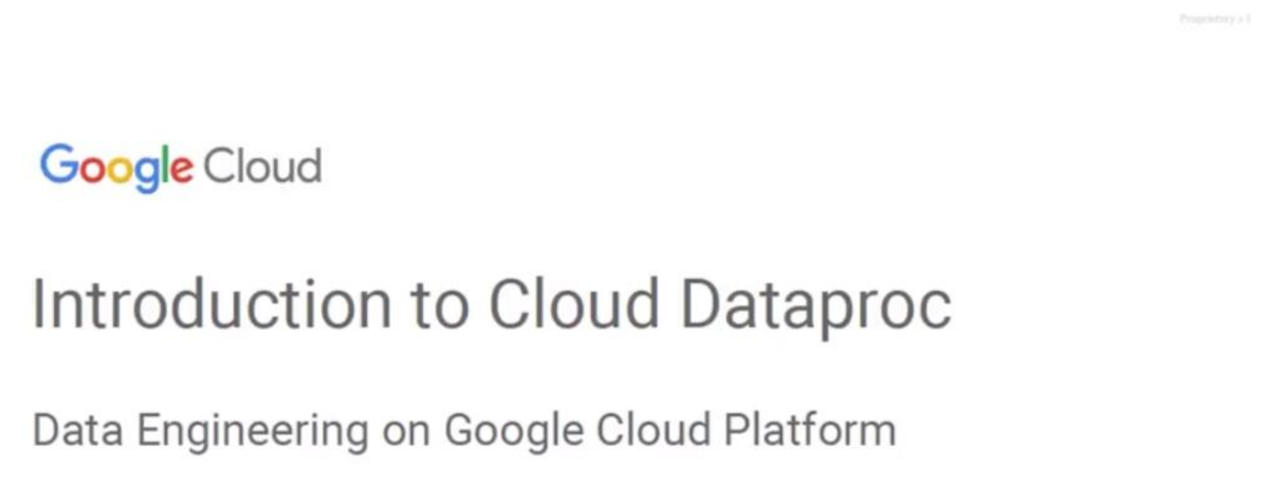
LEVERAGING UNSTRUCTURED DATA WITH CLOUD DATAPROC ON GOOGLE CLOUD PLATFORM

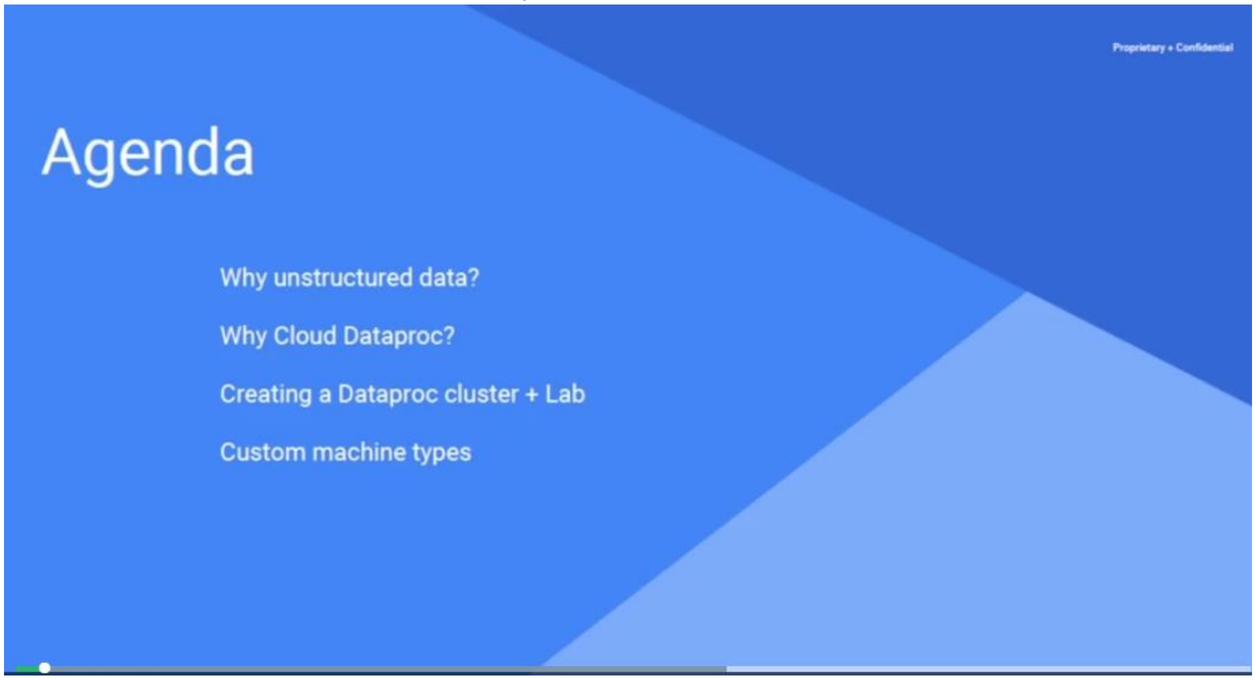


Google Cloud

Introduction to Cloud Dataproc

Data Engineering on Google Cloud Platform

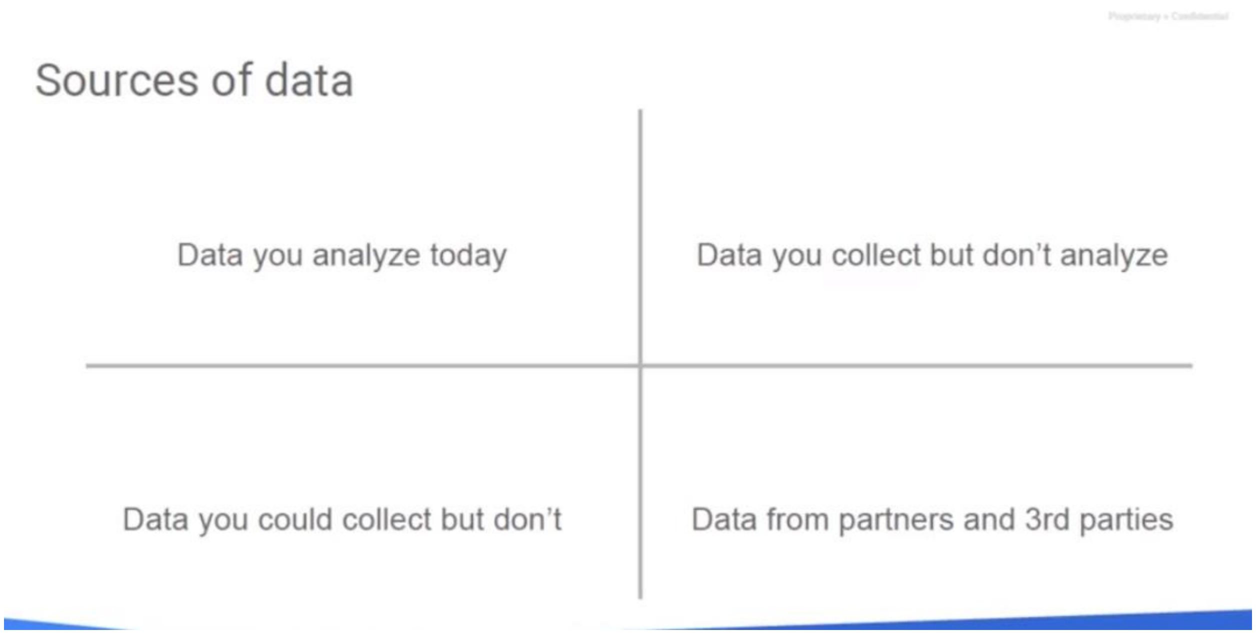
Introduction to Cloud Dataproc


Proprietary + Confidential

Agenda

- Why unstructured data?
- Why Cloud Dataproc?
- Creating a Dataproc cluster + Lab
- Custom machine types

Welcome to Google Cloud platform, this is Grant Morell. Today we're going to look at Cloud Dataproc which is Google's managed Hadoop infrastructure.

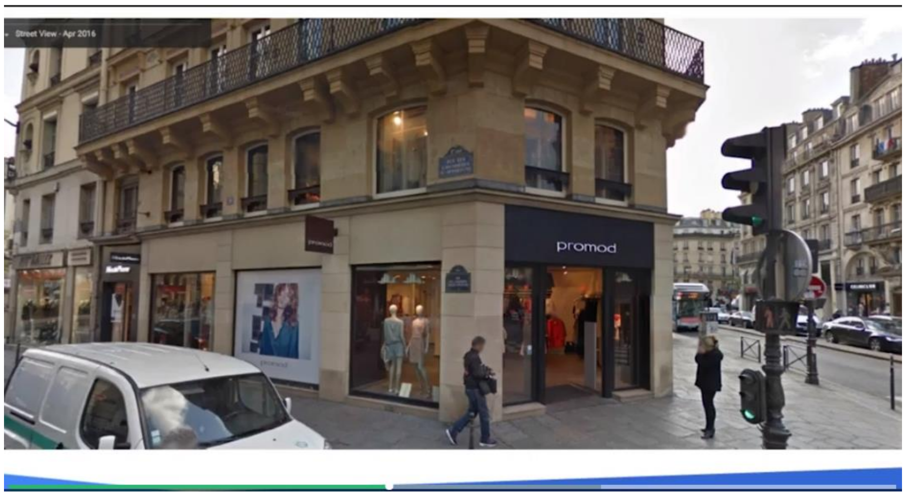

Proprietary + Confidential

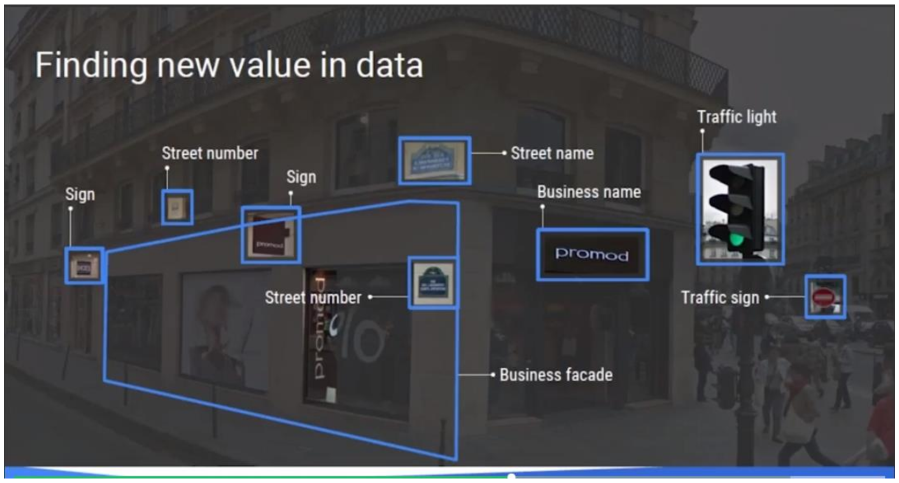
Sources of data

Data you analyze today	Data you collect but don't analyze
Data you could collect but don't	Data from partners and 3rd parties

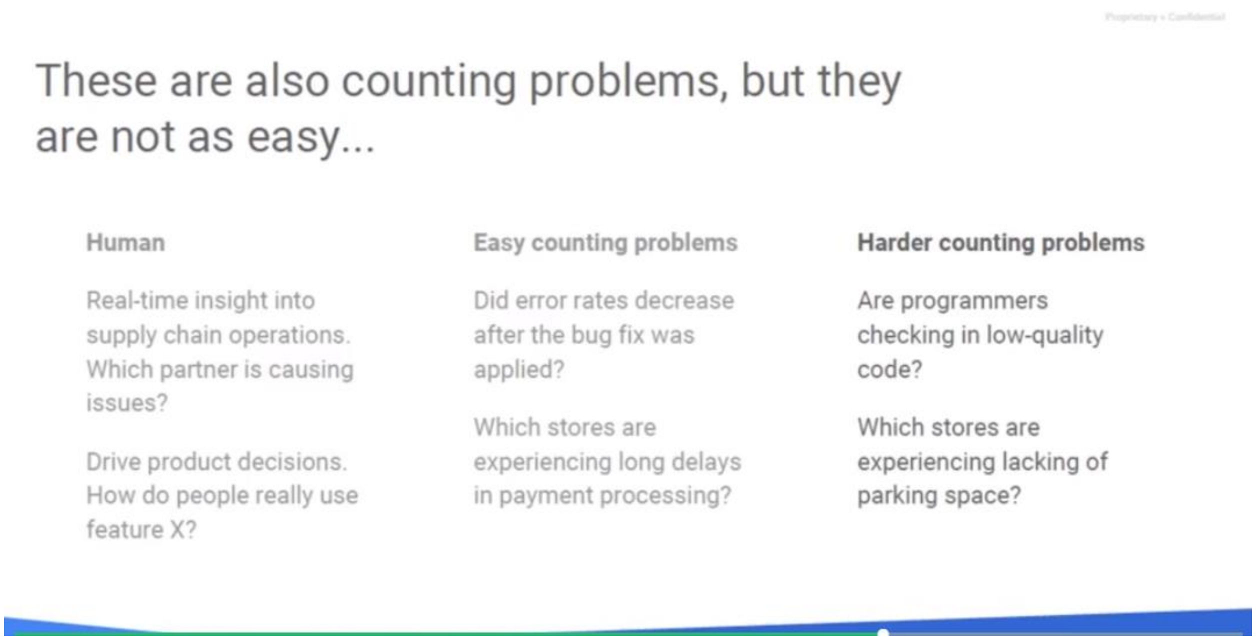
Why Unstructured Data?

Our agenda is to talk about unstructured data, why we would use Cloud Dataproc, how to create a Cloud Dataproc cluster, and some customizations to the computing nodes of the cluster. For the last several years Google has been creating Google Street View by driving specially out of fitted cars with lots of cameras on top around cities and creating street view of those cities. When they initially did this, they were doing it for use with Google Maps and Google Street View, but since the data set has been created now and actually is being updated rather frequently, it could be applied to so many other things beyond what's happening just within Google with it. Look at this image. What can we tell? Well we have street signs. We have street names. We have street numbers. We have business names. We have traffic signals. We have business facades and frontage. We have a one-time snapshot as to how many people are in front of it, and presumably we've got time of day and the data was taken sort of thing. Now I've heard of organizations in New York that are actually using street view to document street parking spaces, disabled parking spaces, curbs, curbs that support wheelchairs, sidewalks, bus stops, windows and even trees, yeah trees. By combining satellite views they can identify trees from overhead, so using Google maps. Then they move down the street view, because from the map they can get the latitude and longitude coordinates and then use the vision API to identify the type of that tree. That way they can actually look at the types of trees and know the population of different types of trees around a city.





The alternative would be to have somebody walk around the city and actually document them all. Now, with something like Google Street View, the streets can be virtually driven and business opportunities can be looked for. We could look at windows you know, going beyond just you know, what we've identified so far, we can look at windows, we can look at railings. We could look at stonework, we could look at painting. You can look at grass mowing that needs be done. There is a lot of data that could actually be determined by just processing an image. Now some big data involves just pure counting. Data error rates go down when a bug fix was applied. Something that a computer could count. You know retail store, calculating delays in payment processing to identify stores experiencing problems with their credit card machines and things like that. Stuff that literally is purely quantitative. Is has a quantity related to it. We need to collect that quantity and we need to look for thresholds.

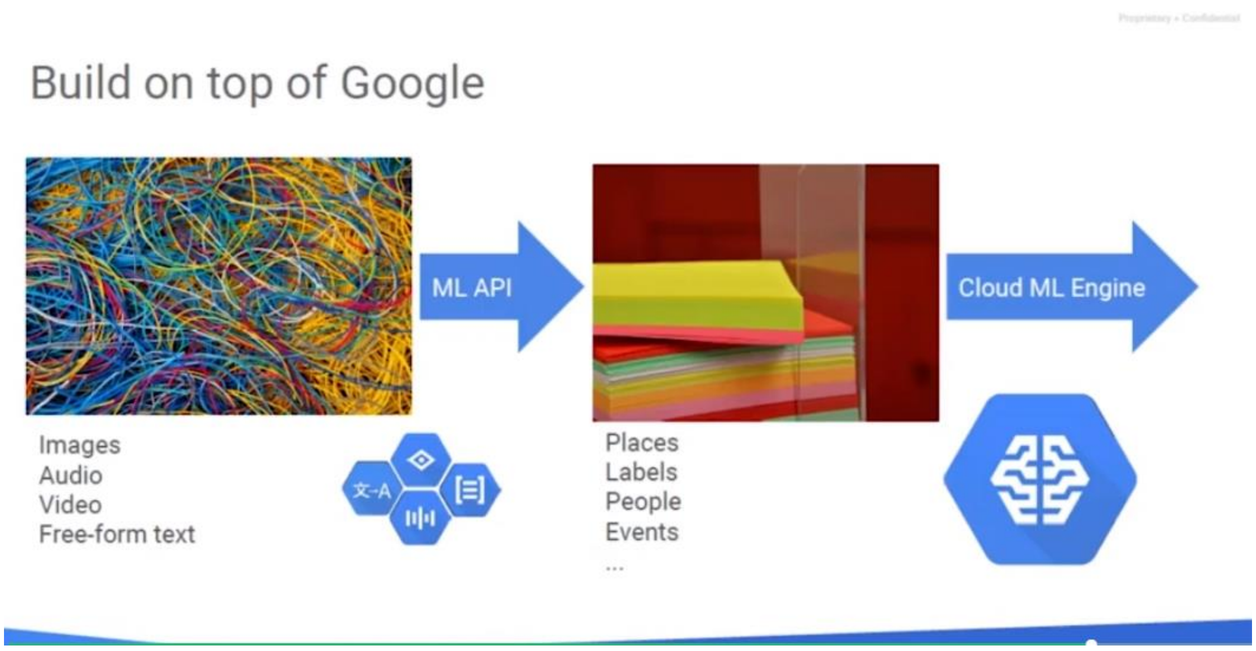


These are also counting problems, but they are not as easy...

Human	Easy counting problems	**Harder counting problems**
Real-time insight into supply chain operations. Which partner is causing issues?	Did error rates decrease after the bug fix was applied?	Are programmers checking in low-quality code?
Drive product decisions. How do people really use feature X?	Which stores are experiencing long delays in payment processing?	Which stores are experiencing lacking of parking space?

But, there are then those problems that require a lot of counting plus the human insight into looking at the data. If I was to ask a question, OK my programmers are checking "code" but are there any specific programmers checking in low quality code. Well how would we know? Obviously, we would need to create some sort of metric or we would have to set aside some sort of metric based on the data we have. We just haven't processed it yet. Now we might say well which of our stores is lacking parking spaces. How would we know? Well that would be video footage of our parking lot and looking and identifying when our parking space is in use, when are they not and is there ever a time of day where we exceed the parking spaces that we actually have allocated and people are waiting or people are leaving?

Unstructured data accounts for 90% of enterprise data*



What Data do Enterprises Analyze?

If you look at the kinds of data that you have in your company. What kind of sources of data do you have? And what kind of data do you actually analyze? You might think of our data that you have and that you analyze, data that you don't have you wish you had, data that you have but for whatever reason you don't analyze and you have data that is probably present in a third party that you could go and acquire.

Let's key off on one aspect of it. The aspect of data that you have that you don't analyze. Why is that? Why would you have data you're storing it, you're keeping it around. You have it it's accessible. But, for whatever reason you don't analyze it. What are some of those reasons? Maybe, the amount of data is too large for you to analyze, maybe it's volume. Maybe it's veracity, maybe you're not sure about how good that data is. Is it worth analyzing? Anything else? Well, perhaps it's the fact that it's coming so fast that trying to keep up with that stream of data. The velocity of that data could be too high. Well, that's one reason. But, I think that in many cases, it's not just about volume or veracity or about velocity. It's something more fundamental than that. It has to do with the kinds of tools that we're familiar with. We're very familiar with analyzing structured data. If it's data in a database, it's a relational data no problem at all. But, if it's data that's not structured, lots of times we just leave it completely unanalyzed. I'm in Google professional services and I talk to customers quite a bit. And, I posed this question to them, what kind of data that you have that you don't analyze? And, they think about it and they talk about data that's too large you know that's not things that are arriving too fast. But, then I say, "well what do you do about your e-mails?" Surely your customers send you e-mails. What do you do about them? And they look at me like I'm crazy. Who analyzes e-mails? How about newsgroups? What do you do with newsgroups? How about photographs that your technicians take on the field? Nope. So, if you have inspections being carried out and you have photographs what do you do with those? This has to do with the tools that we have available to us; unstructured data, images, free form text.

It's genuinely hard. And so, what we're going to be looking at next in the data engineering course, is how to deal with unstructured data.

Even Google Skipped Unstructured Data

Even at Google we face this problem. There was a time that someone at Google had this great idea that we take cameras, put them on top of cars and drive them through every street in the world. Even more amazing to me someone higher up at Google thought that was a really good idea and they would put money into this. Photographs that were taken by those cars got surfaced in Google Maps. When you arrived at your destination, you knew what your destination looked like. It was collected and it is used by whoever is using Google maps. But that was it. There was no technology in place to do anything else with that data. We had all this imagery from all over the world and we're doing nothing with it. But then over time, deep learning came along. Our machine learning improved and we said, "Hey, how about we go back and look at all the imagery that we've collected over time from street view maps?" And we did. We went back to those images and we're looking at those we could say, "Oh, well, here is a street sign. Here is a street number. Here is a business. Here's a road." And we could actually then make it a feedback loop. We could look at these images that were collected by cars driving through and we could basically use them to enhance our maps. That's essentially what you're talking about. You may have unstructured data that you have laying around in your company that you're using maybe for one specific purpose. And usually the purpose is to surface to a human user because a human user can look at that unstructured data and they can make sense of it. But you don't have any automated programs that analyze that unstructured data. What do you want to look at is how we can use machine learning, the advances that have happened in machine learning, to analyze those images, those free-form text, etc. When we are looking at that the thing to realize is that you don't need to start from scratch. You don't have to build an image recognition model in order to take advantage of all the image data that you have. You can use pre-built models and essentially apply them. So those are what we will call the machine learning APIs. Whether it's analyzing free-form text or whether it's analyzing images, we will be able to take advantage of pre-built machine learning models. We'll apply those machine learning models to data that we have around, and we will extract out information that we can then use.

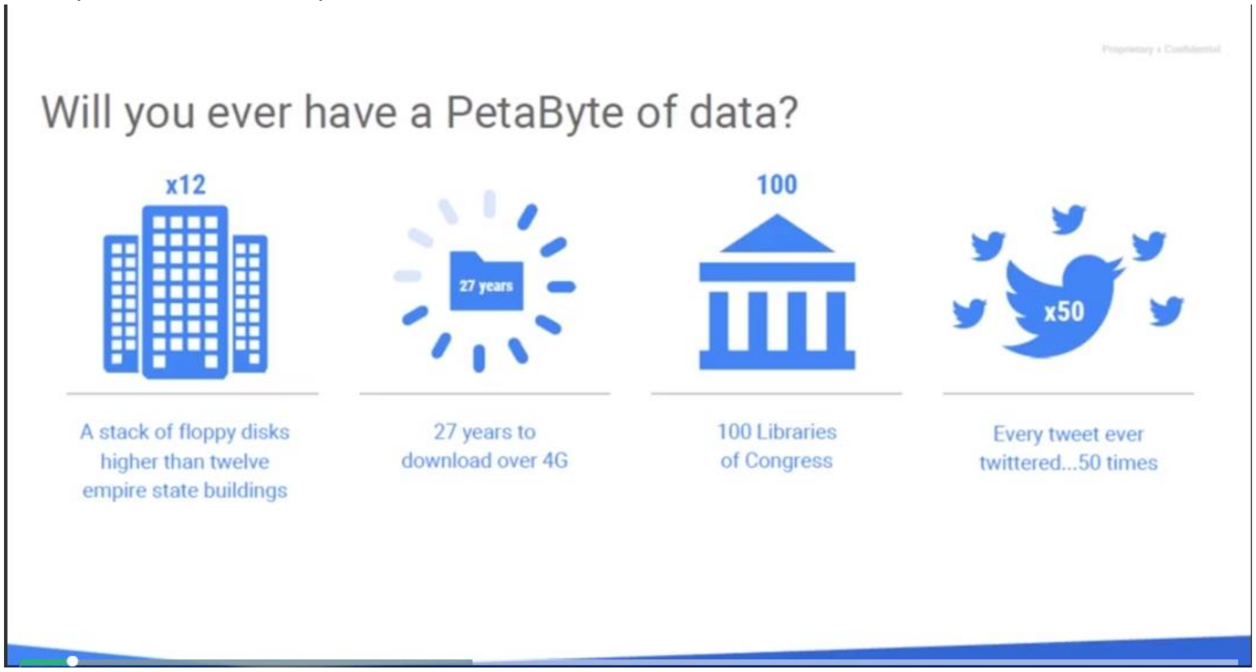
Considering Counting Problems

When you think about analytical tools and much of the analysis that we do, it is about counting. For example, MapReduce counts things. Is it more complex than that? Suppose you're trying to figure out delays in payment processing? Surely a delay in payment processing is not just counting. But, what is it that you actually do when you're trying to figure out delays in payment processing? You are going through every transaction, and for every transaction, you're basically figuring out how long it took you to basically make the payment. And you are taking that and adding it up over all the transactions. You are finding a mean of some kind. That's counting. You're adding, you're counting, now you're basically looking at how many transactions took longer than three days to pay. That's a counting problem. And this is what I would call it easy counting problem and MapReduce is full of easy counting problems. You go through a very large set of data and through that set of data, you basically are looking for specific things and you're counting the number of times they happen.

