Cheatography

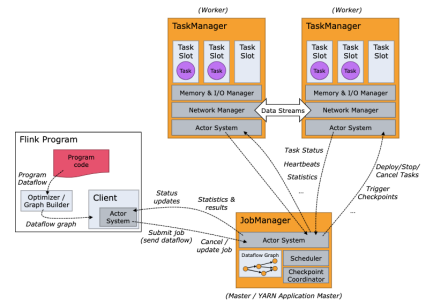
Flink Cheat Sheet

by mliafol via cheatography.com/70309/cs/39328/

Flink's API

SQL	Highlevel Language
Table API	Declarative DSL
DataStream/DataSet API	Core API
Stateful Stream Processing	Lower level building block

Flink architecture



The client is not part of the runtime and program execution, but is used to prepare and send a dataflow to the JobManager. After that, client can disconnect(detached mode), or stay connected (attached mode)

DataSream	
Data Stream	Immutable collections of data that can contain duplicates, can either be finite or unbounded
Flink program	Obtain en execution environment
	Load/create the initial data
	Transformation
	Where to put the result
	Trigger the execution
Flink program executed lazily	do not happen directly. Rather, operation is created and added to dataflow graph

Datasource Overview		
StreamExecution-Environment	getExecutionEnvironment();	
Filebase datasources	env.readFile(fileInputFormat, path, watchType, interval, pathFilter, typeInfo)	watchType: can be ileProcessingMode.PROCESS_CONTINUOUSLY or FileProcessingMode.PROCESS_ONCE
Socket-based:	env.socketTextStream	
Collection based	env. fromCollection, env.fromElements	
Custom source	env.addSource	
A sequence numbers	env.generateSequence(0, 1000)	

Data sink overview	
writeAsText() / TextOutputFormat	Writes elements line-wise as Strings.
writeAsCsv() / CsvOutputFormat	Writes tuples as comma-separated value files.
print() / printToErr()	
writeUsingOutputFormat() / FileOutputFormat	Method and base class for custom file outputs
writeToSocket	
addSink	

Timely Stream processing	
Processing time	**System time of the machine that is executing the respective operation**
	No coordination between streams and machines
	Best performance and lowest latency
	Not provide determinism in distributed and async enviroments

Timely Stream processing (cont)	
Event time	**The time that each individual event occurred on its producing device**
	Extract from the records
	Consistent and deterministic
	High latency while waiting for out-of-order events
Watermark	**A mechanism to measure progress in event time**
	Flow as part of the data stream and carry a timestamp t
	Watermark(t) declares that event time has reached time t, there should be no more elements with timestamp <= t
	Crucial for out-of-order streams
Watermark strategy	TimestampAssigner + WatermarkGenerator
WatermarkGenerator	onEvent: Called for every event
	onPeriodicEmit: call periodically, and might emit a new watermark or not
	punctuate or periodic
WatermarkStrategy.forMonotonousTimestamps();	Event time itself
WatermarkStrategy.forBoundedOutOfOrderness	Watermark lags behind the maximum timestamp seen in the stream by a fixed amount of time

State	
Stateful operator	Remember information acc
Keyed state	Embedded key/value store
	Partitioned and distributed strictly together with the streams
	Only on keyed stream
State persistence	Fault tolerance: stream replay and checkpointing
Checkpoint	Marks a specific point in each of the input streams along with the corresponding state for each operators
	Drawing consistent snapshots of the distributed data stream and operator state
Stream barriers	Injected into the data stream and flow with the records as part of the data stream

State (cont)	
	Separated the records in the data stream into the set of records that goes into the current snapshot, and the records that go into the next snapshot.
	The point where the barriers for snapshot n are injected, is the position in the source stream up to which snapshot cover the data
	Alignment phrase: Receive barrier for snapshot n of one incoming stream, operator need to wait until receive all others input
Snapshot operator state	At the point in time when they received all barriers from input streams and before emitting the barriers to their output streams
	For each parallel stream data source, the offset/position in the stream when the snapshot started
	For each operator, a pointer to the state that was stored
Unaligned checkpoint	Reacts on the first barrier that is stored in its input buffers
Checkpoint	Simple external dependencies
	Immutable and versioned
	Decouple the stream transport from the persistence mechanism
Backpressure	Slow receiver makes the senders slow down in order not to overwhelm the receiver
Snapshot	generic term refer to global, consistent image of a state of a Flink job

RocksDB tunning	
Incremental checkpoints	Record the changes compared to the previous completed checkpoint, instead of producing a full, self-contained backup
Timers	Schedule actions for later => save on healp =>
	state.backend.rocksdb.timer-service.factory =heap
Tunning rocksdb memory	Flink's managed memory to buffer and cache
	Increase the amount of managed memory

Window	
Definition	Split stream into buckets of finite size, over which we can apply computations
Keyed windows	.keyBy().window().[.triger()][.evictor()][.allowedLateness()][.sideOutputLateData()].reduce/aggregate/apply
	Be performed in parallel by multiple tasks
Non-Keyed windows	windowAll().[.triger()][.evictor()][.allowedLateness()][.sideOutputLateData()].reduce/aggregate/apply
	Be performed by a single task (parallelism = 1)
Lifecycle	Created : the first element belong to this window arrie
	Removed: the time passes its end timestamp + allowed lateness
Window Assigner	Responsible for assigning each incoming element to 1 or more windows
	Assign based on time: start timestamp (inclusive) and an end timestime(exclusive)
	TumblingWindows: each element to **a window** of a specified window size. **Fixed size and not overlap**
	SlidingWindows: each element to **windows**. **Fixed size and can be overlapping** (window slide < window size)
	SessionWindows: assigner groups elements by **sessions of activity**. Dont overlap, dont have fixed time. Close when it does not receive elements for a certain period of time
	GlobalWindows: all elements with the same key to **same global window**. Only useful if specify a custom trigger, because it does not have a natural end
Window Functions	Computation that perform on each of windows
ReduceFunction	Incrementally aggregate
	Two elements from the input are combined to produce an output element with the same type
AggregateFunction	Generalised version of a ReduceFunction with 3 types: IN, ACC, OUT
	Methods: creating initial accumulator, merging, extract output
ProcessWindowFunction	Iterable containing all the elements of the window
	Context object with time and state information