Notes

Java 4

Python 4

Reading the required version number of a module 4

Scala 4

General 4

Conversions from Java to Scala Types 4

Quick one liners 4

Reading a file 4

Printing sys properties 4

Dynamic Variables 5

Closures 5

Futures 5

Monads 6

Access 9

Testing 9

Pattern Matching 9

Links 9

Variable Binding 10

Sealed 10

Match Sequences 11

Guards (if statements in cases) 11

Patterns to pick out arguments 11

Options 12

Implicit Class 13

Implicit Functions 13

Implicit parameters 14

Functions 15

Regular Expressions 16

Try-Catch 16

Miscellaneous 16

ScalaTest 17

Example snippets 17

Docker 17

Node 17

Bash-Fu 17

Rerun last command: 17

Last PID variable 17

Pausing a script 17

Sample piping to slack 17

Number of columns in a csv file if the first line is the header 18

Number of rows in a csv file 18

Arithmetic in Bash 18

Which lines are in one file but not the other 18

Left pad with 0 18

Emacs 19

Highlighting 19

Macros 19

Undo 19

copy region to clipboard 19

Vim 19

Postgres 19

Examples 19

Creating btree index 19

Git 20

Creating a new project 20

Cloning 20

Remotes 20

Forks 22

Getting a branch from upstream 23

File Management 23

Logs and Diffs 23

Avoid merge commits 24

Tagging 26

List Tags 26

Create Tag 26

Delete Tag Local 26

Delete Tag Remote 26

Show info using hash 26

Get commit comments between two hashes 27

Spark 27

Spark SQL 27

Change value of one column 27

Quick Examples 27

Writing a spark job 28

Spark Streaming 28

Things to watch for 29

Checkpointing 29

DataFrames 30

Tuning 30

Serialization 30

Memory 30

Questions 31

Kafka 32

What is a log? 32

Confluent Schema Registry 32

Questions 34

Differences from typical messaging system 34

Verbatim from Alexis Richardson and Tim Watson of RabbitMQ: 34

Usage with Avro Schema 35

Use with Spark Streaming 36

What is the Kafka Stream? 37

KafkaRDD 37

Streaming Offsets 38

Questions 38

Configuration Properties 38

Message Delivery 38

“at most once” settings 38

“at least once” settings 39

Data Formats 39

Avro 39

AWS 39

Maven 39

Starting a new Scala Project 39

Starting a new Java Project 39

SBT 40

PIP 42

Unix 42

Ansible 42

SSH 42

Hadoop Data Lake 42

Mac/Unix 44

Searching for a service that is running 44

ssh with a pem key 44

Find Java home by running 44

Zeppelin 44

Spark Machine Learning 45

Data Preperation 45

String columns 45

Interviewing 46

Phone Screening 46

Java Assessment 46

Spark Questions 48

Hive 50

Java Questions 51

Spring Questions 51

Scala Questions 52

Testing 52

Core Principles 52

# Java

On a mac, find your java home with

/usr/libexec/java\_home

# Python

## Reading the required version number of a module

[bitsplit@ip-203-80-238 ~]$ /usr/local/secondlook\_core\_env/bin/python

Python 2.7.5 (default, Aug 9 2016, 05:27:46)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> import pkg\_resources

>>> pkg\_resources.require("secondlook\_core")[0].version

'0.2.10'

# Scala

## General

“def” = evaluate at time of call

“val” = evaluate at time of definition

“lazy val” = evaluate when evaluated the first time

<http://alvinalexander.com/scala/scala-class-examples-constructors-case-classes-parameters>

## Conversions from Java to Scala Types

import scala.collection.JavaConversions.\_

## Quick one liners

### Reading a file

val json = Source.fromFile(("file.json").getLines.mkString

## Printing sys properties

import scala.collection.JavaConversions.\_

for ( (k,v) <- System.getProperties()) println(s"$k: $v")

for ( (k,v) <- System.getenv()) println(s"$k: $v")

## Dynamic Variables

<http://scalageek.blogspot.com/2013/02/when-to-use-dynamic-variables.html>

Will Rose had reviewed my code and we noticed that one of my value objects in Spark is not thread safe. So he suggested using a dynamic variable.

In [src/main/scala/com/capitalone/labs/Driver.scala](https://github.kdc.capitalone.com/SecondLook/avro-auth-stream-scala/pull/8#discussion_r87505):

> @@ -32,7 +35,7 @@ object Driver extends LazyLogging {

> // convert to Avro

> val auths = Transformation.transform(jsonMessages)

> // serialize RealtimeAuth objects into byte arrays

> - val serializedAuths = auths.map(a => AvroSerializer.serialize(a))

> + val serializedAuths = auths.map(a => serialize(a))

You could also just create it once per thread in the Driver object like so:

val serialize = new DynamicVariable[Injection[RealTimeAuth, Array[Byte]](SpecificAvroCodecs.toBinary[RealtimeAuth])

and then

val serializedAuths = auths.map(serialize.value)

I looked up what a dynamic variable is and from the link above it seems to equate a dynamic variable with thread local. But the scala docs for dynamic variable does not give such a straight forward answer: <http://www.scala-lang.org/api/2.12.0-M4/scala/util/DynamicVariable.html>.

## Closures

<http://alvinalexander.com/scala/how-to-use-closures-in-scala-fp-examples>

“To create a closure in Scala, just define a function that refers to a variable that’s in the same scope as its declaration.” Scala will create a “closed” set of variables that the function can access even when the function is passed to other parts of the code where the set of variables are no longer in scope.

## Futures

<http://docs.scala-lang.org/overviews/core/futures.html>

“A placeholder object for a value that may not exist yet”

The block you pass to a Future will run in some thread. You can specify where the thread comes from (new thread, from a pool, the current thread) with the use of an Execution Context object.

Scala has an implementation for the Execution Context that will use a global static thread pool. You an do whatever you want by extending the Execution Context trait or by converting your own Executor to an Execution Context.

“Once a Future is given a value or exception” it becomes immutable.

val f: Future[SomeType] = Future { some task to run asynchronously }

When the task is done, the value “f” will have a Future object holding some return value or an exception if the task threw an exception. You also need to import

import ExecutionContext.Implicits.global

This will provide within your scope an implicit ExecutionContext that has the global thread pool. The Future method actually takes two parameters: your block of code and an ExecutionContext which can be implicitly passed to it.

Since your code will have no idea when the asynchronous task will complete, you have to supply it with a callback that is executed when the task completes. You do this using the Future object (e.g. “f” from the above example).

f onComplete {

case Success(SomeType) => do something

case Failure(Some exception) => do something else

}

Or if you don’t care to handle failures or want to handle them separately, do this

f onSuccess {

case SomeType => do something

}

f onFailure {

case exception => do something else

}

## Monads

<https://medium.com/@sinisalouc/demystifying-the-monad-in-scala-cc716bb6f534#.wlg4hilse>

“Monad is not a class or a trait; monad is a concept. Every “wrapper” that provides us with our two beloved operations, unit and flatMap, is essentially a monad.”

Monads allow a more functional programming style that you would not be able to do without them.

Assume some user service can load a user record from the database and return an Option. Option is conceptually a monad

generic monad:  
--------------  
unit: A => M[A]   
flatMap: (A => M[B]) => M[B]

our monad:  
--------------   
unit: User => Option[User]  
flatMap: (User => Option[User]) => Option[User]

Instead of writing a bunch of nested if-then-else statements to get the user’s grandchild

val result = userService.loadUser(“mike”)

if (result != None) // pseudo code here

then child = result.child

if (child != None)

then child.child

you can write this

UserService.loadUser("mike").flatMap(\_ => \_.child).flatMap(\_ => \_.child)

or with a for-comprehension (that the compile will convert to the flatmap operations above)

for { user <- UserService.loadUser(“mike”)

child <- user.child

grandchild <- child.child } yield grandchild

So, monads allow for “chaining” of operations.

A second example from the same link describes an order of operations:

1. load an order for a particular user
2. load each item in the order
3. purchase each item
4. log each purchase

This example is set up so that each successive step takes as input the output of the previous step. If these were returning single objects in java and each object returned had a method that performed the next step we would simply do

service.step1().step2().step3().step4()

But we are not dealing with single objects here. In the Options example, we can either have Some or None. In List examples, we are plainly dealing with more than one object. And in this example, we are dealing with Future objects which can be either Success or Failure. We also do not have a method implemented on each object that will do the operational steps for us. We have to provide those methods outside of our domain objects (order, item, purchase). The Future monad allows us to chain them together as long as we can define the functions that can convert

A => Monad[B]

Such as

Order => Future[Item]

Item => Future[Purchase]

Purchase => Future[Log]

Once we have these functions, for example

val loadItem: Order => Future[Item] = {

order => ItemService.loadItem(order)

}

val purchaseItem: Item => Future[Purchase] = {

item => PurchasingService.purchaseItem(item)

}

val logPurchase: Purchase => Future[Log] = {

purchaseResult => PurchasingService.logPurchase(purchaseResult)

}

We are allowed to chain them with the flatmap method

OrderService.loadOrder("customerUsername")  
 .flatMap(loadItem)  
 .flatMap(purchaseItem)  
 .flatMap(logPurchase)

We defined the functions and we pass them into each flapmap method.

## Access

From <http://www.ibm.com/developerworks/library/j-scala07298/>

“you can use import anywhere inside the client Scala file, not just at the top of the file and correspondingly, will have scoped relevance”

“use import to bring not just nested types into lexical scope, but any member”

“import can take multiple, comma-separated targets”

“Scala:

* Uses "public" by default
* Specifies "private" to mean "accessible only to this scope"

By contrast, "protected" is definitely different from its counterpart in Java code; where a Java protected member is accessible to both subclasses and the package in which the member is defined, Scala chooses to grant access only to subclasses. This means that Scala's version of protected is more restrictive (although arguably more intuitively so) than the Java version.”