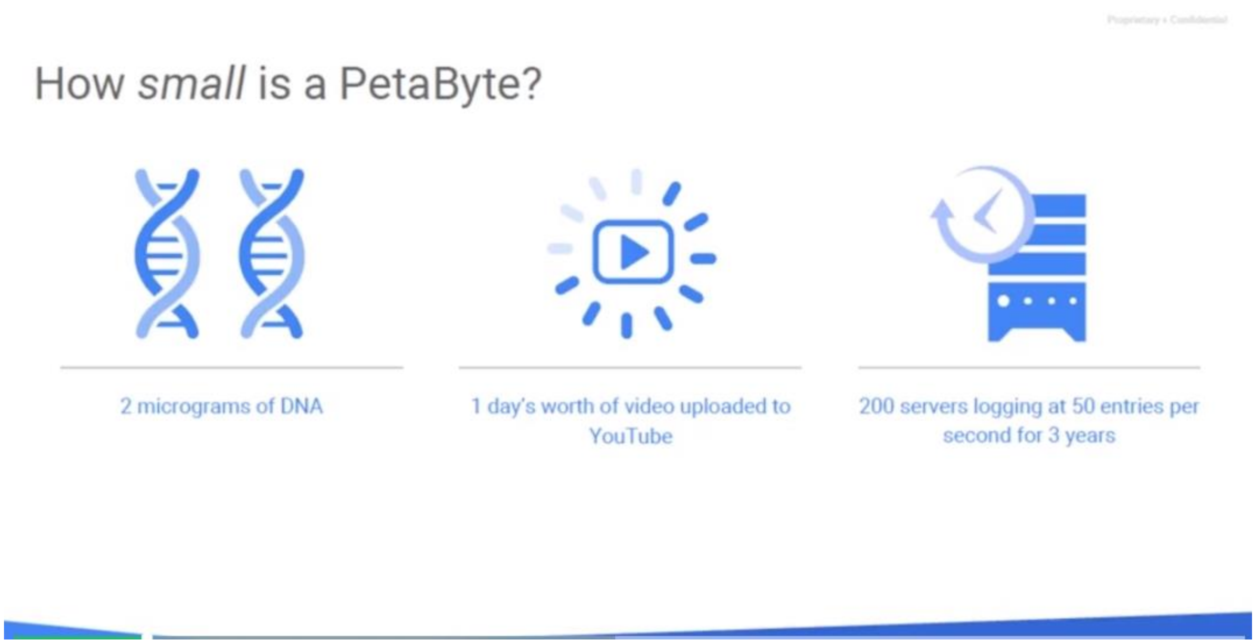
And as long as the things that you are looking for are easy to analyze, it becomes a counting problem.

But there are other kinds of problems that are a little bit harder. Suppose I were to ask you, not for delays in payment processing, but I were to ask you say, well, how often are your programmers checking in low quality code? That is also a counting problem. How often is someone doing something which is a counting problem. The thing that you are counting, low quality code, is extremely hard to just extract from data. Your data or the code. But how can you look at code and say, is this low quality or high quality – that is what makes it a harder counting problem. If the thing that you're trying to detect is harder, then it makes it a harder counting problem, but it's still a counting problem. Lots of analytical tools are about counting problems. But when we look at counting problems, depending on what it is that we count, they could be easy counting problems, or they could be much harder counting problems.

Why Cloud Dataproc?


Will you ever have a PetaByte of data?
x12
A stack of floppy disks higher than twelve empire state buildings
27 years
27 years to download over 4G
100
100 Libraries of Congress
x50
Every tweet ever twittered...50 times

Why Cloud Dataproc? Many users have been using Hadoop and the Hadoop file system as well as Pig, Hive and Spark for many years now running their own clusters and using them for processing large parallel jobs. The question is, will you ever have a petabyte of data?



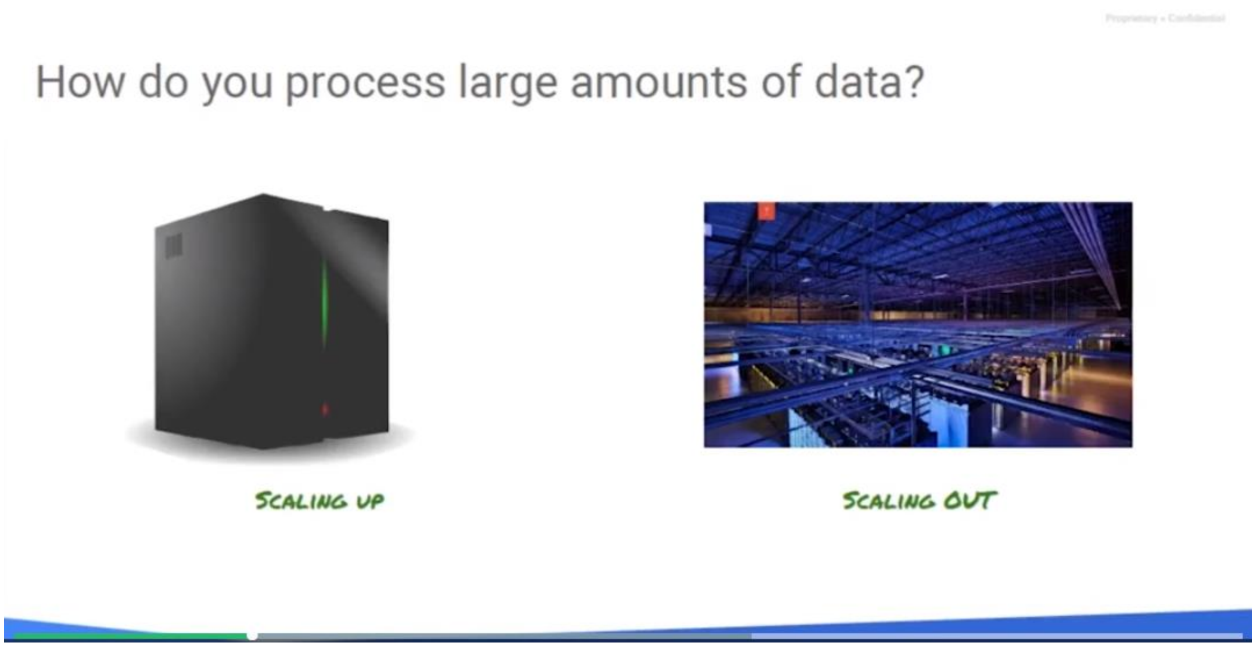
How *small* is a PetaByte?

2 micrograms of DNA

1 day's worth of video uploaded to YouTube

200 servers logging at 50 entries per second for 3 years

Let's look at the scale of that. It's a stack of floppy disks higher than 12 Empire State Buildings. It's 27 years of downloading data over 4G. It's a 100 US Libraries of Congress which maintain copies of all books published in the USA. And every tweet ever Twittered 50 times over. Well, it's probably closer to 49 times these days. It's been a few months since we probably produced this live. But a Petabyte has also two micrograms of DNA. One day's worth of videos uploaded into YouTube. And 200 servers logging 50 entries per second for three years or 600 servers making 50 entries per second for one year.

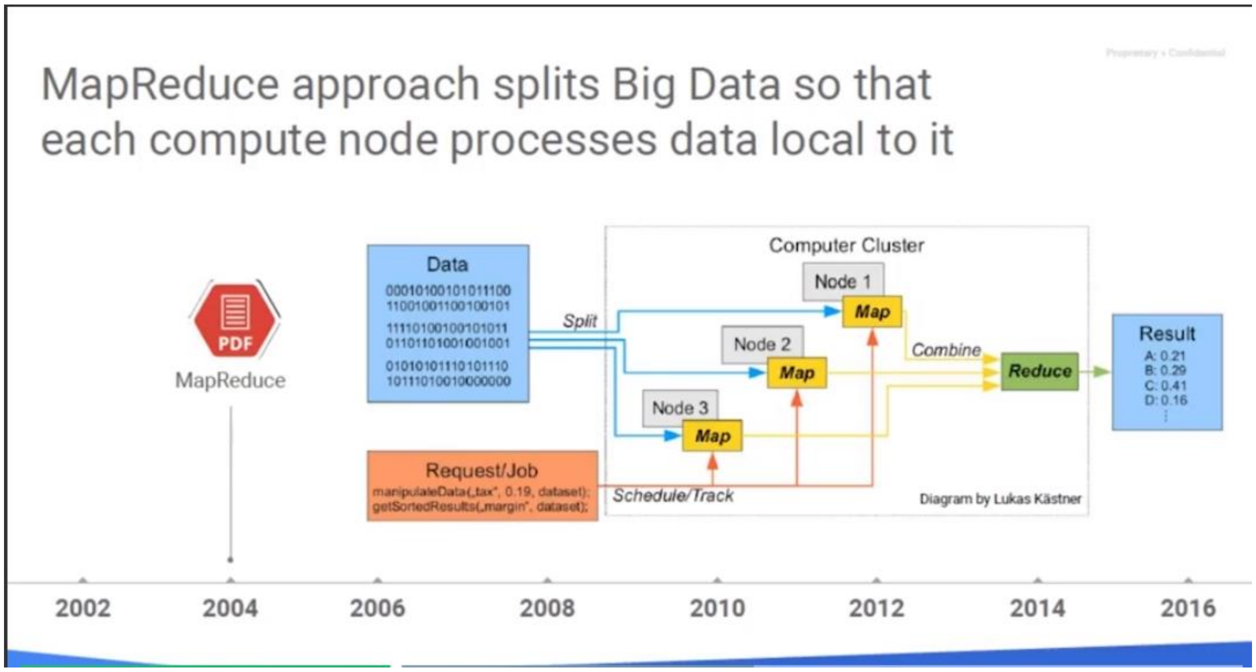


How do you process large amounts of data?

SCALING UP

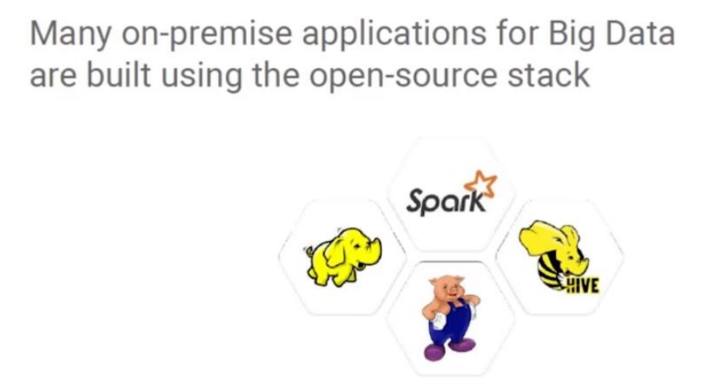
SCALING OUT

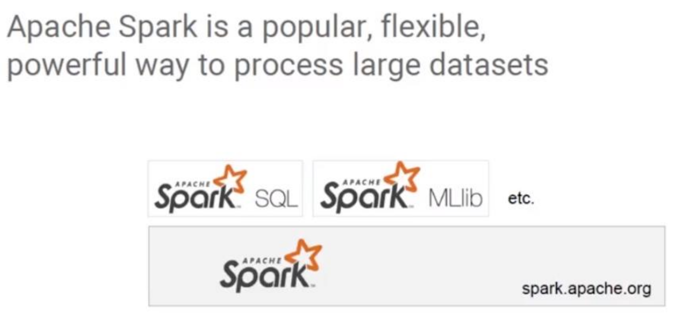
How do you process such large amounts of data? You can scale up by making a bigger machine but how much bigger can you make a single machine? Or you can scale out which lets you scale as large as the parallel nature of the problem you can handle by adding more computers to it. Let's say I have a job where I need to take 50,000 images that we have stored in a Google Cloud Storage bucket and I want to put, say, six different versions of that image, each with different dimensions, for say a web server, you know, clip art, that sort of thing. I want it at a bunch of different resolutions. Now we could try to make one big monolith or monolithic server to do this. Therefore, more CPU cores all tied up and, or make it faster in attempt to process more. But how big can we make it? We can only go so far. But what if we could take five hundred computers and have each one process a hundred images or 5,000 computers and each process 10 images or the absolute extreme, 50,000 computers each processing one image. Well, a lot of this was the problem that Google faced back in the 2000's because they had a massive parallel computing problem. How do they take copies of all the web pages on the internet and then break them down into the keywords so they can make up a search engine? They could build some large computer, buy expensive storage arrays and try to do it with expensive computing resources or they could invent their own parallel architecture to store and process this information. Well, which way did they choose? Well, they built their own parallel processing system and they called that map reduce as in we need to map the data to a model and then we need to reduce it. They also created a distributed file system called Google file system or GFS so that they could store all these files that make up web pages and not have to have one big system that holds those that could bottleneck.

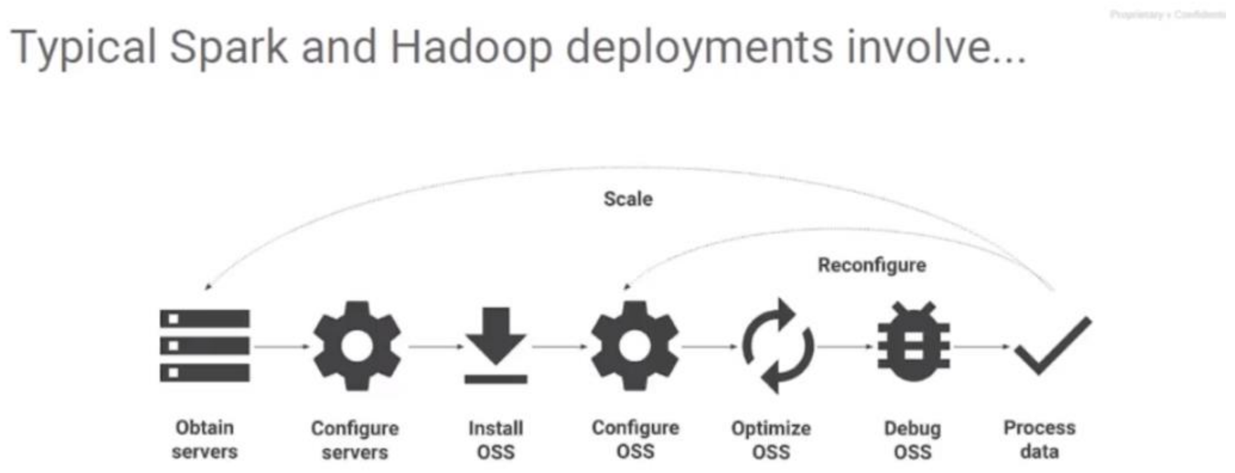


In the mid-2000's after doing all of this, using map reduce and Google file system, they released a couple of papers to the world that showed how they did it. And from that, we now have Apache Hadoop and the Hadoop file system, so thanks Google for that. Now the job of a map reduce is to split the data and map it to a structure and then reduce the structure to a usable result set, often putting it into a database or other structured format.

These days, many on-premise applications for big data are built using the open source stack based on Hadoop and the Hadoop file system. And then higher-level applications which abstract the Hadoop HDFS portion of it, like Pig for data transformation, Hive for data querying on a large scale, and Spark a general purpose open source cluster computing framework. Spark is really popular these days and has database components, machine learning libraries etc. I'd recommend doing some research on spark.apache.org if you've never heard a spark or haven't done too much with it.


Many on-premise applications for Big Data are built using the open-source stack


Apache Spark is a popular, flexible, powerful way to process large datasets


Typical Spark and Hadoop deployments involve...

I am going to set up an environment in your mind to let you think about it for a minute, about running a Hadoop cluster. Imagine a 100 node Hadoop cluster. So that would have 100 workers that basically process the job. It's not uncommon to find a cluster this scale in a medium to large business, and then use it to run parallel computing jobs. Maybe it's a bank balancing their book each night. Deposits vs loans, ensuring they're meeting the regulatory requirements. Or maybe it's a large retailer comparing sales per store, there popular item stores, producing store item pairings, processing all the receipts from the last day, week, month, year etc. Or looking for new insight into their existing data. If they use their Hadoop cluster 24 hours a day, seven days a week. Then they probably need a bigger cluster. They need to add more nodes because who knows? They're basically bottlenecked. Now, if they only use their cluster 10 hours a day then they're wasting 14 hours a day of computing power. They should be taking that into account with our ongoing computing costs. It's like an airline seat that once the door closes, that seat's lost that there's no revenue that can be gained from that seat. Or a hotel room that goes empty for a night. Well, if wasn't used that night, that's lost revenue. And so basically if you're not using a cluster 100 percent of the time you're wasting computing resources and if you are using the cluster 24 hours a day, seven days a week you need a bigger cluster. Either way the on-premise Hadoop isn't going to operate as efficiently and hopefully as cost efficiently as utilizing Google Cloud Platform with Google Dataproc.

Cluster Provisioning Considerations

What is the problem with running everything on one cluster?

Think about it this way. What happens if you don't have enough computing power on that cluster for the number of jobs that you want to run?

Or conversely, what happens if you have way too much computing power for the active processes that you want to run?

And you realize that the same cluster sometimes may be too much and at other times it may be too little.

What you really need is to avoid over-provisioning a cluster or under-provisioning a cluster. You need to dynamically provision that cluster.

Imagine Your Cluster Provisioning

Imagine that you are using a MapReduce cluster, and you realized that you've under-provisioned it, and you need to go get four or five more machines. What do you have to do when you get those five machines? Can you just add them to the cluster?

No, you have to do something else. You have to take the data that you originally had, and you need to reshard the data. You need to take the data, move it around, copy it around.

Now, all the time you're spending recopying the data, redistributing the data, what is the business impact that the cluster is having?

Nothing. It is not actually solving a business problem. All you're doing is you're moving data around.

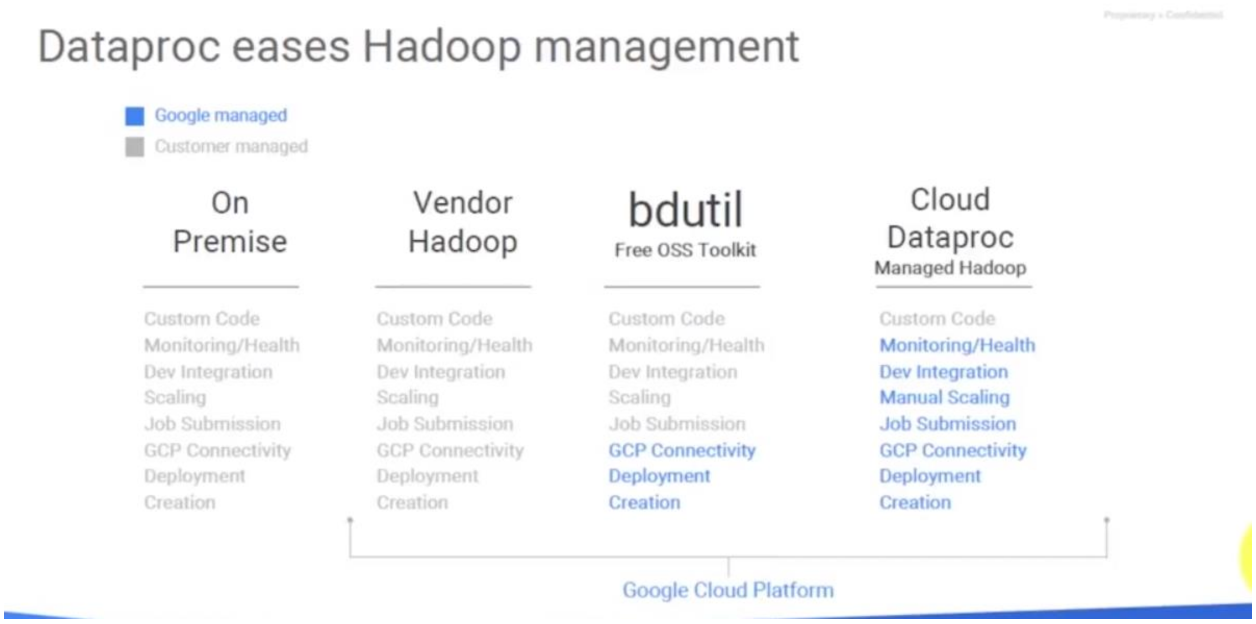
Or think about a cluster that primarily get used during work hours. And someone says, well, I'm noticing that in this cluster, you are not really using it between 10 PM and 6 AM. Let's go find the jobs that we can now run on this cluster, and now you're basically beating down every door in your company trying to increase a utilization of your cluster.

Is that a good use of your time?

What would you rather spend your time doing?

Managing resource allocation and provisioning or doing analytics?

Dataproc Eases Hadoop Management



Dataproc eases Hadoop management

■ Google managed
■ Customer managed

On Premise	Vendor Hadoop	bdutil Free OSS Toolkit	Cloud Dataproc Managed Hadoop
Custom Code	Custom Code	Custom Code	Custom Code
Monitoring/Health	Monitoring/Health	Monitoring/Health	Monitoring/Health
Dev Integration	Dev Integration	Dev Integration	Dev Integration
Scaling	Scaling	Scaling	Manual Scaling
Job Submission	Job Submission	Job Submission	Job Submission
GCP Connectivity	GCP Connectivity	GCP Connectivity	GCP Connectivity
Deployment	Deployment	Deployment	Deployment
Creation	Creation	Creation	Creation

Google Cloud Platform

Dataproc eases Hadoop management. Let us start on this slide here from the left. When you're running Hadoop on premise, you have

- code to worry about
- monitoring the health of the server
- development integration
- scaling
- submit jobs
- handle connectivity
- deployment and creation.

Now, you can move a little bit farther, you could have a vendor do all that for you. And there are lots of vendors that will provide you with turn-key clusters. They handle a lot of the setup and configuration of it, for a price. But then they're also, you still have to worry about, because you're running it on premise, you're still doing all the power, you still have physical servers, you've got all that on site to work with. Now, you could use bdutil, which are a free open source toolkits that can help with some of this sort of stuff, and you can basically build a bit of a solution where some of the pieces can be leveraged from the cloud: So like Google Cloud Platform Connectivity deployment and creation, and you're going to manage your custom code, and the monitoring and the developer integration and the scaling and jobs submission. The ultimate goal, and with the Google Cloud Dataproc that's what we're going to be able to achieve, is basically to offload monitoring health, developed integration, scaling job submission, connectivity, deployment creation, all of that over to Google Cloud platform. We can really concentrate on our custom code pieces, and everything else we just use as a service and pay for. What does a typical Dataproc deployment involve? Well, we go into Google Cloud platform, we could use the web console, we can use G-Cloud or we could use the APIs. We could basically request a cluster be created with several parameters like how many nodes we want on it, and the size of those nodes. We then click Create, and about 90 seconds later our cluster is ready to go. OK, you're saying, "Is that really true?" Well, you bet. Let's give it a try.

Creating a Dataproc Cluster from the Web

https://cloud.google.com/compute/docs/regions-zones/

Here is the general layout of what the Google Cloud Platform looks like right now. Now I will note that just in the last several weeks some of these future regions and zones have actually come online. This map is always changing and I encourage you to go up and take a look at Google Cloud Platform, Zones and Regions page. Let me point you to that really quick. I'm just going to go to my browser, grab a new tab and I'm going to search for GCP zones and regions. And almost always the first article that comes up is the Google Zones and Regions. And when you click on this, it'll actually, just scroll down a little bit and it'll show you all the current locations that you could run Google Cloud Platform resources, although your results may vary a little bit. There are certain regions that can't support some Google products but in the case of Dataproc, it's just compute engine so it's already supported in all these regions. We want to make sure that we put our compute nodes where our data lives.

Google runs a network within a zone, so within a Google Cloud Platform zone the data transmitting between storage and compute is literally passing across what Google calls their petabits bisection network. Basically, they didn't build petabit network cards but when they aggregate all the bandwidth and they look at the capabilities on that it's on the petabit scale. Between regions and also between zones you have less bandwidth. Between zones in the same region that's basically they're different buildings or different segments of buildings with separate isolation across so the bandwidth isn't quite to the petabit scale. And then when you go between regions you're looking at fiber optic connections scattering the globe and Google transmitting it across those network segments. So that's why it's again really important to choose where your data is versus where your data PROC cluster is.

Cluster Configurations and Preemptible Workers

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/9XNY5/cluster-configurations-and-preemptible-workers

Now, when you actually build your Dataproc cluster, you're given the option of three different cluster configurations:

1) A single node for everything. And that's what's called "pseudo mode" in traditional Hadoop structures. Now, that's just for experimentation, that's for working with really small datasets and everything happens on one machine.
2) The more common one is the standard which uses one master and then two or more worker nodes.
3) And then the third is a high availability which actually leverages and/or creates three masters plus all the workers. That way if a master fails, you've still got two other masters that have the data, and they can keep the cluster operation. You haven't lost everything.

If you go with a standard, with one master only and something happens to the master basically anything that wasn't stored outside of that cluster is going to be lost. We can still use the Hadoop file system. In fact, when we were provisioning our disks we ask, "How big do you want the Hadoop file system to be?” If I choose four worker nodes and I give each one 500 gigabytes, presumably that's about two terabytes worth of disk space. But we have to take into account the replication scale so we probably have something more like 600-700 gigabytes of space, because the default replication scale on the Hadoop file system is three copies.

