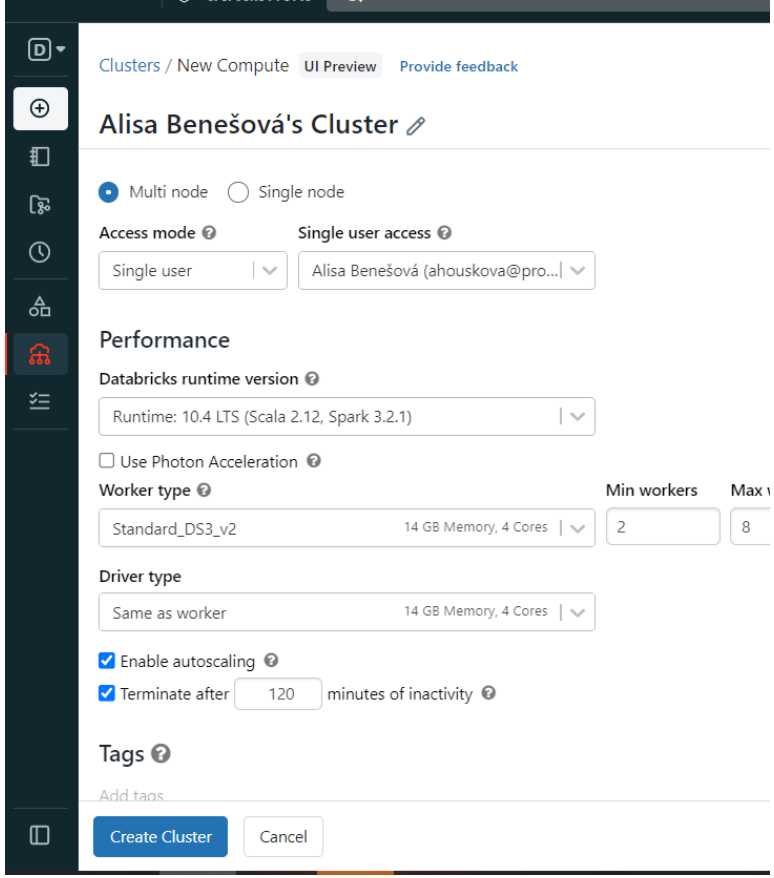
Navigate to Computer and create desired cluster. Be reasonable with resources and termination time.

Be advised, while streaming you will need to terminate the cluster manually, because the termination time is only for clusters that are idle.

For first runs and debug, single node is enough.



Once you are sure with solution, you may test your solution on a stronger cluster with bigger amount of data, see multi node cluster below.



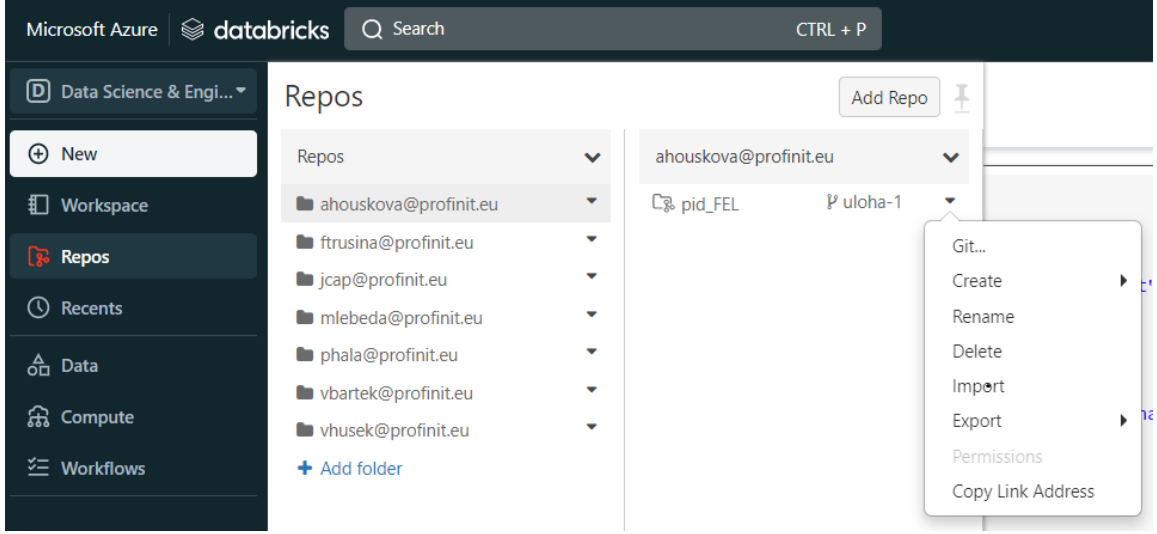
Create a Notebook

You can create either notebook, which is not connected to your repository, or create a Repo files, which you can git control. **We will use git controlled files.** GIT in Databricks environment has some limitations, eg. merging is not possible.