“access modifiers in Scala can be "qualified" with a package name, indicating a level of access *up to* which the member may be accessed”

“the *object-private* specification, illustrated by private[this], which stipulates that the member in question can only be seen by members called on that same object, not from different objects, even if they are of the same type”

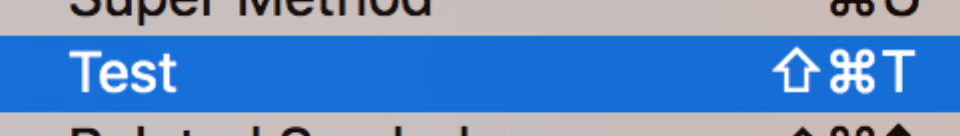
## Testing

Add the following dependency:

“org.scalatest” % “scalatest\_2.10” % “2.1.0” % “test”

to build.sbt and let Idea load the library.

Use Idea to create a test by using “Navigate -> Test”



Your test class should extend org.scalatest.FunSuite and you can do this automatically by selecting ScalaTest as the Testing Library.



## Pattern Matching

### Links

<http://www.artima.com/pins1ed/case-classes-and-pattern-matching.html>

### Variable Binding

“A variable pattern matches any object, just like a wildcard. Unlike a wildcard, Scala binds the variable to whatever the object is. You can then use this variable to act on the object further.”

case Some(x) => println(“I got x: ” + x)

“In addition to the standalone variable patterns, you can also add a variable to any other pattern. You simply write the variable name, an at sign (@), and then the pattern. This gives you a variable-binding pattern. The meaning of such a pattern is to perform the pattern match as normal, and if the pattern succeeds, set the variable to the matched object just as with a simple variable pattern.”

case x @ Some(\_) => println(“I got the entire Option object, not just what is inside: ” + x)

“A constructor pattern looks like

case BinOp("+", e, Number(0)) => println(“good”)

It consists of a name (e.g. BinOp) and then a number of patterns within parentheses (e.g. "+", e, and Number(0)). Assuming **the name designates a case class**, such a pattern means to first check that the object is a member of the named case class, and then to check that the constructor parameters of the object match the extra patterns supplied.”

### Sealed

When using the “match” statement, you must supply every possible match pattern because unlike Java’s switch statement, there is no “default” fall through statement. You would usually do this on your own by specifying a pattern that matches anything at the end.

case \_ => println(“matches everything and anything”)

“In fact, you can enlist the help of the Scala compiler in detecting missing combinations of patterns in a match expression. To be able to do this, the compiler needs to be able to tell which are the possible cases; make the superclass of your case classes **sealed**. A sealed class cannot have any new subclasses added except the ones in the same file.”

**sealed** abstract class Expr

  case class Var(name: String) extends Expr

  case class Number(num: Double) extends Expr

### Match Sequences

Sequence patterns can be thought of as a special case to constructor patterns for case classes.

“You can match against sequence types like List or Array just like you match against case classes. Use the same syntax, but now you can specify any number of elements within the pattern.”

case **List(0, \_, \_)** => println("found it")

This matches any list that starts with zero and has a length of three. To match an arbitrary length list you need to use “\_\*”

case List(0, \_\*) => println("found it")

### Guards (if statements in cases)

“Scala restricts patterns to be linear: a pattern variable may only appear once in a pattern.”

case BinOp("+", x, x) => BinOp("\*", x, Number(2))

This will fail to compile because you used “x” twice in the pattern. But you really want the two arguments to be the same or else you can’t just assume that on the right side of the case statement. That is, if you use

case BinOp("+", x, y)

That will match any two arguments and they don’t have to be the same. To resolve this, add an “if” (before the arrow) to guard against matches that succeed according to your pattern but is not exactly what you wanted.

case BinOp("+", x, y) **if x == y** => BinOp("\*", x, Number(2))

### Patterns to pick out arguments

If you already have a value defined like this

val myTuple = (123, "abc")

And you would like to draw out the data from the value, you can use a pattern like this to assign the data in the tuple to other values.

val (number, string) = myTuple

Now the values “number” and “string” have values from the tuple, 123 and “abc”, respectively.

You can do this with any constructor and not just with tuples

val exp = new BinOp("\*", Number(5), Number(1))

val BinOp(op, left, right) = exp

Now, the values op, left, and right have the data “\*”, Number(5), and Number(1), respectively. A nice way to remember this is to think that the pattern is “de-constructing” the constructor.

## Options

You can treat options **like a sequence**, even though they are not. If there is “something” in it, it behaves like a sequence of one value and if it is “None” then it behaves like an empty sequence.

Because of this, you can write less verbose code than your average match-case blocks. For example:

val result:Option[Type] = functionThatReturnsOption

result.foreach( t => println(“got it:” + t) )

The foreach won’t do anything to an empty list but will do something for the one object in the option.

The other thing you can do with Options is to map them. Options provide a map method that takes any Some[A] to Some[B] and None to None. So if I have an Account object that I just looked up from some DAO and that DAO returned an Option of Some[Account] or None and I wanted the balance (an integer) from the account, I can map the value:

someAcct.map(\_.getBalance)

and this will return either Some[Int] or None

However, if the Account class had a method “address” that also returned an option and I do:

someAcct.map(\_.getAddress)

I would get Some(Some(Address)) or None. I get a double Option[Option[]] because the getAddress method also returns an option. Do get rid of this, I can use the flatmap method in Option to flatten out the nested Option objects.

someAcct.flatMap(\_.getAddress)

You can call “filter” on them like a sequence. The filter method will return the Some[] to you if the predicate returns true; otherwise it returns None or if the option was None to begin with.

someAcct.filter(\_.getState == “MD”)

## Implicit Class

<http://alvinalexander.com/scala/scala-2.10-implicit-class-example>

“In Scala 2.10, you define an *implicit class*, and then define methods within that class to implement the behavior you want.”

## Implicit Functions

From <https://twitter.github.io/scala_school/advanced-types.html>

Implicit functions can automatically plopped into places where the input/output types fit.

implicit def strToInt(x: String) = x.toInt

math.max("123", 111)

The second line would usually fail since the max function expects an integer. But with the implicit function, scala calls it automatically for you.

Implicit functions are helpful with type conversions. You can specify in your class that your client must have an implicit conversion function when they instantiate your class or else a compile time error is generated. You use the **<%** operator, which I think is called “viewable” so type A must be viewable as an Int. I guess the percent sign looks like a pair of glasses.

class Container[A <% Int] { def addIt(x: A) = 123 + x }

The class Container takes a **generic** type that is bounded by the existence of an implicit function to convert a type A to Int. So with the implicit function we already defined above we can create a Container of Strings.

(new Container[String]).addIt("123")

But not a Container of Floats

(new Container[Float]).addIt(123.2F)  
<console>:8: error: could not find implicit value for evidence parameter of type (Float) => Int

## Implicit parameters

They are used as an equivalent to Haskell’s Type Classes. Similar to the above examples, our type class will be a trait. For example, if I created an API that consist of a class “MyApiClass” and a function that will take some parameter of an unknown type (It is some class of my clients, that I can’t know ahead of time) and the function must use that parameter so it can produce an instance of MyApiClass as a result. How would my client be able to use my API? In Java, I would provide an interface that my client will need to implement and the client will need to pass its implementation into my function. This couples my API with my client’s main class hierarchy. In scala, we have another option: specify an implicit parameter in the definition of my API function and the definition of a “type class” trait. Doing that allows the client code to create an object outside of its main class hierarchy that extends my trait to provide the implementation needed for my function to work. In Java, this just means that the client must pass another object into my function and that object will implement my interface instead of the client’s main object. This is also happening in scala however the object is passed in automatically (or implicitly) by the compiler just because the compiler sees the right type of object in scope. So it’s a round about way of doing things which in Java would add the cost of an extra parameter in the client’s method calls. But in scala, that cost is taken up by the compiler.

class MyApiClass

trait CanDoStuffMyApiNeeds[A] { def doStuff(param: A): MyApiClass }

def myApiMethod[T](param: T)(implicit i: CanDoStuffMyApiNeeds[T]) = i.doStuff(param)

class SomeClientUsingMyApi

val client = new SomeClientUsingMyApi

scala> myApiMethod(client)

<console>:13: error: could not find implicit value for parameter i: CanDoStuffMyApiNeeds[SomeClientUsingMyApi]

myApiMethod(client)

// now create an object that extends CanDoStuffMyApiNeeds in scope

implicit object glueClientToApi extends CanDoStuffMyApiNeeds[SomeClientUsingMyApi] { def doStuff(param: SomeClientUsingMyApi) = new MyApiClass }

// call it again and it’s okay

scala> myApiMethod(client)

res1: MyApiClass = MyApiClass@551aa95a

Additionally, if instead of taking a client typed parameter that you can’t know about, your API might provide all of the types that the function can use. Instead of polluting your own types with concerns that might not have anything to do with your classes. Here’s another example from <http://docs.scala-lang.org/tutorials/tour/implicit-parameters>:

abstract class SemiGroup[A] {

def add(x: A, y: A): A

}

abstract class Monoid[A] extends SemiGroup[A] {

def unit: A

}

object ImplicitTest extends App {

implicit object StringMonoid extends Monoid[String] {

def add(x: String, y: String): String = x concat y

def unit: String = ""

}

implicit object IntMonoid extends Monoid[Int] {

def add(x: Int, y: Int): Int = x + y

def unit: Int = 0

}

def sum[A](xs: List[A])(implicit m: Monoid[A]): A =

if (xs.isEmpty) m.unit

else m.add(xs.head, sum(xs.tail))

println(sum(List(1, 2, 3)))

println(sum(List("a", "b", "c")))

}

## Functions

<http://alvinalexander.com/scala/how-to-use-functions-as-variables-values-in-scala-fp>

“Assigning an existing function/method to a function variable”

val c = scala.math.cos \_

The value “c” is now a function that I can pass into another function.