We have a note here saying "Don't use Hadoop file system". Well, why not? That's a great step bringing your on-premise up to dataproc. The problem is that any persistent data that is stored in the Hadoop file system is where you are loading your input files or you are collecting your output files from. That prohibits the ability to literally create the cluster, use it and then destroy it. We're going to have them directly manage the persistent data we store on the Hadoop file system. The better answer, and we're going to look at this in a later video, is to actually change our Hadoop jobs to directly use Cloud Storage instead of the Hadoop file system. It is going to give us the capabilities to do our persistent data to bring it in from Google Cloud Storage, run on the cluster and then store data back out. And so, the cluster itself remained stateless. We don't have to preserve the state of the cluster between each job.

One of the really powerful capabilities of Google Cloud platform is the ability to use preemptible worker nodes. What are preemptible worker nodes? Well, these are basically nodes that Google offers at a significant discount of around 20 to 30 percent of the full price. Preemptible nodes are not unique to Dataproc. You can actually go through compute engine and ask for a preemptible node. And if Google has availability on the computing resources in that zone, they will go ahead and give it to you. It is like buying a last-minute plane ticket when the airline has a bunch of seats open, or a last minute hotel room when the hotel room was going to go empty. You are sort of in the driver's seat on that pricing, you know, because hey, it's going to be lost revenue if they don't use it. So rather than have some sort of weird auction or other sort of bidding war, Google just basically has a flat discount that they offer on it based on purely the computing charges. And it amounts to basically 30 cents on the dollar. I'm going to take a minute and I'm going to go look at compute engine to show you a little bit of details about preemptibles. And then I'm going to adjust the cluster to use them. Let's do a quick demo here because we can only explain so much. I'm going to shift over to my Dataproc cluster. And here's my Dataproc cluster but I'm going to go over to compute engine. First: go to compute engine and create a brand-new computer instance. I'm just asking for instance one. It's going to run a Debian Linux. And we see over here that the cost for one CPU and 3.75 Ghz of RAM in the U.S.

one central or U.S. Central one C is 24.67 per month. Now, this is not a premium image, this is a free operating system image – there is no OS charge for that. When I scroll down the bottom I see there's a dropdown that says "management", "disks", "networking" and "SSH keys". If I click on "management" and scroll down a little bit, we see there's an option for preemptability. I'm going to turn that on. And look what happens to the price. It's now down to 7.70 per month. This is a significant cost over the $25 we saw before.

The idea is you are using computing resources that Google has unallocated at that time. Now, what if somebody comes along and suddenly needs that computing power and Google could get full price for it? Well, they will preempt you. Now, preemptability basically says they send you a soft shut down and you have 30 seconds to finish off whatever you're working on, and then the system will be shut down. Now, another thing is you can only run it for 24 hours. That's the most they let a preemptible instance run for before they'll shut it down. Well, that could be, you know, not ideal for running something like a Web server or things like that. But when it comes to Dataproc that is absolutely perfect because while Hadoop is built to handle the increase and decrease of nodes. Nodes can join the cluster and they can also be taken out of the cluster as needed. Let's now go over to our Dataproc cluster. And let's see where the options for us to add preemptible nodes are in the cluster. Go to the configuration and we see that it says "We don't have any preemptible worker nodes right now". Well, we can solve that. Let's edit that and I'm going to ask for preemptible worker nodes. Now, I don't get to choose specific type of compute because they're going to match the same number of CPUs and memory as my worker nodes. And also, they're not going to have any primary disk assigned to them because they can't be part of the Hadoop file system. And that's normally or that's primarily what the disk is for. Why not? Well, since the preemptible could be taken away at any time, we don't want it to hold any data, even if it's temporary, as part of the cluster. Cick "save".

Let's take a look at the VM instances. Here are the clusters we've submitted the command and in just a minute we should see our Dataproc. If they're available, we'll go ahead and create us four more preemptible workers. I now have the capability to add these preemptibles, and I can also define them when I create my cluster. And basically, they're going to get "let me get my job done faster because I've got more workers". Plus, they're going to let me run it for lower cost because I'm going to pay less for those workers. Let's say I have a job I need to run and it takes two hours If I use four nodes. That is my baseline. And my deadline is I have to produce the results in two hours. I need to find my cluster with four worker nodes right off the bat because there's no guarantee I can get preemptibles. And so, I based it on one master node and two workers. And I don't get any preemptibles. I can't get my job done in the two hours. But based on what I've, you know, calculated things to be one master, four nodes completes my job in two hours. And in this case, I asked for four more workers so presumably we can get those for preemptible workers. We can now get the job done in one hour. Now, what's my cost savings to this? Well, I only had to run the main workers for an hour so half the price for the primary workers. And I got 30 cents on the dollar for the preemptibles. My job finsihes faster, assuming those preemtibles come online, plus I'm saving money.

Can we take this to another extreme? You bet. If you need to run a job and you don't have a tight deadline for when it needs to be done, you just need it to be processed. There's nothing to say that you couldn't build one master, two regular workers because that's the minimum number we need. And then I could build 10 or even 100 preemptible workers. And as preemptible workers can come online, they'll process. Now, a little word of advice on preemptibles. The larger you ask for the machine to be, the less likely you're going to get preemptible workers. Or the more likely they are going to be taken away once they're up and running. Why is that? Well, if you asked for a 16 or a 32 node preemptible worker, that's over half of a physical machine in Google's data center. And more often than not somebody is going to come around who wants a big machine and they're going to need to preempt you to offer them that capability. But if you ask for smaller workers so your job like my image processing where I wanted to process 50,000 images in, you know, and create six different sizes, a bunch of preemptible workers would be fabulous because it can run in parallel and process more and more of my results. I really can design around preemptible workers when time is not a factor. And when time is a factor what I have to do is calculate the minimum number of workers necessary to get my job done in the timeline, and then supplement that with preemptible workers. Let us shift back to our side deck here. We get our jobs done faster and for less cost. And during set up is just under the Advanced Options, you just say how many nodes you want. You don't have to do anything special to that preemptibles, they're going to match the same configuration as the worker nodes.

Customizing a Dataproc Cluster

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/SRQsj/customizing-a-dataproc-cluster

When creating a Dataproc cluster, there are some additional options that can be selected. Let's jump over and I'll pull up the wizard for creating another cluster and I'll discuss these few extra entries for you while we see it in Google Cloud Platform. I am going to go back to Clusters and I'm going to say Create another cluster. There was the name and the zone, and the master nodes, and the cluster node that we talked about. There was the disk, here's the worker nodes. And when I click this drop down here, there's the preemptable node option. There's a cloud storage staging bucket, which it'll automatically create if you don't choose one, it'll just create a unique identifier for it. Here's our networking, so we can decide what networks are we actually going to put it on. Now networks are beyond the scope of this video but there's plenty of resources up within Google Cloud Platform to discuss the network configuration. A default project only has one network, so this is connected on the network in the particular zone we're at. What I'm really excited to tell you about is this image version option. If we drop this down, we actually see that Google is offering at current time five different images. A 0.1, 0.2, 1.0, 1.1 and a preview. I am going to click on the question mark here, and click on Learn More.

It is going to tell me these are the Cloud Dataproc Version List. So what version of software will be on the Dataproc cluster when I create it? The default is obviously the latest version, version 1.1, and that's going to have Apache Spark 2.02, Apache Hadoop 2.73, Apache Pig 0.16, which is pretty much the latest version, and Apache Hive 2.11. It's also going to have the connectors for Google Cloud Storage so that Hadoop can talk to Google Cloud Storage. And remember, that's going to be one of our eventual goals, is to not leverage HDFS because that makes our clusters not stateless anymore and instead begin using cloud storage for the importing and exporting of our data, our persistent data. And then also a big query connector for Hadoop so that it can directly call big query. So, if 1.1's got the latest versions of that, and in fact in this case, it was last updated in April of 2017. Now version 1.0 has slightly older versions of Spark and Pig and Hive. Now, there are also the 0.1 and the 0.2 version which at this point are unsupported because they go way back, and they've got really older versions of the tools. And then at the very bottom, we see that there's actually a preview version. And this has the latest version of Spark, the latest version of Hadoop and also current versions of all the other tools. So, that's what we're actually choosing when we're building our cluster using the graphical interface. And we can also pass those as parameters during creation. Now the only other option there we see is initialization action. And we're going to discuss that in a later video in this series on Dataproc, is just basically a script we want to have run on the workers and the master nodes when they boot up. So that way if we need any custom installation to occur. And then the project access. And this is basically just like when you create a compute node, what automatic access is the software running on the nodes going to be able to leverage? The worker nodes, are they going to have API access to all of Google Cloud services that are present in the project?

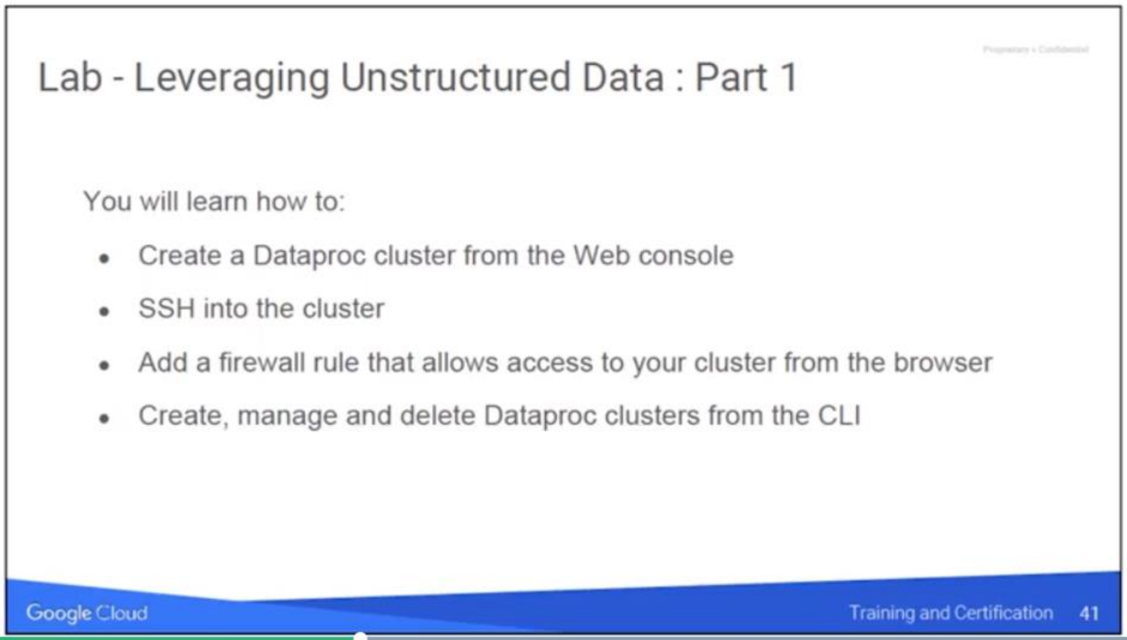
Now, just like everything within Google Cloud Platform, most of the time, the things you can do in the web console can also be done from the gcloud command or the gcloud SDK. And they can also often be integrated in with custom software because we can call the restful services directly. So there's plenty of ways to build and manage your Dataproc clusters.

What does it look like if I was to use the gcloud SDK command? Well, I want gcloud, I want dataproc, and I want clusters, and I want to create, and it's going to be called my-second-cluster. I'm going to put it in the zone us-central1-a, I want the master machine type to be an n1-standard-1, so one CPU with a standard ratio of memory which is 3.75 gigs. The master boot disk size is going to be 50 gigs. I want two workers. I want those workers to be an n1-standard-1, so they're also going to be one CPU with 3.75 gigs of RAM, and their boot disk is going to be 50 gigs in size. Now, are there other options there? Yeah, we don't have an option for preemptible instances, so the default's going to be used, which is none. If we wanted that, we just have to know the appropriate --option for it. Now, how do you get that? Well, you can always run gcloud dataproc --help to get details on that level. gcloud dataproc clusters --help to get details on that. And if you want the really specific commands on the creation command, you just do that gcloud dataproc clusters create --help. And that'll give you all the documentation. And if that's not enough, head up to Google Cloud Platform's documentation on the website, or like I do, search gcp dataproc, and you'll come up with the entire documentation set on Dataproc.

I am now goimg turn you lose on the lab, and we're going to have you create a couple of Dataproc clusters. The first one you're going to do is through the web console, and then we're going to have you SSH into it and just verify it looks good. Now, Hadoop does offer web page management. And so, by default you wouldn't be able to get those webpages just from your on-premise workstation because the firewall is going to be blocking it. They're running on non-standard ports, so like port 8080, 8088 and 50070. We are going to have you make a firewall rule change that allows your browser on your laptop or your desktop that you're using to do this, to actually connect into the resources of that Dataproc cluster. And then last but not least, we're going to have you create, manage, and delete a Dataproc cluster from the command line interface.

Go ahead and pause the video and go tackle Leveraging Unstructured Data Part 1, the lab to create your first and your second Dataproc cluster.

Lab: Create a Dataproc Cluster



Lab - Leveraging Unstructured Data : Part 1

You will learn how to:

- Create a Dataproc cluster from the Web console
- SSH into the cluster
- Add a firewall rule that allows access to your cluster from the browser
- Create, manage and delete Dataproc clusters from the CLI

Google Cloud Training and Certification 41

I am now going to turn you loose on a lab, and we're going to have you create a couple of Dataproc clusters. The first one you're going to do is through the Web console, and then we're going to have the SSH into it and just verify, it looks good. Now, Hadoop does offer web page management. And so, by default, you wouldn't be able to get to those web pages just from your on premise workstation because the firewall is going to be blocking it. They're running on nonstandard ports, so like port 8080, 8088 and 50070. So we're going to have you make a firewall rule chains that allows your browser on your laptop or your desktop that you're using to do this, to actually connect into the resources of that Dataproc cluster. And then last but not least, we're going to have you create manage and delete a Dataproc cluster from the command line interface. So, go ahead and pause the video, and go tackle leveraging unstructured data Part 1: The lab to create your first and your second Dataproc cluster.

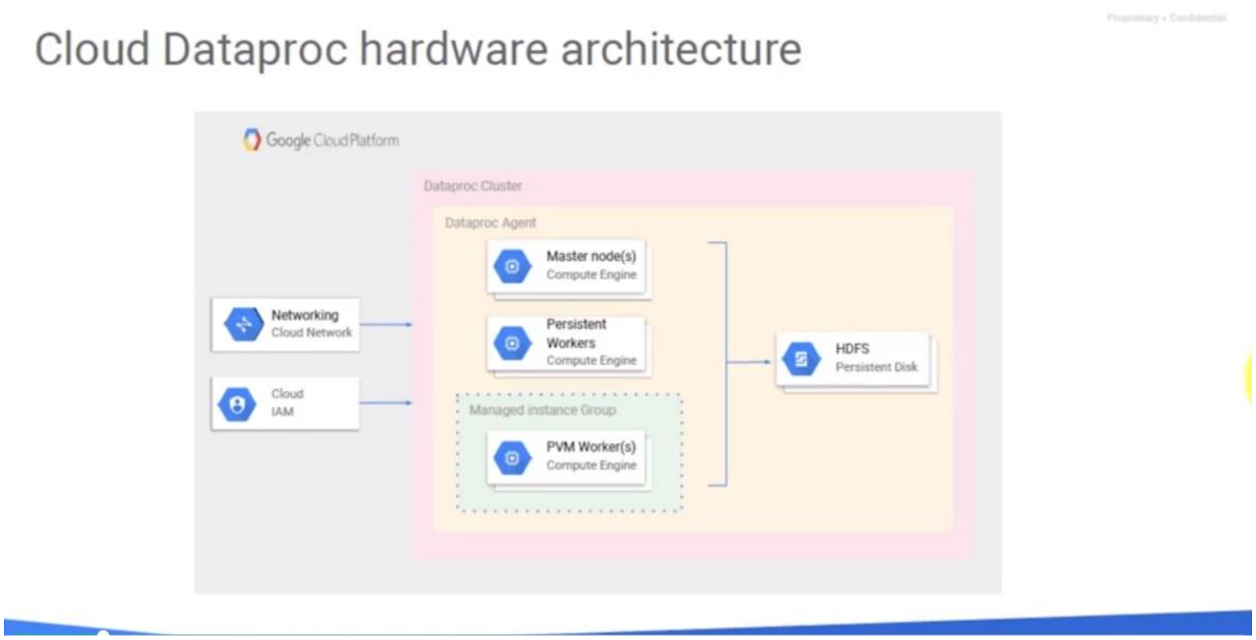
https://codelabs.developers.google.com/codelabs/cpb102-creating-dataproc-clusters/

Review

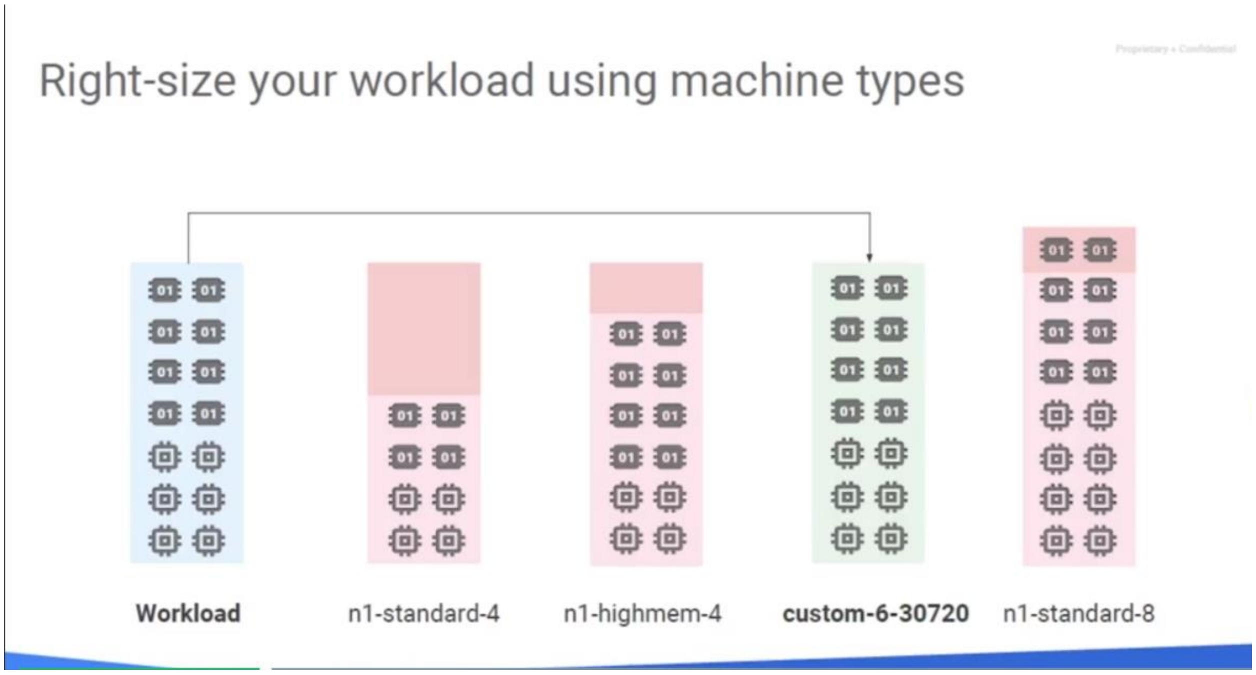
https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/T67Ek/lab-creating-a-dataproc-cluster-review

Notes: Networking is no longer under big data – it is under networking so go to the firewall menu there. Also, when going on the web, port 80 (for html) is not open. You have to go to the ports that Hadoop uses port 8088.

Creating Customer Machine Types



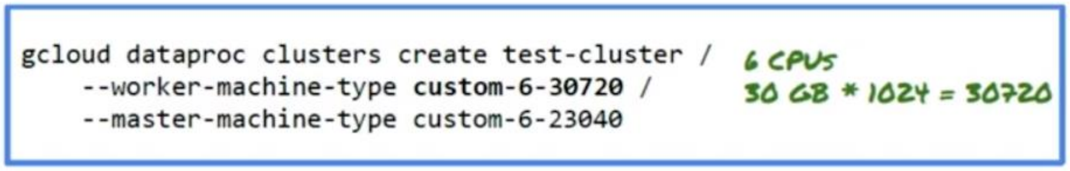
The first Dataproc cluster was created using the graphical interface and the second one using the gcloud command utility. The last part of this video that we're going to talk about is controlling the size of the compute engine nodes that you're using. In our examples, we used one CPU and 3.75 gigs of RAM. And depending on the jobs that you're running, you may need to adjust either the amount of memory or the number of CPU's per node. The Cloud Dataproc hardware architecture is basically we're leveraging Google Cloud for the networking aspect. We're using Google Cloud IAM or Identity and Access Management for security. And then the Dataproc cluster is actually creating compute engine notes for the master, for the persistent workers, as well as the preemptable workers. It's going to be creating, within the cluster, the Hadoop File System. Not showing in this diagram, but should be, we should also see Google Cloud Storage in there. Because that should, ideally, be the place that we persist our input data and send our results.



Right-size your workload using machine types

I want to create a custom machine type where I want to define a specific amount of CPU, and a specific amount of memory. Instead of using the N-one standard one, I use the term custom. And then there's a dash and then there's a number of CPUs, and then there's a dash and the amount of memory to do. Google allows me to actually build the machine to my exact specifications. Now there's a little bit of a caveat with that. In that you may not want to do that because if you're really close to one of the standard building blocks for Google Cloud platform it's going to be cheaper. You can get the right value to pass to creating your machine type. In the menu to create the cluster, I have the option to set the machine type. I have the option to use preemptibles, - remember that from last time so let me turn that off. And that way we can also get an approximate pricing – see the price estimate If I was going to run my dataproc clause for an entire month would be $24.67 per month plus whatever network egress I would have. So, one CPU and if I increase this to two CPUs, we see here's my option two CPUs 7.5 Gig of RAM. And I see my option for four CPU and 15 gigs of RAM. And so on all the way up. Here we see the N one high memory which is two with 13 gigs of RAM. Double what we have. And then if we scroll down even further we can see here are the high CPUs which lower the memory per CPU.

Creating the custom machine type...



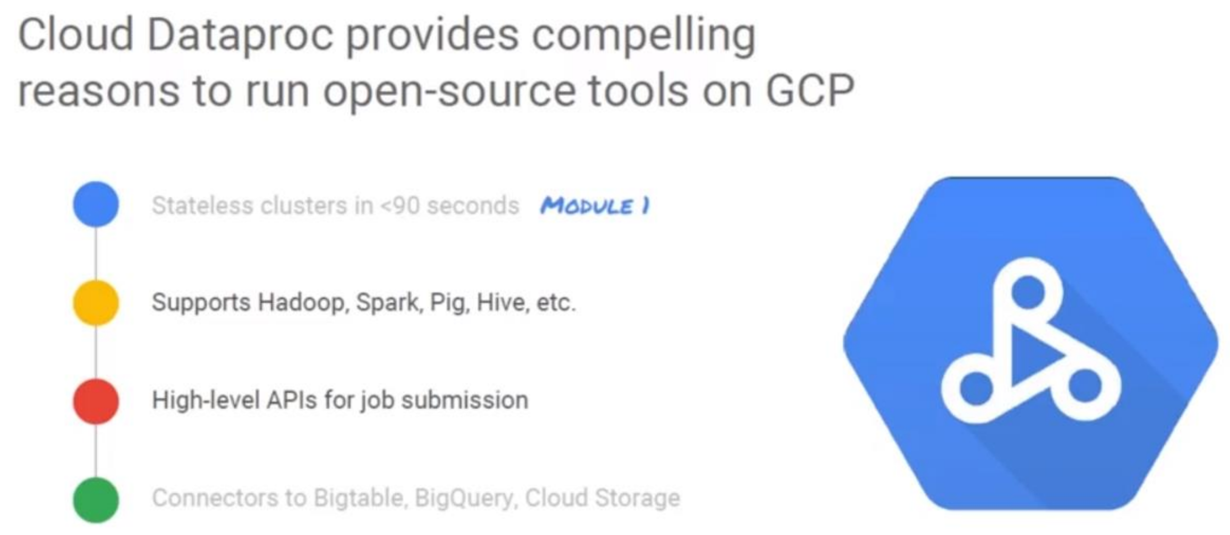
```
gcloud dataproc clusters create test-cluster /
    --worker-machine-type custom-6-30720 /
    --master-machine-type custom-6-23040
```

6 CPUs
30 GB * 1024 = 30720

Google also has an option here for us to customize. I can set amount of memory on the worker nodes, number of CPUs, etc. Notice the warning down the bottom it says, you know what? You can save two dollars and fifty two cents per month by going with a straight N one standard four which is actually going to give me 15 gigs of RAM. And that's because the standard blocks have a lower price than when you customize because basically there's a calculator that Google uses. The number of cores times the core price plus the amount of memory times the memory price. And when you're right around a standard building block it's definitely going to be cheaper to go for the standard one. Now, how do I figure out what exactly is that for the parameter to pass to gCloud create? Well, I'm glad you asked. We're going to scroll down the bottom. And what I can do is, I can ask Google to show me the command line. Now, when they show me the command line for this, this is the command line that would create this compute engine instance. But we're going to use it to pick up the identifier for that custom build we just set: Custom four dash 10240. It is a custom VM rather than a standard, it's got four CPUs and I want 10240 memory. And that's what we would plug in to this value back here when we do gcloud data proc cluster create. We could plug in custom four 10240. Giving me what I need. So, that's what this slides kind of showing you. I just like to show it to you live sometimes. It's more fun for me too. But basically, we went through just creating compute engine instance. We chose the customize option, we specified what we want and then we could steal it out from either the command line or we could have also pulled it from the restful call as well because it's in the blob of JSON data that would be submitted that way. Okay. Well that concludes our video on our first section of Google Cloud DataProc.

