PART 1. LARGE SCALE DATA ANALYTICS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

• PA1
Programming Assignment 1

- Creating N-gram profile for a Wikipedia Corpus
- Due: 5:00PM Sept 30 2019

Objectives
- Basic features of Hadoop distributed file system and MapReduce
- Creating NGram profiles using Hadoop MapReduce

- N-Gram
  - A contiguous sequence of N items from a given sequence of text or speech
  - “We analyze large datasets”
  - 1-grams (aka unigram)
    - We, analyze, large, datasets
  - 2-grams (aka bigram)
    - (__, We), (We, analyze), (analyze, large), (large, datasets), (datasets, __)
  - https://books.google.com/ngrams/info
Programming Assignment 1

- **Requirements**
  1. extracting all distinct unigrams
  2. computing the frequency of each unigram per article and also over the corpus
  3. ranking the unigrams based on these frequencies

---

Structure of PA1: Profile 1

[Diagram showing the process flow of PA1 with Mapper, Sort and eliminate duplicates, Join Reducer, and Output Part A and B nodes.]
Programming Assignment 1

Profile 1
A list of unigrams that occurred at least once in the entire corpus (1G dataset). The unigrams must be sorted in (ascending) alphabetical order. You should eliminate duplicates. The output should be generated using MapReduce. You may store the output in multiple files.

Profile 2
A list of unigrams and their frequencies within the target article. Your software must generate this profile per article. Your list should be grouped by the Document ID (see page 3), and sorted (in descending order) on the frequency of the unigram within the article. This output should be generated using MapReduce. Output may be stored in multiple files.

Profile 3
A list of unigrams and their frequencies within the target corpus. The list of unigrams should be sorted (in descending order) on the frequency of the unigram within the corpus (1G dataset). This output should be generated using MapReduce. Output may be stored in multiple files.

Input Data

• Format

```plaintext
... Title_of_Article-1<====>DocumentID-1<====>Text_of_Article-1
NEWLINE
Title_of_Article-2<====>DocumentID-2<====>Text_of_Article-2
...
```
April 28<====>1639<====>April 28 Events224 – The Battle of Hormozdgān is fought. Ardashir I defeats and kills Artabanus V effectively ending the Parthian Empire.357 – Emperor Constantius II enters Rome for the first time to celebrate his victory over Magnus Magnentius.1192 – Assassination of Conrad of Montferrat (Conrad I), King of Jerusalem, in Tyre, two days after his title to the throne is confirmed by election. The killing is carried out by Hashdashashin.1253 – Nichiren, a Japanese Buddhist monk, propounds Namu Myōhō Renge Kyō for the very first time and declares it to be the essence of Buddhism, in effect founding Nichiren Buddhism.

USA<====>453673<====>USA - The United States of America (USA), commonly known as the United States (U.S.) or America, is a federal republic composed of 50 states, a federal district, five major self-governing territories, and various possessions. At 3.8 million square miles (9.8 million km2) and with over 325 million people, the United States is the world's third- or fourth-largest country by total area and the third-most populous. The capital is Washington, D.C., and the largest city by population is New York City. Forty-eight states and the capital's federal district are contiguous and located in North America between Canada and Mexico. The state of Alaska is in the northwest corner of North America, bordered by Canada to the east and across the Bering Strait from Russia to the west.

Handling input data

- Ignore tense, and gender of words
  - “He” and “She” should be different unigrams
  - “was” and “is” should be different unigrams
- Do not distinct the plural words
  - “name” and “names” should be considered as different unigrams
- Hyphenated words
  - “well-described” should be considered as 1 unigram
- Convert all of the upper cases to the lower case first, if needed
  - “Ngram” and “ngram” should be considered as 1 unigram
- Do not eliminate stop words (the most common words in a language)
  - Include “a” and “the” in your unigram profiles
Output data format

• Profile 1
  Ngram-A NEWLINE
  Ngram-B NEWLINE
  Ngram-C NEWLINE

• Profile 2
  Document-ID TAB ngram TAB frequency NEWLINE

• Profile 3
  Document-ID TAB ngram TAB frequency NEWLINE

Topics

• MapReduce Design Pattern IV. Join Patterns
Join Patterns

- Data is **all over** the place
- “Joins” allow users to create a smaller reference set or filter out or select dataset to discover interesting relationships across datasets
- Joining a terabyte of data onto another terabyte dataset could require up to **two terabytes of bandwidth!**
  - That’s before any actual join logic can be done!

