**Index**

## A

adding Flink internal dependencies, [107](#_bookmark398) aggregation operations, [23](#_bookmark458), [88](#_bookmark22)

Amazon S3, [238](#_bookmark23)

analytical processing, [3](#_bookmark224)

anomaly detection, [6](#_bookmark348)

Apache Calcite, [282](#_bookmark173)

Apache Cassandra, [199](#_bookmark0)-[202](#_bookmark24) Apache Flink

architecture of, [37](#_bookmark133)-[69](#_bookmark134)

basic Flink application, [13](#_bookmark0)-[15](#_bookmark160) benefits of, [1](#_bookmark133), [12](#_bookmark1)

community resources, [283](#_bookmark25) Hello, Flink! example, [79](#_bookmark276)-[83](#_bookmark379) operation of, [245](#_bookmark166)-[280](#_bookmark26)

real-world use cases, [6](#_bookmark27) resources, [281](#_bookmark133)-[283](#_bookmark399)

Web UI, [270](#_bookmark349)

Apache Flink architecture

checkpoints, savepoints, and state recovery, [58](#_bookmark350)-[69](#_bookmark28)

data transfer in Flink, [44](#_bookmark1)-[47](#_bookmark135) event-time processing, [47](#_bookmark192)-[52](#_bookmark459)

state management, [53](#_bookmark1)-[58](#_bookmark174)

system architecture, [37](#_bookmark276)-[43](#_bookmark460)

Apache Hadoop YARN, [4](#_bookmark218), [225](#_bookmark183), [236](#_bookmark29) Apache Kafka

compatibility with Flink, [5](#_bookmark351) sink connector, [190](#_bookmark247)-[193](#_bookmark477)

source connector, [187](#_bookmark225)-[190](#_bookmark226)

Apache ZooKeeper, [42](#_bookmark317), [232](#_bookmark461) application deployment

configuring logging behavior, [279](#_bookmark30) controlling task scheduling, [260](#_bookmark380)-[263](#_bookmark318)

debugging applications, [76](#_bookmark31)

deployment modes, [221](#_bookmark277)-[232](#_bookmark295)

evolving stateful applications, [174](#_bookmark167)-[177](#_bookmark32) executing applications, [83](#_bookmark175)

monitoring clusters and applications, [270](#_bookmark193)-[279](#_bookmark33)

running and managing applications, [245](#_bookmark411)-[260](#_bookmark248)

running applications in IDEs, [75](#_bookmark176)

tuning checkpoint and recovery, [263](#_bookmark400)-[270](#_bookmark168) AssignerWithPeriodicWatermarks function, [52](#_bookmark219),

[112](#_bookmark278)

AssignerWithPunctuatedWatermarks function, [52](#_bookmark279), [114](#_bookmark34)

asynchronous checkpoints, [66](#_bookmark35)

at-least-once result guarantees, [35](#_bookmark2), [66](#_bookmark249) at-most-once result guarantees, [34](#_bookmark319) authentication, [243](#_bookmark302)

## B

backpressure, [22](#_bookmark3)

barrier alignment, [63](#_bookmark381)

batch processing, [281](#_bookmark352)

bounded intervals, [122](#_bookmark36) BoundedOutOfOrdernessTimestampExtractor,

[113](#_bookmark427)

broadcast data exchange strategy, [19](#_bookmark37), [95](#_bookmark462)

broadcast state, [54](#_bookmark353), [93](#_bookmark4), [160](#_bookmark478)

## C

case classes, [99](#_bookmark401), [103](#_bookmark296)

Chandy–Lamport algorithm, [61](#_bookmark320)

CheckpointedFunction interface, [164](#_bookmark38), [203](#_bookmark354) checkpointing and recovery

#### 285

asynchronous checkpoints, [66](#_bookmark35)

barrier alignment, [63](#_bookmark381)

checkpoint barriers, [61](#_bookmark412)

checkpointing algorithm, [61](#_bookmark280)

consistent checkpoints, [59](#_bookmark0)-[61](#_bookmark227) enabling failure recovery, [166](#_bookmark321)

incremental checkpointing, [66](#_bookmark136), [118](#_bookmark5), [170](#_bookmark137),

[243](#_bookmark39), [267](#_bookmark184)

performance implications, [65](#_bookmark447)

receiving checkpoint completion notifica‐ tions, [166](#_bookmark0)

savepoints, [66](#_bookmark303)-[69](#_bookmark28), [167](#_bookmark250)

system configuration, [243](#_bookmark40)

tuning, [263](#_bookmark400)-[270](#_bookmark168)