## Regular Expressions

**val** uniqueIdPattern = "\"uniqueId\":\"([^\"]\*)\"".r  
 **val** uniqueId: Option[String] = **for** {  
 uniqueIdPattern(id) <- uniqueIdPattern.findFirstIn(json)  
 } **yield** id  
 *logger*.info(s"ID: **$**{uniqueId.getOrElse("None")}")

uniqueIdPattern.findFirstMatchIn("""{"hello":"world","uniqueId":"200"}""").get.group(1)

## Try-Catch

import scala.util.Try

import scala.util.Success

import scala.util.Failure

val t: Try[Long] = Try ( “hello”.toLong )

t match { case Success(l) => print

case Failure(e) => e.printStackTrace}

## Miscellaneous

You can treat scala strings as sequences.

The function “filter not” takes a function

val s:String = "hello"

s.filterNot(\_ == 'e')

This also works but is less readable if all you have is one char in the set.

s.filterNot("e".toSet)

This works because Set has an apply function that returns true or false if a char is in the set.

# ScalaTest

## Example snippets

class TransformationSpec extends FlatSpec with Matchers with Eventually with MockFactory {

“This” should “do that” in {

val api = mock[Api]

(api.isUnique \_).expects("hello").returning(true)

}

}

# Docker

docker ps

docker logs –f [CONTAINER ID]

# Node

# Bash-Fu

## Rerun last command:

!!

## Last PID variable

$!

## Pausing a script

read -rsp $'Press enter to continue...\n'

## Sample piping to slack

slack() {

if [[ $1 =~ ^Blocked.\* ]] ; then

emoji="':thumbsdown:'"

else

emoji="':thumbsup:'"

fi

quoted="'"$1"'"

curl -X POST --data-urlencode "payload={'channel': '#mychannel', 'username': 'CXF-212', 'text': $quoted, 'icon\_emoji': $emoji}" https://hooks.slack.com/services/XXX

}

tail -f /mnt/spark-work/app-20160901090106-0368/1/stderr | grep --line-buffered 'insight code=' | awk -F ' - ' '{printf $2"\n";fflush("/dev/stdout");}' | awk -F ', accountId' '{printf $1"\n";fflush("/dev/stdout");}' | while read -r line; do slack "$line"; sleep 2; done

## Number of columns in a csv file if the first line is the header

head -1 MERGED2012\_PP.csv | tr ',' '\n' | wc –l

## Number of rows in a csv file

wc –l FILE

## Arithmetic in Bash

echo $[3706/3]

## Which lines are in one file but not the other

diff --new-line-format="" --unchanged-line-format="" <(sort file1) <(sort file2)

sort in-place with process substitution <( )

In the above *new* and *unchanged* lines are suppressed, so only *changed* (i.e. removed lines in your case) are output.

## Left pad with 0

cat file | awk '{ if (length($1) < 11) { print "0"$1 } else { print $1 } }'

# Emacs

## Highlighting

<http://stackoverflow.com/questions/18090378/turn-on-background-color-when-highlighting-with-c-spc-on-a-mac>

M-x transient-mark-mode to toggle the highlighting

## Macros

c-x, ( to start

c-x, ) to end

f4 to execute

## Undo

c-\_

## copy region to clipboard

M-| and then pbcopy ENTER

# Vim

:set number

:set number!

:tabe <filepath>

:tabn

:tabp

# Postgres

## Examples

with stuff as (select count(\*) as c, unique\_id from "authorization" group by 2) select from stuff where c > 1;

## Creating btree index

create index ix\_auth\_unique\_id on "authorization" (unique\_id);

analyze "authorization"

explain analyze select count(\*) from "authorization" where unique\_id = 'goodnight';

# Git

<https://git-scm.com/docs>

<http://ftp.newartisans.com/pub/git.from.bottom.up.pdf>

## Creating a new project

1. First, you can go into any directory and do a   
   **git init .**  
   to make it a git repo
2. Add files and commit to your local repo
3. Create a repo in your github
4. Add the gibhub remote to your local repo  
   **git remote add origin** [**https://github.kdc.capitalone.com/rhp086/kafkastreams.git**](https://github.kdc.capitalone.com/rhp086/kafkastreams.git)
5. Push what you checked in to the remote  
   **git push -u origin master**

## Cloning

Make your own local repository by copying from a remote:

git clone -o MY\_NAME\_FOR\_REMOTE -b BRANCH <https://github.com/REPO.git> MY\_DIR\_TO\_WRITE\_TO

Or if you grabbed the remote without the branch information like this

git clone -o REMOTE\_NAME URL DIRECTORY

You can fetch the branch later and switch to it with this

git fetch REMOTE\_NAME BRANCH\_NAME

git checkout BRANCH\_NAME

## Remotes

Ask for a list of references on any remote repository

git ls-remote REMOTE\_NAME\_OR\_URL

If you're in a git repository that you cloned, you can leave off the REMOTE\_NAME\_OR\_URL and it'll default to the remote you cloned from.

References are "alias" to sha1 commit ID numbers.

If you're inside a git directory, you can list the remotes your local git is tracking (most likely from cloning, pushing, or pulling)

git remote

If there isn't one listed that you want, you can add it with

git remote add WHATEVER\_NAME URL

After adding a remote, you can fetch and merge it into your repo

Fetch references (branches, tags, updated objects, etc.) from a remote repo to your repo

git fetch REMOTE\_NAME\_OR\_URL

Fetching will get all the changes in references from the remote but it will not touch files in your working directory. You can merge the changes yourself after the fetch with:

git merge REMOTE/BRANCH

The merge will **merge and commit** to your local branch unless you run it with --no-commit option.

This tells me what remote branches I am tracking in my local repo and where it'll push and pull from

git remote show REMOTE\_NAME

List your remotes

git remote -v

Change which the URL of your remotes

git remote set-url REMOTE\_NAME NEW\_URL

Rename your remote

git remote rename OLD\_NAME NEW\_NAME

This tells me which local branch I am currently using

git branch -vv

If I want to add another remote for my local to track:

git checkout --track REMOTE/BRANCH\_IN\_REMOTE

(above) will create a local branch for me and then I can switch to it in my local with

git checkout BRANCH

All in one:

git checkout -t REMOTE\_NAME -b BRANCH\_NAME

Once I am in my local branch I can **reconfigure** where it pushes and pulls from

git branch -u REMOTE/SOME\_OTHER\_BRANCH\_IN\_REMOTE

## Forks

Synching your forked repo with the original

1. Make sure you added the original as a remote (assume you named it ‘origin’)
2. Fetch the original remote/branch (creates a new branch locally for you)
3. Switch over to your forked branch
4. Merge in changes from the original branch

git remote add YOUR\_ORIGIN\_REMOTE\_NAME YOUR\_ORIGIN\_REMOTE\_URL

git fetch YOUR\_ORIGIN\_REMOTE\_NAME

git checkout YOUR\_FORK\_BRANCH

git merge YOUR\_ORIGIN\_REMOTE\_NAME/YOUR\_ORIGIN\_BRANCH\_NAME

The above will synch up your local repo but if you want to update the branch that you’re on, then push the merged changes from your local to your remote.

git push

Merge from the upstream master to your local so that you can commit it back into your forked remote. Note that by default, merge will commit to your local repo if there are no merge conflicts. If there are, you will need to resolve them by hand and then commit them into your repo. Or you can abort the merge and git will return merged files back to the way it was before merging although it may not work all the time, so it is best to only do merges when you have already committed everything to your repo.

git merge upstream/master

## Getting a branch from upstream

git fetch upstream

git checkout -b develop upstream/develop

git push -u myfork develop

## File Management

List what is checked into your repo

git ls-files

Revert a file back to is checked in

git checkout FILE

Remove files from git (not from your working directory)

git rm FILE

Delete a branch on a **remote**

git push REMOTE\_NAME --delete BRANCH\_NAME

Delete a branch locally

git branch --delete BRANCH\_NAME

## Logs and Diffs

One of the more helpful options is -p, which shows the difference introduced in each commit. You can also use -2, which limits the output to only the last two entries:

git log -p -2

Doing a diff on a previous version

git diff HEAD~1:FILE\_PATH YOUR\_FILE\_PATH

For example, diffing what I have locally in my BDFD-284 branch in the src directory to what is in second look master

git diff BDFD-284:src upstream/master:src

git diff upstream/master myfork/BDFD-284

Diff two commits with

git diff COMMIT1\_SHA1 COMMIT2\_SHA1

You can get the hashes by using git log.

## Avoid merge commits

A quick and safe way to do a pull would be to ask git to only pull if it can fast forward. Git will abort the pull if it cannot just fast forward the changes.

git pull –ff-only

<http://kernowsoul.com/blog/2012/06/20/4-ways-to-avoid-merge-commits-in-git/>

Whenever we merge changes in from another branch (e.g. merging updates into our forked repo), the merge happens and then we must commit the merged changes on top of our own changes. And if we later merge our forked branch back to the trunk, the history will show this merge which is just noise since it really represents changes already done to the trunk. To avoid this, we need to run “rebase” which will merge changes from trunk into your local repo at a point that is where your local repo diverged. In other words, it *should* “rewind” your commits, apply the merge, and then “replay” your commits on top of the merged changes.

Loosely, a regular “pull” equals a “fetch” + “merge” and a “pull --rebase” equals a “fetch” + “rebase”. So to understand the difference between a regular pull and a pull with rebase, we should look at the difference between merge and rebase. See <https://www.atlassian.com/git/tutorials/merging-vs-rebasing/conceptual-overview>

One thing I have seen over and over in write-ups about rebasing is to **never** do it in a public branch; it should only be done in your local repo/branch. One gotcha in particular refers to creating a Pull Request. Once created, the Pull Request is now public and doing a rebase here would confuse Git and people who are reviewing your request. So, you should rebase before creating the Pull Request and only do merges afterwards.

