**Welcome** to Module 13, Web Services. In this module, we're going to look at how we can use a Suitelet or a RESTlet as a web service. And when I say web service, that will be a script or any service off of the web that you can call so that you can pass processing through that service.

So in this module, we're going to look at how you can create a Suitelet web service. That's basically a Suitelet that does not have any user interface. We're also going to look at a RESTlet. So here we're going to also look at the difference between a RESTlet and the Suitelet, and then also how you can call a web service-- either a Suitelet or a RESTlet-- from a client-side script. The process of calling a Suitelet or a RESTlet is the same, so using one or the other would be up to you. It's important for us to decide which script we're going to use, so we're going to compare the two.

Now, what is the web service? Similar to what we saw in our Suitelet discussion, a web service is simply just a client sending a request to the server and then the server sending a response. But unlike a Suitelet, where we're actually displaying something on the screen, this is more of asking for information from the server and then the server responding in the back end.

So you can use a web service, for example, if you want to execute some of your process on the server instead of directly executing that on the client. But again, the system is pretty much the same. You would be sending a request from your client, and then the server would be sending a response.

Here are some of the reasons why you would want to use web services. One would be if you want to offload processing to another script. Let's say if you want to have a user event script. And on your user event script, maybe that's running too long. You can push that into a Suitelet or a RESTlet. So that's something that you can do. Or maybe if you have a scheduled script, you can actually call a Suitelet from a scheduled script.

Another thing that you can use a web service script for is that you can actually hide the execution from the client-side script so that it is nicely hidden in the server. Now, this is something that you want to do especially if you have very important business processes that are running on the client-side script, because maybe you have a process where you need to compute for something or validate something when you load a page, or maybe you have a validate field or save record validation that needs to happen on the client side.

If your script is on the client side, as we all know, we can just open the debugger. And since the scripts are actually on the browser, if you know what you're doing-- for example, if you're an experienced developer or you actually know how to debug scripts-- you can just open the implementation and see the source code. So if you have anything there, you can easily just change some values. Or maybe if you're hiding some values, those values will be exposed. And that's going to be a problem for you.

Now, to solve that, what you can do is you can push the processing to the server by using web services so that even if they try to look at your code they can see the code no problem, but when they look at that the only thing that they're going to see is that they're going to see a call to the server. And when you look at that, the actual process would be hidden, so it's not going to be a security problem.

Another thing that you can use web services for is to connect a third-party system to NetSuite. So just in case you have other pieces of software or other systems that you want to connect to NetSuite, you can use web services for that. There are a lot of options for web services in NetSuite, and SuiteScript in terms of Suitelet and a RESTlet is only one option.

Another option that you have for web services would be SuiteTalk. And in fact, if you try to make an integration, or if you're thinking about web services, the first thing that you would probably think of in terms of NetSuite would be SuiteTalk.

Here are some of the examples of other web services options that we have. So we have SuiteTalk, as I've mentioned. We have a RESTlet. We also have a Suitelet. And we have something that's fairly new, or at least it's relabelled. So now we have something called the SuiteAnalytics Connect. Let's start with that.

SuiteAnalytics Connect is a tool for you so that you can connect directly to the database for fetching information. SuiteAnalytics Connect uses ODBC and JDBC for your connection. So if you're developing using Java, or if you're developing using C#, or .NET, or any technology that supports ODBC and JDBC, then you can use SuiteAnalytics Connect. But do remember that SuiteAnalytics Connect is read-only. So that means that you cannot write directly to the database, but you can use that to get information.

It's called SuiteAnalytics Connect because it's mainly used for reporting purposes. So if you remember our discussion, SuiteAnalytics is a tool that is used not exactly to customize NetSuite, but it allows you to get targeted information. So saved searches, dashboards or reports, those are part of the SuiteAnalytics toolset.

Now, in terms of actual integration, you have three different options. So you have SuiteTalk, RESTlets, and also Suitelets. Now what is the difference between those three? Full integration can happen using a Suitelet. For example, you can push data into NetSuite using a Suitelet, and you can also pull data from NetSuite using a Suitelet. Now, a Suitelet has a limitation regarding the data. Suitelets would only be available publicly. And you don't have any authentication with a Suitelet.

Now, if you're dealing with a script calling a client-side script, or a user event script, or a scheduled script or whatnot, you can have that call a Suitelet without any problems because you're already logged into NetSuite and you're assumed to be using the system. So security would not be an issue there. But outside of NetSuite that can be a problem. So if you are going to use a Suitelet for integration, you have to make sure that you're exposing only data that you can expose, so things that you can do publicly.

So for example, you can use a Suitelet if you want to create a public API or a public RSS feed. So that's an example that I gave previously. So if you want to maybe allow your subscribers or allow your partners to get information about your product so that they can sell your products or they can help you advertise your products, then that would be perfect for that. So you can use a Suitelet for it.

But if you need more security-- for example, I'm going to give some partners information [AUDIO OUT] include prices or pricing information. Sometimes those are things that you don't want to share. Then you can't really use a Suitelet because if you do that, then again, Suitelet would be public. So the two options that you have now would be SuiteTalk and RESTlet. So you can use both of that for integration.