1. Reduce Side Join Pattern
2. Replicated Join Pattern
3. Composite Join Pattern
4. Cartesian Product Pattern
MapReduce Design Patterns IV: Join Patterns

2. Replicated Join

Replicated Join

• Special type of join operation between **one large and (many) small data set(s)** that can be performed on the map-side

• Mapper
  • Reads all files from the distributed cache during the setup phase
    • Sorting them in to in-memory lookup tables
  • Performs mapper process
    • Joining data
  • If the foreign key is not found in the in-memory structure?
    • The record is either omitted or output (based on the join type)

• **No combiner/partitioner/reducer needed**
### Structure of the replicated join pattern

![Diagram showing the replicated join pattern](image)

### Hadoop DistributedCache

- Provided by the Hadoop MapReduce Framework
- Caches **read only text files, archives, jar files etc.**
- Once a file is cached for a job using Distributed cache
  - Data will be available on **each data node** where map/reduce tasks are running
Working with DistributedCache

• Make sure
  • Your file is available and accessible via http:// or hdfs://

• Setup the application’s JobConf in your Driver class

  DistributeCache.addFileToClasspath(new Path("/usr/datafile/XYZ"))

Size of DistributedCache in Hadoop

• Size
  • Default size of the Hadoop distributed cache is 10GB
  • Configurable in mapred-site.xml

• Data consistency
  • Hadoop Distributed Cache tracks the modification of timestamps of the cache file

• Overhead
  • Object serialization
Using **DistributedCache** for replicated join

- **A small file** is pushed to **all map tasks** using **DistributedCache**
- **Useful for join between a small set and a large set of data**
  - e.g. user information vs. transaction records, user information vs. comment history
- **Mapper Code**
  - **Setup phase**
    - User data is read from the DistributedCache and stored in memory
    - (userID, records) pairs are stored in a HashMap for data retrieval during the map process
  - **Map phase**
    - For each input record (from the large dataset), the user information is retrieved from the HashMap
    - Assemble a joined record

---

**Mapper Code**

- Join (Comment, user, on userID)

```java
public static class ReplicatedJoinMapper extends Mapper<Object, Text, Text, Text> {
  private static final Text EMPTY_TEXT = new Text("");
  private HashMap<String, String> userIdToInfo = new HashMap<String, String>();
  private Text outValue = new Text();
  private String joinType = null;

  public void setup(Context context) throws IOException, InterruptedException {
    Path[] files = DistributedCache.getLocalCacheFiles(context.getConfiguration());
  }
```

[1/3]
Mapper Code

- Join (Comment, user, on userID)

```java
// Read all files in the DistributedCache
for (Path p : files) {
    BufferedReader rdr = new BufferedReader(new InputStreamReader(new GZIPInputStream(new FileInputStream(new File(p.toString())))));
    String line = null;
    // For each record in the user file
    while ((line = rdr.readLine()) != null) {
        // Get the user ID for this record
        Map<String, String> parsed = transformXmlToMap(line);
        String userId = parsed.get("Id");
        // Map the user ID to the record
        userIdToInfo.put(userId, line);
    }

    // Get the join type from the configuration
    joinType = context.getConfiguration().get("join.type");
}
```

```java
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    Map<String, String> parsed = transformXmlToMap(value.toString());
    String userId = parsed.get("UserId");
    String userInformation = userIdToInfo.get(userId);
    // If the user information is not null, then output
    if (userInformation != null) {
        outvalue.set(userInformation);
        context.write(value, outvalue);
    } else if (joinType.equalsIgnoreCase("leftouter")) {
        // If we are doing a left outer join,
        // output the record with an empty value
        context.write(value, EMPTY_TEXT);
    }
}
```
MapReduce Design Patterns IV: Join Patterns