CheckpointListener interface, [166](#_bookmark177)

classloading, [239](#_bookmark355)

cluster monitoring with REST API, [253](#_bookmark228) code examples, obtaining and using, [xi](#_bookmark41), [72](#_bookmark229) command-line client

cancelling applications, [250](#_bookmark6)

deleting savepoints, [249](#_bookmark322)

listing running applications, [248](#_bookmark323) scaling applications in and out, [251](#_bookmark324)

starting applications from savepoints, [250](#_bookmark297) taking and disposing savepoints, [249](#_bookmark6) Windows set up, [247](#_bookmark138)

complex event processing (CEP), [6](#_bookmark325), [281](#_bookmark251)

composite keys, [103](#_bookmark42) connected broadcast state, [160](#_bookmark478) ConnectedStreams object, [91](#_bookmark5) consistency guarantees

exactly-once state consistency, [184](#_bookmark43) for source and sink connectors, [186](#_bookmark161) idempotent writes, [184](#_bookmark413)

transactional writes, [185](#_bookmark230)

consistent checkpoints, [59](#_bookmark0) containers

bundling applications in, [258](#_bookmark44) Docker, [258](#_bookmark281)-[260](#_bookmark248)

Kubernetes, [259](#_bookmark231)

Minikube, [260](#_bookmark45)

count-based windows, [25](#_bookmark428), [123](#_bookmark326)

CPU resources, [240](#_bookmark46)

createTypeInformation macro function, [101](#_bookmark402) credit-based flow control, [45](#_bookmark232)

## D

data dependencies, [17](#_bookmark403)

data exchange strategies, [19](#_bookmark0), [94](#_bookmark178)

data infrastructures

analytical processing and, [3](#_bookmark224) data pipelines, [8](#_bookmark0)

event-driven applications, [6](#_bookmark194)

lambda architecture, [10](#_bookmark282)

microservices design pattern, [2](#_bookmark429), [6](#_bookmark382)

streaming analytics, [8](#_bookmark283)

traditional design, [2](#_bookmark47)

data parallelism, [18](#_bookmark404), [40](#_bookmark252)

data pipelines, [8](#_bookmark0)

data sinks, [17](#_bookmark448), [23](#_bookmark139), [184](#_bookmark1)

data sources, [17](#_bookmark430), [23](#_bookmark48), [184](#_bookmark1) data streams

definition of term, [20](#_bookmark49) operations on, [22](#_bookmark356)

data transfer

credit-based flow control, [45](#_bookmark232) record batching technique, [44](#_bookmark1) task chaining, [46](#_bookmark0)

data types

case classes, [99](#_bookmark401), [103](#_bookmark296)

creating TypeInformation, [100](#_bookmark463)

explicitly providing TypeInformation, [102](#_bookmark7) overview of, [97](#_bookmark414)

POJOs, [100](#_bookmark8), [103](#_bookmark296)

primitive types, [98](#_bookmark415)

serializing, [98](#_bookmark233)

special-purpose types, [100](#_bookmark383)

tuples, [98](#_bookmark479), [103](#_bookmark0)

data warehouses, [3](#_bookmark327) dataflow programming

data exchange strategies, [19](#_bookmark0)

data parallelism and task parallelism, [18](#_bookmark404) dataflow graphs, [17](#_bookmark357)

DataSet API, [281](#_bookmark352) DataStream API

data types, [97](#_bookmark414)-[102](#_bookmark234)

defining keys and referencing fields, [102](#_bookmark358) function implementation, [105](#_bookmark1)-[107](#_bookmark50) functions

AggregateFunction, [127](#_bookmark220)

AssignerWithPeriodicWatermarks, [52](#_bookmark219),

[112](#_bookmark431), [189](#_bookmark51)

AssignerWithPunctuatedWatermarks, [52](#_bookmark279), [112](#_bookmark195), [189](#_bookmark51)

BroadcastProcessFunction, [116](#_bookmark140), [145](#_bookmark196), [161](#_bookmark416)

CoFlatMapFunction, [91](#_bookmark52)

CoMapFunction, [91](#_bookmark52)

CoProcessFunction, [116](#_bookmark140), [120](#_bookmark253), [145](#_bookmark196)

FilterFunction, [86](#_bookmark9), [105](#_bookmark53)

FlatMapFunction, [86](#_bookmark298), [155](#_bookmark141)

JoinFunction, [147](#_bookmark359)

KeyedBroadcastProcessFunction, [116](#_bookmark140),

[145](#_bookmark196), [161](#_bookmark416)

KeyedProcessFunction, [116](#_bookmark140)-[118](#_bookmark54)

KeySelector, [104](#_bookmark169), [117](#_bookmark284)

MapFunction, [85](#_bookmark55), [105](#_bookmark53)

ParallelSourceFunction, [202](#_bookmark56)

ProcessAllWindowFunction, [116](#_bookmark140)

ProcessFunction, [105](#_bookmark53), [116](#_bookmark140)

ProcessJoinFunction, [116](#_bookmark140), [146](#_bookmark197)

ProcessWindowFunction, [116](#_bookmark140), [127](#_bookmark285), [129](#_bookmark360)

ReduceFunction, [89](#_bookmark57), [127](#_bookmark220)

RichFlatMapFunction, [106](#_bookmark286)

RichMapFunction, [106](#_bookmark286)

SinkFunction, [206](#_bookmark58)-[207](#_bookmark254)

SourceFunction, [52](#_bookmark59), [111](#_bookmark39), [202](#_bookmark56)-[204](#_bookmark464) Hello, Flink! example, [79](#_bookmark276)-[83](#_bookmark379)

including external and Flink dependencies, [107](#_bookmark398)

methods

DataStream.assignTimestampsAndWa‐ termarks(), [112](#_bookmark60)