MODULE 2: RUNNING DATAPROC JOBS

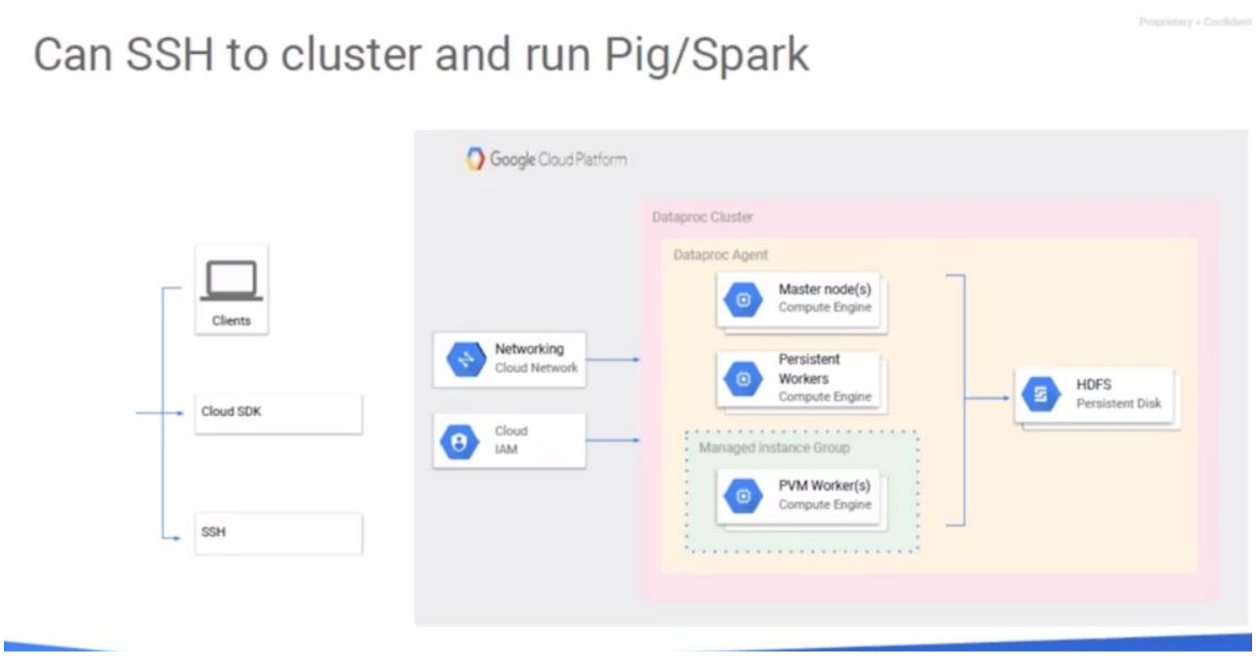
Overview



Cloud Dataproc provides compelling reasons to run open-source tools on GCP

● Stateless clusters in <90 seconds MODULE 1
● Supports Hadoop, Spark, Pig, Hive, etc.
● High-level APIs for job submission
● Connectors to Bigtable, BigQuery, Cloud Storage

Welcome to Google Cloud Platform. This is Grant Moyle. In this video, we are going to look at running Dataproc jobs on Google Cloud Platform. Dataproc is Google's managed Hadoop infrastructure so it's a fabulous platform for running your open source parallel computing jobs. In a previous video we discussed how to create a stateless cluster in less than 90 seconds using Dataproc. In this video we're going to discuss how to run our jobs on Dataproc clusters, how to submit jobs via a high-level APIs, and also how to optimize the use of Google Cloud Storage as a replacement for your persistent data instead of using Hadoop file system. Once a Dataproc cluster has been created and then started, you can SSH into the master node, or in the case of a highly available cluster, any of the master nodes. From there, you can interact directly with Hadoop, run Pig/Spark, or Hive drops. Since all of these are preloaded, you can also interact with the Hadoop file system directly, or HDFS, as we'll hear it referred to, as that has been provisioned on your master and persistent worker nodes. The only nodes that don't have HDFS will be the preemptible nodes because they could be preempted at any time, so it doesn't make sense to have their disks associated with the Hadoop file system. Now the best way to experience SSH in any other cluster is to do it yourself. So, I'm going to just turn you loose on a lab, and in this lab, you're going to create a data plot cluster using Google Cloud Shell. Create a Google Cloud Storage Bucket and clone a repository. From there, you connect into the master node via SSH and run a Python Spark read evaluate process loop interpreter. Then you'll run a Pig job that'll use data stored in the file system. And then you'll destroy the cluster since you'll all be done with it. Pause the video and go complete the lab titled Leveraging Unstructured Data Part 2.


Proprietary + Confidential

Can SSH to cluster and run Pig/Spark

You can SSH into the cluster and it can run by Spark. Why would you do that? Why would you want to SSH into the cluster to that particular machine and run a program on the interpreter? Normally, the way you interact with a cluster is when you use a notebook or you submit a job into the Spark cluster but there are times when it's advantageous to be able to simply drop into the shell of the machine. Drop into Spark so that you can do a quick exploration. You can try-- you could try a couple of lines of Spark code. You can basically see if something works, if the data is in place, how long something takes. For those kind of quick experimentation, it can be very helpful to be able to SSH into the cluster. Start the PySpark interpreter and try out your quick experiments.

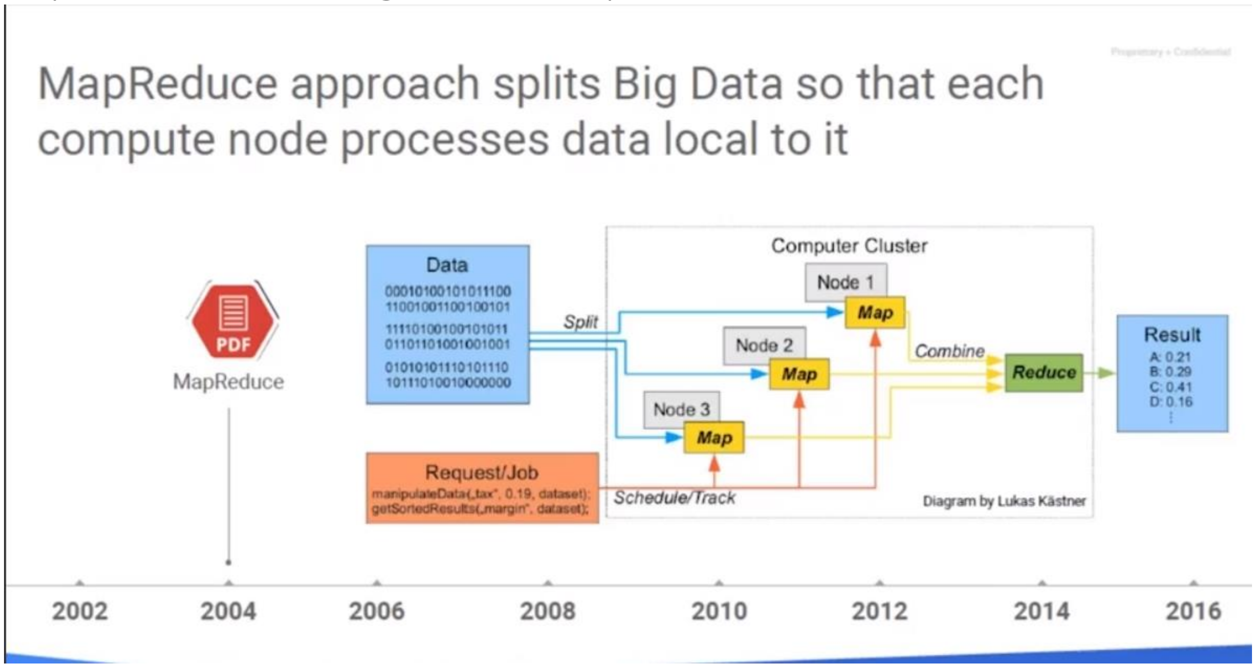
Lab 2a: Running Pig and Spark programs

https://codelabs.developers.google.com/codelabs/cpb102-running-pig-spark/#0

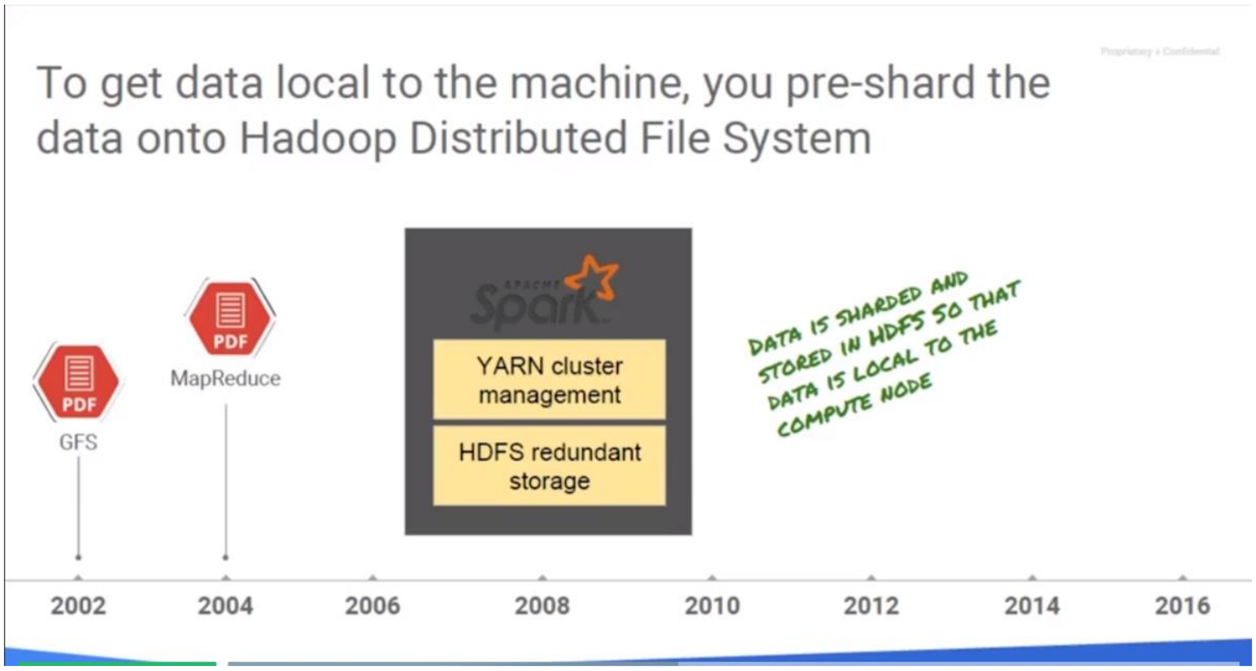
Review:

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/I1R2C/lab-review

Separation of Storage and Compute


MapReduce approach splits Big Data so that each compute node processes data local to it

Now, let's talk about the separation of storage and compute. Often, the first stage of moving from on-premise Hadoop and Spark is to make your existing jobs run as is. You don't want to change too many things at once. But once you get your data cluster processing your existing jobs, it's time to optimize them for our Google Cloud platform. Hadoop clusters typically rely on the Hadoop file system, or HDFS, which takes files and spreads them across the disks of the nodes. This allows for parallel read and write operations and also allows for larger disk sizes than you could create with a single disk. It also allows for the worker nodes to read/write files locally when operating. While this is advantageous for temporary data, if you utilize it for storage of persistent data either as your inputs to the cluster or the outputs after the job is done. If any of that is stored in the Hadoop file system, you cannot create or destroy the cluster – it needs to be running all the time. Well, that kind of defeats the purpose of utilizing Cloud platform. You might recall from a previous video, that the basis for Hadoop and the Hadoop file system were Google's papers that they published back in the early 2000s on the Google File System and MapReduce. Which the Open Source community took and built Hadoop, Hadoop File System and evolved into Spark.


To get data local to the machine, you pre-shard the data onto Hadoop Distributed File System

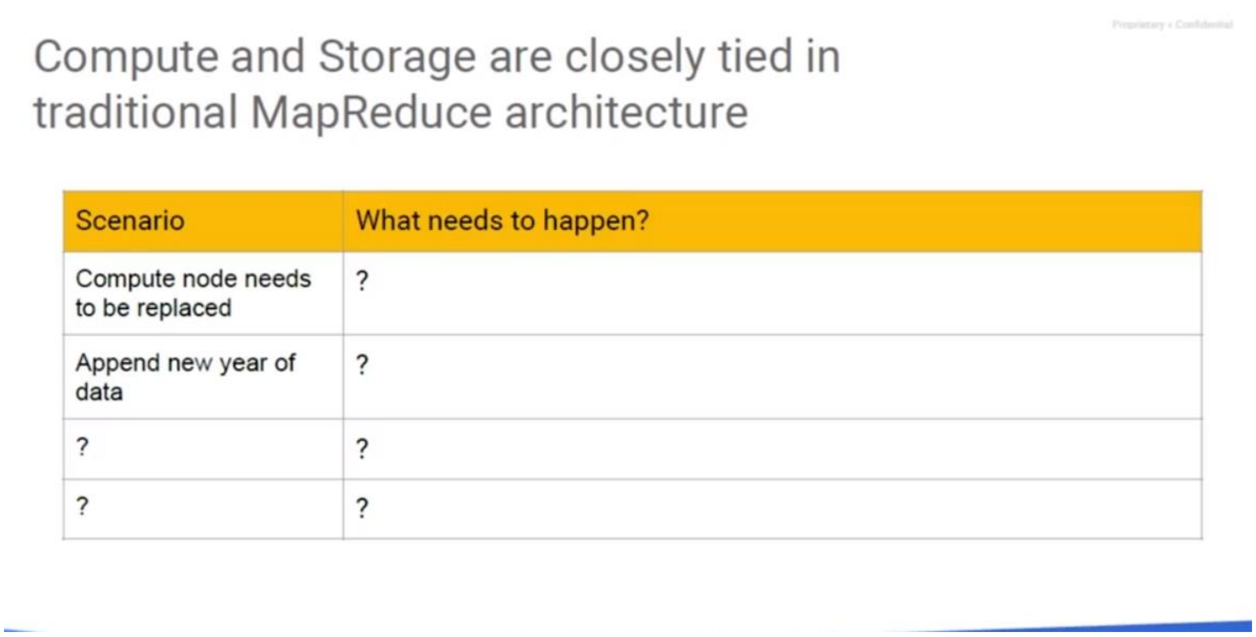
Proprietary + Confidential

YARN cluster management

HDFS redundant storage

DATA IS SHARDED AND STORED IN HDFS SO THAT DATA IS LOCAL TO THE COMPUTE NODE

When talking about Hadoop and Spark, we often refer to YARN or Yet Another Resource Negotiator, for managing our cluster. We utilize the Hadoop File System for persistent storage, in addition to using it for temporary storage. You might keep your input files on that HDFS, which shards that data and spreads it across all the disks that make up the Hadoop File System - all the disks on the compute nodes. And then when you execute your Hadoop job, they will read the input files, process and often put their output at the end of the job. Now, at the end of the job, since the files are sitting on the Hadoop File System, you going to have to go collect those. Well, the input data and the output data satisfies the characteristics of persistent data, because you need it before and after your job is actually run. This is going to present an issue if we want to create the cluster on the fly. Remember, we can have it running in as little as 90 seconds, process our data and then get the output, then destroy the cluster, optimizing the use of our Cloud resources.



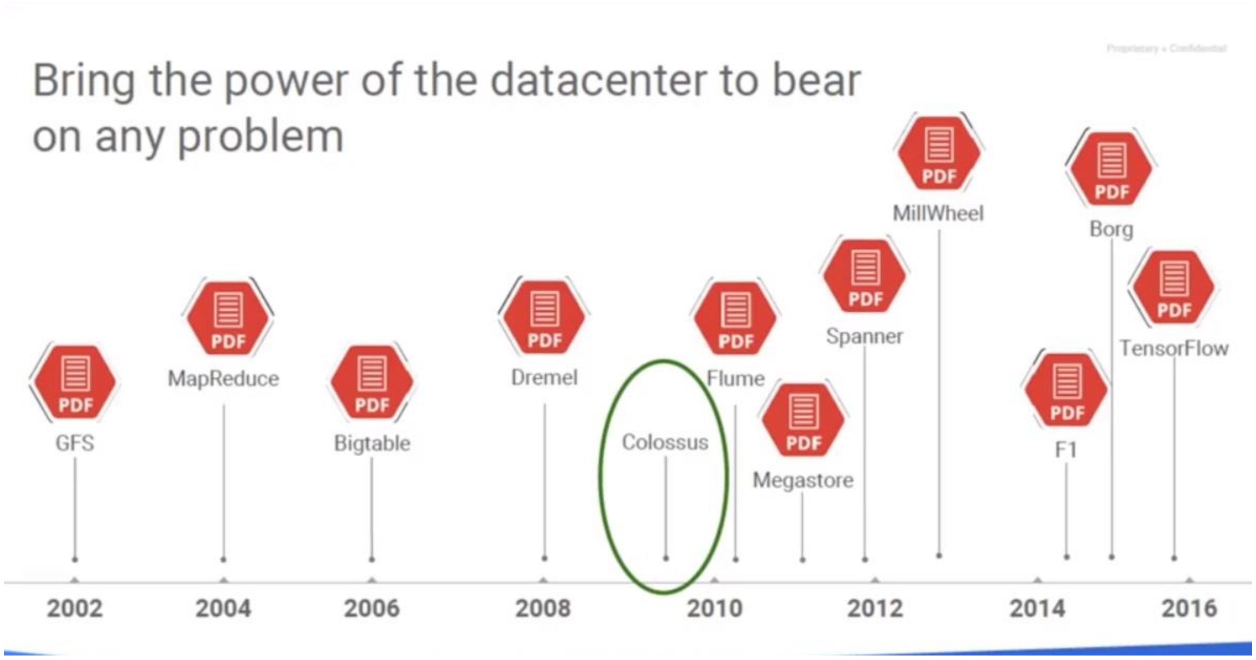
Compute and Storage are closely tied in traditional MapReduce architecture

Scenario	What needs to happen?
Compute node needs to be replaced	?
Append new year of data	?
?	?
?	?

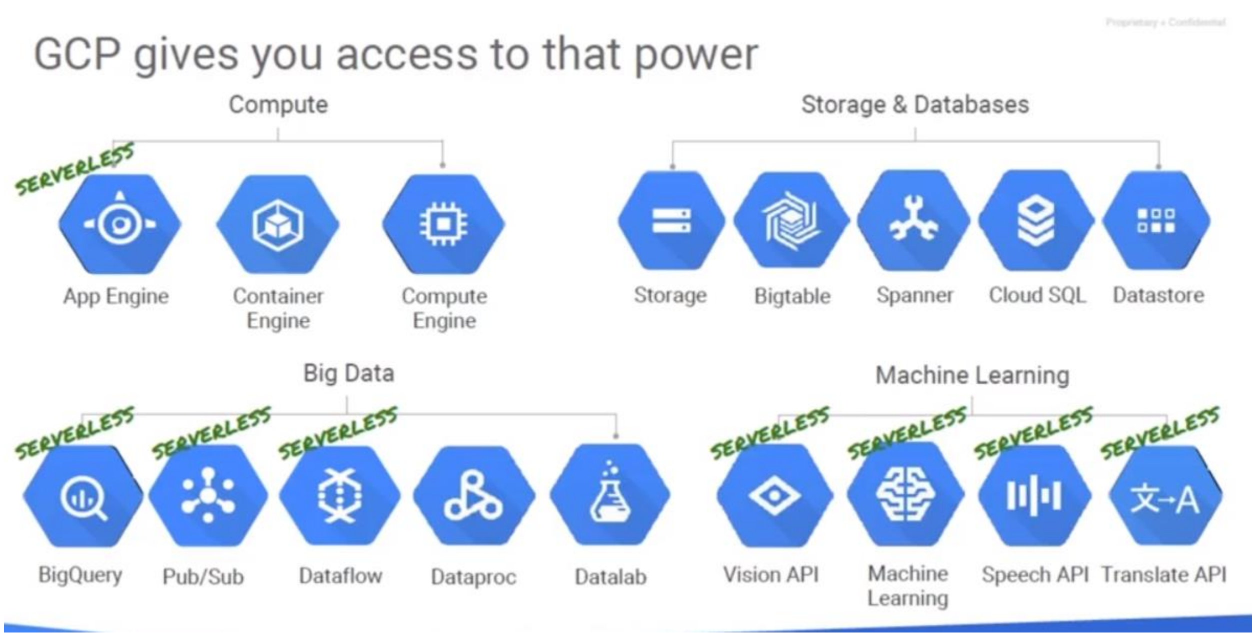
If you think about it, compute and storage, in a traditional cluster, are closely tied together or coupled together. When a compute node needs to be replaced, the shards of the file that are stored on those disks, must be replicated somewhere else. If we want to add new data, well, we have to store or append it to the files in the Hadoop file system. The Hadoop file system must always be running. Or whenever we want to perform a file operation. And we may have to increase the size of the disks if our new data that we want to add is larger than we have allocated across Hadoop file system. We have to pay to keep the nodes running all the time, which is fine on-premises because they're your physical hardware. But it goes against the cloud model.

We ingest our data from Hadoop file system. And then, we write it back out to the HDFS. If I am running a ten node Hadoop cluster, the data is sharded across the ten nodes. And let's assume that the replication factor of three which is basically what the Hadoop file system does, it shards the data and places a copy of each shard on at least on around three notes. Well, we have to have three times the amount of storage allocated, so I want to have 10 terabytes. I actually need to allocate 30 terabytes plus I need to have some room for growth.

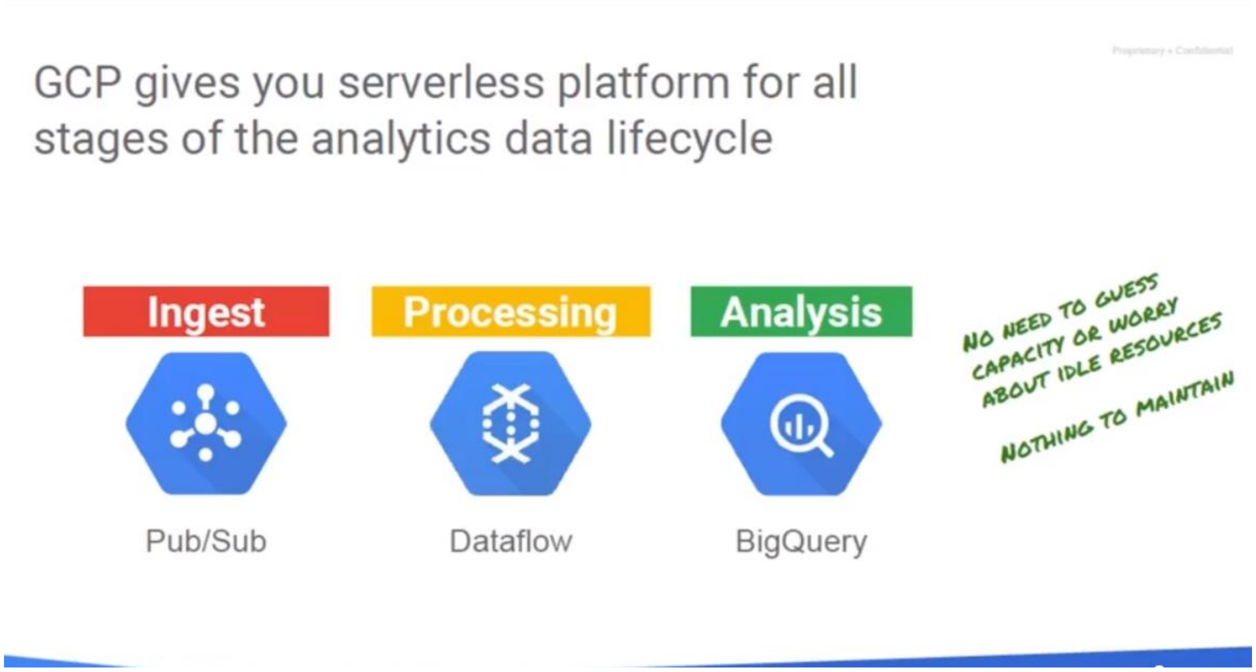
When the nodes replace the system as to ensure the replication factor is kept at that same level. And since the data prop machines are based on compute engine nodes, the discs assigned to the Hadoop file system are built based on allocation not usage. There are a lot of downsides to the traditional HDFS model. Now, it is not going to completely go away, but it is going to go away for the purpose of our persistent input and our persistent output. What we would really like to see is storage based on our usage, not allocation that goes along with a good cloud theme. And the ability to quickly increase or potentially decrease the use of that resource, and have our costs adjust accordingly. Google Cloud Storage is built on top of Google Cloud Storage buckets – it is the evolution of the Google file system, and these days it's a product within google called Colossus. Now Colossus offers a global file system over a petabit scale network. I've heard that petabit term a lot lately from Google, both in storage, you know they're putting a petabyte of new data into YouTube every day. It also forms the back end of their network. They didn't go out and build pedabit network cards or anything like that, but when they actually looked at their overall network backplane that connects the compute engine with your storage, they can pump pedabits worth of data around there per second.