3. Composite Join

Composite Join

- Joins very large datasets together
  - And if the datasets are sorted by foreign key
- No shuffle and sort needed
- Each input dataset must be partitioned and sorted in a specific way and divided into the same number of partitions
Structure of the composite join pattern

Dataset A
Foreign keys

- hash(fk) % 5 = 0
  - Adam
  - James

- hash(fk) % 5 = 1
  - Bradley
  - Stella
  - William

- hash(fk) % 5 = 2
  - Andrew
  - Donald
  - Peter
  - Wade

- hash(fk) % 5 = 3
  - Chris
  - Denis

- hash(fk) % 5 = 4
  - Frank
  - Fred
  - Nicholas

Dataset B
Foreign keys

- Adam
- James
- Bradley
- James

- Bradley
- Donald
- Chris
- Denis
- Frank
- Fred
- Nicholas

The number of output part files is equivalent to the number of map tasks
Joining process (Inner Join) - Within a group

Joining process (Inner Join) - Between groups

Hash function $h(fk) = fk \mod 10$

There is no overlapping range

Between groups.

No computation across the sub-tables will be needed.
Joining process (Inner Join)

For N items in the left dataset and M items in the right dataset
Naive approach (without sorting and exact matching partitioning) takes O(NM)
With the composite join, it takes O(N)

<table>
<thead>
<tr>
<th>UserID</th>
<th>Credit Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Master</td>
</tr>
<tr>
<td>13</td>
<td>Master</td>
</tr>
<tr>
<td>14</td>
<td>Visa</td>
</tr>
<tr>
<td>15</td>
<td>Visa</td>
</tr>
<tr>
<td>15</td>
<td>Discover</td>
</tr>
<tr>
<td>17</td>
<td>Visa</td>
</tr>
<tr>
<td>17</td>
<td>Discover</td>
</tr>
<tr>
<td>20</td>
<td>Visa</td>
</tr>
<tr>
<td>20</td>
<td>American Express</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>Payments</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>FC Utility</td>
</tr>
<tr>
<td>12</td>
<td>FC Utility</td>
</tr>
<tr>
<td>13</td>
<td>FC Utility</td>
</tr>
<tr>
<td>13</td>
<td>Coloradoan</td>
</tr>
<tr>
<td>13</td>
<td>Netflix</td>
</tr>
<tr>
<td>13</td>
<td>Xcel</td>
</tr>
<tr>
<td>13</td>
<td>Honda Finance</td>
</tr>
<tr>
<td>14</td>
<td>Xcel</td>
</tr>
<tr>
<td>19</td>
<td>FC Utility</td>
</tr>
<tr>
<td>20</td>
<td>Xcel</td>
</tr>
</tbody>
</table>

Example: Composite user comment join

- Preconditions
  - User and comment datasets should be preprocessed by MapReduce
  - CompositeInputFormat will be used
  - Output should use the TextOutputFormat
    - Key: userID
    - Value: userXML or comment XML
    - Hadoop KeyValueTextOutputFormat can parse these information

- Note: Since CompositeInputFormat uses Text objects as the key for comparisons
  - Align your keys as Text objects (rather than LongWritable)
  - e.g. “12345” comes before “2”
Driver Code [1/2]

```java
public static void main(String[] args) throws Exception {
    Path userPath = new Path(args[0]);
    Path commentPath = new Path(args[1]);
    Path outputDir = new Path(args[2]);
    String joinType = args[3];
    JobConf conf = new JobConf("CompositeJoin");
    conf.setJarByClass(CompositeJoinDriver.class);
    conf.setMapperClass(CompositeMapper.class);
    conf.setNumReduceTasks(0);
    // Set the input format class to a CompositeInputFormat class.
    // The CompositeInputFormat will parse all of our input files
    // and output
    // records to our mapper.
    conf.setInputFormat(CompositeInputFormat.class);
    conf.setInputFormat(CompositeInputFormat.compose(joinType,
        KeyValueTextInputFormat.class, userPath, commentPath));
    TextOutputFormat.setOutputPath(conf, outputDir);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    RunningJob job = JobClient.runJob(conf);
    while (!job.isComplete()) {
        Thread.sleep(1000);
    }
    System.exit(job.isSuccessful() ? 0 : 1);
}
```

