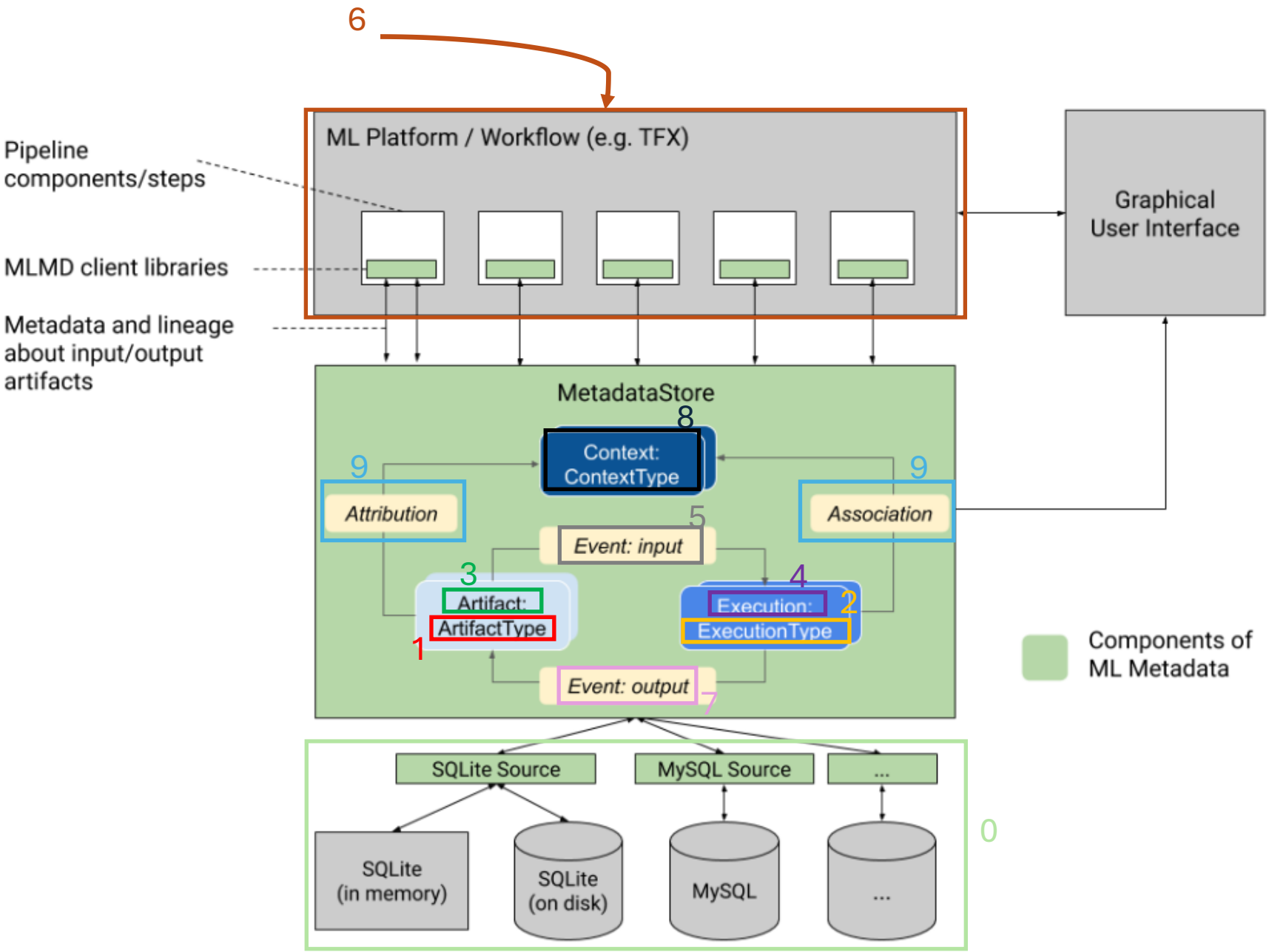
6
ML Platform / Workflow (e.g. TFX)
Pipeline components/steps
MLMD client libraries
Metadata and lineage about input/output artifacts
Graphical User Interface
MetadataStore
8
Context: ContextType
9
Attribution
9
Association
5
Event: input
3
Artifact:
ArtifactType
1
4
Execution:
ExecutionType
2
Event: output
7
Components of ML Metadata
SQLite Source
MySQL Source
...
SQLite (in memory)
SQLite (on disk)
MySQL
...
0

In [3]:
```
# Instantiate a connection config
connection_config = metadata_store_pb2.ConnectionConfig()

# Set an empty fake database proto
connection_config.fake_database.SetInParent()

# Setup the metadata store
store = metadata_store.MetadataStore(connection_config)
```