But you don't see Colossus as a product option when you go up to Google Cloud platform. Well, you're not going to. That's an internal name for the platform that basically has replaced the Google File system, and is running behind the scenes on a lot of Google technologies, including Google Cloud storage buckets. So, if we can incorporate cloud storage buckets into our either inputs or outputs for our data-plot cluster, we could replace the Hadoop file system dependency with Google Cloud storage.



Google Cloud Storage is just one of the serverless options within Google Cloud Platform. The benefits of the serverless options like App Engine, which we use for Web and application hosting, BigQuery, which we use for data warehousing and it's a direct, and we can query like it's a SQL database. Pub/Sub for enterprise middleware or message-oriented middleware. Dataflow for pure elastic pipeline's based on Apache Beam, and then all the machine learning APIs. We've got all of those in the serverless space where you pay for what you use. Well with Hadoop, we're very still much in the driver seat. We have to specifically create the cluster to our specifications - the number of nodes, how big the nodes are, etc. We submit our job, we monitor the job, and then when we're done we have to go and destroy the cluster. But the long-term goal might be to move to a pure serverless platform that just allows you to submit a job, writing it in Java or Python using Apache Beam, submit it to Google Cloud platform and letting Google handle all of the scale with that. Even better, we might want to feed in that data in batch form from Google Cloud storage, stream it in via Pub/Sub, process it with Dataflow, and then store the results out to BigQuery or BigTable, and we have nothing to maintain. Now that would be sort of the end goal of data analysis on Google Cloud platform, but we can't necessarily get there right away. We've got to go through a series of steps. If my customer has an existing Hadoop infrastructure, they run Spark jobs. Well I'm going to want to support those in Google Cloud platform first, before I go and try to recode them to a brand new platform. Think of it as a series of steps to move us up to the Cloud.


GCP gives you serverless platform for all stages of the analytics data lifecycle
Proprietary + Confidential
Ingest
Processing
Analysis
Pub/Sub
Dataflow
BigQuery
NO NEED TO GUESS CAPACITY OR WORRY ABOUT IDLE RESOURCES
NOTHING TO MAINTAIN

The problem of compute and storage rigidity – scenario 1

One problem, and we've been saying this over and over again. One problem with the traditional MapReduce architecture is how closely tied compute and storage are to each other.

As an example, let's say we have a cluster and we bring in one new year of data.

What is it that you have to do? You've just brought in new data. What do you have to do? Perhaps you have to take that year of data and sort it. You have to store it on all of your compute nodes. New storage often implies new compute nodes. There is that tie again, right? New data, new compute.

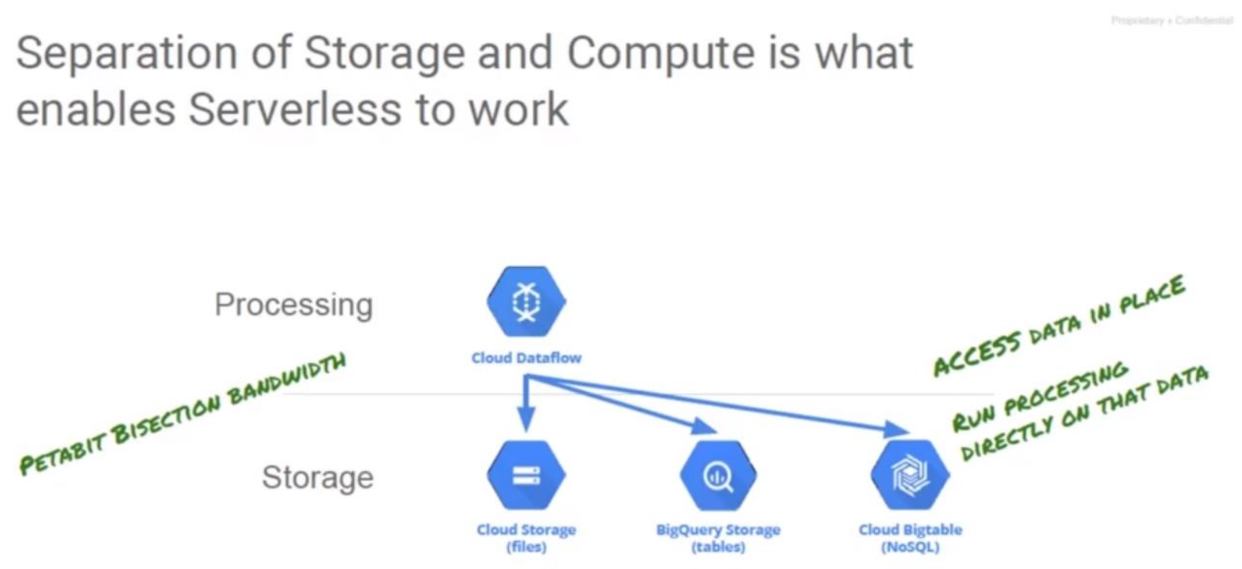
The problem of compute and storage rigidity – scenario 2

For example, you run into a problem and one of your computer nodes goes down; it needs to be replaced - a problem on the compute site. What happens? All the data that was stored on that compute node now need to be stored somewhere else. There is that tie again, compute and storage are too closely tied and that's the start of many other problems. Can you think of any other scenarios?

Moving to a Serverless World

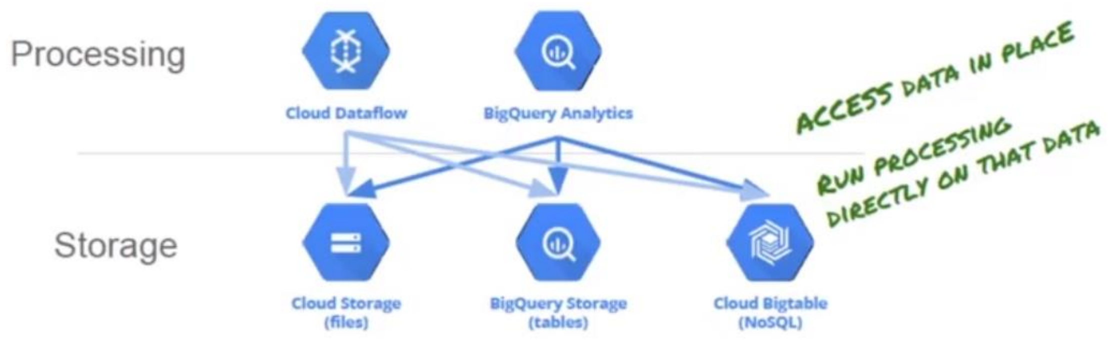
A serverless design is going to focus on getting the inside out of the data instead of the administration of the servers giving you a near infinite scale, because Cloud Storage, Pub/Sub, Dataflow, BigQuery, all of these runs on Google's infrastructure which can scale to really whatever you need. You also pay for what you use, not for what you allocate. This allows you to experiment with small sets of data, and once you're happy, you can go a very large scale very easily, giving you the speed and freedom as well as controlling your costs. In a serverless world we process using Dataflow and we get our input/output from Google's Cloud Storage for files and basically BLOB storage, or binary large objects. We use BigQuery for traditional database queries based on SQL. We use Pub/Sub for streaming of our data and we can even use Cloud Bigtable for NoSQL.

Bigtable is Google's managed HBase compatible database. It is a drop-in replacement for HBase. What was HBase? HBase was a database that's often utilized in a Hadoop file system type infrastructure. Because the HBase database gets spread across the Hadoop file system and the database itself is queried using the Hadoop worker notes. So HBase, just like the Hadoop file system gets closely coupled in an on-premise Hadoop infrastructure because the database file is spread across that Hadoop file system.


Proprietary + Confidential
Separation of Storage and Compute is what enables Serverless to work
Processing
Cloud Dataflow
Storage
Cloud Storage (files)
BigQuery Storage (tables)
Cloud Bigtable (NoSQL)
PETABIT BISECTION BANDWIDTH
ACCESS DATA IN PLACE
RUN PROCESSING DIRECTLY ON THAT DATA

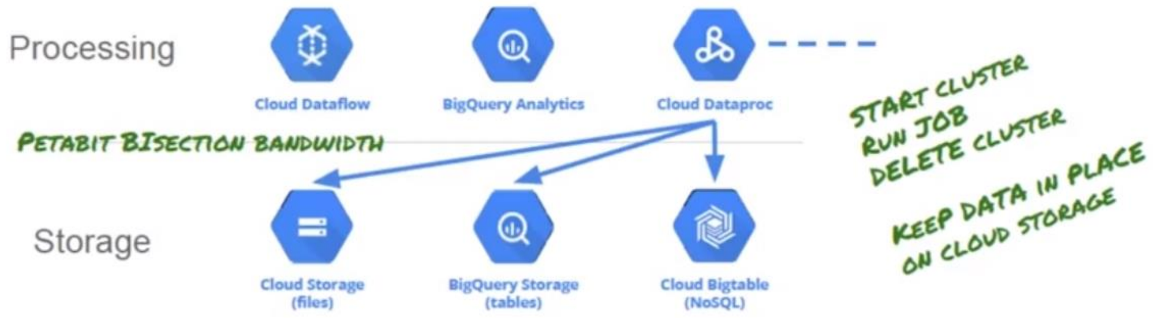
When we start to do something like Dataproc and where our goal is to decouple the storage in the data we can do that because we can move the Hadoop file system data that's used for input and output over to Cloud Storage and we can decouple HBase because we instead can introduce Bigtable. It's drop in replacement uses the same API, so we update a couple references to where we're storing the data and presto we can decouple that as well.

BigQuery also separates compute and storage

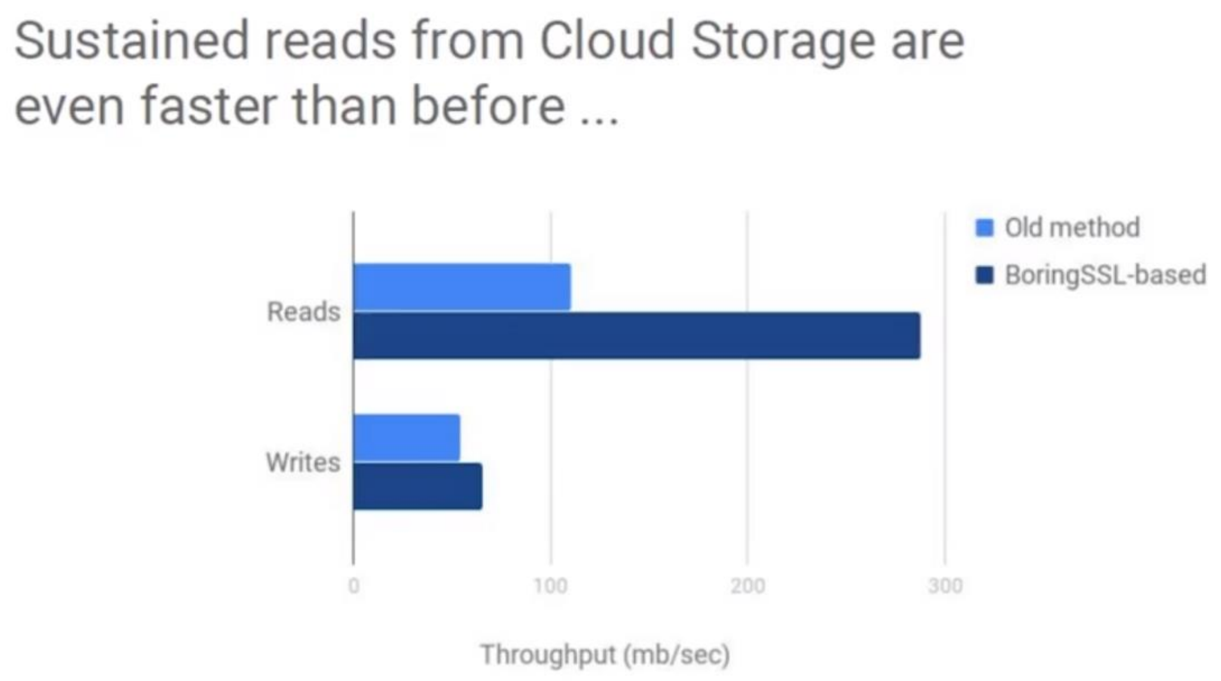


If we look at the architecture of BigQuery, basically BigQuery is a lot like a parallel computing system. We have nodes, the workers that basically process your query. They're basically referred to as slots in the BigQuery terminology. And then we have the storage that BigQuery uses, a highly optimized storage platform that Google uses for holding the database structures in the corresponding column or formats. BigQuery separates its storage from the database engine. We could submit jobs and get results back very quickly because Google can throw thousands of slots against the data and it could scale for large scale databases even up to the petabyte scale, so it's perfect for a data warehouse.

Cloud Dataproc provides the ability for Spark programs to also separate compute & storage

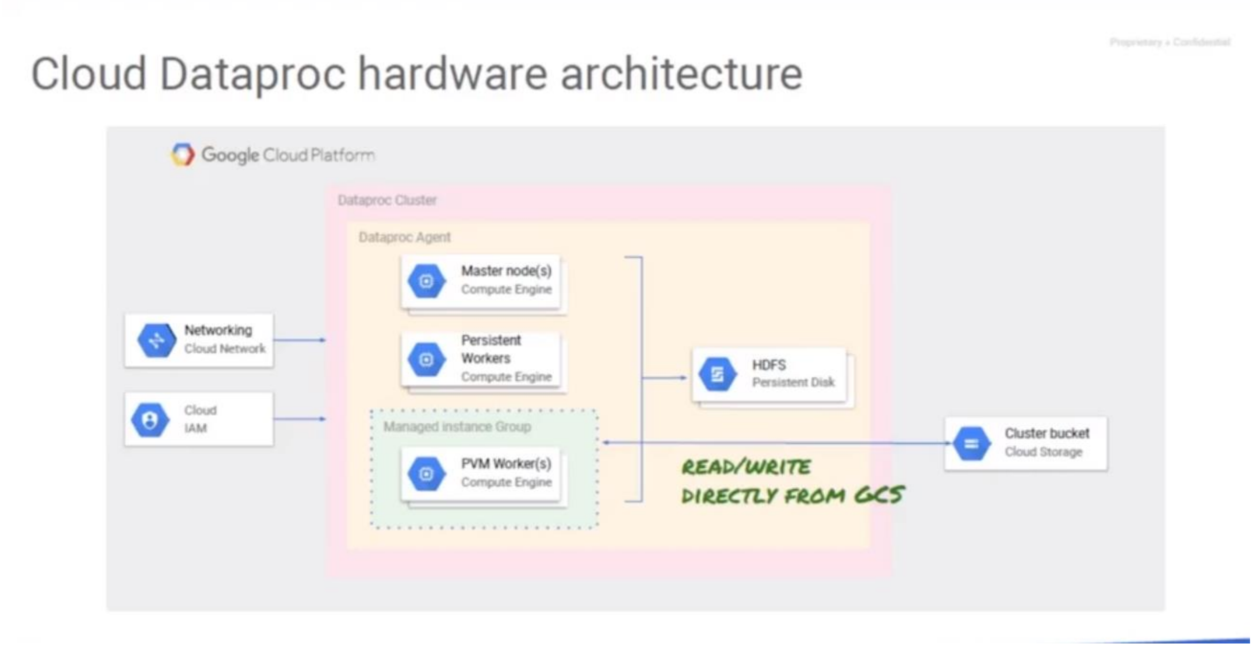


Getting back to Hadoop and Spark clusters, can I separate my storage and my compute once I move Dataproc up to Google Cloud Platform or once I move my Hadoop up to Dataproc on Google Cloud Platform? You bet. With Spark, which runs on top of Hadoop, we just need to modify the jobs so they're going to use our persistent storage coming from Google Cloud Storage for the files, and databases like BigQuery or BigTable for our database needs. Now, our processes have to change a little bit. This is our second big step in moving on-premise Hadoop up to the cloud. The first step was bringing our on-premise Hadoop cluster into Dataproc, making sure our jobs run and using HDFS as is. The second step would be to rework our persistent storage model, to use Google Cloud Storage instead of HDFS for obtaining the input files. And then our end result is writing those back out to Cloud Storage. With Dataproc, this actually isn't very difficult, because the libraries for talking to Google Cloud Storage are already present on the workers and the master notes. We just replace references that say, hdfs:// which is how we describe a path in the Hadoop file system with gs:// paths. And now we're taking advantage of Google's petabit bisection bandwidth that exists within the compute and storage back plane of Google Cloud platform.

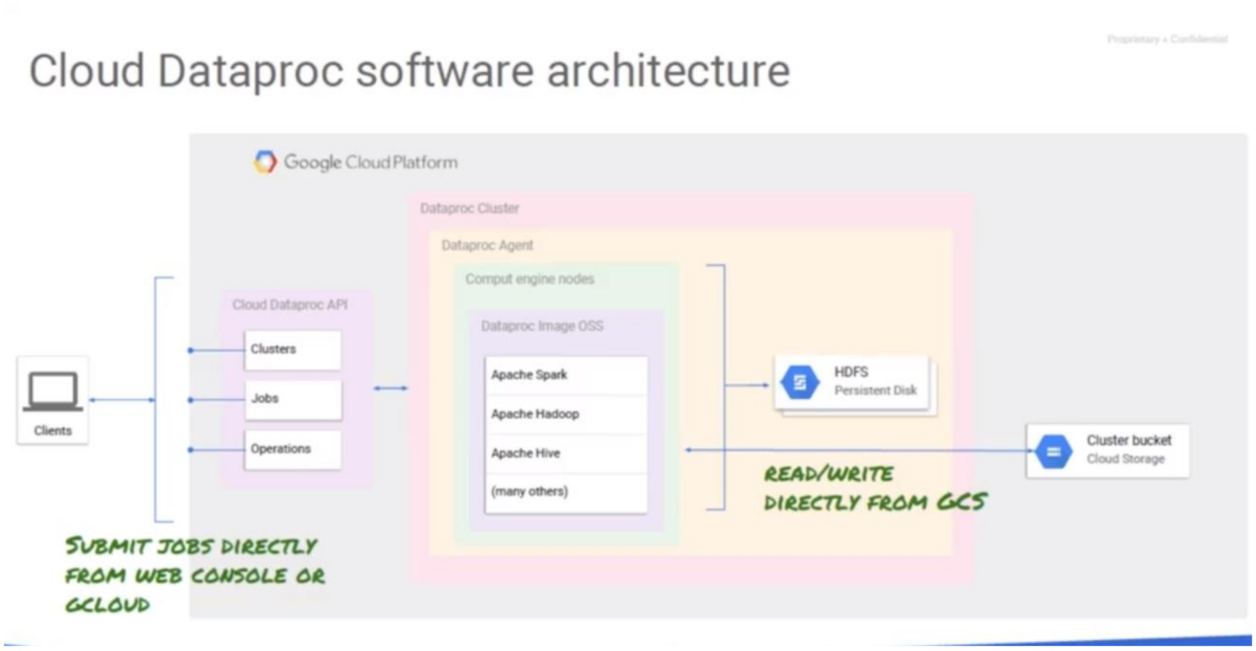


Is Cloud Storage really that fast? You bet. Google spent years optimizing their internal networks. Google uses a lot of software defined networking. Google builds their own routers and the software that runs on top of it. So, if we go out and buy a traditional router from say, Cisco, they provide us a firmware or the software running on the router and it's got to be flexible enough to meet a variety of customers' needs. There is code in there for SNMP and older network protocols and all this sort of stuff. Well, when Google goes to work on their network stack, they don't even bother to put that code into the stack if they're not going to use it. The core code running on their routers, basically, is stripped down to only the features they need. If they are not using something like SNMP, that doesn't get rolled into the code. If they are not supporting an old protocol like, heaven forbid, IPX/SPX or something like that, it's not in the code. So only the pieces of code need to be in there when it gets compiled up. So that improves performance, another false step. And also makes it very scalable, reliable, etc. Now, they put all this on top of that, they put a highly optimized storage protocol that drives Google Cloud Storage and that sustain read and write operations are faster than ever.

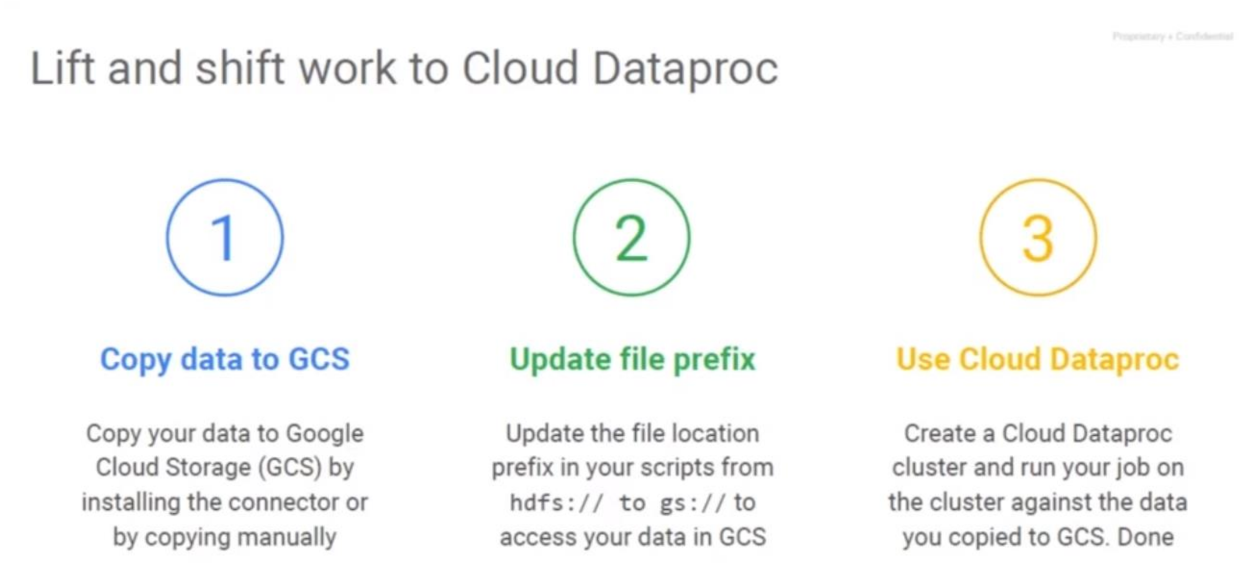
Submitting Jobs with Dataproc and Cloud Shell


Cloud Dataproc hardware architecture
Proprietary + Confidential
Google Cloud Platform
Dataproc Cluster
Dataproc Agent
Master node(s) Compute Engine
Persistent Workers Compute Engine
Networking Cloud Network
Cloud IAM
HDFS Persistent Disk
Managed instance Group
PVM Worker(s) Compute Engine
Cluster bucket Cloud Storage
READ/WRITE DIRECTLY FROM GCS

Let us look at submitting jobs to a Dataproc cluster using the Web interface for Google Cloud Platform and also using Google Cloud Shell. If we make changes to our input and output sources replacing the HDFS with GS references, we can now read and write any persistent data to Google Cloud Storage. We can reduce or eliminate the need for the Hadoop file system. It is just being used for temporary working space on the persistent workers. You might remember HDFS couldn't be run on preemptable worker nodes because the preemptable nodes could be taken away at any time which would cause the Hadoop file system to have to replicate the charge to other nodes. Moving persistent files to Google cloud storage is going to allow us to use less persistent workers and more preemptables since we won't now have a bottleneck potential on the Hadoop file system. You could build a cluster that uses for example, one master two workers so you have a bare minimum Hadoop file system for temporary working space and then tens or even hundreds of preemptable nodes optimizing your costs utilization. And since your input and output are now coming from Google Cloud Storage, you won't have any bottlenecks for the input or the ultimate writing of your output data.



To submit a job to dataproc, we can use the web console or we can submit jobs via G-cloud. Jobs are sent to the master node which then distributes the work. Hadoop, Spark, Pig, and all of those can now write directly to Google Cloud storage. We really just need to submit jobs and have the appropriate references.