This input format scans through the left dataset and Identifies matching in the right dataset. It generates input value, (item from The left dataset, item from the right Dataset)

Driver Code [1/2]

```java
// The composite input format join expression
// will set how the records
// are going to be read in, and in what input format.
conf.set("mapred.join.expr",
    CompositeInputFormat.compose(joinType,
        KeyValueTextInputFormat.class, userPath, commentPath));
TextOutputFormat.setOutputPath(conf, outputDir);
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(Text.class);
```
Mapper Code

```java
public static class CompositeMapper extends MapReduceBase
        implements Mapper < Text, TupleWritable, Text, Text > {
    public void map(Text key, TupleWritable value, OutputCollector < Text, Text > output, Reporter reporter)
        throws IOException {
        // Get the first two elements in the tuple and output them
        output.collect((Text)value.get(0), (Text)value.get(1));
    }
}
```

MapReduce Design Patterns IV: Join Patterns

4. Cartesian Product
Cartesian Product

- Pairs every record from multiple inputs

- Applicability
  - You want to analyze relationships between all pairs of records
  - e.g. Correlation analysis
Performance Analysis

- Cartesian product requires a massive data space
  - To compute a Cartesian product between N records and M records
  - N x M data points are created
    - Requires many Map slots and a very long time

Example: Cartesian Product

- Comments comparison
  - Inspects a pair of comments and determines how similar they are
  - If they are similar enough, the pair is stored in the output file

- During the setup phase
  - getInputSplits
    - Creates the cross-product of the input splits and sets into a list of CompositeInputSplits
HDFS Block Size vs. Hadoop Input Split

- **Block Size**
  - Physical Location where the data been stored (default size of the HDFS block is 128MB)
  - All blocks of the file are of the same size except the last block
  - HDFS distributes these blocks (chucks) across the cluster
  - Achieves more parallelism and fault tolerance (with replication)

- **Input Split**
  - Logical representation of block
    - More or less than a Block Size
  - Used during data processing in MapReduce program or other processing techniques
  - InputSplit does not contain actual data but a reference to the data
  - Split acts as a broker between the block and mapper
    - During MapReduce execution, Hadoop scans through the blocks and create InputSplits and assign them to individual mapper

Example of Using Input Split [1/2]

- **1.28GB file**
  - Divided into 10 blocks (128MB)
  - `InputFormat.getSplits()` is responsible for generating the input splits
    - Default: creates one input split for each HDFS block

- **Case 1. If input split is not specified**
  - HDFS block size will be the split size
  - 10 mappers are initialized to load the file
  - Each mapper loads one block

- **Case 2. If the start and end positions of the records are not in the same block?**
  - `InputSplit` provides the Start and End positions (offsets)
  - Mapper is going to load the block of data according to start and end offset values
Example of Using Input Split [2/2]

- **1.28GB file**
  - Divided into 10 blocks (128MB)
  - `InputFormat.getSplits()` is responsible for generating the input splits
    - Default: creates one input split for each HDFS block

- Case 3. If the specified split size was not correct
  - The whole file will form one input split and processed by one map
  - Requires processing very large file

- Case 4. If your resource is limited and you want to limit the number of maps?
  - Specify the Split size as 256MB
  - Only 5 maps will be executed

From Input Splits to Mapper tasks

- Hadoop’s Interface `InputFormat<K,V>`
  - `RecordReader<K,V>`
  - `InputSplit[]`

- Validate the input-specification of the job
- Split-up the input file(s) into logical `InputSplits`
  - Assign to an individual Mapper
- Provide the `RecordReader` implementation to the user to glean input records from the logical `InputSplit` for processing Mapper
RecordReader<\(K, V\)>