Here is an example of something I have done. I have created a branch off of master on the pubic repository. Locally, I have both master and my branch checked out. I synched up both master and my branch locally. I made changes to my branch locally and I committed locally. Someone else made changes to master on the public repo. I switched to my local master and did a pull to update my local master with the new commits. I switch back to my local branch and issue a

git rebase master

After that, my local commits look like that happen after the last commit to master by someone else. So when I merge my branch into master, it will look the same way.

|  |  |
| --- | --- |
| My local started with this: |  |
| Master was updated to this: |  |
| After the rebase, my local repo looks like this: |  |

Instead of doing a pull on my local master, I have also done a fetch followed by a merge.

git checkout master

git fetch upstream

git merge upstream/master

git checkout BRANCH

A link that Alexis sent me:

<https://coderwall.com/p/euwpig/a-better-git-log>

It has an alias command for “lg” that will print the log in ascii art:

git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

## Tagging

### List Tags

git tag

### Create Tag

Note to self: for secondlook core, update the version number first

git tag TAG\_NAME

git push [REMOTE\_NAME] [TAG\_NAME]

### Delete Tag Local

git tag –d TAG\_NAME

### Delete Tag Remote

git push --delete [REMOTE\_NAME] [TAG\_NAME]

## Show info using hash

git show 4e4a96cfb4e1eb55d34148410c6a7a0d3e49eb09

## Get commit comments between two hashes

git log --pretty=oneline ebf2a62e5fdbede99dd89f869b88e7ed6408c2cc...a500843a0256a0f5b4c234c55646167379f36a45

# Spark

Spark application = a Driver program that can execute operations in parallel across nodes in a cluster.

## Spark SQL

DataFrame = A relational table

## Change value of one column

val d = sqlContext.read.parquet("s3a://card-secondlook/dev/posted\_data/instnc\_id=20160327000000\_12")

d.withColumn("accountId", when($"accountId" === "01772914196", lit("10000520196")).otherwise($"accountId"))

d.select("accountId").show()

res1.select("accountId").show()

res3.write.parquet("s3a://card-secondlook/dev/posted\_data/instnc\_id=20160327000000\_12F")

## Quick Examples

val rdd = sc.parallelize(Array("one two three", "two three four", "three four five", "four five six", "five six seven", "six seven eight"))

Gives you an RDD[String]

// Creates a one column "table" with the column name as "value"

// To do something like: select count(\*) from TABLE group by value;

sparkSession.readStream // gives you a DataFrameReader

.format("socket") // gives you a DataFrameReader

.option("host", "localhost") // gives you a DataFrameReader

.option("port", 9999) // gives you a DataFrameReader

.load() // gives you a DataFrame aka DataSet[Row]

.as[String] // gives you a DataSet[String]

.flatMap(\_.split(" ")) // gives you a DataSet[Array[String]] first then flattening it to DataSet[String]

.groupBy("value") // gives you a RelationalGroupedDataset

.count() // gives you a DataFrame

## Writing a spark job

1. You’ll need a main method
2. Your object should extend Scala Logging SLF4J’s LazyLogging
3. Create a new SparkConf object
4. If you’re doing spark streaming, create a new StreamingContext with the config object, else create a new SparkContext with the config object.
5. Start off with some data source and create a base RDD and pipeline your transformations and actions over the RDD.
6. If this is spark streaming, you’ll need to call the start method on the streamingContext and the awaitTermination method on the streaming context object or you can call stop on the SparkContext object if this is a stand alone application.

## Spark Streaming

First gather your spark configuration and create a spark streaming context. Each JVM may only have one streaming context created. But you may create multiple streams from the one streaming context. From the context you can create different types of data streams. Spark streams are actually micro-batches of discretized data that is fed to your program at a time interval you specify.

These micro-batches of discretized data are called “Discretized Streams” or “DStreams.” The first DStream created by the context is a special stream of type “input stream.” To process the DStream, your program will transform the first DStream into another DStream. Your program will continue to process the data chaining a series of transformations for each DStream. Remember that each transformation you provide is a function. But the function isn’t executed, they are just recorded. When the stream is started, your functions are applied to the “stream” of micro-batches of RDD sets. The stream of data is a sequence of RDDs where each RDD is the set of data for that time interval.

Spark doesn’t start listening for data from the stream until you start the “receiver” thread to do so. This is done by calling “start” on the streaming context. Once you start the thread, you need to put your main thread on hold so that it doesn’t exit the program. This is done by calling “awaitTermination” on the streaming context. You wouldn’t need to do this if you were executing this on a REPL since you won’t be exiting the program. If you start another stream, then another receiver thread will be created. These receiver threads are like any other executor in Spark. So you’ll need to make sure you have enough executors to run both the receiver threads and the normal work. You should setup all of your streams and start all of the receiver threads at once with the streaming context.

DStream turns a stream of data into one RDD at a time and hands it to your program. It “batches” the stream and each batch is one RDD.

Each batch executes a “computes” method to generate one RDD and submits the RDD in a Spark job. The stream will call “compute” over batches until the streaming context is stopped.

DStream API is similar to RDD API. You can reuse your RDD-based code and apply it to DStream - a stream of RDDs - with no changes at all (through [foreachRDD](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-streaming-operators.html#foreachRDD))

It runs [streaming jobs](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-streaming.html#Job) every [batch duration](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-streaming.html#batch-interval) to pull and process data (often called *records*) from one or many [input streams](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-streaming-inputdstreams.html).

Data is being streamed to Spark. Spark collects the data up to the batch interval time and creates an RDD. While Spark is collecting the data within this batch interval, it breaks up the data into Blocks. The break up is done by time also, so we also have a Block interval, which defaults to 200ms. Each Block become the data that is sent to a partition. This makes the data stream discretized and distributed.

**Batch Interval (aka batchDuration)**

**Batch Interval** is a property of a Streaming application that describes how often an RDD of input records is generated. It is the time to collect input records before they become a [micro-batch](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-streaming.html#micro-batch).

You use **stream operators** to apply **transformations** to the elements received (often called **records**) from input streams and ultimately trigger computations using **output operators**.

## Things to watch for

Make sure that the Processing Time for each batch can be done within 80% of your batch interval time or else batches will start to queue up.

From the Completed Batch list, you can drill down to the Batch Details and from there drill down to the Job Details and from there you can get to the Task Details of each Job Stage. Batch Details will show you information about the input data spark read during the batch. The Job Details will show a DAG of transformations and actions taken on the DStream.

## Checkpointing

Check pointing is the act of saving progress data to HDFS or some other distributed file system. The data can be just metadata so that a failed node can be brought back up and have enough metadata information to reconstruct the driver application. Or the data can be the RDD chain.

You first have to configure your check point directory using the spark context. The directory should be an HDFS path.

Follow directions here: <http://spark.apache.org/docs/latest/streaming-programming-guide.html#checkpointing>

## DataFrames

Here is some test code

val df = Seq((0, "hello", 21.99), (1, "world", 19.44), (2, "My Spotify Shop", 1.99)).toDF("id", "merchantName", "transactionAmount")

def in(name: String) : Boolean = { ".\*Spotify.\*".r.findFirstIn(name).isDefined }

val u = udf(in \_)

df.filter(u(df("merchantName"))

or

df.filter(udf(in \_).apply(df("merchantName")))

## Tuning

For Spark, you want to focus on memory and network bandwidth. Shrinking the data you keep and/or transmit is key.

### Serialization

serialization can serialize anything the implements Serializable but it is slow and the format is bloated. Kryo is faster and more compact but doesn’t handle all types. Kryo also works best if you register the classes to be serialized. Spark pre-registers a few common classes for you. Classes that are not registered requires the full class name to be saved with the object which will take up space.

Serialization is used when shuffling data and serializing RDDs to disk.

When you use the cache() method it calls the persist() method with the default storage level with is MEMORY\_ONLY. This level does not serialize the object first. To use something like MEMORY\_ONLY\_SER, you need to call persist.

Sometimes recalculating data for a partition is faster than writing and reading from disk. So be cautious with spilling to disk options.

### Memory

Consider if your dataset will fit in memory and how much GC pressure will your data apply.

Config parameters:

spark.memory.fraction: Portion of memory Spark can use of the total heap for execution and storage (cache) memory.

spark.memory.storageFraction: Portion of the spark memory space that can be used for storage memory and once allocated cannot be evicted to make more execution memory.

JVM’s NewRatio parameter defaults to 2 which means the Old generation is 2/3 of the heap space. Your Old generation should be as large or larger than your spark.memory.fraction or else too many full collections will happen. So if you see a lot of full GC’s you should increase the NewRatio or lower your spark.memory.fraction setting.

GC performance depends on the number of objects it needs to traverse. So if you can avoid using objects (an array of primitive instead of a linked list of objects) will help speed up GC. You can get stats on how often the GC is running with these parameters: -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps. The goal when tuning the GC is to have only truly old RDDs in the Old generation and make enough space in the Young generation for all short lived objects during task execution.

Greater number of minor collections means that the Young generation (Eden) isn’t big enough. The amount of memory needed to run a task can be used to configure the Young generation by multiplying for 4/3 and setting that to –Xmn.

OOM exception can come about when the combination of data sets a task is working with is too large to fit into memory. Shuffle operations require a Hash table to be built that can be large depending on the data it must work with. Increasing the number of partitions will shrink down the data that any one task has. This will also recruit more cores to the task and increase parallelism.

Spark needs to serialize task objects to send over from the driver to the worker. If there are large objects in the driver that are wrapped up in the task, you should use a broadcast variable instead.

## Questions

1. What is a window?

A window in your stream is a in memory union of all RDD’s that span a “window length” (number of batch intervals to accumulate) that is refreshed at a configured interval count call the “slide interval.” One thing to note is that the refresh count makes overlaps in your window. For example, if the batch interval is 1 second, the window length is 60 seconds, and the slide interval is 3 seconds, then the first window will have RDDs 1-60 and the second window will have RDDs 4-63. The RDDs 4-60 overlap between the two windows; hence the name sliding window.