DataStream.broadcast(), [92](#_bookmark221), [95](#_bookmark462), [161](#_bookmark304)

DataStream.connect(), [91](#_bookmark5)

DataStream.filter(), [85](#_bookmark384)

DataStream.flatMap(), [86](#_bookmark298), [91](#_bookmark52)

DataStream.global(), [96](#_bookmark10)

DataStream.keyBy(), [87](#_bookmark480), [92](#_bookmark11), [94](#_bookmark361), [103](#_bookmark235)-[104](#_bookmark255),

[154](#_bookmark198)

DataStream.map(), [84](#_bookmark481), [91](#_bookmark52)

DataStream.partitionCustom(), [96](#_bookmark61)

DataStream.process(), [130](#_bookmark385)

DataStream.rebalance(), [95](#_bookmark10)

DataStream.rescale(), [95](#_bookmark62)

DataStream.shuffle(), [94](#_bookmark449)

DataStream.split(), [93](#_bookmark256)

DataStream.union(), [90](#_bookmark63)

KeyedStream.max(), [88](#_bookmark236)

KeyedStream.maxBy(), [88](#_bookmark305)

KeyedStream.min(), [88](#_bookmark199)

KeyedStream.minBy(), [88](#_bookmark257)

KeyedStream.reduce(), [89](#_bookmark200)

KeyedStream.sum(), [88](#_bookmark179)

SplitStream.select(), [93](#_bookmark287)

setting parallelism, [96](#_bookmark258)

time characteristics, [109](#_bookmark362)

timestamp and watermark access, [50](#_bookmark4), [111](#_bookmark259)

transformation operations, [83](#_bookmark432)-[96](#_bookmark64)

debugging applications, [72](#_bookmark142), [76](#_bookmark31)

delayed events, [29](#_bookmark405) (see also late elements) dependencies, [107](#_bookmark398), [236](#_bookmark143)

deployment modes (see also application deployment; set up)

Apache Hadoop YARN, [225](#_bookmark183) Docker, [223](#_bookmark417), [258](#_bookmark281)-[260](#_bookmark248)

Kubernetes, [228](#_bookmark260)-[232](#_bookmark295), [259](#_bookmark231)

standalone clusters, [221](#_bookmark418) development environment

bootstrapping Flink Maven projects, [76](#_bookmark306) running and debugging applications, [72](#_bookmark142) software required, [71](#_bookmark288)

disk storage, [242](#_bookmark185)

Dispatcher, [39](#_bookmark12)

distributed data stream processing, [37](#_bookmark276) distribution transformations

broadcast, [95](#_bookmark462)

custom, [96](#_bookmark61)

global, [96](#_bookmark10)

overview of, [94](#_bookmark178)

random, [94](#_bookmark449)

rescale, [95](#_bookmark62)

round-robin, [95](#_bookmark10)

Docker, [223](#_bookmark417), [258](#_bookmark281)-[260](#_bookmark248)

## E

eager watermarks, [31](#_bookmark65)

end-to-end result guarantees, [35](#_bookmark328), [184](#_bookmark1)-[186](#_bookmark180)

event logs, [5](#_bookmark261), [35](#_bookmark35)

event-driven applications, [6](#_bookmark194) event-time processing

benefits of Flink for, [47](#_bookmark192) configuration of event-time, [110](#_bookmark66) event time, [29](#_bookmark307), [31](#_bookmark308)

timestamp assignment and watermark gen‐ eration, [52](#_bookmark0), [111](#_bookmark40)

timestamps, [47](#_bookmark465)

watermark propagation and event time, [49](#_bookmark185)-[51](#_bookmark466)

watermarks, [48](#_bookmark162)

event-time windows, [29](#_bookmark363)

EventTimeSessionWindows assigner, [139](#_bookmark364)

evictors, [136](#_bookmark201), [144](#_bookmark67)

evolution of stateful applications, [174](#_bookmark167)-[177](#_bookmark32) exactly-once result guarantees, [35](#_bookmark163), [60](#_bookmark467)

exactly-once state consistency, [58](#_bookmark350)-[65](#_bookmark202), [184](#_bookmark43), [264](#_bookmark203)

execution environment, [81](#_bookmark68)

ExecutionGraphs, [41](#_bookmark204)

external dependencies, [107](#_bookmark398), [187](#_bookmark69), [236](#_bookmark143) external systems

application consistency guarantees, [184](#_bookmark1)-[186](#_bookmark180) asynchronously accessing, [216](#_bookmark70)-[219](#_bookmark71)

custom sink functions, [206](#_bookmark58)-[216](#_bookmark10) custom source functions, [202](#_bookmark56)-[206](#_bookmark13) provided connectors, [186](#_bookmark365)-[202](#_bookmark24)

extract–transform–load (ETL), [3](#_bookmark386)

