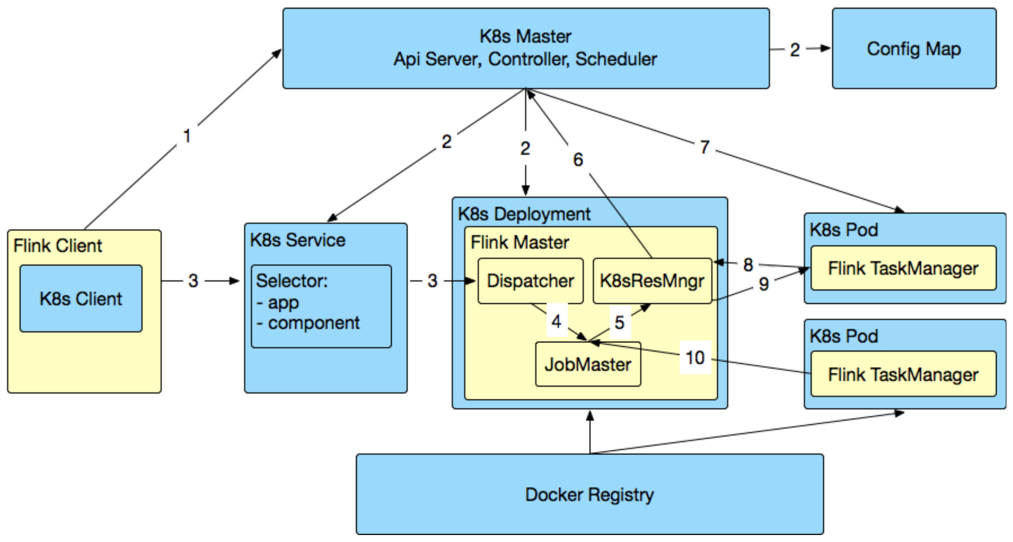
Apache Flink 1.10.0 Features [Set 1]

Apache Flink 1.10.0 is a major release with many significant features & I drafted first list of notable features according to me. Set 2 will follow next.

Kubernetes on Flink

1.10.0 Flink has active kubernetes integration, where ResourceManager can launch Kubernetes Pods. Flink on Kubernetes is now available.
This feature is based on *Natively Running Flink on K8s*.

Follow doc.



Supporting Features -

- KubernetesClusterDescriptor & ResourceManager to deploy K8s Session

```
FlinkKubeClient flinkKubeClient = getFabric8FlinkKubeClient();
                    final KubernetesClusterDescriptor descriptor = new
KubernetesClusterDescriptor(FLINK_CONFIG, flinkKubeClient);

ClusterSpecification clusterSpecification = new ClusterSpecification.ClusterSpecificationBuilder()
          .setMasterMemoryMB(1234)
          .setTaskManagerMemoryMB(1235)
          .setNumberTaskManagers(1)
          .setSlotsPerTaskManager(1)
          .createClusterSpecification();

// Deploy a Session Cluster
descriptor.deploySessionCluster(clusterSpecification)
// Killing a Cluster
descriptor.deploySessionCluster(clusterSpecification)
```

- CLi Tools to start/stop Kubernetes Session, entrypoint scripts
- Uniform Kubernetes Executor naming
- Provision for keeping memory for hvm off-heap
- Dockerfile for building image
- Documentation

Phase 2 Advanced features is already on the pipeline.

Pluggable Module

This feature makes Flink to provide interface for custom pluggable Modules like Hive.

Main Motivations:

1. Enable users to integrate Flink with cores and built-in objects of other systems, so users can reuse whatever they are familiar with in other SQL systems seamlessly as core and built-ins of Flink SQL and Table
2. Empower users to write code and do customized development for Flink table core

seperate blog

Integration of Apache Hive

Integration of Hive inside Flink Ecosystem is by itself a huge task, but 1st version of support is now available.
First set of features available-

Support of simple Hive UDF
Support of Hive GenericUDF, UDTF, UDAF
Data & Metadata Interoperability
Data type compatibility

Python User-Defined Stateless Function UDF

Flink now supports Python UDF using Beam portability framework.

User-Defined AggregateFunction, Python DataStream API support and Pandas support is not available in this release.

```
env.register_function("one", udf(
        lambda: 1, input_types=[], result_type=DataTypes.BIGINT(), deterministic=True))
t = self.env.from_elements([(1, 2), (2, 5), (3, 1)], ['a', 'b'])
```

```
t.select("one()").insert_into("Results")
```

Fine Grained Resource Management

Fine Grained RM is a feature to provide custom resource usages while defining an UDF function. This gives more control to user for task definition.