1. What are Stateful operations in spark streaming like (mapWithState, updateStateByKey) for?  
    The updateStateByKey method is there to keep some data for you for the entire life of the DStream from RDD to RDD. As the method name shows, you will be maintaining state against key/value RDDs. Spark will save a single state for each key. The method will require you to provide a function to tell it how to update the state.
2. What is the difference between RDD.foreach and foreachPartition.  
    The foreach method will execute your function against every element for all partitions in the RDD, handing your function the element. The foreachPartition will execute your function against a partition that is represented by an iterator of elements in that partition. This extra segregation allows you to prepare things like database connections once per partition in your function. Otherwise, the function to the foreach method will prepare the database connection for every element. You won’t be able to pass in a connection to these lambdas.

## Structured Streaming

Main idea: abstract away the notion of streaming and use the “Table” as the main abstraction. This Table is “unbounded” and is append only. This unifies batch data with streaming data.

Instead of Batch Interval, they have renamed it Trigger Interval. When possible new data is appended to the table.

When you select from this table, you get a result table. Both the input table and the result table are updated at the Trigger interval.

If you write out your result table, you can choose to write out the “complete” table each Trigger Interval time or just the updates (i.e. new rows) with “append” mode.



Spark will take your batch query and “incrementalize” it to be applied to every batch of new data that arrives.

It will automatically checkpoint for you allowing to start at where it left off and it also de-duplicates when writing to sinks.

# Kafka

A centralize strategy for data integration across (enterprise, system, etc.) based upon real-time logs.

## What is a log?

* Time ordered, queue of immutable messages. Because messages are ordered, messages can be assigned a sequential offset.
* A log can be partitioned into many logs so that it can scale.
* The partitioned group falls under a generic topic which is just a category of data.
* They are used for replication (moving data from one place to another) and to bring multiple systems into consistency with one another.

Think of your enterprise systems as one big distributed system and you need a way to keep all the parts in the system in synch on the same set of data. Jay Kreps proposes to use a commit log to keep them in synch. Other ways to do this is to copy large files around from one system to another. But that is additional work, slow, done one by one, and not real time. The other way is to use a traditional messaging system. But they don’t scale out to handle large amounts of data.

Stream processing is just Batch processing if your batch is never ending and you can’t just cut it off to run a batch.

<http://docs.confluent.io/3.1.2/quickstart.html>

## Confluent Schema Registry

<http://docs.confluent.io/3.0.0/schema-registry/docs/serializer-formatter.html>

“When sending a message to a topic t, the Avro schema for the key and the value will be automatically registered in the schema registry under the subject t-key and t-value, respectively, if the compatibility test passes.”

<https://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one/>

“Writing Avro records to Kafka is as simple as configuring a producer with the Schema Registry serializers and sending Avro objects to Kafka. Schemas are automatically registered, so the whole process of pushing new schemas to production is seamless. Same goes when reading records from Kafka.”

<http://stackoverflow.com/questions/31204201/apache-kafka-with-avro-and-schema-repo-where-in-the-message-does-the-schema-id>

“In general what's happening when you send an Avro message to Kafka:

1. The encoder gets the schema from the object to be encoded.
2. Encoder asks the schema registry for an id for this schema. If the schema is already registered you'll get an existing id, if not - the registry will register the schema and return the new id.
3. The object gets encoded as follows: [magic byte][schema id][actual message] where magic byte is just a 0x0 byte which is used to distinguish that kind of messages, schema id is a 4 byte integer value the rest is the actual encoded message.

When you decode the message back here's what happens:

1. The decoder reads the first byte and makes sure it is 0x0.
2. The decoder reads the next 4 bytes and converts them to an integer value. This is how schema id is decoded.
3. Now when the decoder has a schema id it may ask the schema registry for the actual schema for this id. Voila!”

See: <https://github.com/confluentinc/schema-registry/blob/master/avro-serializer/src/main/java/io/confluent/kafka/serializers/AbstractKafkaAvroSerializer.java#L59>

Given that this logic is inside the KafkaAvroSerializer, you will need to use this in your producer for automatic schema registry. In other words, using a byte array serdes will not work. You can probably register it on your own though.

The KafkaProducer requires you to configure a serializer. When you call the “send” method the producer will call the “serialize” method on your configured serializer. The KafkaAvroSerializer will grab the schema from your message (GenericRecord is a GenericContainer which has a method to get schema). The KafkaAvroSerializer calls the schema registry to “register” the schema. The registry will return an ID back. This ID is then added to the byte stream (after the “magic byte”) that will be written to the Kafka datum writer (in AbstractKafkaAvroSerializer).

On the flip side, the KafkaConsumer’s “poll” method will eventually call the deserializer’s “deserialize” method with the topic and the bytes. For the KafkaAvroDeserializer, this call eventually gets down to the AbstractKafkaAvroDeserializer where it creates a ByteBuffer out of the byte array and separates out the magic byte and the 4 byte INT that represents the schema ID. Using the schema registry it will get the schema and create a DatumReader from it with a null “reader schema”. The DatumReader reads the bytes and returns the Avro record object back.

## Questions

1. TTL settings: is it per topic or for the entire cluster?  
    It is per topic
2. How do you specify a “reader” schema for avro data?  
    This seems to be purely an avro functionality. Kafka does this automatically for you.
3. Can you inspect messages for the schema ID?  
    The only way seems to be to read the message as a byte array and parse out the ID yourself. Otherwise, the KafkaAvroDeserializer does this for you but will not give you back the ID. The other way is to get the GenericData.Record object back by using the KafkaAvroDeserializer and grabbing the schema out of it. Then you can use the Registry to query for the ID.
4. How does the consumer grab the offset number?  
    You call ConsumerRecord.offset method.
5. How does the schema registry work?
6. How does spark structured streaming use consumer groups?
7. How do I reset the offset of a consumer group I want to reuse?

## Differences from typical messaging system

* Cluster based, not server based (allows scaling for high write throughput)
  + Analogous to HDFS, Kafka is good at persisting data across the cluster
  + Able to replay the data
  + Message systems get slower as more data is written to it
* Characteristics of a database: ordered messages and persistent
* Provides failover strategies with replicas
* Avails itself to concurrency through partitioning
* Messaging systems are more concern for delivery guarantees and requires consumer acknowledgments so that messages can be removed. Kafka’s concerns are centered around persisting a log of messages. Consumers need to keep track of where they are in the log (offsets).

### Verbatim from Alexis Richardson and Tim Watson of RabbitMQ:

Kafka writes all incoming data to disk immediately, and then figures out who sees what. So it is much more like a database than Rabbit, in that new consumers can appear well after the disk write and still subscribe to past messages. Instead, Rabbit which tries to deliver to consumers and buffers otherwise. Persistence is optional but robust and a feature of the buffer ("queue") not the upstream machinery. Rabbit is able to cache-on-arrival via a plugin, but this is a design overlay and not particularly optimal.

Kafka is a client server system with end to end semantics. It defines order to include processing order, and keeps state on the client to do this. Group management is via a 3rd party service (Zookeeper? I forget which). Rabbit is a server-only protocol based system which maintains order on the server and through completely language neutral protocol semantics. This makes Rabbit perhaps more natural as a 'messaging service' eg for integration and other inter-app data transfer.

Rabbit does NOT store multiple copies of the same message across queues, unless they are very small (<60b, iirc). A message delivered to >1 queue on 1 machine is stored once. Metadata about that message may be stored more than once, but, at scale, the big cost is the payload.

RabbitMQ will delete messages from its message store as soon as a relevant consumer has seen and ACK'ed them, which \*requires\* tracking consumer state in the broker.

Right, but just to be clear, unless Kafka starts to fsync for every single published message, you are /not/ going to offer the same guarantee. In this respect, rabbit is clearly putting safety above performance when that's what users ask it for, which is fine for some cases and not for others. By way of example, if you're using producer/publisher confirms with RabbitMQ, the broker will not ACK receipt of a message until (a) it has been fsync'ed to disk and (b) if the queue is mirrored, each mirror has acknowledged receipt of the message. Again, unless you're fsync-ing to disk on each publish, the guarantees will be different - and rightly so, since you can deal with re-publishing and de-duplication quite happily in a system that's dealing with a 7-day sliding window of data and thus ensuring throughput is more useful (in that case) than avoiding data loss on the server.

## Usage with Avro Schema

**Backwards compatibility**: message created with older schemas can be deserialized with the new schema.

Use case: The newest schema can be used against all old and new data.

**Forward compatibility**: message created with the new schema can be deserialized with older schemas.

Use case: Consumers coded against the old schema can still read messages created with the new schema so that Producer and Consumer can evolve schemas separately.

**Full compatibility**: both backwards and forwards compatible

One place you can set the compatibility setting is the schema-registry.properties file that you would pass to the schema registry starter script (see <http://docs.confluent.io/1.0.1/schema-registry/docs/config.html)>. The default is “backward” compatibility. If a producer tries to write a message with a schema that breaks the current schema, the producer receives a “IncompatibleSchemaException: New schema is incompatible with the latest schema” and the message never makes it into kafka. For example, not providing a default value for a new field.

## Use with Spark Streaming

Two approaches: old = receiver based (Kafka pushes messages to Spark), new = direct (Spark pulls Kafka messages)

Receiver:

Spark has a built-in Kafka receiver that we can use. Data received is stored in Spark executors and then jobs will process the data. You might lose data if executors die unless you enable “write ahead logs,” which will write the data to disk (i.e. you write the messages to disk). The disk must be in some distributed file system (e.g. HDFS, S3).

When the driver process fails, all the executors running in a standalone/yarn/mesos cluster are killed as well, along with any data in their memory.

Using write ahead logs will make things slower. To recoup some of the speed you will need to either create more receivers and/or have faster drives.

Kafka partitions on the consumer side dictates how many messages can be read at the same time on that topic. The number of partitions should be at least the number of distributed consumers.

RDD’s are partitioned across the spark nodes so that they may operate in parallel when each RDD “executes.”