0

In [4]:

```
# Create ArtifactType for the input dataset
data_artifact_type = metadata_store_pb2.ArtifactType()
data_artifact_type.name = 'DataSet'
data_artifact_type.properties['name'] = metadata_store_pb2.STRING
data_artifact_type.properties['split'] = metadata_store_pb2.STRING
data_artifact_type.properties['version'] = metadata_store_pb2.INT

# Register artifact type to the Metadata Store
data_artifact_type_id = store.put_artifact_type(data_artifact_type)
```

1A

```
# Create ArtifactType for Schema
schema_artifact_type = metadata_store_pb2.ArtifactType()
schema_artifact_type.name = 'Schema'
schema_artifact_type.properties['name'] = metadata_store_pb2.STRING
schema_artifact_type.properties['version'] = metadata_store_pb2.INT

# Register artifact type to the Metadata Store
schema_artifact_type_id = store.put_artifact_type(schema_artifact_type)
```

1B

```
print('Data artifact type:\n', data_artifact_type)
print('Schema artifact type:\n', schema_artifact_type)
print('Data artifact type ID:', data_artifact_type_id)
print('Schema artifact type ID:', schema_artifact_type_id)
```

1A
1B
1A
1B

```
Data artifact type:
 name: "DataSet"
properties {
  key: "name"
  value: STRING
}
properties {
  key: "split"
  value: STRING
}
properties {
  key: "version"
  value: INT
}

Schema artifact type:
 name: "Schema"
properties {
  key: "name"
  value: STRING
}
properties {
  key: "version"
  value: INT
}

Data artifact type ID: 10
Schema artifact type ID: 11
```

In [5]:
```python
# Create ExecutionType for Data Validation component
dv_execution_type = metadata_store_pb2.ExecutionType()
dv_execution_type.name = 'Data Validation'
dv_execution_type.properties['state'] = metadata_store_pb2.STRING

# Register execution type to the Metadata Store
dv_execution_type_id = store.put_execution_type(dv_execution_type)

print('Data validation execution type:\n', dv_execution_type)
print('Data validation execution type ID:', dv_execution_type_id)
```

2

```
Data validation execution type:
 name: "Data Validation"
properties {
  key: "state"
  value: STRING
}

Data validation execution type ID: 12
```

In [6]:

```
# Declare input artifact of type DataSet
data_artifact = metadata_store_pb2.Artifact()
data_artifact.uri = './data/train/data.csv'
data_artifact.type_id = data_artifact_type_id
data_artifact.properties['name'].string_value = 'Chicago Taxi dataset'
data_artifact.properties['split'].string_value = 'train'
data_artifact.properties['version'].int_value = 1

# Submit input artifact to the Metadata Store
data_artifact_id = store.put_artifacts([data_artifact])[0]

print('Data artifact:\n', data_artifact)
print('Data artifact ID:', data_artifact_id)
```

3

```
Data artifact:
 type_id: 10
uri: "./data/train/data.csv"
properties {
  key: "name"
  value {
    string_value: "Chicago Taxi dataset"
  }
}
properties {
  key: "split"
  value {
    string_value: "train"
  }
}
properties {
  key: "version"
  value {
    int_value: 1
  }
}

Data artifact ID: 1
```

In [7]:
```
# Register the Execution of a Data Validation run
dv_execution = metadata_store_pb2.Execution()
dv_execution.type_id = dv_execution_type_id
dv_execution.properties['state'].string_value = 'RUNNING'

# Submit execution unit to the Metadata Store
dv_execution_id = store.put_executions([dv_execution])[0]

print('Data validation execution:\n', dv_execution)
print('Data validation execution ID:', dv_execution_id)
```

4

```
Data validation execution:
 type_id: 12
properties {
  key: "state"
  value {
    string_value: "RUNNING"
  }
}

Data validation execution ID: 1
```

In [8]:
```
# Declare the input event
input_event = metadata_store_pb2.Event()
input_event.artifact_id = data_artifact_id
input_event.execution_id = dv_execution_id
input_event.type = metadata_store_pb2.Event.DECLARED_INPUT

# Submit input event to the Metadata Store
store.put_events([input_event])

print('Input event:\n', input_event)
```