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>close() - Close this InputSplit to future operations.</td>
</tr>
<tr>
<td>(K)</td>
<td>createKey() - Create an object of the appropriate type to be used as a key.</td>
</tr>
<tr>
<td>(V)</td>
<td>createValue() - Create an object of the appropriate type to be used as a value.</td>
</tr>
<tr>
<td>long</td>
<td>getPos() - Returns the current position in the input.</td>
</tr>
<tr>
<td>float</td>
<td>getProgress() - How much of the input has the RecordReader consumed i.e.</td>
</tr>
<tr>
<td>boolean</td>
<td>next(K key, V value) - Reads the next key/value pair from the input for processing.</td>
</tr>
</tbody>
</table>

InputSplit

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>getLength() - Get the total number of bytes in the data of the InputSplit.</td>
</tr>
<tr>
<td>String[]</td>
<td>getLocations() - Get the list of hostnames where the input split is located.</td>
</tr>
</tbody>
</table>
Input Format Code

```java
public static class CartesianInputFormat extends FileInputFormat {
    public static final String LEFT_INPUT_FORMAT = "cart.left.inputformat";
    public static final String LEFT_INPUT_PATH = "cart.left.path";
    public static final String RIGHT_INPUT_FORMAT = "cart.right.inputformat";
    public static final String RIGHT_INPUT_PATH = "cart.right.path";

    public static void setLeftInputInfo(JobConf job, Class <? extends FileInputFormat> inputFormat, String inputPath) {
        job.set(LEFT_INPUT_FORMAT, inputFormat.getCanonicalName());
        job.set(LEFT_INPUT_PATH, inputPath);
    }
    
    public static void setRightInputInfo(JobConf job, Class <? extends FileInputFormat> inputFormat, String inputPath) {
        job.set(RIGHT_INPUT_FORMAT, inputFormat.getCanonicalName());
        job.set(RIGHT_INPUT_PATH, inputPath);
    }

    public InputSplit[] getSplits(JobConf conf, int numSplits) throws IOException {
        // Get the input splits from both the left and right datasets
        InputSplit[] leftSplits = getInputSplits(conf, conf.get(LEFT_INPUT_FORMAT),
                                               conf.get(LEFT_INPUT_PATH), numSplits);
        InputSplit[] rightSplits = getInputSplits(conf, conf.get(RIGHT_INPUT_FORMAT),
                                                  conf.get(RIGHT_INPUT_PATH), numSplits);
        // Create our CompositeInputSplits, size equal to
        // left.length * right.length
        CompositeInputSplit[] returnSplits =
            new CompositeInputSplit[leftSplits.length * rightSplits.length];
        }
```

```java
int i = 0;
// For each of the left input splits
for (InputSplit left : leftSplits) {
    // For each of the right input splits
    for (InputSplit right : rightSplits) {
        // Create a new composite input split composing of the two
        returnSplits[i] = new CompositeInputSplit(2);
        returnSplits[i].add(left);
        returnSplits[i].add(right);
        ++i;
    }
}

// Return the composite splits
LOG.info("Total splits to process: " + returnSplits.length);
return returnSplits;
```

```java
public RecordReader getRecordReader(InputSplit split, JobConf conf, Reporter reporter) throws IOException {
    // Create a new instance of the Cartesian record reader
    return new CartesianRecordReader((CompositeInputSplit)split, conf, reporter);
}
```
private InputSplit[] getInputSplits(JobConf conf, String inputFormatClass, String inputPath, int numSplits) throws ClassNotFoundException, IOException {
    // Create a new instance of the input format
    FileInputFormat inputFormat =
        (FileInputFormat)ReflectionUtils.newInstance(Class.forName(inputFormatClass), conf);
    // Set the input path for the left data set
    inputFormat.setInputPaths(conf, inputPath);
    // Get the left input splits
    return inputFormat.getSplits(conf, numSplits);
}