## F

failed processes

automatic recovery from, [232](#_bookmark366) enabling failure recovery, [166](#_bookmark321) restarting, [42](#_bookmark0), [268](#_bookmark329)

restoring application state, [59](#_bookmark0) failure-rate restart strategy, [269](#_bookmark14) fast forwarding, [30](#_bookmark299)

fields, referencing, [102](#_bookmark358) filesystems

filesystem configuration, [237](#_bookmark72)-[239](#_bookmark15)

sink connector, [196](#_bookmark73)-[198](#_bookmark468)

source connector, [194](#_bookmark0)-[196](#_bookmark16)

filter transformation, [85](#_bookmark330)

fixed-delay restart strategy, [269](#_bookmark17) flatMap transformation, [86](#_bookmark144)

FlinkCEP, [282](#_bookmark331)

flow control, credit-based, [45](#_bookmark232) forward data exchange strategy, [19](#_bookmark74) framework deployment style, [39](#_bookmark433) FsStateBackend, [169](#_bookmark332), [267](#_bookmark75)

full window functions, [127](#_bookmark285)

functions (see also under DataStream API) applying functions on windows, [127](#_bookmark186)-[133](#_bookmark262) function classes, [105](#_bookmark53)

implementing stateful functions, [154](#_bookmark76)-[166](#_bookmark263) lambda functions, [106](#_bookmark77)

process functions, [116](#_bookmark78)

rich functions, [106](#_bookmark237), [116](#_bookmark238) fundamentals

dataflow programming, [17](#_bookmark277)-[19](#_bookmark205) processing streams in parallel, [20](#_bookmark1)-[27](#_bookmark309) state and consistency models, [32](#_bookmark79)-[35](#_bookmark469) time semantics, [27](#_bookmark367)-[32](#_bookmark16)

## G

Gelly, [282](#_bookmark450)

GenericWriteAheadSink, [209](#_bookmark368) global partitioning strategy, [96](#_bookmark10)

global windows, [26](#_bookmark369)

GlobalWindows assigner, [139](#_bookmark187)

graph processing, [281](#_bookmark251)

## H

Hadoop distributed filesystem (HDFS), [4](#_bookmark264), [236](#_bookmark29),

[238](#_bookmark3)

HBase, [236](#_bookmark29)

Hello, Flink! example

applying transformations, [82](#_bookmark80)

data type, [79](#_bookmark276)

emitting results, [82](#_bookmark451)

executing applications, [83](#_bookmark175) execution environment set up, [81](#_bookmark68) reading input streams, [81](#_bookmark419)

typical application structure, [80](#_bookmark470) highly available (HA) setups

configuring HA Flink clusters, [232](#_bookmark366) HA Kubernetes setup, [235](#_bookmark333)

HA standalone setup, [233](#_bookmark434) HA YARN setup, [234](#_bookmark334) restarting failed processes, [42](#_bookmark0)

## I

idempotent operations, [184](#_bookmark413) idempotent sink connectors, [207](#_bookmark310) importing a project into an IDE, [72](#_bookmark229) incremental aggregation functions, [127](#_bookmark220)

incremental checkpointing, [66](#_bookmark136), [118](#_bookmark5), [170](#_bookmark137), [243](#_bookmark39),

[267](#_bookmark184)

IngestionTime characteristic, [110](#_bookmark145)

installation, [71](#_bookmark288), [221](#_bookmark277)

interval join, [145](#_bookmark335)

## J

JAR file, [14](#_bookmark5), [38](#_bookmark146), [39](#_bookmark433), [77](#_bookmark222), [81](#_bookmark206), [83](#_bookmark370), [108](#_bookmark164), [254](#_bookmark387), [258](#_bookmark452)

Java Virtual Machine (JVM), [12](#_bookmark406), [38](#_bookmark81), [75](#_bookmark482), [221](#_bookmark430),

[239](#_bookmark82)

job parallelism, [40](#_bookmark265), [96](#_bookmark258), [247](#_bookmark471), [251](#_bookmark324), [257](#_bookmark420)

JobGraph, [41](#_bookmark147)

JobManager, [38](#_bookmark83), [42](#_bookmark239), [221](#_bookmark311)-[235](#_bookmark289) joining streams on time, [145](#_bookmark148)-[148](#_bookmark84)

## K

Kerberos authentication, [243](#_bookmark302)

key groups, [56](#_bookmark336), [168](#_bookmark240), [252](#_bookmark241)

key-based data exchange strategy, [19](#_bookmark149) keyBy transformation, [87](#_bookmark266)

keyed state, [54](#_bookmark435), [154](#_bookmark85)

KeyedStream transformations, [87](#_bookmark0)

keys, defining, [102](#_bookmark358)

Kryo serialization framework, [98](#_bookmark242) Kubernetes, [228](#_bookmark260)-[232](#_bookmark295), [259](#_bookmark231)

## L

lambda architecture, [10](#_bookmark282)

lambda functions, [84](#_bookmark150), [88](#_bookmark86), [106](#_bookmark77) late elements

definition of term, [49](#_bookmark87), [148](#_bookmark88)

dropping, [148](#_bookmark337)

redirecting, [148](#_bookmark453)

updating results by including, [150](#_bookmark89) latency

average, maximum, and percentile, [20](#_bookmark407) definition of term, [20](#_bookmark207)

effect of watermarks on, [115](#_bookmark90) importance of low latency, [21](#_bookmark18) monitoring, [278](#_bookmark0)

processing-time windows, [31](#_bookmark371)

versus throughput, [22](#_bookmark91)

leaking state, [171](#_bookmark338)

library deployment style, [40](#_bookmark92) ListCheckpointed interface, [158](#_bookmark93)