Proprietary + Confidential

Lift and shift work to Cloud Dataproc

1. **Copy data to GCS**

   Copy your data to Google Cloud Storage (GCS) by installing the connector or by copying manually

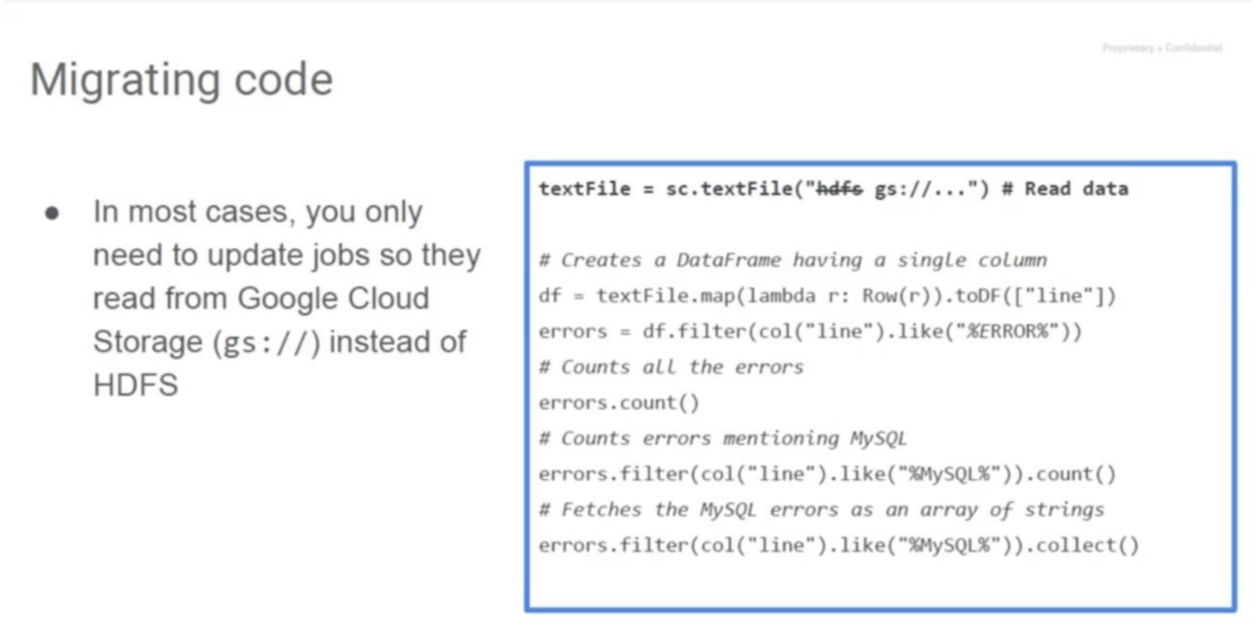
2. **Update file prefix**

   Update the file location prefix in your scripts from `hdfs://` to `gs://` to access your data in GCS

3. **Use Cloud Dataproc**

   Create a Cloud Dataproc cluster and run your job on the cluster against the data you copied to GCS. Done

How do we move dataproc from on premise? We use the idea of lift and shift. First, we want to copy the data that we're going to need the cluster to utilize in a Google Cloud Storage bucket. We could do this by uploading the files manually or we could install a connector to do it. The second step would be to update in our code the references so we're no longer pointing to hdfs:// and instead, we're pointing to the correct path in gs://, our new cloud storage location. And third, we create a dataproc cluster, we run the job, and when we're done we delete the cluster. Now, if we read and write data to an H base database, what we're going to want to migrate that H base database up into a Bigtable and update the references. It uses the same API as H base, so it's a drop-in replacement when you utilize Bigtable. The code changes should be minimal for that.



Migrating code

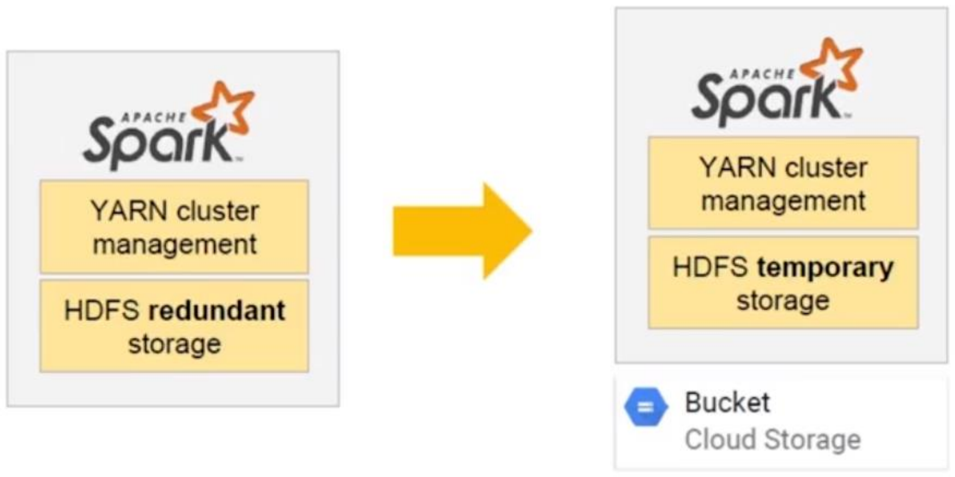
Proprietary + Confidential

- In most cases, you only need to update jobs so they read from Google Cloud Storage (`gs://`) instead of HDFS

```
textFile = sc.textFile("~~hdfs~~ gs://...") # Read data

# Creates a DataFrame having a single column
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))
# Counts all the errors
errors.count()
# Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count()
# Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect()
```

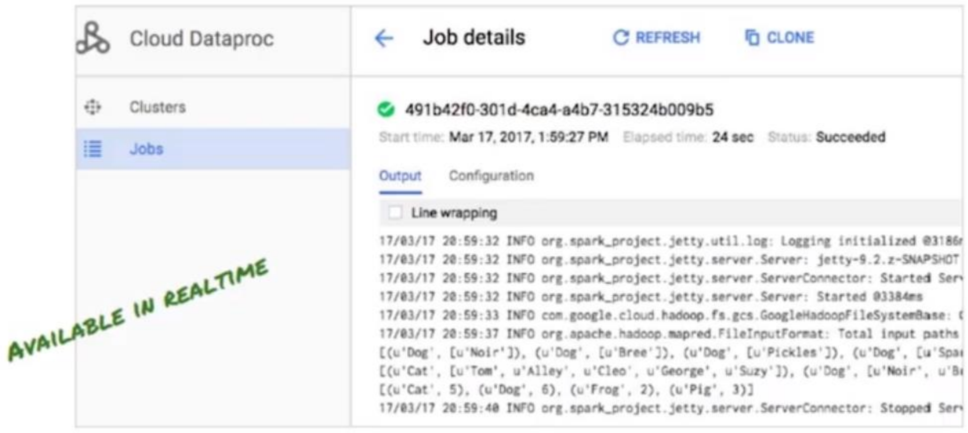
If we read and write data to sequel as part of our jobs, we might move our data into BigQuery so that we can query it very quickly and get back out our data sets and also on the output we can output our information into big query if we are going to be using it for analytics. Or if we're going to be using it for real time solutions, we can output our results into Bigtable. For example, let's say we currently run a 50 node Hadoop cluster at our organization and each day we are in a series of 10 jobs. It takes several hours to run. We run them one after the other. We submit our jobs at 5:00 pm, in about 2:00 am our jobs are done, and we're dealing with them the next morning. Well, unless your output of one job is depended on, or it is an input to the next job, why not run those 10 jobs in parallel instead of running them in sequence? So, we create 10, 15 node Hadoop clusters or hack make the clusters larger, submit the jobs in parallel, and they get your results much faster. We can have 10 jobs at 5:00 every day we fire up 10 clusters and we submit each job to a different node. If they're using cloud storage while cloud storage is going to provide the data to all of the nodes. And if we're outputting to cloud storage, all our results can go back to cloud storage. Since we create, use, and destroy the cluster, we're only going to pay for the compute time we're using. And we're not going to worry about having to worry about the cost of running that when the systems are idle because by deleting the cluster we give the resources back to Google. With this in mind, we can get our results faster. If we had those 10 jobs were taken a total of seven hours to run or nine hours, instead, we could get them done in an hour or two and we can have our results by 7:00 pm. Migrating your cloud dot or migrating your code to the cloud is really easy.

Why change hdfs:// to gs://?



Look at the code on the above. The first line has a reference where it's basically getting its text files from an HDFS file store. You see we've crossed out HDFS and instead we're now referencing gs:// and the path which would identify the Google Cloud Storage bucket and then the path within that bucket to the files. Once we do this, we've now reduced our use of HDFS to just temporary storages that are persistent and we've made our cluster stateless. So, that's what we did by changing the HTFS to GS. Migrating the code is really easy. Look at the code on the right. The first line has a reference to an HDFS path. Just replace it with the new path and identify the Google Cloud Storage bucket and then the path within the bucket to the files. So, once we do this, we've reduced our use of HTFS to temporary storage instead of persistent storage which makes our cluster stateless. We can now easily create, use it, and then destroy it. And the persistent data is in the cloud storage bucket. This may be as far as we need to go to moving our Bigdata processing jobs up to the cloud. But, maybe we want to go a little bit farther. Well, the next step beyond that would be to actually rewrite the existing jobs or tackle brand new jobs by writing the code in Apache Beam and running them on serverless dataflow. But that's beyond the scope of this video.

Monitor logs of submitted jobs from web console



491b42f0-301d-4ca4-a4b7-315324b009b5

Start time: Mar 17, 2017, 1:59:27 PM Elapsed time: 24 sec Status: Succeeded

```
17/03/17 20:59:32 INFO org.spark_project.jetty.util.log: Logging initialized @3186r
17/03/17 20:59:32 INFO org.spark_project.jetty.server.Server: jetty-9.2.z-SNAPSHOT
17/03/17 20:59:32 INFO org.spark_project.jetty.server.ServerConnector: Started Ser
17/03/17 20:59:32 INFO org.spark_project.jetty.server.Server: Started @3384ms
17/03/17 20:59:33 INFO com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystemBase: (
17/03/17 20:59:37 INFO org.apache.hadoop.mapred.FileInputFormat: Total input paths
[(u'Dog', [u'Noir']), (u'Dog', [u'Bree']), (u'Dog', [u'Pickles']), (u'Dog', [u'Spa
[(u'Cat', [u'Tom', u'Alley', u'Cleo', u'George', u'Suzy']), (u'Dog', [u'Noir', u'B
[(u'Cat', 5), (u'Dog', 6), (u'Frog', 2), (u'Pig', 3)]
17/03/17 20:59:40 INFO org.spark_project.jetty.server.ServerConnector: Stopped Ser
```

The last part before I turn you loose on the lab is to discuss how we monitored the jobs running on cloud dataproc. After submitting a job, it can be monitored by the dataproc job sections of the web console. We can also monitor the cluster compute usage using the web UI either through the dataproc image or right down through the individual compute nodes through compute engines monitoring. Now, we can also monitor them using stack driver or Google's monitoring platform.

Lab - Leveraging Unstructured Data : Part 3

- Create a Cloud Storage bucket to store job input, output, and application files
- Submit jobs using the Web Console
- Submit jobs using the CLI
- Monitor job progress and view results

Now I'm going to set you loose on the second lab of this chapter, Leveraging Unstructured Data - Part Three. In this lab you're going to utilize a Google Cloud storage to store job input, output, and the application files, submit the job by the Web console, and submit another job through the cloud shell. Throughout the process your monitor your job process and view the results.

Lab – Leveraging Unstructured Data, Part 3

https://codelabs.developers.google.com/codelabs/cpb102-running-dataproc-jobs/#0

Review: https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/xC6ZO/lab-review

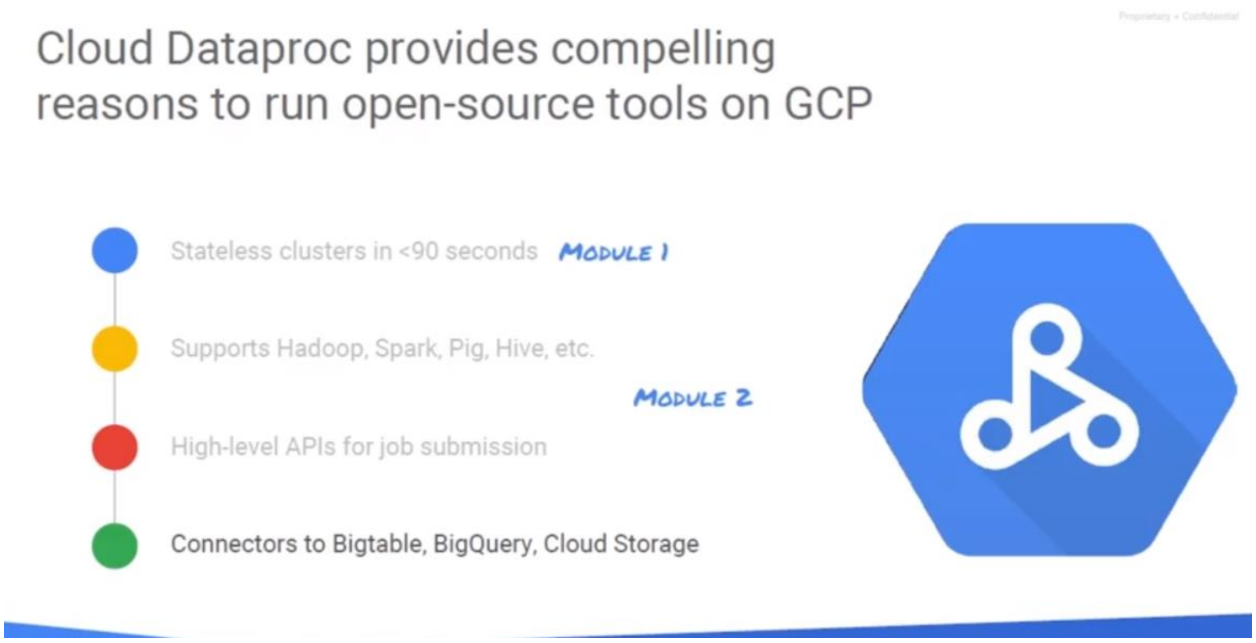
MODULE 3: LEVERAGING GCP

Introduction to Leveraging GCP

In this course, we consider migration from an on-premise Hadoop cluster to GCP. Take the code that already runs on your on-premise cluster plus the Spark, Pig or Hive, etc. code. The one change what we're asking you to make is to change it where you're reading from HDFS. Don't read from HDFS, instead replace reading from HDFS to reading from GCS, all of a sudden things get much better economically. You don't need to have a cluster up all the time. You don't need to have a cluster that people are competing for. You don't need to over provision the cluster for those small periods of peak usage. You get much better economic benefits, you get ephemeral clusters by doing one simple thing; changing your input from HDFS to GCS.

If you're migrating from on-premises where you're doing Hadoop workloads to a public cloud, replace HDFS reads by GCS reads. This allows you to take advantage of the extremely high sustained read performance that GCS provides. And this allows you to create clusters that are job specific. You create a cluster, you run the job and you delete the cluster. So that's basically what the last couple of chapters have been about. But now, migration has been done. You are on GCP. What else can you do? What we are going to now look at is how to leverage the power of GCP beyond the on-premises jobs and migrating them. You now have your on-premises jobs, you've migrated them to the Google Cloud and they are now running on the Google Cloud. Now that you have jobs that are running on Google Cloud, let's not treat your Hadoop and Spark jobs in isolation. Let's look at how you can take advantage of the rest of the capabilities of the platform.

Customizing Clusters: Leveraging Google Cloud Platform, Part 1

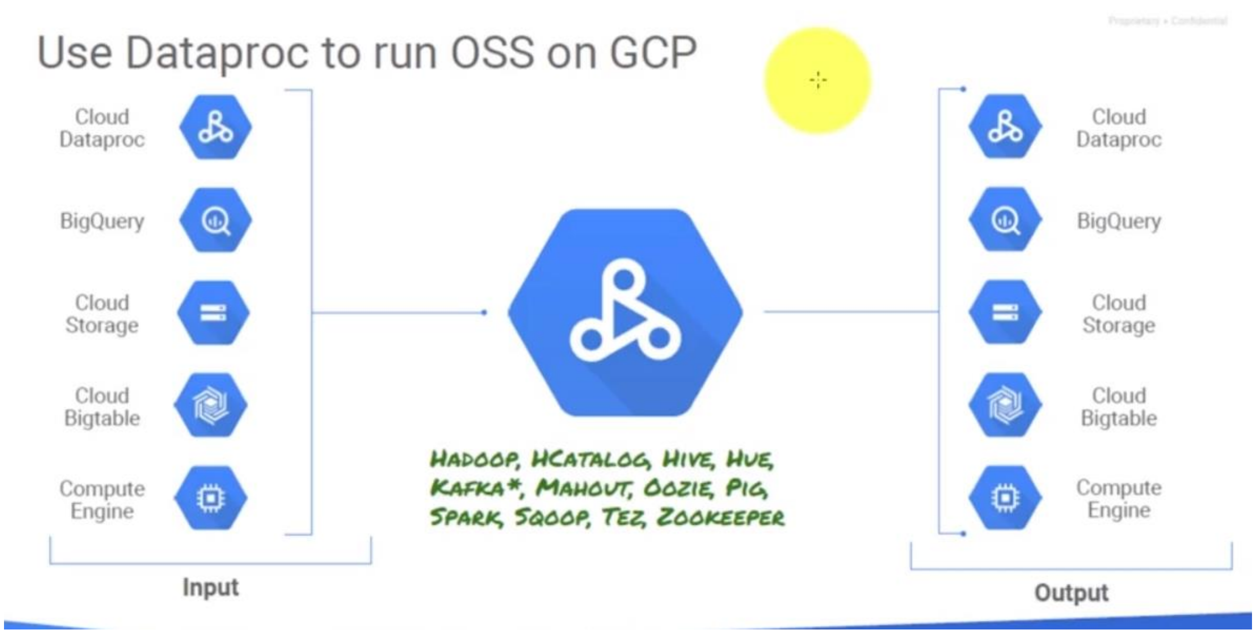


Cloud Dataproc provides compelling reasons to run open-source tools on GCP

- Stateless clusters in <90 seconds MODULE 1
- Supports Hadoop, Spark, Pig, Hive, etc.
- High-level APIs for job submission MODULE 2
- Connectors to Bigtable, BigQuery, Cloud Storage

Proprietary + Confidential

In this video we're going to take a look at leveraging Google Cloud platform for processing our big data jobs on Dataproc and also customize the initialization of our Dataproc master and worker nodes so that we can include other open source software or custom initialization. So far in the Dataproc videos, we have discussed building stateless Hadoop in less than 90 seconds. We've discussed running standard Hadoop, Spark, Pig, and Hive, which are all included in the base Dataproc configuration and submitting jobs by a high level APIs. In this video, we're going to look at customizing the initialization of our Dataproc clusters using scripts or programs that we're going to load into Cloud storage. And we're also going to take a look at how BigQuery and other Google Cloud platform services can be leveraged from Dataproc. Let's look at customizing our cluster.



Dataproc uses Apache Bigtop

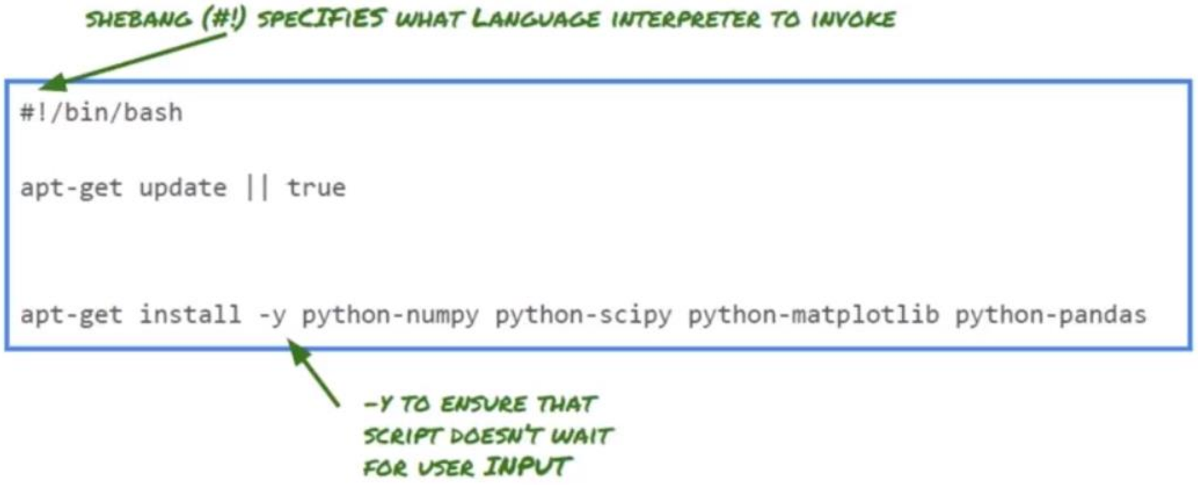
- Conservative about which packages are installed by default.

To install software on a Dataproc cluster

1. Write an executable program (bash, python, etc.)
2. Upload it to Cloud Storage
3. Specify the GCS location in the Dataproc creation command

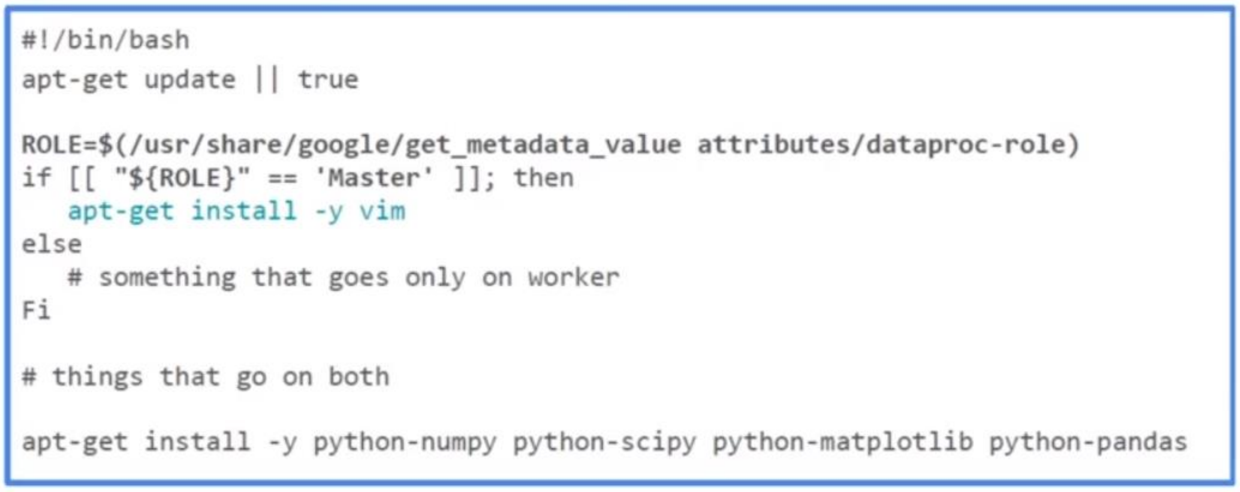
Customizing Clusters: Leveraging Google Cloud Platform, Part 2

Write an executable script that runs as root


SHEBANG (#!) SPECIFIES WHAT LANGUAGE INTERPRETER TO INVOKE
-Y TO ENSURE THAT SCRIPT DOESN'T WAIT FOR USER INPUT

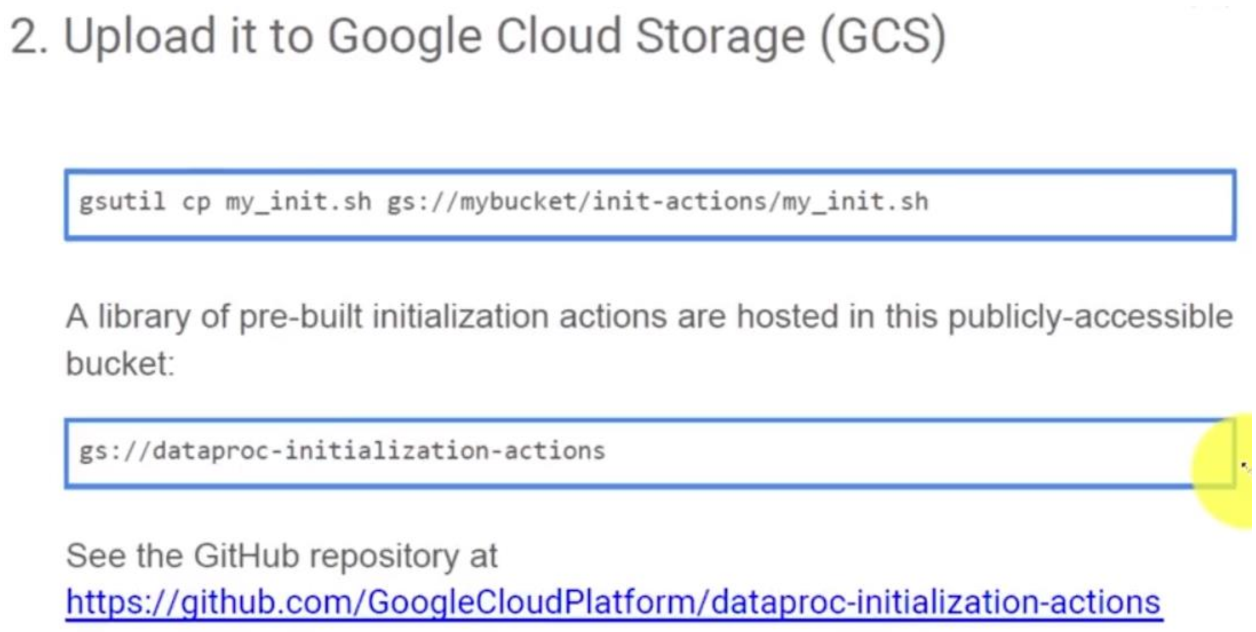
```
#!/bin/bash

apt-get update || true


apt-get install -y python-numpy python-scipy python-matplotlib python-pandas
```

This is a bash script that runs apt-get to install a Python module. To update the cluster nodes and then also do an apt-get install and install a series of libraries that we need: numpy, and scipy, Matplotlib, and Python Pandas. These four components are going to be installed as well as an apt-get update. Now you'll notice the apt get update has a pipe true on it and that's going to pass in the true value so that way it isn't waiting for user input. And you'll notice that the apt get installed, we have a minus Y and that basically tells the script that it should go ahead and confirm yes to any options. Otherwise it's likely to get stuck waiting for input and our initialization will have to eventually timeout on that and continue onward and we wouldn't necessarily want that.



```
#!/bin/bash
apt-get update || true

ROLE=$(/usr/share/google/get_metadata_value attributes/dataproc-role)
if [[ "${ROLE}" == 'Master' ]]; then
   apt-get install -y vim
else
   # something that goes only on worker
Fi

# things that go on both

apt-get install -y python-numpy python-scipy python-matplotlib python-pandas
```

What if I only want to install software on my master node? Google Cloud platforms dataproc visualization uses the same script for both the master and worker nodes. They don't differentiate between the two that way if you need to install something on both nodes, you don't have to do it in two places. The downside to that is what you just want to do something on the master, you need to have your script figure out which one's the master or which ones are the master if you're running a multi node master. The way we do distinguish the nodes is through the metadata. You see how we're asking or resetting the value of roll equal to the metadata value attributes dataproc-role. If the roll comes back as master, we're going to go and install VI or vim there - see the apt get -y vim. It's going to install that and confirm yes to any prompt that appears. We're not going to install anything specifically on the worker nodes but if we look at the very last line of this script, we're going to go out and put numpy, scipy, matplotlib and pandas on both nodes because they're likely to both need those set of libraries for when they're executing a job.



