Male Speaker: IBM Cognos 8 Deployment Best Practices for Performance and Scalability written by Mark McFadden and Dean Browne. So, today we are going to discuss quickly Cognos 8 architecture, some deployment drivers, why you would choose one deployment over another, what are some of the options you may want to choose, and discuss quickly some basic *tunings[Phonetic]* just for Cognos 8 software. Cognos 8 is based upon a service oriented architecture, which allows each service to be broken into its own functional area and role. This allows for easy distribution, for redundancy, security or scalability and easy separations of the security in the data layers. This slide here demonstrates basically our three tiered architecture. On the top of the slide, you will see client based layer where you would have such things as Controller, Framework Manager, Transformer, Planning. Then you have your web server layer where you have your gateway and possibly a load balancing software dispatcher, you could have routers and firewall between each of these layers and you have your application tier; in your application tier, you have the various services which do the bulk of the work, data movement services, the *courier[Phonetic]* services which handles the MDD, MDX, the SEQUEL calls for LTP or OLAP data sources, your presentation layer which handles the rendering of your data, the jobs and scheduling where you can do a lot of batch bursting and scheduling, your content managers service, which manages your content, data storage, and various other services. Then you have your data tier, which can be separated by various routers and/or firewalls if you like. You can have encryption built between that layer for further security in that layer typically of the content store, the database, which has all the information and security about the users and the applications as well as your read and write OLAP data sources and relational data sources. Quick description of the basic Cognos 8 components and what they typically do. The gateway, so the bulk of the work done by gateway are things such as encrypting passwords to ensure security, extract information required to submit request to the Cognos 8 server, attaches environment variables to the web server as a default name space to the request to make sure that security authentication is performed, pass the request to the Cognos 8 Dispatch for processing via gateway or load balancing dispatcher. The Content Manager is responsible… basic responsibilities are things such as security, configuration of data, models, metrics, report specification, report outputs. The dispatcher which does the peer-to-peer communication between different application layers is a Java based service. It self registers every 30 seconds or so with the Content Manager and information is pulled to the Content Manager. So, speaking of the dispatcher, as I said before, the dispatcher is a multithreaded Java process. It's responsible for the routing of requests throughout your tier. So, typically you will have gateway server or servers on the gateway; typically, we would recommend using *[Inaudible]* software load balancing dispatcher that comes with the product; that dispatcher relative to calls in a round robin format although with some of the newer versions of Cognos 8 such as Cognos 8.4, the algorithm has been changed slightly and you can utilize a slightly smarter round robin dispatching that actually pulls each individual dispatcher to make sure that it is not overloaded before it sends requests. This can eliminate such things as server processing capacity analysis, where if you have a heterogeneous report server and server type of infrastructure, you may have to spend time determining do I give two requests to this server and one to this server because of the power or the speed of the server. This new algorithm will actually be able to replace that type of process. So, the BIBusTKServerMain, we are going to discuss here as one of the

most important processes of Cognos 8. It's the main server process of the report server. It defaults when you install the product to two BIBusTKServerMains per report server. That can easily be modified and often it can be increased for greater throughput for requests. It's *spawned[Phonetic]* by the Java dispatcher and it can be *[Inaudible]* to the configuration settings. This process is responsible for creating, maintaining, and releasing the connection pools of the various data sources. So, the BIBusTKServerMain threading model can be considered, each thread considered a unit of work. Basically by default, a BusTKServerMain has four low affinity threads and one high. The low affinity threads do the bulk of the work; they typically take the request and they perform functions such as report execution that are creating through studios. The high affinity thread usually is given requests that you want to put back to the same via BusTKServerMain that issued the initial requests, things such as you finish… you are executing the report, you finished looking at it, you want to click return to portal, you want to make sure all your connections that you have utilized are cleaned up; obviously, you want to send that back to the same BIBusTKServerMain that we initiated the whole request. So, that's high affinity call or such things as a page down into report, much caching might have been done by the initial request. This will improve performance. So, we will try to push this as a high affinity thread. So, the BIBusTKServerMain, we are spending a bit of time on it because it's one of the components that can easily be scaled out to improve performance. Some of the *interpolicy[Phonetic]* requirements that *may spawn[Phonetic]* you to choose a different type of deployment, are usually driven by two main