ListState, [54](#_bookmark267), [55](#_bookmark290), [154](#_bookmark388), [158](#_bookmark93)

load rebalancing, [95](#_bookmark94)

local filesystems, [237](#_bookmark436)

local recovery, [269](#_bookmark291)

logging, [279](#_bookmark30)

## M

maintainability, ensuring

defining maximum parallelism of keyed state operators, [168](#_bookmark240)

specifying unique operator identifiers, [168](#_bookmark0), [251](#_bookmark19)

stateful applications, [167](#_bookmark208)-[169](#_bookmark95)

map transformation, [84](#_bookmark481)

MapState, [55](#_bookmark312), [154](#_bookmark483) Maven projects

bootstrapping Flink Maven projects, [76](#_bookmark306) importing, [72](#_bookmark229)

memory

configuring, [241](#_bookmark5)

JVM heap memory, [242](#_bookmark96) major consumers of, [241](#_bookmark97) requirements, [240](#_bookmark313)

MemoryStateBackend, [169](#_bookmark332), [266](#_bookmark421)

MergingWindowAssigner interface, [139](#_bookmark339), [142](#_bookmark151)

metrics system, [273](#_bookmark73)-[277](#_bookmark454)

microbatching, [44](#_bookmark98)

microservices design pattern, [2](#_bookmark429), [6](#_bookmark382)

Minikube, [260](#_bookmark45) monitoring

applications using REST API, [254](#_bookmark372) clusters using REST API, [253](#_bookmark228) Flink Web UI, [270](#_bookmark349)

latency, [278](#_bookmark0)

metrics system, [273](#_bookmark73)-[277](#_bookmark454) multistream transformations

combining events of two streams, [90](#_bookmark373)-[93](#_bookmark99) filtering and replicating events, [93](#_bookmark100) merging DataStreams, [90](#_bookmark63)

## N

no-restart strategy, [269](#_bookmark101)

## O

OnTimerContext object, [117](#_bookmark7) open source stream processing, [9](#_bookmark437) OpenStack Swift FS, [238](#_bookmark152) operation

configuring logging behavior, [279](#_bookmark30) controlling task scheduling, [260](#_bookmark380)-[263](#_bookmark318) monitoring clusters and applications,

[270](#_bookmark193)-[279](#_bookmark33)

running and managing applications, [245](#_bookmark411)-[260](#_bookmark248)

tuning checkpoints and recovery, [263](#_bookmark400)-[270](#_bookmark168) operator state, [54](#_bookmark0), [154](#_bookmark102), [158](#_bookmark93), [164](#_bookmark103)

operators

built-in, [22](#_bookmark389)

defining windows operators, [122](#_bookmark300) definition of term, [17](#_bookmark403)

input and output streams, [23](#_bookmark390) time-based, [51](#_bookmark340)

out-of-order records, [29](#_bookmark307)-[30](#_bookmark374), [48](#_bookmark472), [204](#_bookmark464)

## P

parallel windows, [26](#_bookmark408) parallelism

defining maximum of keyed state operators, [168](#_bookmark240)

scaling applications in and out, [251](#_bookmark324), [257](#_bookmark341) setting, [96](#_bookmark258)

partition watermarks, [50](#_bookmark268)

partitioning (see also distribution transforma‐ tions)

global partitioning strategy, [96](#_bookmark10) rescale partitioning strategy, [95](#_bookmark62)

pattern detection, [6](#_bookmark325), [282](#_bookmark331) performance

aspects affecting, [169](#_bookmark104) choosing state backends, [169](#_bookmark209)

choosing state primitives, [55](#_bookmark210), [171](#_bookmark0) configuring state backends, [266](#_bookmark342) effect of task scheduling on, [260](#_bookmark380) preventing leaking state, [171](#_bookmark338)

periodic watermarks, [112](#_bookmark278)

POJO data type, [100](#_bookmark8), [103](#_bookmark296) primitive data types, [98](#_bookmark415) process functions

CoProcessFunction, [120](#_bookmark253) emitting to side outputs, [119](#_bookmark211) TimerService and timers, [117](#_bookmark7) uses for, [116](#_bookmark78)

processing slots, [40](#_bookmark269)-[43](#_bookmark460), [240](#_bookmark46), [262](#_bookmark105)

processing time, [29](#_bookmark73), [31](#_bookmark308), [109](#_bookmark438)

ProcessingTime characteristic, [109](#_bookmark438)

ProcessWindowFunction interface, [130](#_bookmark20)

projects, importing, [72](#_bookmark229)

punctuated watermarks, [114](#_bookmark34)

## Q

queryable state

architecture and enablement of, [177](#_bookmark455) exposing, [179](#_bookmark106)

querying from external applications, [180](#_bookmark153) uses for, [177](#_bookmark107)