Repos/choose_your_repo/choose_your_branch/Create/Notebook

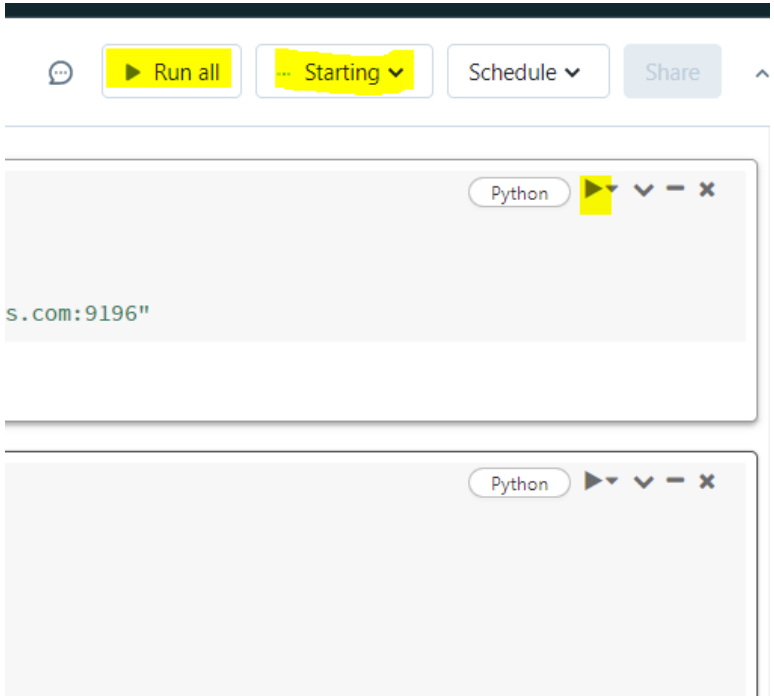
You can Create branch via *Git...* option or in git gui/console.

When you want to use Databricks cluster and Databricks features, it is not neccessary to use Databricks GUI. It is also possible to use Visual Code, with simple Azure verification and Databricks plugin. Feel free to explore, but it is beyond this course.



Once your Notebook is created, you are free to code and run it against cluster.

Starting a cluster will take some time, few minutes.



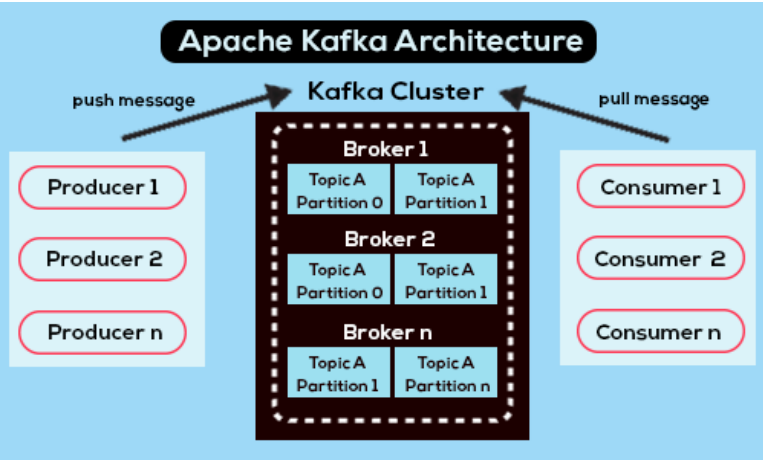
Let's get started with kafka

Kafka brief review

Kafka is

- Scalable - distributed architecture capable of handling huge amount of incoming messages in high speed.
- High Throughput - Thousands messages per second, depends on the cluster.
- Low Latency - the latency = time take for a system to process a single event is under 10ms.
- Fault Tolerance - it is possible to use replication, which helps Kafka handle failures at nodesin the cluster. Durability - Kafka can persist data for desired amount of time.
- Reliability - all of above makes Kafka very popular, the most used and reliable messaging system.

Ability to handle real-time data: Kafka supports real-time data handling and is an excellent choice when data has to be processed in real-time.



Broker Broker works as a container that can hold multiple topics with different partitions.

Topic Kafka stores its data in streams, called topics. Topics are append-only, immutable logs of events. Typically, events of the same type, or events that are in some way related, would go into the same topic.

Consumer Can read/pull/subscribe present data from kafka topics.

Producer Can write/push/publish new data to kafka topics.

Run Kafka

Download desired version in https://kafka.apache.org/downloads When on local, you will need to start zookeeper and kafka server. For this, you will need to navigate into your kafka folder in console.

```
tar -xzf kafka_2.13-3.3.1.tgz

cd kafka_2.13-3.3.1 seznamu


# Start the ZooKeeper service
bin/zookeeper-server-start.sh config/zookeeper.properties

# Start the Kafka broker service
bin/kafka-server-start.sh config/server.properties
```

Upravíte dvojitým kliknutím (nebo stisknutím klávesy Enter)

Create a topic

We will be using local server as a bootstrap server, so the address is localhost and standard port 9092.

```
bin/kafka-topics.sh --create --topic first-topic --bootstrap-server localhost:9092
```

Create some events

Once you start console producer (it is prepared script for reading from console), it will wait for your input. You can kill it with `Ctrl+C`.

```
bin/kafka-console-producer.sh --topic first-topic --bootstrap-server localhost:9092
First message
Second message
```

Read events from topic

Open a new console window, not to kill producer. Run console consumer to read messages you have sent earlier. You can read from beginning (by adding --from-beginning). Once script started, you should see appearing messages sent earlier, or real time in the producer window.

```
1 bin/kafka-console-consumer.sh --topic first-topic --from-beginning --bootstrap-server localhost:9092
```