There is a setting when creating Kafka streams that dictate the number of Kafka partitions to be consumed per topic. This Kafka setting does not correlate to RDD partitions. This controls the **number of threads** used in a single receiver to listen on the number of Kafka partitions. Also, number of receivers ≠ number of RDDs, so their numbers have nothing to do with one another. Even though you may have multiple threads/core in a single machine receiving messages from Kafka, it’s not distributed over the cluster unless you create multiple Kafka DStreams (i.e. creating multiple RDDs). There is a standard process for doing this where you create multiple RDDs and then “union” them together.

This approach uses Zookeeper to keep track of message offsets. There is a small chance that offset tracking may become inconsistent (after failures) with what has actually been consumed (i.e. you might consume the same message twice).

Direct:

Spark periodically queries Kafka for messages.

Spark will create as many RDD partitions as there are Kafka partitions (meaning the reading of messages will be distributed across the cluster).

Data retention is now a Kafka problem since you are pulling from Kafka and not getting pushed messages that you have to manage with a write ahead log.

Uses Spark checkpoints to keep track of message offsets to avoid possible inconsistencies between zookeeper and Kafka. However, if you have other tools that rely on zookeeper offsets, you will need to update them yourself.

### What is the Kafka Stream?

You create a Kafka stream using the KafkaUtils singleton from the Spark API. The stream is actually just a bunch of “micro batches” that Spark will feed your program. The stream is called an “Input DStream” and you should map it to a regular DStream. The Input DStream will consist of tuples: a key and the message itself. If the message in the Kafka topic is an avro object, the message will be a byte array.

The Kafka DStream has a “compute” method that runs on the **Driver** and is responsible for connecting to the Kafka partition and topic to get the newest offset value and then to create a KafkaRDD with those offset values as the “ending” offset. The “starting” offset is by default the “largest” or the same as the ending offset; which means the next micro-batch will be whatever is written to the Kafka topic next. Or you can set the “auto.offset.reset” to “smallest” which will start the micro-batching from the very first message still in the topic (see Kafka retention period). If you want to provide your own starting offset, you need to use the TopicAndPartition object to specify it.

If the Driver fails, the offset value it gets from Kafka at each time interval will be lost because they float around in the Driver’s memory. Remember that a DStream is a series of RDD’s (an RDD per interval). So while the Driver is running, each RDD is a continuous moving window of message offsets. However, when the Driver fails and you have to restart it, it will just start at whatever the largest offset is. It is possible that you may skip over messages or process messages again. In order for your process to start back up where it left off, you will need to persist the offset values somewhere. In direct stream, the offset value is not stored in Zookeeper and you don’t store the messages in a “write ahead log.” You can opt to store the offsets in Zookeeper, if you like. The advantage of doing this is that other Zookeeper clients can make use of those offsets.

### KafkaRDD

Remember:

* RDD is a set of data per time interval
* The data set of an RDD can be distributed over the cluster, each distribution is call a partition.
* KafkaRDD partitions map to Kafka partitions. Each partition has its own OffsetRange that specifies the topic, the partition number, the starting offset, and the ending offset.

The KafkaRDD implements the HasOffsetRanges interface that can be used to get an array of OffsetRange objects. This array stores offsetRanges for each RDD partition. Remember that there is a one-to-one correspondence between an RDD partition and a Kafka partition. If you had to call the KafkaRDD constructor yourself, you would need to hand it this array of offsetRanges so that it knows all of the offsets to use when creating its partitions. But since the “direct” KafkaStream created the RDD for you, you can ask it for the ranges; just remember it is an array of offset for all of its partitions.

### Streaming Offsets

#### Questions

When we create a direct kafka stream and provide it offsets, is that a one time thing that happens at the start since it’s when we create the stream? More generally, does the input DStream only happen once? The other DStreams are a series of transformations that will occur at each time interval over a particular RDD.

When calling foreachRDD on a kafka stream, is that called at every interval (aka at every new RDD)?

How do I use createDirectStream with specified offsets? Specifically, why do I need a message handler for?

Should I use ZkClient?

Should I process the DStream first and then call foreachRDD?

#### Configuration Properties

Kafka parameter: auto.offset.reset

Set to “largest” or “smallest”; the default is largest.

Spark parameter: spark.streaming.kafka.maxRatePerPartition

Adjusts the maximum number of messages from a partition per micro-batch.

### Message Delivery

Spark streaming of Kafka topics can either be “at most once” or “at least once” according to the configuration you give it.

At most once means that you don’t care to lose messages.

At least once means that you don’t care to process the same message again.

If you want “exactly” once, the best you can do is set up your configuration to “at least once” and make your “side effects” idempotent. In order words, there’s no such thing as “exactly” once processing.

#### “at most once” settings

set spark.task.maxFailures to 1 (kill job on task failures)

set spark.speculation to false

set auto.offset.reset to largest

#### “at least once” settings

You need to persist the offsets you have processed or set auto.offset.reset to smallest to reprocess everything again.

## Sample Confluent Shell Commands

bin/kafka-avro-console-consumer --property schema.registry.url=http://localhost:8082 --topic s1 --bootstrap-server localhost:9092

# Data Formats

## Avro

Provides compression

Row based storage

Provides schema definition (all data MUST have a schema)

Provides a mechanism for schema evolution

Can generate generic or specific POJOs

Schema packed with data or stored in registry or added to HBase and Cassandra

Many third party tools have support for it

Can be used with RPC

# AWS

aws ec2 describe-instances --instance-ids "`curl http://169.254.169.254/latest/meta-data/instance-id`" --region "us-east-1"

# Maven

## Starting a new Scala Project

1. mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.capitalone.[app] -DartifactId=[artifact]
2. cd into new [artifact] directory
3. git init .
4. update <url> in pom.xml
5. Add .gitignore and add at least target/ and the idea/ folder and \*.iml files
6. Add and commit your files to your local repo
7. Create a repo in your github
8. git remote add origin <https://github.kdc.capitalone.com/rhp086/kafkastreams.git>
9. git push -u origin master

## Starting a new Java Project

#1 above will create a java project for you.

In general you can use the interactive mode by just doing: mvn archetype:generate (without -B -- batch mode)

Then you can pick a scala archetype and provide your group and artifact ID.

Look for "net.alchim31.maven:scala-archetype-simple"

## Mixing Java and Scala in one project

Follow this: <http://davidb.github.io/scala-maven-plugin/example_java.html>

# SBT

The % method is used to construct an Ivy module ID from strings

Keys have an overloaded method called in used to set the scope. The argument to in can be an instance of any of the scope axes.

For example, set the name scoped to the Compile configuration

name in Compile := "hello"

it’s important to understand that in and := are just methods, not magic. Scala lets you write them in a nicer way, but you could also use the Java style:

name.in(Compile).:=("hello")

To change the value associated with the compile key, you need to write compile in Compile or compile in Test. Using plain compile would define a new compile task scoped to the current project, rather than overriding the standard compile tasks which are scoped to a configuration.

Assignment with := is the simplest transformation, but keys have other methods as well. If the T in SettingKey[T] is a sequence, i.e. the key’s value type is a sequence, you can append to the sequence rather than replacing it.

* += will append a single element to the sequence.
* ++= will concatenate another sequence.

Unmanaged dependencies work like this: add jars to lib and they will be placed on the project classpath.

Dependencies in lib go on all the classpaths (for compile, test, run, and console).

There’s nothing to add to build.sbt to use unmanaged dependencies, though you could change the unmanagedBase key if you’d like to use a different directory rather than lib.

sbt uses Apache Ivy to implement managed dependencies

you can simply list your dependencies in the setting libraryDependencies

The key “libraryDependencies” is a settings key of a sequence (aka list) of “module IDs”

Module ID objects are created with the “%” method and can be chained together like this:

libraryDependencies += groupID % artifactID % revision % configuration

Using ivy, sbt will download your dependencies from either its standard list of repositories or ones you described and store them in the “.ivy2” directory in your home directory. The “update” task will do this. The “compile” task depends on “update”

Of course, you can also use ++= to add a list of dependencies all at once:

libraryDependencies ++= Seq(  
  groupID % artifactID % revision,  
  groupID % otherID % otherRevision  
)

resolvers does not contain the default resolvers; only additional ones added by your build definition.

sbt combines resolvers with some default repositories to form externalResolvers.

Therefore, to change or remove the default resolvers, you would need to overrideexternalResolvers instead of resolvers.

If you want a dependency to show up in the classpath only for the Test configuration and not the Compile configuration, add % "test" like this:

libraryDependencies += "org.apache.derby" % "derby" % "10.4.1.3" % "test"

If your project is in directory hello, and you’re adding sbt-site plugin to the build definition, create hello/project/site.sbt and declare the plugin dependency by passing the plugin’s Ivy module ID to addSbtPlugin:

addSbtPlugin("com.typesafe.sbt" % "sbt-site" % "0.7.0")

If you’re adding sbt-assembly, create hello/project/assembly.sbt with the following:

addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.11.2")

Not every plugin is located on one of the default repositories and a plugin’s documentation may instruct you to also add the repository where it can be found:

resolvers += Resolver.sonatypeRepo("public")

# PIP

# Unix

List services

startKafkaConsumer "SECONDLOOK\_DETECTION\_OUTPUT"

68 pid1=$!

69 startKafkaConsumer "SECONDLOOK\_DETECTION\_FILTERED\_OUTPUT"

70 pid2=$!