## R

random data exchange strategy, [19](#_bookmark212), [94](#_bookmark449)

real-time applications, [21](#_bookmark108)

real-time recommendations, [6](#_bookmark292)

recovery (see checkpointing and recovery) relational queries, [281](#_bookmark251)

rescale partitioning strategy, [95](#_bookmark109) ResourceManager, [38](#_bookmark270), [221](#_bookmark343)-[232](#_bookmark295) REST API

accessing, [252](#_bookmark314)

managing and monitoring applications, [254](#_bookmark372) managing and monitoring clusters, [253](#_bookmark228)

result guarantees, [34](#_bookmark213), [184](#_bookmark1)-[186](#_bookmark180)

ResultTypeQueryable interface, [102](#_bookmark110)

rich functions, [106](#_bookmark237), [116](#_bookmark238)

RocksDBStateBackend, [169](#_bookmark332), [267](#_bookmark181)

rolling aggregations, [23](#_bookmark458), [88](#_bookmark22)

round-robin distribution transformation, [95](#_bookmark10)

## S

SAM (single abstract method) interfaces, [84](#_bookmark111) savepoints

deleting with CLI, [249](#_bookmark322) parameters required, [167](#_bookmark250)

starting applications from, [67](#_bookmark439), [250](#_bookmark297)

structure of, [246](#_bookmark301)

taking and disposing with CLI, [249](#_bookmark6) taking and disposing with REST API, [256](#_bookmark315) using, [67](#_bookmark91)

versus checkpoints, [66](#_bookmark303), [246](#_bookmark214) Scala DataStream API, [84](#_bookmark150) security, [243](#_bookmark302)

session windows, [26](#_bookmark112), [126](#_bookmark271) set up

deployment modes, [221](#_bookmark277)-[232](#_bookmark295)

filesystem configuration, [237](#_bookmark72)-[239](#_bookmark15) highly available setups, [232](#_bookmark366)-[236](#_bookmark113)

integration with Hadoop components, [236](#_bookmark29) system configuration, [239](#_bookmark114)-[244](#_bookmark272)

side outputs, [119](#_bookmark211) sink connectors

Apache Cassandra, [199](#_bookmark0)-[202](#_bookmark24)

Apache Kafka, [190](#_bookmark247)

custom, [206](#_bookmark58)-[216](#_bookmark10)

filesystems, [196](#_bookmark73)-[198](#_bookmark468)

idempotent sink connectors, [207](#_bookmark310) transactional sink connectors, [209](#_bookmark47)

sliding windows, [25](#_bookmark422), [125](#_bookmark223)

slot-sharing groups, [262](#_bookmark105)

software requirements, [71](#_bookmark288) source code, downloading, [73](#_bookmark5) source connectors

Apache Kafka, [187](#_bookmark225)-[190](#_bookmark226)

custom, [202](#_bookmark56)-[206](#_bookmark13)

filesystem, [194](#_bookmark0)-[196](#_bookmark16) provided by Flink, [186](#_bookmark365)

timestamps and watermarks, [52](#_bookmark115), [204](#_bookmark464) special-purpose data types, [100](#_bookmark383)

split transformation, [93](#_bookmark100)

SplitStream, [93](#_bookmark391)

SQL, [282](#_bookmark173)

SSL authentication, [244](#_bookmark215)

standalone clusters, [221](#_bookmark418) state and consistency models

consistency guarantees, [184](#_bookmark1)-[186](#_bookmark180)

result guarantees, [34](#_bookmark213)

role of state management, [27](#_bookmark116), [32](#_bookmark79) supporting stateful operators, [32](#_bookmark440) task failures, [33](#_bookmark188)

state backends, [55](#_bookmark441), [169](#_bookmark209), [243](#_bookmark40), [266](#_bookmark342)

state checkpointing, [56](#_bookmark117) state management

evolving stateful applications, [174](#_bookmark167)-[177](#_bookmark32) keyed state, [54](#_bookmark435), [154](#_bookmark85)

operator state, [54](#_bookmark0)

overview of, [53](#_bookmark1)

purpose of, [33](#_bookmark3)

scaling stateful operators, [56](#_bookmark243) state backends, [55](#_bookmark441), [169](#_bookmark209), [243](#_bookmark40), [266](#_bookmark342)

state partitioning, [33](#_bookmark92) (see also keyed state) state primitives, [55](#_bookmark210), [171](#_bookmark0)

state recovery, [33](#_bookmark118), [58](#_bookmark350) stateful functions

declaring keyed state at RuntimeContext, [154](#_bookmark85)

implementing operator list state with List‐ Checkpointed interface, [158](#_bookmark93)

receiving checkpoint completion notifica‐ tions, [166](#_bookmark0)

using CheckpointedFunction interface, [164](#_bookmark38) using connected broadcast state, [160](#_bookmark478)

stateful operators and applications enabling failure recovery, [166](#_bookmark321)-[167](#_bookmark154) ensuring maintainability, [167](#_bookmark208)-[169](#_bookmark95)

evolving applications, [174](#_bookmark167)-[177](#_bookmark32) implementing stateful functions, [154](#_bookmark76)-[166](#_bookmark263) performance and robustness of, [169](#_bookmark104)-[174](#_bookmark119) queryable state, [177](#_bookmark107)-[182](#_bookmark170)

scaling, [56](#_bookmark243), [251](#_bookmark324), [257](#_bookmark341) stateful stream processing

basics of, [4](#_bookmark392)

data pipelines, [8](#_bookmark0)

event-driven applications, [6](#_bookmark194)