5

```
Input event:
 artifact_id: 1
execution_id: 1
type: DECLARED_INPUT
```

In [9]:

```
# Infer a schema by passing statistics to `infer_schema()`
train_data = './data/train/data.csv'
train_stats = tfdv.generate_statistics_from_csv(data_location=train_data)
schema = tfdv.infer_schema(statistics=train_stats)

schema_file = './schema.pbtxt'
tfdv.write_schema_text(schema, schema_file)

print("Dataset's Schema has been generated at:", schema_file)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.8 interpreter.

WARNING:tensorflow:From /opt/conda/lib/python3.8/site-packages/tensorflow_data_validation/utils/stats_util.py:247: tf_record_it
erator (from tensorflow.python.lib.io.tf_record) is deprecated and will be removed in a future version.
Instructions for updating:
Use eager execution and:
`tf.data.TFRecordDataset(path)`

WARNING:tensorflow:From /opt/conda/lib/python3.8/site-packages/tensorflow_data_validation/utils/stats_util.py:247: tf_record_it
erator (from tensorflow.python.lib.io.tf_record) is deprecated and will be removed in a future version.
Instructions for updating:
Use eager execution and:
`tf.data.TFRecordDataset(path)`

Dataset's Schema has been generated at: ./schema.pbtxt
```

6

This happens up there in the figure as part of ML flow (not ML metadata)

In [10]:
```
# Declare output artifact of type Schema_artifact
schema_artifact = metadata_store_pb2.Artifact()
schema_artifact.uri = schema_file
schema_artifact.type_id = schema_artifact_type_id
schema_artifact.properties['version'].int_value = 1
schema_artifact.properties['name'].string_value = 'Chicago Taxi Schema'

# Submit output artifact to the Metadata Store
schema_artifact_id = store.put_artifacts([schema_artifact])[0]

print('Schema artifact:\n', schema_artifact)
print('Schema artifact ID:', schema_artifact_id)
```

3

```
Schema artifact:
 type_id: 11
uri: "./schema.pbtxt"
properties {
  key: "name"
  value {
    string_value: "Chicago Taxi Schema"
  }
}
properties {
  key: "version"
  value {
    int_value: 1
  }
}

Schema artifact ID: 2
```

In [11]:
```
# Declare the output event
output_event = metadata_store_pb2.Event()
output_event.artifact_id = schema_artifact_id
output_event.execution_id = dv_execution_id
output_event.type = metadata_store_pb2.Event.DECLARED_OUTPUT

# Submit output event to the Metadata Store
store.put_events([output_event])

print('Output event:\n', output_event)
```

7

```
Output event:
 artifact_id: 2
execution_id: 1
type: DECLARED_OUTPUT
```

In [15]:
```
# Mark the `state` as `COMPLETED`
dv_execution.id = dv_execution_id
dv_execution.properties['state'].string_value = 'COMPLETED'

# Update execution unit in the Metadata Store
store.put_executions([dv_execution])

print('Data validation execution:\n', dv_execution)
```

4

```
Data validation execution:
 id: 1
type_id: 12
properties {
  key: "state"
  value {
    string_value: "COMPLETED"
  }
}
```

In [16]:
```
# Create a ContextType
expt_context_type = metadata_store_pb2.ContextType()
expt_context_type.name = 'Experiment'
expt_context_type.properties['note'] = metadata_store_pb2.STRING

# Register context type to the Metadata Store
expt_context_type_id = store.put_context_type(expt_context_type)
```

8

Similarly, you can create an instance of this context type and use the `put_contexts()` method to register to the store.

In [17]:
```
# Generate the context
expt_context = metadata_store_pb2.Context()
expt_context.type_id = expt_context_type_id
# Give the experiment a name
expt_context.name = 'Demo'
expt_context.properties['note'].string_value = 'Walkthrough of metadata'

# Submit context to the Metadata Store
expt_context_id = store.put_contexts([expt_context])[0]

print('Experiment Context type:\n', expt_context_type)
print('Experiment Context type ID: ', expt_context_type_id)

print('Experiment Context:\n', expt_context)
print('Experiment Context ID: ', expt_context_id)
```

8

```
Experiment Context type:
 name: "Experiment"
properties {
  key: "note"
  value: STRING
}

Experiment Context type ID:  13
Experiment Context:
 type_id: 13
name: "Demo"
properties {
  key: "note"
  value {
    string_value: "Walkthrough of metadata"
  }
}

Experiment Context ID:  1
```

In [18]:
```
# Generate the attribution
expt_attribution = metadata_store_pb2.Attribution()
expt_attribution.artifact_id = schema_artifact_id
expt_attribution.context_id = expt_context_id

# Generate the association
expt_association = metadata_store_pb2.Association()
expt_association.execution_id = dv_execution_id
expt_association.context_id = expt_context_id

# Submit attribution and association to the Metadata Store
store.put_attributions_and_associations([expt_attribution], [expt_association])

print('Experiment Attribution:\n', expt_attribution)
print('Experiment Association:\n', expt_association)
```

9

```
Experiment Attribution:
 artifact_id: 2
context_id: 1

Experiment Association:
 execution_id: 1
context_id: 1
```