factors. One would be a pure load/performance requirement and the other one might be a load segregation due to various reasons whether it's segregation of data, due to business units, the security of that data, maybe this financial data doesn't want to be seen by others, maybe some HR data that doesn't want to be seen by the masses; these are some of the main… probably the two main reasons why you would choose one deployment over another. In this particular scenario, it's basically a single platform. Typically, in this environment here you have… it's driven by *raw[Phonetic]* performance scalability. You will note here that we have a gate in our web server layer, we have a gateway; you get multiple gateways; the gateway you request comes out. You need to go straight… usually goes into the... it can go to the Content Manager. As you here, there is only one Content Manager because there only could be one live Content Manager at any given time though you may have multiple redundant Content Managers to *fail over to[Phonetic]*. Your report server, *there can be many[Phonetic]* report servers and typically we do distribute our report servers because they are very, very easily distributed out due to our SOA architecture. The routing of the calls can be driven to specific report service if you so choose; say for instance you… you have PowerPlay 8 and you want specific servers to run PowerPlay 8, you can force that type of routing. This diagram here is just a, you know, a quick graphical diagram of a simple configuration built primarily for scalability performance that I set up in 2008 to measure scalability on AIX 6.1 platform on a Power series hardware. So, as you can see here we have multiple gateways with load balancing dispatches on each. Often, *if you have[Phonetic]* very, very high scalability you may need multiple gateways, not because they necessarily require a lot of hardware, lot of resources, but there often are connection or thread limits to a gateway that one has to consider. We typically don't you recommend using the CGI single threaded process because multiple CGIs can be spawned consuming excessive

memory for no reason. We typically prefer to recommend multithreaded web servers such as *ISAPI or Apache mod[Phonetic]* for better scalability using less resource consumption. The report server/presentation services application tier is really working most of our horsepower, most of the CPU, most of RAM goes there because there that's the area where we can spawn multiple instances and get greater scalability. We can scale up and scale that within a system relatively easily. The Content Manager layer is the layer that manages the content store data. We typically recommend one CPU or one core of Content Manager hardware to four of report servers; 4:1 ratio, and then of course we had our data tier separated from the rest of the application tier. So, here is a diagram basically that promotes one platform, but broken into different groups. So, what might drive something like this is that you have multiple business units. They want to share the costs of the administration and the hardware. So, they are broken and separated based on their security. So, their routing requests are based upon the security access of each individual group, gets routed to the gateways, maybe the same gateway, maybe different gateways and then depending on the security clearance, they go through different report servers if you like. You can route them through the same report server and based what they can see… based upon their security policies, where based upon they can see, they only get routed request to specific report servers, which could go to two different data layers. A good example of this might be that you have you know an executive group, high level execs, that may have one service specifically in an environment they want to have so that systems always available just for them to get quick access and great performance or you could have… for instance, you would only have report server AA we will say and all you guys go to that one and you only add on more servers if you started to increase the number of users, you can add on more servers. This one here is a multi platform, the environment where basically the platforms are almost 100% separate from each other based upon users and this particular case, they have their own individual Content Managers. The only thing to make sure in common is their LDAP servers with different user groups in that server. So, this again… *this is one[Phonetic]* you may have *[Inaudible]* this might be based upon hardware costs, administration costs, you know services' maintenance, software. This might be an excellent example of a production and a development environment. So, you may have a smaller development environment, but before you go live to your production environment, you want to make sure that everything is tested, it runs fine, then you will swap over or you may have a hot and cold standby, you may have your hot environment you worked on with a backup on either side. So, there could be many reasons, but typically you are looking at two separate environments here, but you quickly switch over from one to the other. So, we will discuss quickly some basic tuning suggestions for Cognos 8 software. So, some of the things we have to cover are some common questions that I get in performance is, "What can be quickly tweaked to improve performance in Cognos 8? You know what are the buttons that we can hit to improve performance and *knobs turned[Phonetic]*?"*11:37* Second is "When shall I scale up or scale out on my hardware to get an improved performance or larger user community to have good performance and how do I build you know some fail over and redundancy in my systems?" So, one of the easiest and first place *[Inaudible]* the way to look at is, is really modification of the report server. So, here are some of the basic tuning ideas that I think about on the report server layer. As we discussed previously, the BIBusTKServerMain, which is a process that does the