71 approvalCode1=`sendAuth vim

systemctl -all list-units spark\*

Start service

dzdo systemctl start spark-worker.service

# Ansible

Running a playbook:

ansible-playbook playbooks/myplaybook.yml -i inventories/INVENTORY\_FILE

# SSH

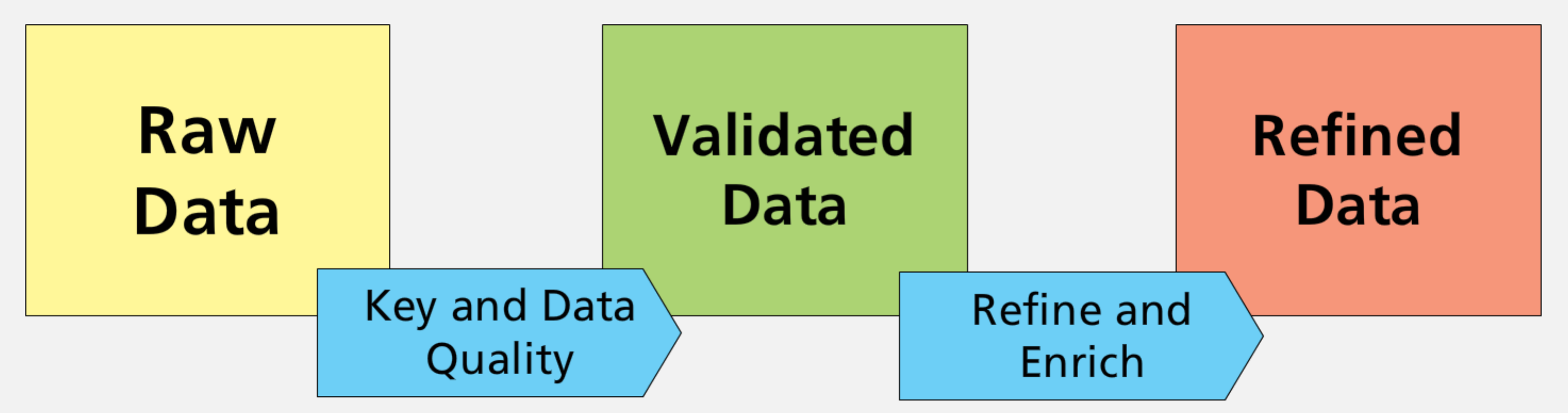
Tunneling to Second Look QA server

ssh -t -L4242:localhost:4242 rhp086@card-bastion-ql.kdc.capitalone.com ssh -L 4242:localhost:22 secondlook@10.203.80.238

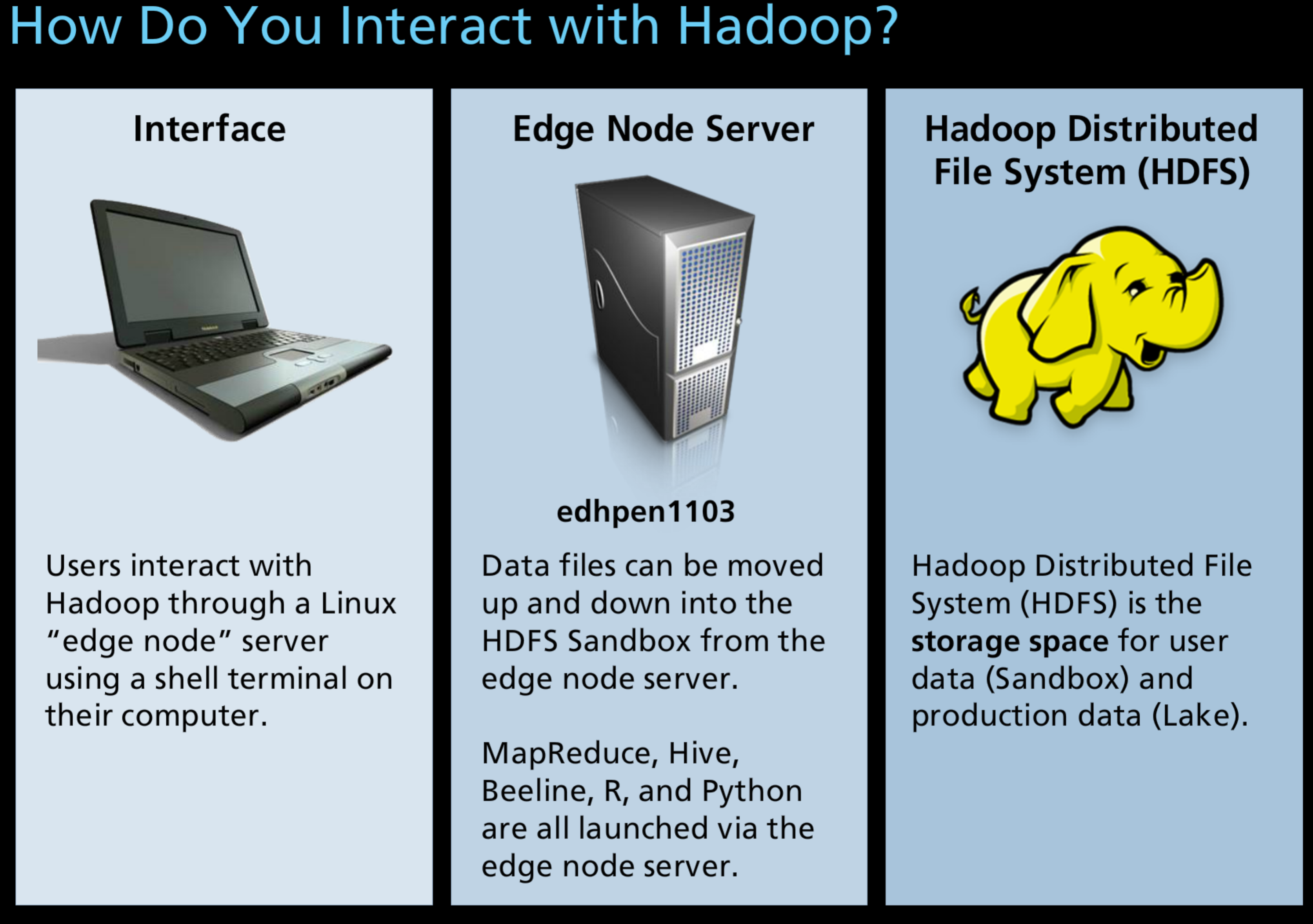
# Hadoop Data Lake

The Lake refers to production data in Hadoop.

Three states of data:



All data is logged in the Registry when created. The Registry is a home grown tool.



# Mac/Unix

## Searching for a service that is running

sudo systemctl status spark-worker.service

## ssh with a pem key

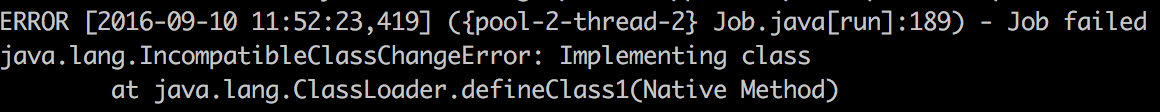
ssh -i secondlook.pem ec2-user@10.203.80.251

## Find Java home by running

/usr/libexec/java\_home

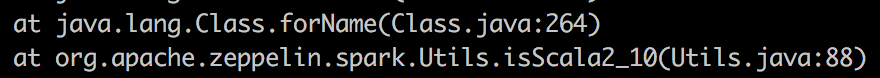
# Zeppelin

When I first installed Zeppelin, I kept getting this error after doing a simple println:

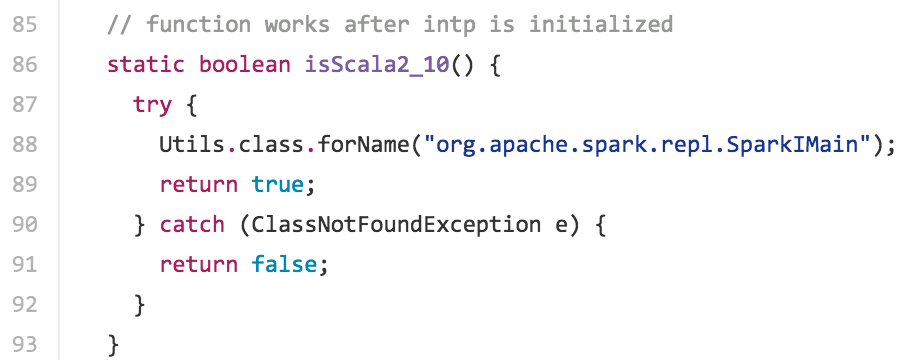


It was the most annoying thing.

The line that caused this was



Looking at <https://github.com/apache/zeppelin/blob/v0.6.1/spark/src/main/java/org/apache/zeppelin/spark/Utils.java>:



It fails to load a Spark class because the version of the class it tried to loaded had some changes to a non-private variable that went from static to non-static or vice versa. I finally realized that it was because I had SPARK\_HOME defined in my environment and that pointed to a Spark version that was compiled with Scala 2.10. All I had to do was “unset” the environment variable (or alternatively download a new version Spark). Since Zeppelin comes with a version of Spark, unsetting it worked.

# Spark Machine Learning

## Data Preperation

## String columns

Most of the time you need to convert strings into some type of numeral representation. You can do this with the StringIndexer class by giving it the string column name and the output column name that it was add to your data frame.

# Interviewing

## Phone Screening

* Ask for a past project where candidate had to learn a new skill. How did the candidate go about doing that?
* Ask for a past project where the candidate had a major impediment. How did the candidate work through that?
* Ask the candidate for some of his software principles.

Adcock, Lee: (9:55 AM)

Mainly we are trying to move towards less quizzing them on Java specifics, and getting a better read on their **ability to learn quickly**, their **effectiveness on past projects**, and **understanding of comp sci concepts**.

Adcock, Lee: (9:57 AM)

I like to start with finding a project they've worked on that was a highlight for them, and then dig in as far as we can to see how it works, what they did to make it work, where the challenges were, what role they played, how they influenced it.  All working back to those three things we value, learning quickly, getting results, and a concepts.

Adcock, Lee: (9:59 AM)

Look for a time in their resume where they started using a technology that they didn't use on the previous job, and **ask about the process of learning** and applying the new skills.  How did they learn it, what role did they play in educating team members, how long did the process take.

Adcock, Lee: (10:00 AM)

Try to dig up a example of where they were part of **making a key decision**, and then get them to share the considerations and factors that weighed into the decision.

Adcock, Lee: (10:02 AM)

Then how they communicated the results, how they influenced the larger group to take their decision, how what the final result was.