2. Upload it to Google Cloud Storage (GCS)

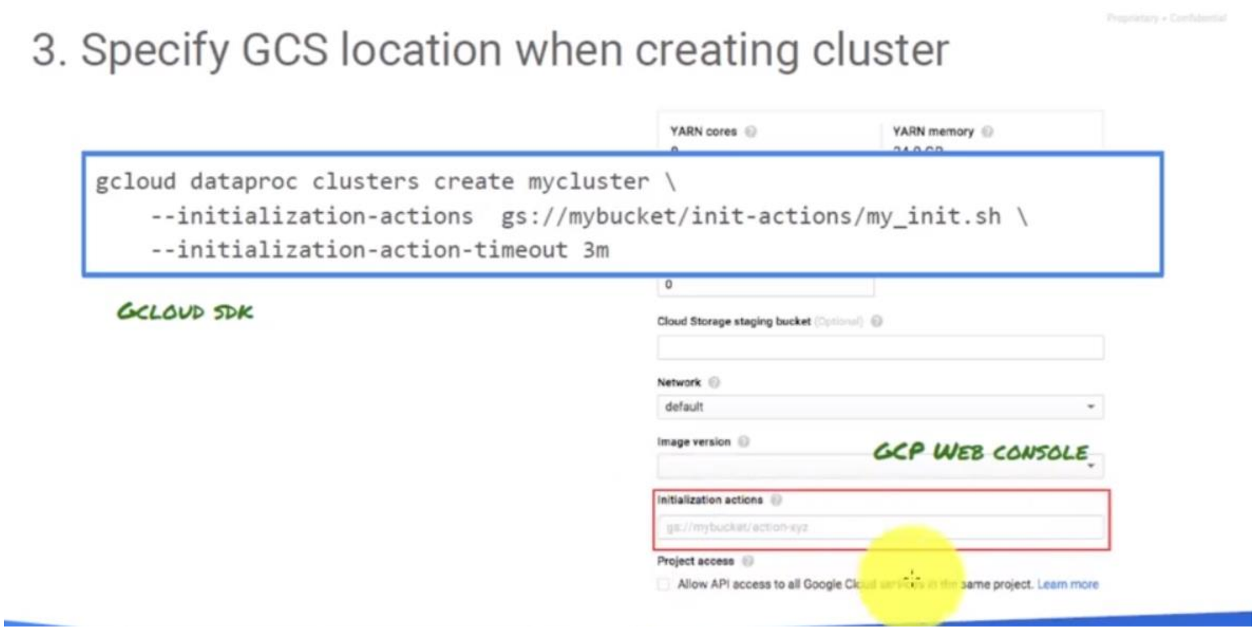
```
gsutil cp my_init.sh gs://mybucket/init-actions/my_init.sh
```

A library of pre-built initialization actions are hosted in this publicly-accessible bucket:

```
gs://dataproc-initialization-actions
```

See the GitHub repository at
https://github.com/GoogleCloudPlatform/dataproc-initialization-actions

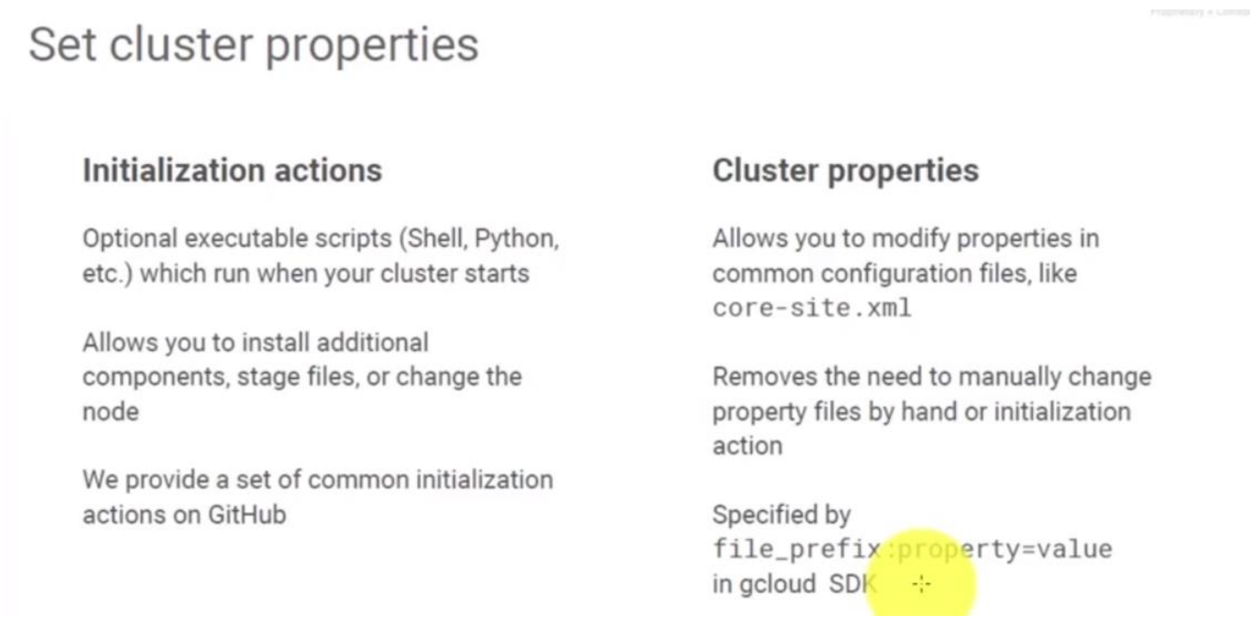
After we create the script or the program, we need to save it and then we need to upload it to Google Cloud Storage. Here we're seeing the gsutil cp command, we're taking the script we have locally called my_init.sh and we're going copy that up to a cloud storage bucket and into a folder or directory, and then the file itself. Now, what about if I want to initialize standard things like flanker, oozie? Do I really have to go write those scripts from scratch? No you don't. Google has a git repository that contains the initialization scripts for a lot of open source software that we run on Dataproc. In addition to that, Google has created a public facing project where you have read-only access to the storage bucket and it actually has files in there as well. On this GitHub site, Google has the scripts necessary to initialize Datalab, Flink, Kafka, Oozie, Tez, Stackdriver, Zeppelin, Zookeeper - you can play around with a whole bunch of initializations. And in addition to that, if we go take a look, they've also put it as I said into a bucket. Let's take a look at what's actually sitting in that bucket. I am going to flip back and I'm going to go to my cloud flat platform project, which I have over here, and I'm going to open up Cloud Shell. So Cloud Shell is going to initialize here for me and we're going to go ahead and take a look. Let me grab that other screen so I have it somewhat up on my screen here. So we want to look at that bucket called gs://dataproc-initialization-actions. So if I want to see what's actually in there I'm just going to do a gsutil ls and we're in Cloud Storage, so we want gs://dataproc-initialization-actions and we will hit Enter. Now if we have permissions to that, presumably we should get back a listing and sure enough there it all is. We've got a listing of all of the directories and the files located there. So let's say we want to initialize Datalab. Well, let's take a look to see what's in the Datalab directory. And sure enough, there is a read me and there's also a datalab.sh. Let's go and change our command to cat - gsutil cat gs://dataproc-initialization-actions/datalab - let's look at datalab.sh. There is a whole series of commands, so let's see what this is. We are running a bash script here that Google created that goes ahead and runs a series of commands to bring down the docker image for Datalab and then initialize it and make it available to us. That is pretty impressive that all that is done by just pointing to an existing script. We don't have to figure out how to initialize that - Google's going to do that for us; that is the initialization action. When we actually go to create the Dataproc cluster we can specify that we want to use gs://dataproc-initialization-actions/datalabs/datalabs.sh and we'll actually do that in the upcoming lab.



3. Specify GCS location when creating cluster

```
gcloud dataproc clusters create mycluster \
    --initialization-actions  gs://mybucket/init-actions/my_init.sh \
    --initialization-action-timeout 3m
```

This is what the actual Dataproc initialization looks like. Remember, we have created a Dataproc cluster from the command line before using the GCloud command - dataproc clusters create mycluster. I want the initialization actions to pull from Google's cloud storage, and the name of the bucket and then what we actually want to perform. In this case, we are actually adding another line where we're asking for an initialization-action-timeout of three minutes. If for some reason a step in that or the script or program takes more than three minutes it's going to be canceled. Now it's not going to destroy the cluster or anything like that, it's just not going to complete that step. But that's better than having it wait endlessly before it comes online. Now if you're going to use the web console, you can also specify the initialization action. And as you add initialization actions and hit Enter, it'll give you another blank line because you can actually support multiple initialization actions. Now, if you need to handle multiple initialization actions from the GCloud command, just put a comma after each reference to the initial initialization action. We will have one initialization action parameter, and then you'll list all of the various files separated with commas and no extra spaces, just comma between each of the gs references. Initialization actions are going to give us optional execution scripts we want to run which are going to run on each node of the cluster. We can install additional files, we can stage files, we can change the node configuration. Basically, anything you want it to do programmatically within the cluster. And Google does provide for a set of common initialization actions by the GitHub and the bucket. However, you also have the option that what if you needed to change a specific configuration property of the cluster - cluster properties. For example, there is a particular parameter you needed to add or update in core-site.xml. There is nothing to say you couldn't do it through initialization action but to facilitate it and make it easier, Google allows you through cluster properties to set the file and its prefix, colon and then a property in what you want the value to be. This can be done through the Cloud SDK or the gcloud command line. Right now, it's not in the graphical user interface.



Set cluster properties

Initialization actions

Optional executable scripts (Shell, Python, etc.) which run when your cluster starts

Allows you to install additional components, stage files, or change the node

We provide a set of common initialization actions on GitHub

Cluster properties

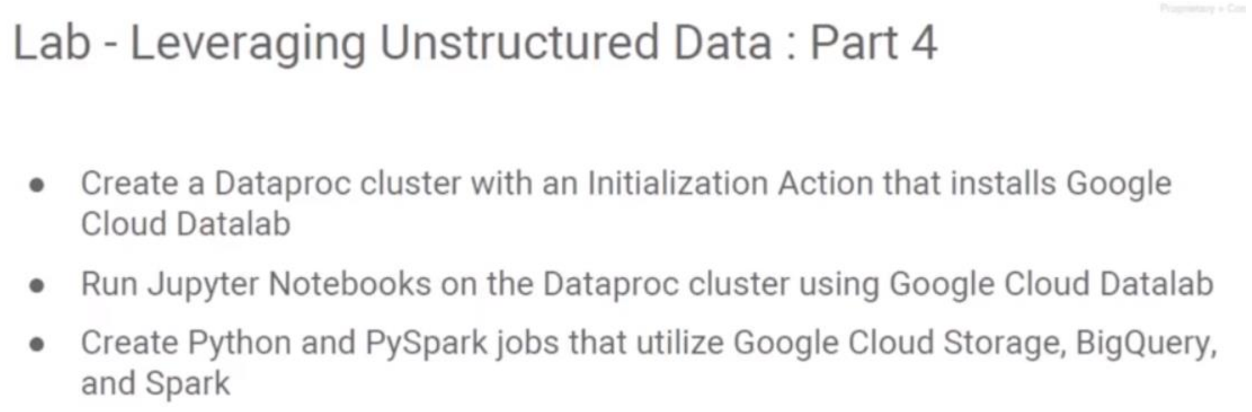
Allows you to modify properties in common configuration files, like `core-site.xml`

Removes the need to manually change property files by hand or initialization action

Specified by `file_prefix:property=value` in gcloud SDK

Head over to quick labs and complete the lab on Leveraging Unstructured Data: Part 4. This will have you create a new Dataproc cluster. It'll initialize Cloud Datalab on there, so that you have Datalab up and running. You will leverage the code to have it install using the GitHub or that public bucket initialization script. Plus, you'll create your own initialization script to put some other components on there. Then, you'll work with a Jupiter notebook to create a Python and PySpark jobs that will utilize Cloud Storage. They'll run a BigQuery operation and run some Spark on your Dataproc cluster. Now, one quick thing is when you're utilizing your project for this lab if you haven't gone into BigQuery and initialized it, basically open BigQuery and confirm any prompts. You want to do that before you run the lab so that way the BigQuery commands will run successfully.

Lab Overview: Leveraging Google Cloud Platform Services



Lab - Leveraging Unstructured Data : Part 4

- Create a Dataproc cluster with an Initialization Action that installs Google Cloud Datalab
- Run Jupyter Notebooks on the Dataproc cluster using Google Cloud Datalab
- Create Python and PySpark jobs that utilize Google Cloud Storage, BigQuery, and Spark

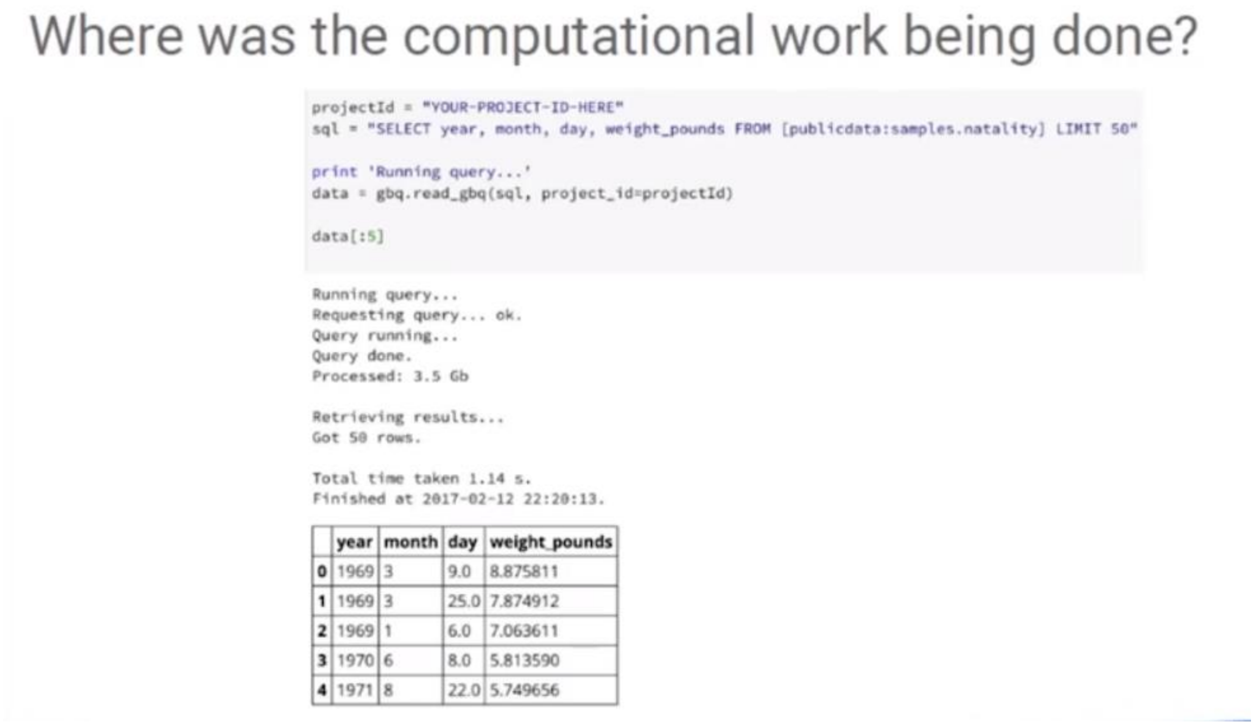
https://codelabs.developers.google.com/codelabs/cpb102-dataproc-with-gcp/#0

Review: https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/0ddz5/lab-review

Big Query Support: Why Process Data in Spark

Let’s say you want to process some BigQuery dataset in your Spark cluster. Why would you ever want to do that? Remember that BigQuery is serverless, so you can run a query on BigQuery scaled out to thousands of nodes for a few seconds and come back to you with the answer. That is a whole lot more scalable than doing some similar thing with Spark SQL for example. Why would you have data that's in BigQuery and process it using Spark? It’s not really a good thing for doing SQL statements but maybe what you want to do is that you want to run a Spark machine learning program. And at that point because you want to use the Spark library on data that exists in the BigQuery, you need to have this connection between a BigQuery dataset and your Spark program.

BigQuery Support



Where was the computational work being done?

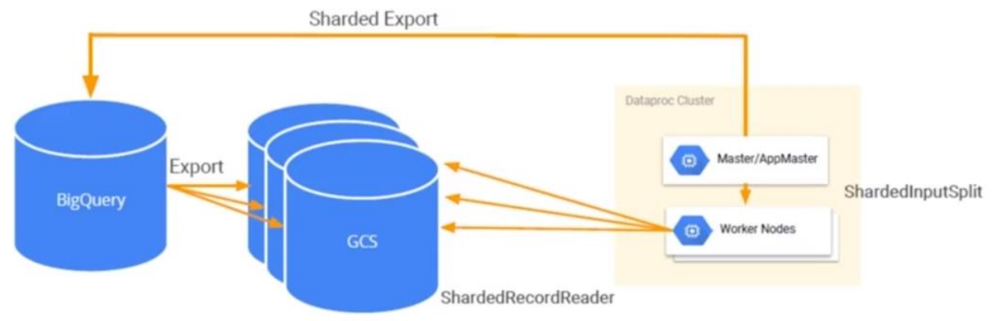
```
projectId = "YOUR-PROJECT-ID-HERE"
sql = "SELECT year, month, day, weight_pounds FROM [publicdata:samples.natality] LIMIT 50"

print 'Running query...'
data = gbq.read_gbq(sql, project_id=projectId)

data[:5]
```

```
Running query...
Requesting query... ok.
Query running...
Query done.
Processed: 3.5 Gb

Retrieving results...
Got 50 rows.

Total time taken 1.14 s.
Finished at 2017-02-12 22:20:13.
```

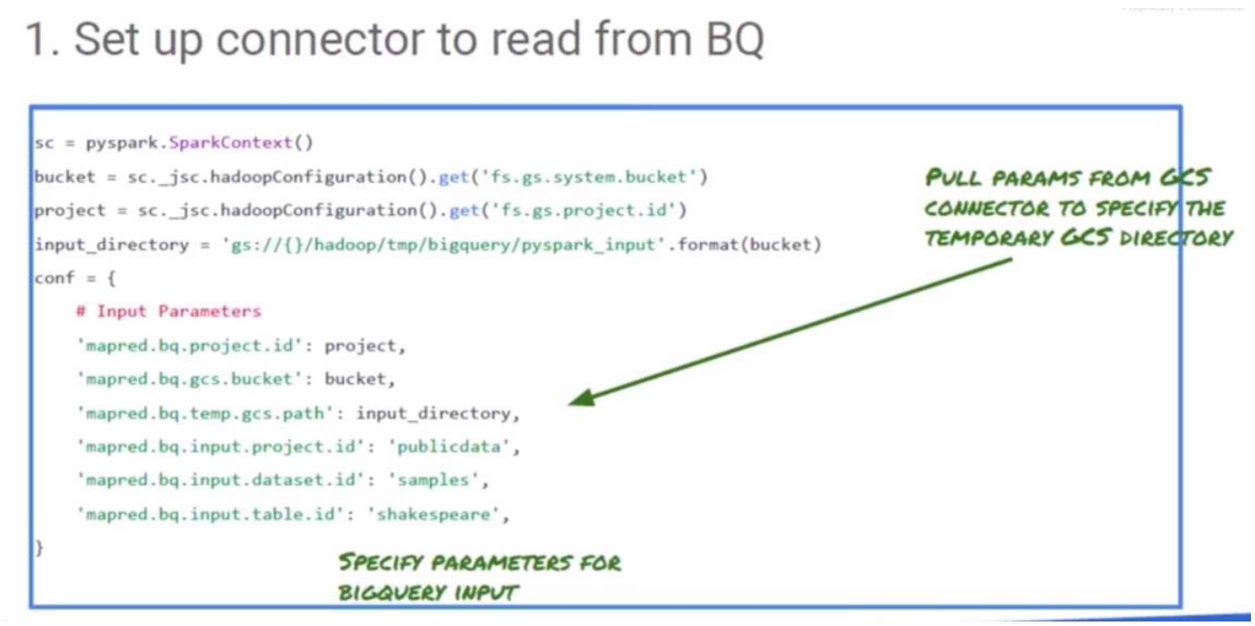
	year	month	day	weight_pounds
0	1969	3	9.0	8.875811
1	1969	3	25.0	7.874912
2	1969	1	6.0	7.063611
3	1970	6	8.0	5.813590
4	1971	8	22.0	5.749656

Let's discuss BigQuery support and the integration with Hadoop and Spark when running on cloud dataproc. In the last lab, we were working with the Jupyter Notebook and where was the computational work actually happening? Well, it depends on which part of the Jupyter Notebook we're talking about. When you were running the BigQuery portion, the Jupyter Notebook was just submitting a query over to BigQuery, and that all happens on the Google side, and then the results were passed back to us. If we're running some of the NumPy or SciPy or the Pandas code, well, that would all execute on the master node just as Python code. And, if we were executing into the Spark, that was actually being passed off to the workers and the worker nodes were executing that. There is work happening everywhere.

Hadoop/Spark jobs can read from BigQuery, but go through a temporary GCS storage



How do I directly integrate BigQuery with Hadoop and Spark jobs? Hadoop and Spark jobs don't know how to communicate with BigQuery. The answer to that is to actually use Google Cloud Storage in the middle. The idea is you run a BigQuery job and have the export store the results to Google Cloud Storage. Then you bring the output into Hadoop via the Google Cloud Storage. Likewise, the output from our worker nodes is stored into Google Cloud Storage in a format that is suitable to import into BigQuery. Is this new – no. Most of the time when we're using Python or Spark over an old traditional on-premise, we would use the Hadoop file system as that middle ground. Because that Hadoop file system basically is part of the cluster node, we want something that's happening external to the cluster because BigQuery is going to store its results to BigQuery. It really doesn't know how to work with Hadoop file system. Likewise, we want the results to be available at the end of our Dataproc job so that we can clear out and destroy the Dataproc cluster when we're done with it. No reason to keep it around if we're not using it when we can initialize a new one in 90 seconds or less.



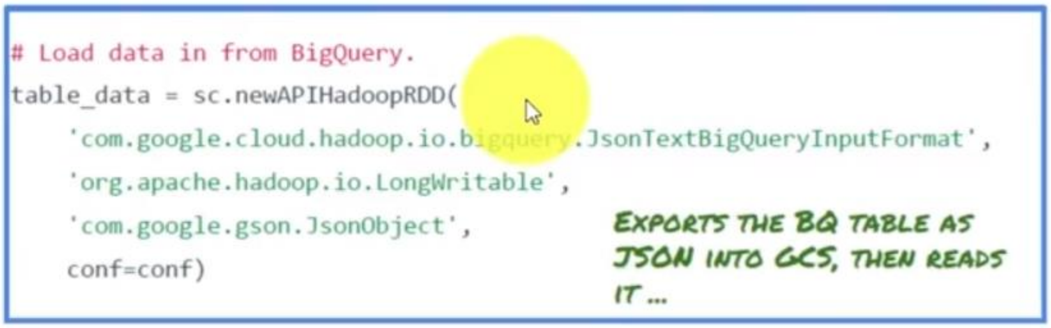
1. Set up connector to read from BQ

```
sc = pyspark.SparkContext()
bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
conf = {
    # Input Parameters
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'publicdata',
    'mapred.bq.input.dataset.id': 'samples',
    'mapred.bq.input.table.id': 'shakespeare',
}
```

PULL PARAMS FROM GCS CONNECTOR TO SPECIFY THE TEMPORARY GCS DIRECTORY
SPECIFY PARAMETERS FOR BIGQUERY INPUT

First thing, assume that we have put all the data from our big query results into cloud storage. That's outside of the data proc cluster and it is really easy to do. We split the data out of big query. Now we have to set up a connector within our Hadoop framework to basically go bring from the input from the cloud storage. Note we have Spice marketing getting some initialization, we've got a bucket so we're identifying the buckets that contains the data. We identify the project we're using, the input directory that's going to be made up of the cloud storage bucket, and the appropriate path to bring in the input. We have the project, the bucket and the input. We don't call the big query from here, because we assume BigQuery already has done it, BigQuery dropped its data in an appropriate format into the bucket.