Following are the user-defined custom resources currently available -

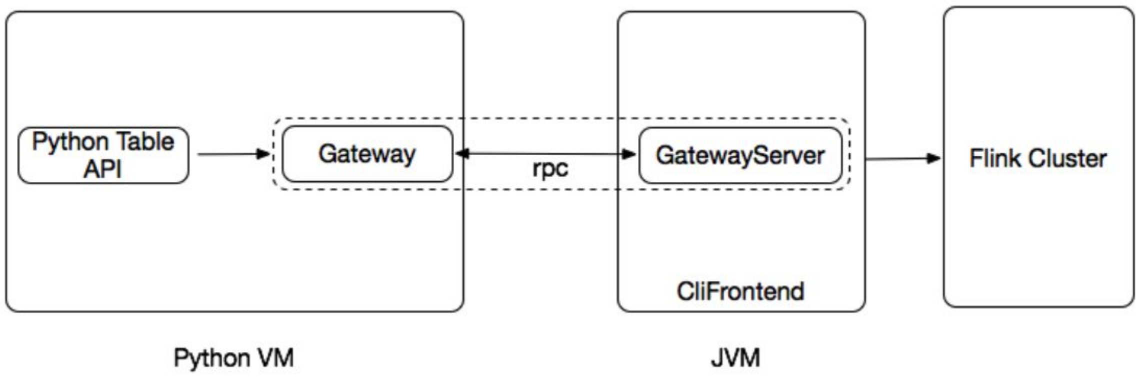
- CPU Cores
- Managed/Unmanaged on-heap & off-heap
- Network Memory, Direct & native Memory
- Extended Resources like GPU

A Resource Specification interface is defined ResourceSpec.

Customized Checkpoint Policy for File Size

Now Flink supports a new Rolling Policy for file size limit. This policy prevents file size from growing too big.
Follow up interface OnCheckpointRollingPolicy

Python Support in Flink API
Flink now brings more broad support of Python for its Table APi.



Flink's goal for Python Table API as follows:

- Users can write Flink Table API job in Python, and should mirror Java / Scala Table API
- Users can submit Python Table API job in the following ways:
  - Submit a job with python script, integrate with `flink run`
  - Submit a job with python script by REST service
  - Submit a job in an interactive way, similar `scala-shell`
  - Local debug in IDE.
- Users can write custom functions(UDF, UDTF, UDAF)
- Pandas functions can be used in Flink Python Table API

Spreading slots across TaskExecutors

This feature brings evenly distribution of slots across all available TaskManagers.

Now Flink's ResourceManager tries to fulfill slot requests with slots whose TaskExecutor's utilization is minimal if there are multiple slots matching the resource requirements.

Introduction of `ClusterOptions.EVENLY_SPREAD_OUT_SLOTS_STRATEGY`.

```
SlotPoolResource slotPoolResource = new
SlotPoolResource(LocationPreferenceSlotSelectionStrategy.createEvenlySpreadOut());
```

Data Compression to Shuffle

Flink now supports Data Compression to Shuffle, which can really reduce disk IO and Network IO. This feature is going to boost performance in all shuffle heavy operations like Join.

Basic Vector & Matrix for ML

Flink now provides Sparse/Dense Vector & Dense Matrix for basic operations related to machine learning.

```
DenseMatrix matA = new DenseMatrix(new double[][]{
                    new double[]{1, 3, 5},
                    new double[]{2, 4, 6},
          });
DenseMatrix matB = DenseMatrix.ones(2, 3);

//Dense Vector
DenseVector vec = new DenseVector(new double[]{1, 2, -3});

// SparseVector
SparseVector sv = new SparseVector(4, new int[]{0, 2}, new double[]{1., 1.});
```

Job Listener to notify state

Flink now provides a JobListener that can be registered by users to get notifications when the status changes.
Any Notebook like Zeppelin, where user can submit jobs, but the framework needs to handle the job in order to manage.

```
env.registerJobListener(new JobListener() {
          @Override
          public void onJobSubmitted(JobContext jobContext, Throwable throwable) {
                    submissionLatch.trigger();
          }

          @Override
          public void onJobExecuted(JobExecutionResult jobExecutionResult,
Throwable throwable) {
                    executionLatch.trigger();
          }
});
```

Create/Drop table support for SqlClient

Flink SQL Client supports create table and drop table

```
CREATE TABLE … WITH

DROP TABLE [IF EXISTS] …
```

Python API functions

Python table API now supports following matching scala/java functions -
Temporary Table & View functions
Insert_into & from_path.

```
…with_schema(Schema()
          .field("a", DataTypes.INT())
          .field("b", DataTypes.STRING())) \
.create_temporary_table("temporary_table_2")

#view
env.create_temporary_view(
"temporary_view_2",
t_env.from_elements([(1, 'Hi')], ['a', 'b']))

#from_path
env.from_path("temporary_view_1")

#insert_into
env.insert_into("Sinks", t_env.from_elements([(1, "Hi", "Hello")], ["a", "b", "c"]))
```

Support to parse watermark in SQL

Support to parse watermark Syntax like following

```
CREATE TABLE tbl (
        x0 bigint,
        x1 varchar,
        x2 boolean
        WATERMARK FOR x0 AS (SELECT x1 from tbl)
) WITH (...)
```

Developer Changes

- Introduction of new Executor Interface for abstraction the execution/submission logic from environments.
- Addition new Executor Interface in SQL cli
- Flink Table Api Migration from Scala -> Java, ensuring future stability and contributions.
- Flink supports ARM based Memory Architecture.
- Vectorized ORC InputFormat for blink Runtime

Other Notable Changes

- Flink now added a new connector to support Elasticsearch 7.x
- Support Any Consistency Level for InfluxDB
- Bug: Flink-s3-fs-hadoop library breaks in plugins mechanism
- SqlTimestamp supports nanoseconds' precision
- Simplification of Join Condition in Logical Phase