Flink introduction, [12](#_bookmark1)-[15](#_bookmark160)

open source stream processing, [9](#_bookmark437) streaming analytics, [8](#_bookmark283)

traditional data infrastructures, [2](#_bookmark47)-[4](#_bookmark316) stateless operations, [22](#_bookmark442)

stream processing benefits of, [1](#_bookmark273)

DataStream API, [79](#_bookmark133)-[108](#_bookmark443)

external systems, [183](#_bookmark166)-[219](#_bookmark293)

Flink architecture, [37](#_bookmark133)-[69](#_bookmark134)

Flink development environment, [71](#_bookmark166)-[77](#_bookmark444) Flink operation, [245](#_bookmark166)-[280](#_bookmark26)

Flink set up, [221](#_bookmark133)-[244](#_bookmark393) fundamentals, [17](#_bookmark133)-[36](#_bookmark155)

future directions, [281](#_bookmark133)-[283](#_bookmark399)

stateful operators and applications, [153](#_bookmark133)-[182](#_bookmark344) stateful stream processing, [1](#_bookmark133)-[15](#_bookmark345)

time handling, [109](#_bookmark133)-[152](#_bookmark120)

StreamExecutionEnvironment, [109](#_bookmark394), [166](#_bookmark473), [269](#_bookmark121)

streaming analytics, [8](#_bookmark283)

Swift FS, [238](#_bookmark152) system architecture

application deployment, [39](#_bookmark423) components of Flink setups, [38](#_bookmark47) highly available setup, [42](#_bookmark0), [232](#_bookmark366)-[236](#_bookmark113)

task execution, [40](#_bookmark244) system configuration

checkpointing and state backends, [243](#_bookmark40) configuration files, [239](#_bookmark114)

CPU, [240](#_bookmark46)

disk storage, [242](#_bookmark185)

Java and classloading, [239](#_bookmark274)

main memory and network buffers, [240](#_bookmark313) security, [243](#_bookmark302)

## T

Table API, [282](#_bookmark122)

task chaining, [46](#_bookmark0), [261](#_bookmark47)

task parallelism, [18](#_bookmark404), [40](#_bookmark252) task scheduling

defining slot-sharing groups, [262](#_bookmark105) effect on performance, [260](#_bookmark380)

TaskManagers, [38](#_bookmark409), [40](#_bookmark244), [42](#_bookmark123), [221](#_bookmark343)-[232](#_bookmark295)

Threads, [41](#_bookmark346)

throughput, [21](#_bookmark171) time handling

configuring time characteristics, [109](#_bookmark343)-[115](#_bookmark424) handling late data, [148](#_bookmark124)-[151](#_bookmark456)

joining streams on time, [145](#_bookmark148)-[148](#_bookmark84) process functions, [116](#_bookmark1)-[121](#_bookmark125)

window operators, [122](#_bookmark126)-[145](#_bookmark127) time semantics

event time, [29](#_bookmark307), [47](#_bookmark192)-[52](#_bookmark459)

processing time, [29](#_bookmark73)

processing time versus event time, [31](#_bookmark308) role in stream processing, [27](#_bookmark116)

time-critical applications and, [27](#_bookmark474) watermarks, [30](#_bookmark425), [48](#_bookmark162)

time-based operators, [51](#_bookmark340) (see also time han‐ dling)

time-based tumbling windows, [25](#_bookmark156), [124](#_bookmark91)

TimeCharacteristic.EventTime, [111](#_bookmark9)

TimeCharacteristic.ProcessingTime, [111](#_bookmark9) timers

checkpointing of, [117](#_bookmark445)

on nonkeyed streams, [117](#_bookmark275) registering in the past, [118](#_bookmark128) synchronization of, [117](#_bookmark216)

timestamps and, [117](#_bookmark375)

TimerService, [117](#_bookmark7)

TimestampAssigner interface, [111](#_bookmark259) timestamps

assigning, [111](#_bookmark40) event time and, [29](#_bookmark307)

event-time processing and, [47](#_bookmark465) timers and, [117](#_bookmark375)

watermark generation and, [52](#_bookmark0) transformation operations

basic transformations, [84](#_bookmark410)

basics of, [23](#_bookmark189)

distribution transformations, [94](#_bookmark178)

KeyedStream transformations, [87](#_bookmark0)

multistream transformations, [90](#_bookmark0)-[94](#_bookmark157)

overview of, [83](#_bookmark432)

purpose of, [82](#_bookmark80) triggers

custom, [140](#_bookmark426)

early firing, [142](#_bookmark395)

implementing, [140](#_bookmark129)

purpose of, [140](#_bookmark6)

