Tools for Assisted Spark Migrations

From Spark 2.1 to the "future"*

Because frankly no one likes legacy code

One of my nightmares involves me waking up in the basement of some office building to the sound of a motorola pager going off (shout out to my ex-amazon friends) to fix an out of memory exception in a Spark 1.6 job. Also for some reason I'm out of coffee. It's not a good day. But together we can prevent that version of the future.

Also seriously I don't want to backport another Spark PR to Spark 2.4 god damn it.

The adventure ahead!*

- Meeting the "narrator"**
- Our goal: to save ourselves from legacy software
- Our challenge: developers (and data scientist) are…. lets say lack excitement to upgrade legacy pipelines.
- Our tools: some janky scala, python, and perl code (insert evil laughter)
- My side quest: to promote my books (you should like totally buy them)
- Will we triumph? Only if we scope our OKRs down enough.

*Shout out to past us for causing some of this "adventure."

Who am I?

- My name is Holden Karau
- Pronouns are she/her
- Apache Spark PMC (think committer with tenure)
- Contributor to a lot of other projects
- co-author of High Performance Spark, Learning Spark, and Kubeflow for Machine Learning
- Twitter: @holdenkarau
- Slideshare http://www.slideshare.net/hkarau
- OOS Livestreams: https://youtube.com/user/holdenkarau
- Github https://github.com/holdenk
- Related Spark Videos http://bit.ly/holdenSparkVideos
- Do not speak for my employer

Soooo before we go forward:

- Raise your hand if your using Apache Spark
- Keep your hand up if your still using a Spark before 3.0
- Keep your hand up if your still using a Spark before 2.4
- I really hope there are still some hands up because otherwise this talk is less useful…..

I have a problem, let's fix it with computers

- "I played a role in helping ... become the deprecation-happy prima donnas that they are today, when I built Grok, which is a source-code understanding engine that facilitates automation and tooling on source code itself" https://medium.com/@steve.yegge/dear-google-cloud-your-deprecation-policy-is-killing-you-ee7525dc05dc
- Oh hey that sounds familiar
- Wait "killing you" -- that doesn't sound good -- w/e

What's the OSS version of grok?

- Scala
  - scalafix
- Python
  - 2to3
  - Python-Modernize
  - Rope
  - bowler
  - etc.
- Go
  - gofix
- Generic-ish
  - sourcegraph

Hmmmm those are some interesting names

- Most created to upgrade language versions
- Or fixes

How does this relate to Spark?

- We deprecate & then delete a lot of stuff
  - I know we say binary compatibility but look at the exceptions
  - Then look everything labeled experimental
  - Then everything labeled developer API
- Ok, let's try and use a tool to fix everything, what could go wrong?

What do they look like [Scala]

```scala
override def fix(implicit doc: SemanticDocument): Patch = {
    val readerMatcher =
SymbolMatcher.normalized("org.apache.spark.sql.DataFrameReader")
    val jsonReaderMatcher =
SymbolMatcher.normalized("org.apache.spark.sql.DataFrameReader.json")
    val utils = new Utils()

    def matchOnTree(e: Tree): Patch = {
      e match {
        case ns @ Term.Apply(jsonReaderMatcher(reader), List(param)) =>
```

What do they look like [Scala] (continued)

```
          param match {
            case utils.rddMatcher(rdd) =>
              (Patch.addLeft(rdd, "session.createDataset(") +
Patch.addRight(rdd, ")(Encoders.STRING)") +

utils.addImportIfNotPresent(importer"org.apache.spark.sql.Encoders"))
            case _ =>
              Patch.empty
          }
```

What do they look like [Scala] (continued)

```
      case elem @ _ =>
        elem.children match {
          case Nil => Patch.empty
          case _ => elem.children.map(matchOnTree).asPatch
        }
    }
  }
  matchOnTree(doc.tree)
}
```

What do they look like [Python] w/Bowler

```
migration_pairs = [("toDegrees", "degrees"), ("toRadians", "radians")]
def make_func_rename_query(old_name, new_name):
    print("Making query to rewrite " + old_name + " to " + new_name)
    return (
        Query(sys.argv[1])
        .select_function(old_name)
        .rename(new_name)
        .execute(interactive=(not no_prompt), write=False, silent=silent)
    )
```

Semi-automated as an option

Do types help?

- Yes!
- >10% of function names are not unique in PySpark
- Especially complicated with PySpark code with mixes pandas & Spark code since borrowed function names & concepts

Ok but like what about running it on a project?

- https://github.com/holdenk/spark-testing-base
- https://github.com/holdenk/spark-testing-base/pull/330
- ^ Click above and we can walk through some of the automatic adventures

Let's skim some rules:

https://github.com/holdenk/spark-upgrade/tree/main/scalafix/rules/src/main/scala/fix

Ok, but where doesn't this work well?

- Dependencies
  - In my super informal survey of folks the #1 reason blocking upgrade was ElasticSearch connector
- Programming language version change
  - The reality is there's a lot of Scala 2.11 code out there, our resources are focused on 2.12->2.13 migration's but folks are further back
  - Scala version change was the #2 reason blocking Spark upgrades for folks
- API changes (what this "solves") came in #3
- Imports[ish]

Wait a second language version? I thought this fixed that?

- Yes, search "migrate Python 2 to 3" and cry
- The Scala tools are focused on migrating to dotty, many people are on waaaay older versions
- It might not be as cool as helping people migrate to 3 but helping people get to 2.12 is needed

How do we know if it worked?