public static class CartesianRecordReader < K1, V1, K2, V2 > implements RecordReader < Text, Text > {
    // Record readers to get key value pairs
    private RecordReader leftRR = null, rightRR = null;
    // Store configuration to re-create the right record reader
    private FileInputFormat rightFIF;
    private JobConf rightConf;
    private InputSplit rightIS;
    private Reporter rightReporter;
    // Helper variables
    private K1 lkey;
    private V1 lvalue;
    private K2 rkey;
    private V2 rvalue;
    private boolean goToNextLeft = true, alldone = false;
public CartesianRecordReader(CompositeInputSplit split, JobConf conf, Reporter reporter) throws IOException {
    this.rightConf = conf;
    this.rightIS = split.get(1);
    this.rightReporter = reporter;
    // Create left record reader
    FileInputFormat leftFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(conf.get(CartesianInputFormat.LEFT_INPUT_FORMAT)), conf);
    leftRR = leftFIF.getRecordReader(split.get(0), conf, reporter);

    // Create right record reader
    rightFIF = (FileInputFormat) ReflectionUtils.newInstance(Class.forName(conf.get(CartesianInputFormat.RIGHT_INPUT_FORMAT)), conf);
    rightRR = rightFIF.getRecordReader(rightIS, rightConf, rightReporter);
    // Create key value pairs for parsing
    lkey = (K1) this.leftRR.createKey();
    lvalue = (V1) this.leftRR.createValue();
    rkey = (K2) this.rightRR.createKey();
    rvalue = (V2) this.rightRR.createValue();
}
public boolean next(Text key, Text value) throws IOException {
    do {
        // If we are to go to the next left key/value pair
        if (goToNextLeft) {
            // Read the next key value pair, false means no more pairs
            if (!leftRR.next(lkey, lvalue)) {
                // If no more, then this task is nearly finished
                alldone = true; break;
            } else {
                // If we aren't done, set the value to the key and set
                // our flags
                key.set(lvalue.toString());
                goToNextLeft = alldone = false;
                // Reset the right record reader
                this.rightRR = this.rightFIF.getRecordReader(this.rightIS, this.rightConf, this.rightReporter);
            }
        }
    }
    // Read the next key value pair from the right data set
    if (rightRR.next(rkey, rvalue)) {
        // If success, set the value
        value.set(rvalue.toString());
    } else {
        // Otherwise, this right data set is complete
        // and we should go to the next left pair
        goToNextLeft = true;
    }
    // This loop will continue if we finished reading key/value
    // pairs from the right data set
    } while (goToNextLeft);
    // Return true if a key/value pair was read, false otherwise
    return !alldone;
}
Driver Code

```java
public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {
    // Configure the join type
    JobConf conf = new JobConf("Cartesian Product");
    conf.setJarByClass(CartesianProduct.class);
    conf.setMapperClass(CartesianMapper.class);
    conf.setNumReduceTasks(0);
    conf.setInputFormat(CartesianInputFormat.class);
    // Configure the input format
    CartesianInputFormat.setLeftInputInfo(conf, TextInputFormat.class, args[0]);
    CartesianInputFormat.setRightInputInfo(conf, TextInputFormat.class, args[0]);
    TextOutputFormat.setOutputPath(conf, new Path(args[1]));
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(Text.class);
    RunningJob job = JobClient.runJob(conf);
    while (!job.isComplete()) {
        Thread.sleep(1000);
    }
    System.exit(job.isSuccessful() ? 0 : 1);
}
```
Mapper Code (comparing comments)

```java
public static class CartesianMapper extends MapReduceBase implements Mapper<Text, Text, Text, Text> {
    private Text outkey = new Text();
    public void map(Text key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
        // If the two comments are not equal
        if (!key.toString().equals(value.toString())) {
            String[] leftTokens = key.toString().split("\s");
            String[] rightTokens = value.toString().split("\s");
            HashSet<String> leftSet = new HashSet<>(Arrays.asList(leftTokens));
            HashSet<String> rightSet = new HashSet<>(Arrays.asList(rightTokens));
            int sameWordCount = 0;
            StringBuilder words = new StringBuilder();

            for (String s : leftSet) {
                if (rightSet.contains(s)) {
                    words.append(s + ",");
                    ++sameWordCount;
                }
            }
            // If there are at least three words, output
            if (sameWordCount > 2) {
                outkey.set(words + "\t" + key);
                output.collect(outkey, value);
            }
        }
    }
}
```
Questions?