Trigger API, [140](#_bookmark457)

TriggerResult values, [140](#_bookmark172)

tumbling windows, [25](#_bookmark130), [124](#_bookmark91)

TumblingEventTimeWindows, [124](#_bookmark217)

TumblingProcessingTimeWindows, [124](#_bookmark217)

Tuple data types, [98](#_bookmark479), [103](#_bookmark0)

two-phase-commit (2PC), [185](#_bookmark376)

TwoPhaseCommitSinkFunction, [212](#_bookmark131)

TypeInformation, [100](#_bookmark463)-[102](#_bookmark234)

## U

unbounded datasets, [32](#_bookmark190)

union list state, [54](#_bookmark294), [164](#_bookmark158)

unique operator identifiers, [68](#_bookmark475), [168](#_bookmark0), [251](#_bookmark19) updates

modifying state of operators, [176](#_bookmark47) removing state from applications, [175](#_bookmark377) savepoints and, [174](#_bookmark167)

without modifying existing state, [175](#_bookmark165) user-defined functions, [23](#_bookmark245)

user-defined functions (UDFs), [82](#_bookmark378), [84](#_bookmark18), [105](#_bookmark1)

## V

ValueState, [55](#_bookmark246), [154](#_bookmark347)

## W

watermarks accessing, [50](#_bookmark4)

effect on streaming applications, [115](#_bookmark21) generation and timestamp assignment, [52](#_bookmark0),

[111](#_bookmark40)

periodic watermarks, [112](#_bookmark278)

propagation of, [49](#_bookmark185)

properties of, [48](#_bookmark396)

punctuated watermarks, [114](#_bookmark34)

purpose of, [30](#_bookmark425), [115](#_bookmark90)

window assigner, [122](#_bookmark397)

window function, [122](#_bookmark446)

window join, [146](#_bookmark476) window operations

applying functions on windows, [127](#_bookmark186)-[133](#_bookmark262) basics of, [24](#_bookmark191)

built-in window assigners, [123](#_bookmark182)-[127](#_bookmark159) customizing window operators, [134](#_bookmark0)-[145](#_bookmark127) defining window operators, [122](#_bookmark300) GlobalWindows assigner, [139](#_bookmark187)

time characteristics and, [109](#_bookmark343) uses for, [122](#_bookmark126)

window lifecycle, [137](#_bookmark91)

WindowAssigner interface, [137](#_bookmark132)

WindowFunction interface, [130](#_bookmark20)

write-ahead-log (WAL), [185](#_bookmark376)

## Y

YARN, [225](#_bookmark183), [236](#_bookmark29)

# About the Authors

### **Fabian Hueske** is a committer to and PMC member of the Apache Flink project and has been contributing to Flink since its earliest days. Fabian is cofounder and soft‐ ware engineer at Ververica (formerly data Artisans), a Berlin-based startup that fos‐ ters Flink and its community. He holds a PhD in computer science from TU Berlin.

**Vasiliki Kalavri** is a postdoctoral fellow in the Systems Group at ETH Zurich, where she uses Apache Flink extensively for streaming systems research and teaching. Vasia is a PMC member of the Apache Flink project. An early contributor to Flink, she has worked on its graph processing library, Gelly, and on early versions of the Table API and streaming SQL.

**Colophon**

The animal on the cover of *Stream Processing with Apache Flink* is a Eurasian red squirrel (*Sciurus vulgaris*). Most arboreal squirrels in temperate Asia, Europe, and the Americas are of the genus *sciurus*. *Vulgaris* means “common” in Latin, and the Eura‐ sian red squirrels are a regular sight throughout Europe and northern Asia.

### Eurasian red squirrels have a white ring around their eyes, a bushy tail, and a tuft of fur over their ears. Their coloring ranges from light red to black on their heads and backs. The fur on their bellies is cream-colored or white. In the winter, the squirrel’s fur grows a little taller and longer above their ears and around their paws to protect them from the cold. They spend much of the winter curled up in nests, called *dreys*.

Unless they are mating or raising young, a Eurasian red squirrel’s dreys are meant to house one animal. Although they live alone, their ranges often overlap because they are so populous. On average, females give birth to five young per litter twice per year. Young squirrels leave the mother’s nest after about two months. Eurasian red squir‐ rels face many predators, including birds, snakes, and mammals, so only one quarter of the young reach one year of age.

Eurasian red squirrels sustain themselves on seeds, acorns, and nuts. They have also been caught licking tree sap, but they don’t often experiment with new foods. These squirrels can grow about 9 to 10 inches long from head to hind, with a tail of about the same length. They weigh 8 to 12 ounces and can live up to 12 years. In the wild their lifespan is expected to be only 4–7 years.

What allows these tree-dwellers to scale trunks, descend headfirst, and leap across overhanging branches is a combination of their sharp, curved claws and long, fluffy tails. Eurasian red squirrels have excellent agility and balance.

Many of the animals on O’Reilly’s covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover illustration is by Karen Montgomery, based on a black and white engraving from Wood’s *Animate Creation*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Con‐ densed; and the code font is Dalton Maag’s Ubuntu Mono.