heavy lifting of report execution, ad hoc querying, how I look at that typically is I will look at the available memory in a system and say in this case, there is a example here, the system has 32 gig of available memory, I subtract off two gigs for the maximum size of a Java application, for a *[Inaudible]* application it is two gigs, so I subtract that off if you have one instance. I divide that by the number of… by the general size of my BIBus. So, I have run my application for a few hours, few days in steady state. I will look at the average size of my BIBusTKServerMain. Say it's 500 or 600 meg, so I take the high number, 600 meg in this particular case, do the quick math and left with 50 BIBus that I can spawn without actually chewing up all the available memory of my system. Now, to be safe, I probably take an additional two gig from that just to make sure that I don't get anywhere where I need to be swapping the disc, but in anyway you would be left with now 45 to 50 BIBusTKServerMain that could potentially spawn on the system. So, if you multiply the number of BIBusTKServerMain by the number of low threads, that comes to… in this case it's 50 times four, giving you 200 concurrent users. So, that's the number of users that you can run and executing reports and doing the heavy lifting that you typically do with Cognos 8 at any given second on your system without queuing. This doesn't take in consideration pure Java calls such as log in, log out, navigation or viewing safe content. Java tuning - one area over the years that we have noted that really improved scalability is understanding a bit about your application servers and a bit about tuning the java. So, in most application servers such as you know IBM WebSphere or BEA WebLogic, your thread pool web container, MAX, is typically default to 50; 50 threads can quickly be consumed by *a product that has fast response of this[Phonetic]* as Cognos 8. So, after you know 20, 30 concurrent users, often I have seen customers *[Inaudible] where their threads are exhausted[Phonetic]* and they start to receive time outs and poor performance and often not with great amount of information *[Inaudible]* to explain why. By modifying that default thread pool to say a number like 500, which we usually increase to... I have been able to scale up to you know in the order thousands of users, concurrent users, without exhausting threads, so that's a thing someone might even consider when they are tuning their application. As well, when you are tuning an application such as WebSphere or you know you are going to a new… newer version of an application server such as WebSphere from 6.0 to 6.1, you can often run the issues, known issues, because you know the basic algorithm for those software may have changed. So, in this particular case here, we went from WebSphere 6.0 to 6.1 and we noted we are tuning up lot of memory and there appeared to be like a memory leak, but actually what it was that if you didn't set the web container channel right types to sink, it wasn't handling the large SOAP *bodies[Phonetic]* that are used in some of the calls of Cognos 8 very well. So, by setting that to sink, it reduces the size of those SOAP *bodies[Phonetic]* and increases overall performance as well keeps the memory down to at least flat memory footprint. As well, a fairly commonly known performance improvement can be setting your JVM heap max and min to equal size to minimize the amount of effort to increase the memory as well as decrease of the garbage collection at the Java layer. This does have definite performance improvements, but the caveat I must mention is that as you scale some high user loads often, there is not enough memory available for the JNI calls that are into Cognos 8 code and that can cause some issues; so, you may want to consider dropping that if you are scaling very, very high and *run into those[Phonetic]* of issues. On the 64 bit *hybrid[Phonetic]* Cognos 8 have done some

testing and we have been able to significantly increase the size of the min and max heap settings to you know in the order of two gig to eight gig or even higher on some of our layers and this vastly improved performance in areas that are pure Java such as viewing *say it's a content[Phonetic]* and this was most likely due to the decrease in garbage collection and the locking of the memory during the java garbage collection. As well, it's always good to keep your JRE up to the latest and greatest for the... your application server for your JVM because often any old leaks… any leaks that were in some of the old code had been cleaned up and fixed. So, if you it perceivably into your Java layer by sometimes moving to the latest and greatest JRE, you can get rid of these leaks; we have seen that you know quite a few times in our testing and lastly, anytime you are going to you know scale out to better scalability and… by adding multiple instances of an application server either on a single system or distributed across different systems is really based upon the combination of response times with *resources[Phonetic]* and the user load. So, that's something you want to consider is if you want to increase your user community, you have lots of resources available, you can always add another application server instance and have you know your report servers whatever into that instance. This next slide discusses some basic tuning specific to AIX. Again, in that engagement we did in 2008 for AIX 6.1 on the Power 6 hardware, we came up with some excellent tuning