## Java Assessment

Introduction for Interviewer:

Discuss past projects with candidate, if there is a project that is a good fit for assessing the four criteria, it can be used for following questions. Otherwise, if the candidate needs a starting point, provide a scenario or well-known problem and generic data set (not puzzles). The goal is to make data management through software. Some examples of these types of scenarios might include the following (see appendix for problem set details):

Travel between Cities:

* matrix of distances between cities. [Appendix A]

Traffic Signals:

* intersection managed with traffic signals with varying timing. [Appendix B]

Car Rental: inventory of cars and reservations [Appendix C]

Model the Data:

* Persistent database storage
* Software class objects

Innovation and Collaboration/Teamwork (10 min):

Dev Ops:

* How would you coordinate the development of this project across multiple members on a team?
* How would you break up the work and why?
* How many developers, and with what skill sets, would you want on a team?

Passion:

* What new technologies would you want your team to experiment with?
* How would you promote innovation and open thinking on your team?

Design: What development paradigms have you been part of in the past, like Waterfall or Agile? Which would you choose for this project and why?

Deployment and Release (5 Minutes):

Dev Ops:

* How would you coordinate code management and deployment?
* What tools would you choose for source management and why?
* What tools do you currently use for deployment?
* What are their strengths and weaknesses?
* How can automation play a role in a build and deployment process?

Passion:

* How might you learn about new CICD tools and techniques?
* When was the last time you had to learn a new technology to overcome a problem?
* How did you decide the technology was a good solution to your problem?
* How did you ask for help?

Quality (10 min):

Design:

* How would you debug errors in your application? What steps do you take as you analyze a software problem? What tools would you use? What do you do if the root cause is the business intent or an external dependency on another team? Tell me about a time you had to debug an issue.
* How do you design for fault tolerance to address platform instability?
* How do you ensure long-term maintainability of your application code?
* What are some ways we can ensure quality in the application we are creating? What stages in the development process can be a source of bugs? How could we detect these bugs?

Dev Ops:

* What unit testing frameworks have you used? What role would mocking play in your testing?
* What is the importance of code coverage, and how is it measured? What is a reasonable level of code coverage? What is the difference between line coverage vs branch coverage?

Perf & Scalability:

* How would you ensure your application would scale to support the user load?

Passion:

* What are some different options for open source logging, monitoring, or runtime performance analysis that you might consider using?

Distributed Systems / System Integration (15 min):

Design:

* How would you expose this data to external consumers and data providers?
* Would you use a REST or SOAP service? What factors would influence this choice?
* Design the resource endpoint ontology and http verbs to provide read and write access.
* If you later realized you made the wrong choice, how would you go about the process of migrating from one type of service to the other?
* What security concerns would you need to address?
* How would you manage error handling and invalid data?
* How would you manage sessions and session state? What is the role of the session? When is it appropriate to have a stateless application?
* What framework might you use to make web service development easier? What are the strengths and weaknesses of this framework?

Perf & Scalability:

* What impact do these choices make on performance? How can the web service design impact performance?

Passion:

* Have you written code that consumes public APIs?

Data Modeling and Application Design (15 minutes):

Design:

* How would you handle object relational mapping?
* How did you make choices on data types?

Perf & Scalability:

* How does this design address performance?

## Spark Questions

<https://www.dezyre.com/article/top-50-spark-interview-questions-and-answers-for-2016/208>

What is Spark? Why should we use it?

Candidate should say something about its speed advantage over traditional map reduce. Speed is from savings gotten from less IO traffic, Spark RDDs are lazy and will not need to constantly write and read intermediate results. RDDs are lazy in the sense that they keep the instructions provided to them (lineage graph) and will only execute those instructions when needed. Another advantage is that RDDs can be persisted and replayed if a node goes down. Spark is also arguably easier to develop.

Many map reduce solutions out there are meant for batch processing. Spark speed and availability of Spark streaming allows for real time processing.

Compare Transformations vs. Actions on RDDs?

Resilient Distributed Dataset: RDDs are partitioned/spread across a cluster.

Transformations produce another RDD. They are immutable, you can only transform it (i.e. create a new one).

Actions are actual computations on a RDD to return a result. This is when the lineage of transformation instructions is executed

What are DStreams?

Discretized Stream: a sliding window of “micro” batched data handed to RDDs actions. It uses “micro” batches in the sense that the size is small enough to be considered “real time.”

How does Spark relate to Hadoop?

Spark uses HDFS to work on data in any Hadoop data format. It can run on a Hadoop cluster with over YARN or with Spark standalone mode.

How is Spark deployed?

Locally: over a number of cores

Standalone mode: install spark on every machine and then “start” the master and slaves on each machine. Submit applications in either “client” mode or “cluster” mode. Client mode = driver is launched on the submission client process. Cluster = driver is launched by a worker node and the submission client process can exit

YARN: (cluster management package that was extracted from Hadoop) Tell spark submit to use Hadoop YARN by specifying “yarn” as the master. You can run in “cluster” or “client” modes. Cluster = Driver runs in YARN’s master and your submission client can exit. Client = Driver runs in your submission client process and the YARN master is only used when requesting resources.

Mesos: Mesos master replaces the spark master. Spark installed on Mesos slaves and use the SPARK\_HOME env var.

EC2 Scripts (AWS EC2’s): automatically sets up Spark and HDFS on your EC2 cluster. Set your AWS\_ACCESS\_KEY and AWS\_SECRET\_ACCESS\_KEY before running the scripts.

What is a pair RDD?

A special RDD for processing key/value pairs more efficiently. We can create one by simply mapping a RDD to a function that produces key/value pairs (produce tuple in scala). Any function that operates on a pair RDD must take a tuple as input. With this type of RDD, you can aggregate, reduce by key, fold by key, combine by key, joins, grouping, counts, collecting by key, looking up by key, etc.

What is a broadcast variable?

Variable in memory on every worker in the cluster so you don’t have to pass the variable to every task. They are READ ONLY. Use cases may include storing large amounts of data, store look up tables, avoid looking up by key in RDDs,

What is an accumulator?

A variable that can be added to in parallel. Tasks can add to the accumulator but not read from it. Only the Driver can read the value of an accumulator. They have to be given a name if you want to see the value in the Spark UI.

When does Spark execute a shuffle?

On certain RDD operations that require nodes to work on particular sets of data that are scattered across other nodes.

Difference between Driver, Worker, Executor, and Task?

Driver: main process to convert the main program into tasks.

Executor: Processes that are running on a worker. They have a copy of the RDDs in memory. They execute tasks.

Worker: Represents a node (a machine) in the cluster

Task: some unit of work to be executed.

What’s the difference between Avro, Json, and Parquet formats?

Avro: stores schema with data in binary for storage and transmission. Allows your program to created strong-typed variables that refer to the data. Retrieval gets the entire record (row based).

Parquet: Stores column values across all rows next to each other so that sequential access for a few particular columns of all your records is fast.

If someone deploying a Spark application sees a java.io.NotSerializableException, what would you tell the person might be the problem? What would you propose to be a solution?

## Hive

<https://www.quora.com/What-are-the-best-practices-around-designing-tables-on-Hive-Also-are-there-any-tips-tricks-on-improving-hive-performance-I-understand-that-there-may-not-be-any-silver-bullets-but-any-pointers-would-be-very-helpful>

Best Practices

Avoid text formats for storage. Use Row Columnar files (RCFile) format because they are optimized for Hadoop.

Avoid using S3 as main storage, use HDFS.

Partition data if it is large to allow Hive queries to smaller data sets.

Use small tables on left side of joins and enable auto optimization so that Hive will cache the left table in memory.

Execute independent Map Reduce stages in parallel

Enable processing of batch rows together

## Java Questions

<http://www.java-success.com/a-q10-top-50-core-java-interview-questions/>

In Java, what purpose does the key words **final**, **finally**, and **finalize** fulfill?

Can you describe “method overloading” versus “method overriding”? Does it happen at **compile time** or **runtime**?

Static class loading throws “***NoClassDefFoundError***” if the class is not found and the dynamic class loading throws “***ClassNotFoundException***” if the class is not found.

What tips would you give to someone who is experiencing a class loading or “Class Not Found” exception?

<http://www.java-success.com/scenarios-java-multithreading-interview-questions/>

Can you give some scenarios where you had used multi-threading in Java applications?

Can you give some scenarios where you used the synchronized keyword in Java?

Can you describe the multi-threading issues due to deadlocks, thread starvation, and thread contention?

## Spring Questions

What is the Spring MVC framework?

What is IOC?

Annotations: @Component, @Controller, @Repository, @Service, @RequestMapping, @Autowired, @Required

What does the ViewResolver do?

Validator interface and making it into a bean.

## Scala Questions

What is the difference between implicit parameters and implicit object conversions?

What are implicit object conversions used for?

What are implicit parameters used for?

Implicit parameters: automatically pass missing parameters to a function. You can annotation and val or def with “implicit” which tells the compiler that those are candidates to be used in implicit parameter. This remove some “repeated” code clutter when you have to pass an argument like a database connection to every DAO function. Instead, the connection can be implicit and the method calls will be cleaner.

Implicit object conversion: a def marked with “implicit” that converts any type A to any type B. The compiler will call this conversion def whenever a method is called on objects of type A but the method is not defined on type A. Once this conversion happens, the method is called on the converted type B object. This allows API providers to ship with their API some trait implementations so that the user of the client doesn’t have to create common implementations.

What are implicit classes?

Makes the primary constructor an implicit conversion. Almost the same use case as implicit object conversions but you can define the class that you’re converting to all in one step instead of creating a def for the conversion and then later creating the class that has the extended behavior.

What is the difference between def and val?

How about lazy val

What is the “with” keyword used for?

What are the access modifiers in Scala?

“default” = public

private

protected

Can be assigned to members of packages, classes or objects.

Can specify a scope which gives access up to a package, class, or object.

Describe how you would handle actions that may throw exceptions?

Describe the “apply” function.

## Testing

What are mock objects? Why are they used?

Is there a difference between a mock and a stub?

## Core Principles

What are the core design principles you use?