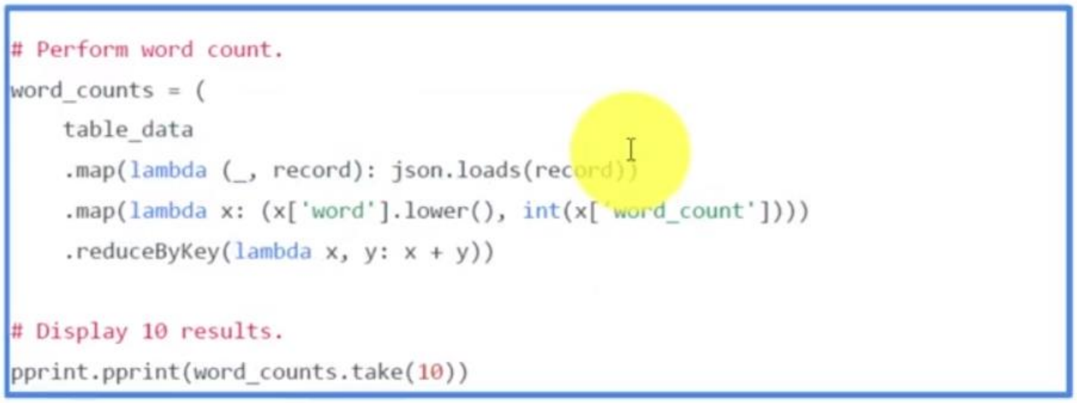
2. Load data using the BigQuery connector as a RDD



```
# Load data in from BigQuery.
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf)
```

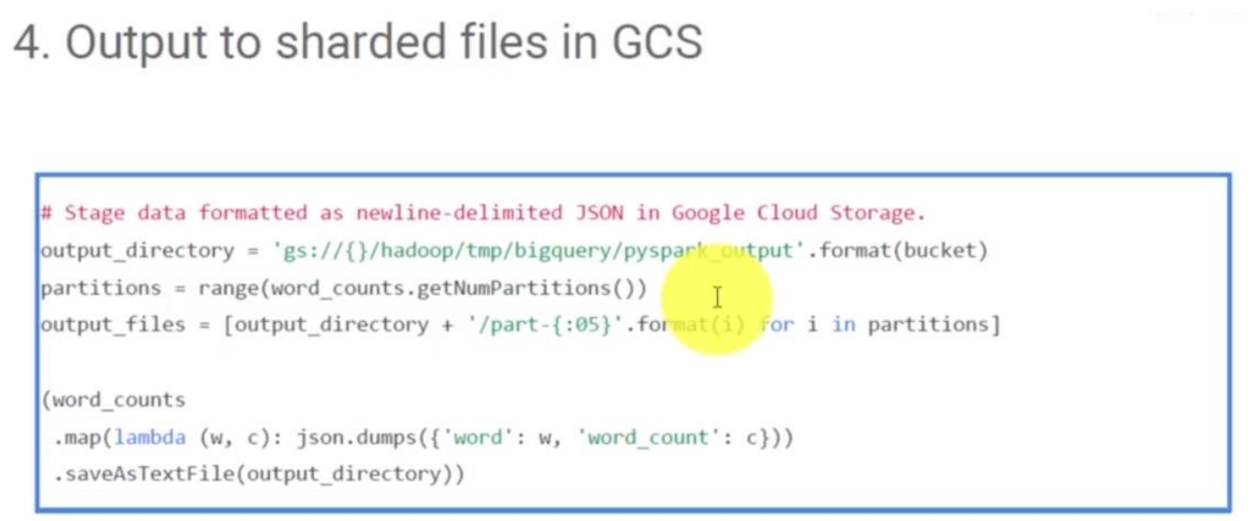
The next thing we're going to do is, we're going to load that data into Hadoop. We are going to load it in as a resilient distributed data (RDD) set. It is JSON data we import and Hadoop knows how to interpret it. We might see a reference here to a BigQuery, but Hadoop isn't directly knowledgeable about what big queries doing, it imports a class, pulling in the BigQuery input format of our data. But it doesn't actually execute the query. The query was done long before that.

3. The Spark code is as normal



```
# Perform word count.
word_counts = (
    table_data
    .map(lambda (_, record): json.loads(record))
    .map(lambda x: (x['word'].lower(), int(x['word_count'])))
    .reduceByKey(lambda x, y: x + y))

# Display 10 results.
pprint.pprint(word_counts.take(10))
```

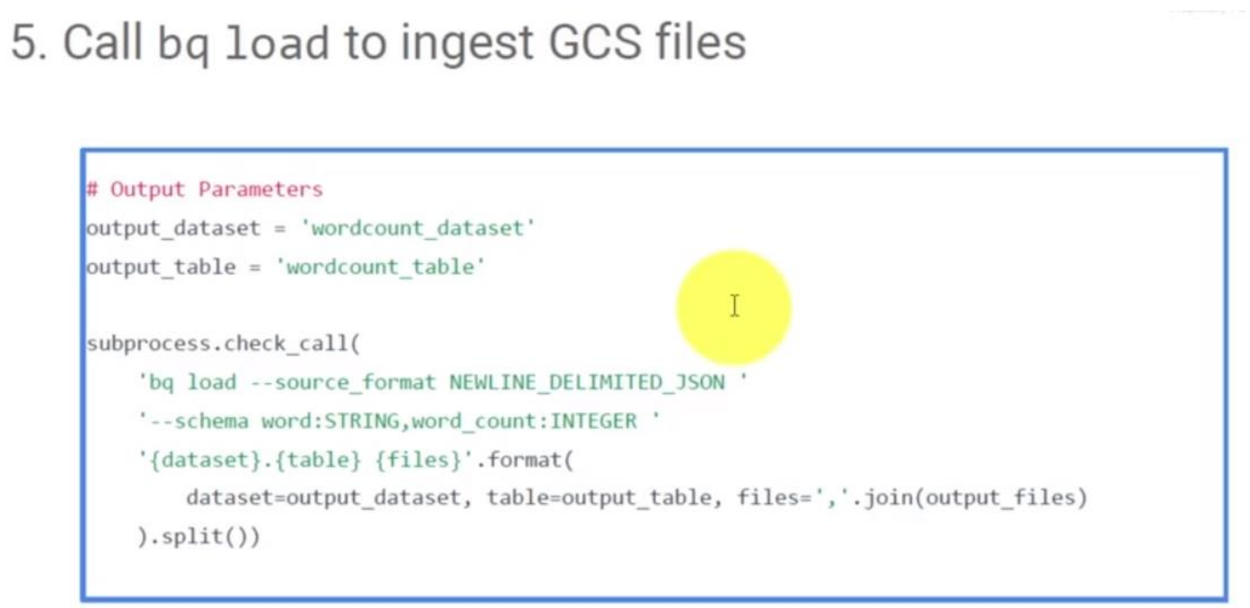
The spark code is going to be the same as usual, list is going to work with the data set that we just loaded in. Consider the example where we do word counts. We have functions to load the records and perform a word count and then a reduction. Because that's going to execute on Sparc, the output is going to be shorted into multiple pieces to be sent to club storage.



4. Output to sharded files in GCS

```
# Stage data formatted as newline-delimited JSON in Google Cloud Storage.
output_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_output'.format(bucket)
partitions = range(word_counts.getNumPartitions())
output_files = [output_directory + '/part-{:05}'.format(i) for i in partitions]

(word_counts
 .map(lambda (w, c): json.dumps({'word': w, 'word_count': c}))
 .saveAsTextFile(output_directory))
```

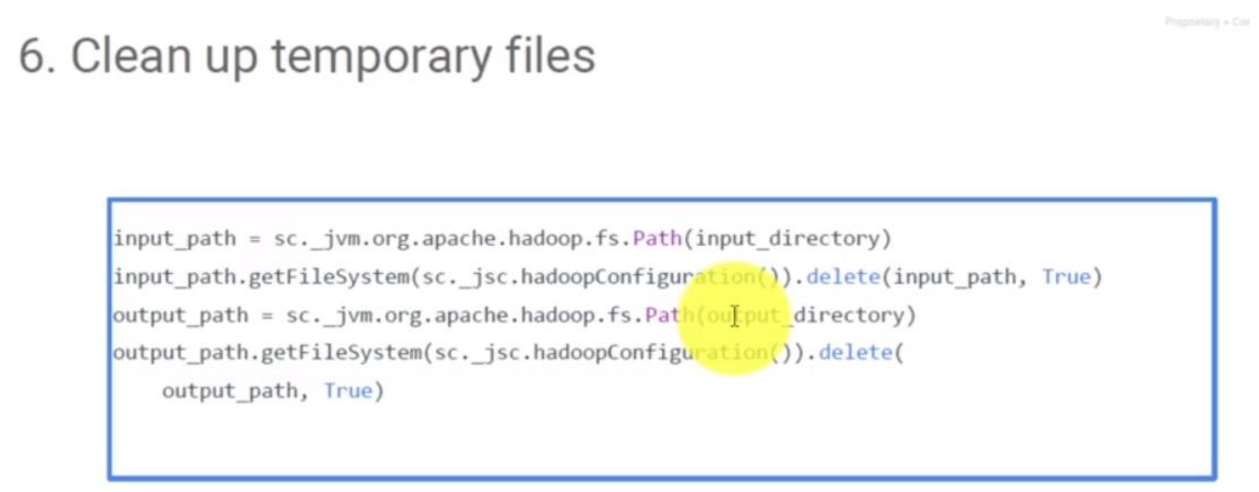
Why? Well because, SPARC is parallel processing it, you would not want them all. All the results trying to go back into the same file, because cloud storage is an immutable blob storage in that, each operation would replace the file that was there before. It you are a pending data what an append operation looks like with something like cloud storage, is you have to read the data that is there, append it to the file in memory, then write it back out to cloud storage. And if you've got four or eight or more nodes all doing that simultaneously, you're going to wind up with too many cooks in the pot all changing the same file. What you want is xargs, you want each worker to basically work with a different file, and you'll wind up with a series of files in cloud storage instead of just one. You end up with part zero of ten, part one of ten, part two of ten, part three of ten, etc. Then collect that information in the end.



5. Call bq load to ingest GCS files

```
# Output Parameters
output_dataset = 'wordcount_dataset'
output_table = 'wordcount_table'

subprocess.check_call(
    'bq load --source_format NEWLINE_DELIMITED_JSON '
    '--schema word:STRING,word_count:INTEGER '
    '{dataset}.{table} {files}'.format(
        dataset=output_dataset, table=output_table, files=','.join(output_files)
    ).split())
```

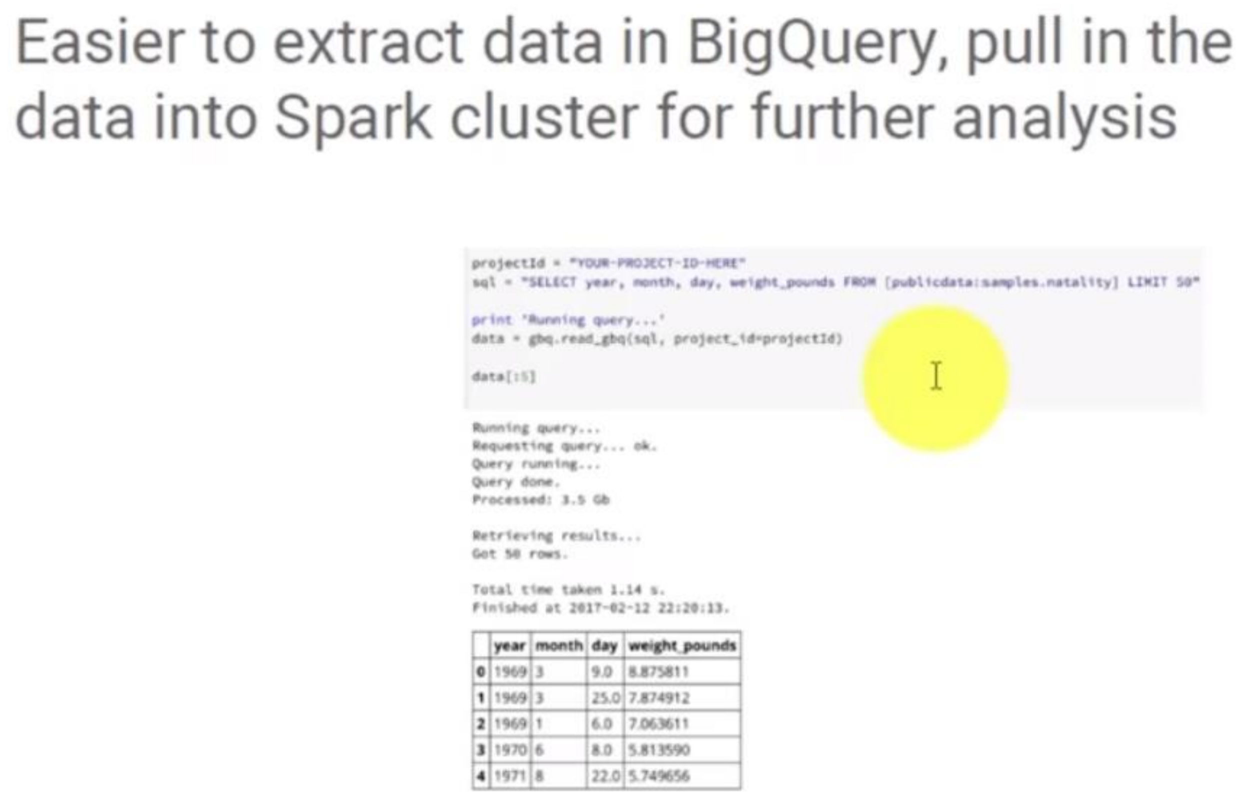
We have our output being started being put out to the output. Then, if we want to bring it into BigQuery at this point, we will import the data in, and based on how we output it to the cloud storage, the shards are import and a schema is applied resulting with the data now in BigQuery.



6. Clean up temporary files

```
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)
output_path = sc._jvm.org.apache.hadoop.fs.Path(output_directory)
output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
    output_path, True)
```

When we're done, then we should cleanup. We're going to want to clean up our files, because we now have the result in big query, we don't need the temporary files that we created along the way. Often, it's easier to extract data in BigQuery, pull the data into the Spark cluster for further analysis, and then possibly put it back out to BigQuery if you can't do everything you want in BigQuery.



Easier to extract data in BigQuery, pull in the data into Spark cluster for further analysis

```
projectId = "YOUR-PROJECT-ID-HERE"
sql = "SELECT year, month, day, weight_pounds FROM [publicdata:samples.natality] LIMIT 50"

print 'Running query...'
data = gbq.read_gbq(sql, project_id=projectId)

data[:5]
```

```
Running query...
Requesting query... ok.
Query running...
Query done.
Processed: 3.5 Gb

Retrieving results...
Got 50 rows.

Total time taken 1.14 s.
Finished at 2017-02-12 22:20:13.
```

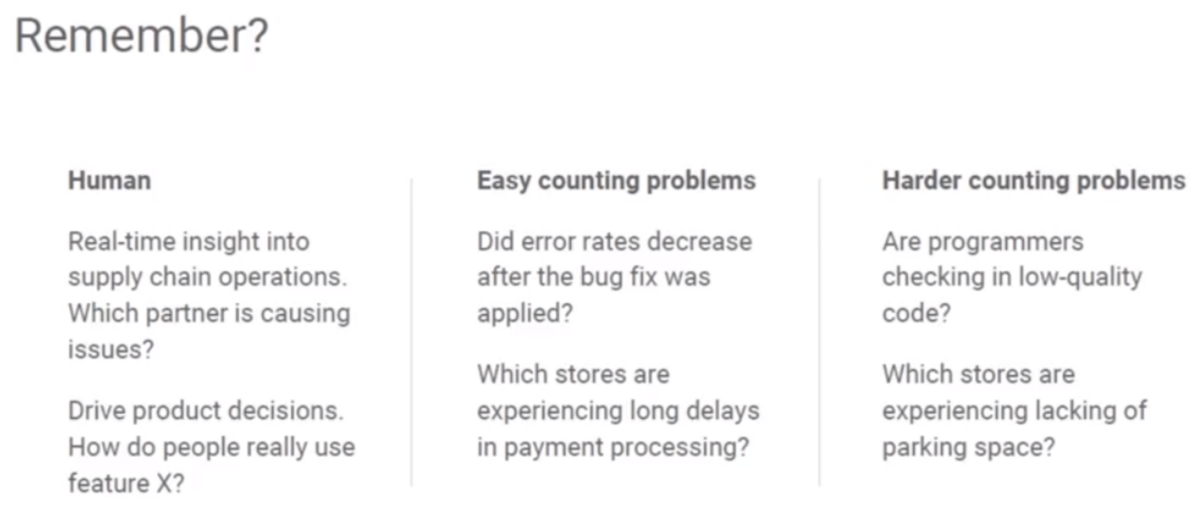
	year	month	day	weight_pounds
0	1969	3	9.0	8.875811
1	1969	3	25.0	7.874912
2	1969	1	6.0	7.063611
3	1970	6	8.0	5.813590
4	1971	8	22.0	5.749656

Tips for Interacting with BigQuery

The easiest way to interact with a BigQuery dataset from within Spark is to essentially use the BQ package in Python to execute the sequence statement on BigQuery. Convert the result set to be a pandas data frame. If you have a pandas data frame, you can then go ahead and operate upon it using Spark. That is the most common and best way to deal with the integration between BigQuery and Spark. This only works if you are retrieving a subset of your entire table. If you are retrieving aggregate and computing against the aggregates, the sums and counts etc. in BQ in BigQuery, then you can send the results of the query into a panda data frame. This does not work on a complete BigQuery dataset because a pandas data frame has to fit in memory. It is unlikely that your BigQuery dataset is going to be small enough to fit in memory.

MODULE 4: ANALYZING UNSTRUCTURED DATA

Introduction to Analyzing Unstructured Data


Remember?

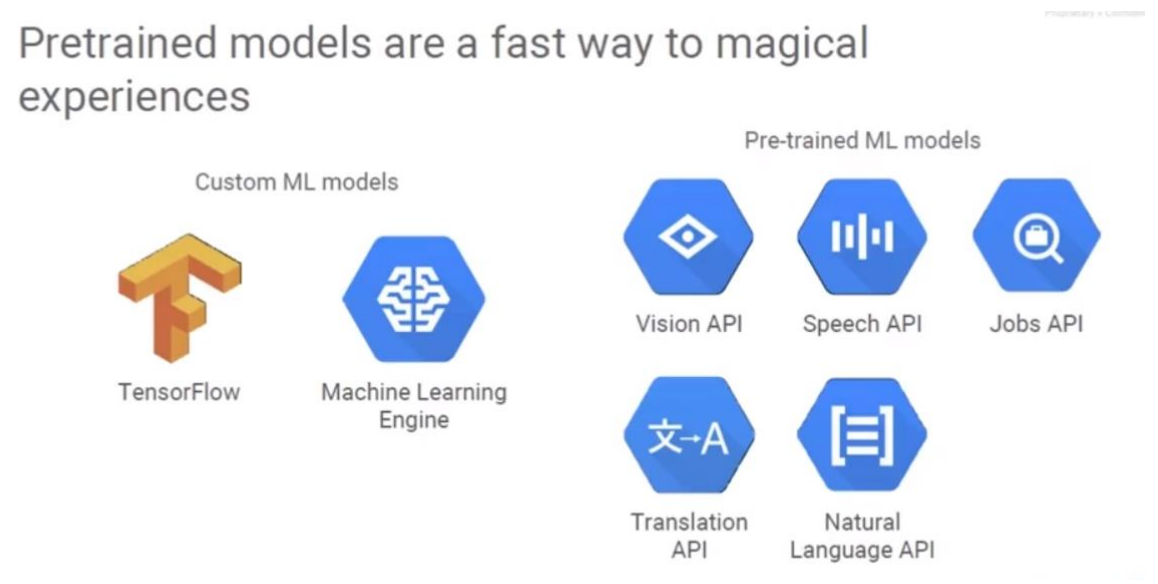
Human	Easy counting problems	Harder counting problems
Real-time insight into supply chain operations. Which partner is causing issues?	Did error rates decrease after the bug fix was applied?	Are programmers checking in low-quality code?
Drive product decisions. How do people really use feature X?	Which stores are experiencing long delays in payment processing?	Which stores are experiencing lacking of parking space?

In this lesson, we look at analyzing unstructured data on Google Cloud Platform using the machine-based APIs. This slide comes from the previous lesson on Dataproc. Humans are great at deriving insight by looking at chart and graphs and picture. Computers are great at counting things, but how do we bridge the gap between the two? That is where machine learning fits in, modelling the way humans learn, just we want to do it with computers. As was said in Wired magazine in May of 2016, soon, we won't program computers, we'll train them like dogs. Well, how do you train a dog or maybe a cat or a spouse - you reward good behavior and you discipline bad behavior. Well, it's not quite like that, it's more like how our children learn. You show them lots of pictures of dogs and tell them that they're dogs. Then, show them lots of pictures of other animals that are not dogs and slowly, their brain builds a model for it. Rinse and repeat, this time for cats and birds and snakes and cows.

Twenty years ago, we didn't have the computing power. Back then, if a lab got $100,000 grant, it could buy six servers. Essentially, you got six CPUs and built some simple learning models and crunch away at data for days upon days. But we could never really touch vision or speech, etc. These days, you don't need to build those, they are available for us to use directly as in this vision API, the speech API, the natural learning API or the translate API. And more APIs are being added all the time. You may have heard about the newly announced jobs API, which is plug and play for helping companies do hiring and resumes and job application processing.

Infuse your business with machine learning

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/F78GB/infuse-your-business-with-machine-learning



Google Cloud Platform has lots of options for machine learning, starting with APIs that do what you need. There is no need to train the system, just pass them an image, or sub text, or audio, and receive the response with the data processed for you in a structured form. For those organizations that need more than the prebuilt models, there is TensorFlow, which is an open source for seeing learning library, and Google's Machine Learning Engine and Cloud ML, and that whole packaging there. We focus on APIs that use machine learning. Let us start with something simple we can easily understand -t he Google Translate API.

This is a library of all the API's that Google makes available for Google Cloud platform projects. There is a section called Google Cloud Machine Learning, looks like a computer brain there. And there is the Vision API, the natural language, the speech, the translation, and the Machine Learning Engine APIs to use on your own models. Google's APIs are exposed via a REST interface. A web application or your internal application can make an https request to Google endpoint. They submit the data and then they receive back a block of data, much like you use a website except it's computer to computer communication. The machine translation API has five different functions available to it. It actually has language detection to detect the text within your request, and there's two variations on that. There's two variations on translating text. And then one to sort of show you the list or to allow you to query for the list of language pairs that can be translated.

Authorize Google to execute this; in the training area of Google, we simply Authorize and Execute the Google Translate and the APIs. What happened when we sent this request. We did a get Https://translation.googleapis.com/language/translate/v2. That is the URL it was submitted to and submitted the parameters q equals where is the bathroom and target equals es. If we use this in production, we would also have to provide an API key so Google knew to build this back to our project. Now that seems simple enough, it's a request and down here is the response we received back.

This is an example of how to use the Translate API, but this isn't the only way it can be used. This is an application that requires the assembly of an URL, submission to the translation.googleapi.com website as a web type service or a RESTful service and gets back a block of data along that same pathway. It's very firewall-friendy - it's just an https request.

The Vision API, that's going to be a little trickier because we have to pass it a graphic. In the case of Vision, we're going to upload the image to Google Cloud Storage bucket, and then submit our request pointing to that image. The Google Cloud Vision API exposes one RESTful service input. The payload returned is very rich with data.

The last part is to look at the JSON data. There is a block of data that's usable for our programmers. We use unstructured data in the form of a photograph and turned it into some form of structure.

This is not a massive undertaking as far as software development goes we leveraging the APIs. For example, it is not necessary to build the model, Google already did that. That is the key about the machine-learning based APIs: Google has basically run the millions upon millions of images through Google's machine-learning algorithms to basically come up with all of our labels and categorizations, etc.

What does this let us do? Unstructured data is everywhere, images, audio, video, free form text. We process them through a machine learning API call and we receive structured data in the form of places or labels or people or events or texts, etc. It makes the data a lot easier to use. The speech API is just as easy to use as the Cloud Vision API or the translation API. Speech can be submitted either synchronously - pass it a file and you get back the text, or asynchronously. There's actually a great demo up on Google Speech API site, where you can speak into your microphone, and Google in near real time will translate that back as to what you're saying - take speech and turn it into text. You can create bots which process the things you say and the system will then perform the task that it identifies you want to do.

With other various APIs, Google leverages the vision API. For example, for conference rooms, Google have something called Meeting Nanny and in monitors of a conference room is actually in use. Another service called Ocado uses natural language processing to route customer service emails better. Wootric collects both numeric data and qualitative data from feedback surveys and processes the data with the natural language API. Wootric uses the NLP APIs and custom machine learning models to get sentiment, to get the topic and possibly even the support personnel if there's names mentioned in the text.

Lab – Leveraging Unstructured Data, Part 5

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/SmACx/lab-review

https://www.coursera.org/learn/leveraging-unstructured-data-dataproc-gcp/lecture/SmACx/lab-review