- Hope is not a plan
- Tests? (See https://github.com/holdenk/spark-testing-base)
- lakeFS or Iceberg + side by side runs https://github.com/holdenk/spark-upgrade/tree/main/pipelinecompare
  - Demo?
- Validation queries
  - SodaCL
  - https://datatest.readthedocs.io/en/latest/intro/pipeline-validation.html
  - Your favourite internal tool

In conclusion:

- The good news is we haven't made a system so powerful we can change APIs left and right
- The bad news is we haven't made a system so powerful we can change APIs left and right
- The excellent news is: my dog is cute
- Even more excellent: there is time

  left to tell you about:

  DistributedComputing4Kids!*

http://www.distributedcomputing4kids.com

The start of their adventure

Alex and her friend Ziva play most weekends. This weekend, they want to take photos for Instagram. They want their photos to be of a topic that is popular with their friends. To know what is popular, they want to see the recent top tags used by their many friends. The Instagram app doesn't show what is trending with just their friends. So they decide to try and figure this out.

Looking up the top tags for each of their friends, one at a time, would take far too long, as Alex and Ziva have many friends. Think superstars like Ariana Grande or JoJo Siwa level friends. They would also need to add up the results and sort them. So Alex and Ziva ask some of their friends, the four garden gnomes, to help them.

Garden gnomes normally don’t spend a lot of time on Instagram. Instead, they are on Instagnome. But these garden gnomes love working together on projects almost as much as they love tea. They like tea a lot because it can get chilly outside.

Our garden gnome friends offer to help.

In exchange for some tea, the gnomes agree to help Alex and Ziva. The gnomes will look up the tags from recent posts of Alex and Ziva's friends. Ziva asks all of the gnomes to install Instagram on their phones.

The gnomes start by downloading Instagram. While it installs, Alex and Ziva write down a list of their friends' Instagram accounts. They use a stack of index cards so the list can easily be split up and divided among the gnomes. Splitting up a single page of paper would be difficult to get right.

Getting to work

The gnomes talk to decide who should be the leader for this project. A leader is needed so that they can make choices quickly. They agree that the gnome with the blue hat should be the leader.

She looks at the stack of cards to guess how many each gnome should take so that it is equal. If the gnomes don't get a fair share, then those with fewer cards will finish before the others. It takes a gnome a minute to handle one card. If one gnome takes 30 cards and another gnome 60 cards, the first gnome would have to wait another half another hour for her friend to finish. The gnome with fewer cards would also get bored and drink all the tea.

Each gnome, except for the leader, grabs a share of the cards. The leader is busy keeping track of the work. Once each of them has Instagram installed and gets their share of cards, they start looking up each account. When they load the account, they check below the most recent photo to see the tags and write down each tag on a new card. So, if underneath the photo it said, "Morning with the #bestdog #coffee," the gnome would write down "#bestdog" and "#coffee." It would look something like this:


#bestdog
#coffee


#bestdog

Figuring out their local tag counts

Each gnome now has a list of tags. Using this they want to find the number of times each tag was used. Each gnome grabs a fresh stack of cards and goes through their list of tags.

They know they'll need to combine the counts, so they give each new tag a card. When they make a new card, they write the tag at the top and put a tally mark (a slanted line, like '/') on the bottom. This way, a single card keeps track of the tag and its count. If the gnomes find a tag they've seen before, they go back and add another tally mark.

One of the gnomes got tired

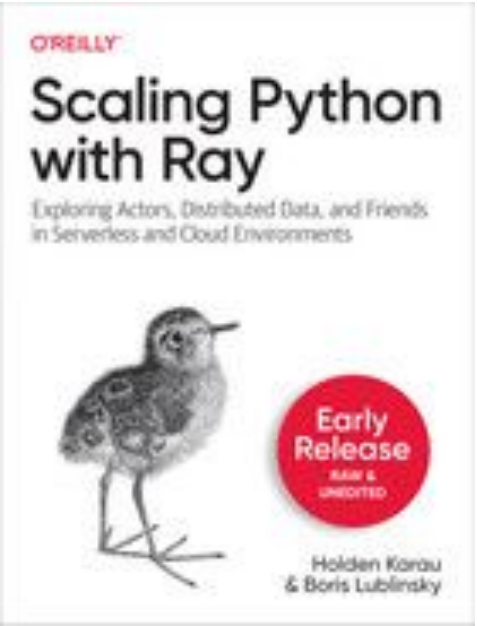
Just like when playing a game or doing a group project, sometimes someone has to leave. The gnome in the red hat was tired from being up late reading. Partway through trying to sort his cards, he fell asleep. The leader asked the gnome in the green hat to take over for him. Ze agrees and picks up the cards.

Ok that's as far as we got with the illustrations so far

- But you can join in - https://docs.google.com/document/d/1MZ6-cVTN8EgXCLsXAGpSp38n0FtkudM1SXRabeNJk_4/edit?usp=sharing
- Currently the examples are in Python (I know I know)
  - I think we can generate the same book for Python and Scala
  - But not a lot of other Scala resources for kids – http://blog.schauderhaft.de/2011/02/06/teaching-a-kid-scala/
  - https://github.com/holdenk/distributedcomputing4kids

But most importantly….

Buy several copies of my books :p (or read them on safari, I think I get money from that?)




Ok, thanks!
Q&A?