suggestions along the lines of memory management and thread handling and if you note there the AIX thread underscore you know environment variables are the ones that really increased the number of threads and the use of the threads via products as well as the malloc and multi heap, increases the size of blocks of memory that was utilized by the C++ code. It did have significant improvement on our performance of Cognos 8, but what the one caveat is that the increasing the size of the memory blocks may have a negative impact if you already have existing functionality or reports that consume a significant amount of memory for your BIBusTKServerMain. This may cause you to push that memory up to higher levels, which could cause some instability in your products. So, something you might want to look at and then consider; definitely, you can improve performance, but you have always got to weigh that against your application that you are running yourself and the type of reports that you have into your system and lastly, I want to point out the *U[Phonetic]* limits on most UNIX operating systems are by default set quite low. We strongly recommend that those are increased to unlimit or very high amounts as you can see here if you expect any type of high scalability on your application. So, this slide discusses a very easy trick that we have done on Windows before that increases throughput. Again, Cognos 8 is a very fast responsive product that tends to eat up connections very quickly and release connections very quickly and on Windows by default, the number of connections are set to 2000 with a time to release that connection once it's not used anymore and allowed to be reestablished with default to two minutes. So, those settings are TCP, IP registry settings, the max user port, and the TcpTimedWaitDelay. So, we typically increase the max user port to 60… max of that system, which is *65, 534[Phonetic]* and we decrease the wait delay for the connection to release after it's completed from two minutes down to 30 seconds. This will significantly improve any type of scalability on Windows. So, this slide here I just quickly want to talk about some of the measurements and for scalability characteristics that we do for Cognos 8 products to ensure that as a user community, increase the number of users on a finite system, they can estimate the type of responses they will be getting in terms of

response times and/or if they decide to scale out and add more hardware, that that hardware will make a significant impact on performance. So, three of the main characteristics that we looked at is predictable or predictive scalability characteristics, horizontal and vertical scalability characteristics. So, predictable scalability basically is when we test with a finite set of hardware, we scale from a low user to a very high user load in proportional amounts. We measure this time. We want to ensure that our scaling, our response times to user load is linear such that at any given time we can estimate if I go from this amount of users to this amount of users, this is the amount of response *delta[Phonetic]* I will see and you know we would like to measure this until we either get a satisfactory number or we hit a point where at an inflection point, potentially the response time is *[Inaudible]* and then we try to determine where that point is and then anything under that, we try to… we can determine this linear scalability. We try to come up with a model of a user per CPU response time. The other two type of scalability characteristics is really for scaling out more and that is horizontal scalability where basically if I increase my hardware and I keep my user load constant, I should see a proportional improvement in performance meaning that if I have a hundred users and my response time is ten seconds, and if I had a second identical hardware, I would expect to see that performance to increase and the response time to decrease proportionally, so from ten seconds to maybe five seconds. We measure these type of things and we are able to achieve that type of horizontal scalability and as well vertical scalability or proportional scalability is if I am satisfied with my response times for my current user load, but I know that I am going to be increasing my user load to a certain amounts, say for instance, I am moving from 100 concurrent users to 200 concurrent users in the near future and I want to… I want to be able to maintain my responsiveness that I have currently on my system, I may consider adding a secondary piece of hardware. So, I want to make sure that if I double my hardware and I double my user load, that the response times stay constant, so that would be a measurement of vertical or proportional scalability. So, that's something that we do often here at Cognos against our products on different platforms and yes, we have noted that our product does scale extremely well in most cases in those three areas. So, in summary, I would just like to touch back. We have covered off deployments, what are the kinds of deployments that you can set your environment up with and what would be the reasons for these choices, usually they are business… to suit your business need, it might be used on security, it might be based on some cost constraints, it might be based solely on scalability and performance. We talked about easy areas you may want to go in and tune such as report server which is one of the easier areas to tune and get some of the best results from it. We discussed some specific environment modifications to various operating systems that can affect performance and scalability and lastly, we discussed a little bit of our model for scaling and measuring scalability and when to scale out on a system. Thank you very much and I hope this helps you.