Now, this is the tricky part. When would you use a Suitelet and when would you use a RESTlet? Because you can use one or the other for pretty much anything. So if you want to have a full integration, you can have that on a SuiteTalk application or a RESTlet application. If you want to do a simple integration where maybe you're just fetching data from NetSuite and then synchronizing that with your system-- so if you have a third-party system and then somebody enters data on that system, it would also be pushed to NetSuite-- you can do that on both SuiteTalk and a RESTlet. So that, again, could be really tricky. And it's kind of up to you in how you want to decide. Now, the decision would be based on what you want to do. So let's go ahead and try to compare the two and see what is the difference between using a SuiteTalk application or a SuiteTalk integration and a RESTlet integration.

Again, with both SuiteTalk and RESTlet, they both offer authentication. So before you can connect to the server, you need to make sure that you put in your username, password, account ID, and your role. So you need to make sure you have your role enabled for web services.

Now, with a RESTlet, you can call this internally. So if you call the RESTlet internally, you don't need an authentication because you're logged into the browser and you've already signed in on the account. So that's not an issue. But if you are going to call that outside of NetSuite, that's a different story, which means that you have to assemble your own header so that you can pass in those four information-- login, password, account ID, and the role.

Now regarding the technology that these two uses, SuiteTalk uses SOAP. That's the technology that is used for a lot of web services. And this is one of the most common technologies before REST became mainstream. So with SOAP, what you have is that you have a server that would be giving you information about what are the things that you can do. Now, with SOAP, this is a format that's pretty much the same regardless of which server you're connecting. So this is an XML-based messaging system, and then you have a certain format for that XML message. So you have a header tag. And then you also have a body tag where your header contains your authentication information. Then your body tag would contain your request.

But with a RESTlet-- so the reason why it's called a RESTlet because it's based on the REST technology. So there's really no advantage for one over the other. They're pretty much the same. They're just two different technologies. SuiteTalk just uses SOAP, and then RESTlet uses REST.

Now, because of the nature of the technology, the kind of message that you're sending is also different. So SOAP, again, is based on XML. And you have a certain pattern for that XML message. But with a RESTlet, it's mainly using JSON. You're not limited to using JSON with RESTlet though, because with a RESTlet you can send pretty much any kind of data that you want. So if you want legacy SV export kind of thing, that's something that you can also do with a RESTlet. But for the most part, commonly for developers people then to use JSON because JSON is a good format. It's easy to manipulate, and it's also nice and structured. So if you are going to use RESTlet, I do recommend using JSON.

JSON is also automatically handled by the system. So if you attempt to, say, return an object-- say I try to load an employee record and then I return my employee record to the request. So if I do that, the system would automatically convert that JavaScript object into a JSON string so that you can process that in the other end, which is, again, very convenient.

Now, aside from the data type that you are going to send to the client, you also have the type of requests that you can do. In SuiteTalk you can only do one kind of request, which is a POST request. So it's always going to be a POST request if you're dealing with SOAP.

With a RESTlet you have a little bit more leeway where you can use other HTTP verbs. So you have four options here-- the GET, POST, PUT, and DELETE. This can look like an advantage. Well, this is actually an advantage over a RESTlet, but it's not that big of an advantage because, for the most part, people would be using either GET and POST anyway. PUT and DELETE are sometimes used, but they're not often used. Functionalities that are used for PUT and DELETE are mimicked inside your GET and POST calls. So people just mainly use GET and POST. Those are the two common HTTP verbs that are being used in web services.

Now, this is where the difference makes sense. With SuiteTalk, because of the nature of SuiteTalk, that means that you have a defined contract. And then with a RESTlet, since you are going to write your script, it's more free-form. Now, what does that mean exactly? What does it mean when we say that SuiteTalk has a defined contract and then RESTlet means free-form?

So anyway, let me just describe what it means to have a defined contract. Defined contract means that anything you do on the system would be provided by the server. So the server would be giving you a WSDL. That's kind of the nature of SOAP. A WSDL would be a definition. So it's basically an XML definition that tells you these are the kinds of messages that you can send that the server would understand. These are the kinds of operations that you can do.

For example, if you want to fetch a record, this is what you're going to use. If you want to search, this is what you're going to use. If you want to do anything, this is what you're going to use. Meaning everything that you want to do will be controlled by the system, or will be provided by the system.

Now, that is a good thing for you because you don't have to worry about anything. For example, if I'm building an application, I would have to build a client, meaning if I'm creating a Java application or a C# application, I will build my client who would be getting information from the server or pushing information to the server. Since SuiteTalk has already defined the contract, or told the client what it can do, then all the client needs to do is just call that server. That means I don't need to worry about anything that I need to configure on the server. All I need to do is use it. The contract is already there.

Comparing that with a RESTlet, the problem with the RESTlet, or at least comparing that with SuiteTalk, is that since it's free-form, I don't have anything on the server. My web services starts from scratch, meaning my web services in RESTlet is zero. If I want something done, I would have to build that on the server along with my client. So I'm building two things. I'm building my client, and I'm building my web service. So if I need to load a record, I don't just call that load record. I have to build a script that would be loading a record, that would be returning that record to my client. So that's double the work.

That might seem like a disadvantage, but that can also be an advantage to developers because if you have a complex call-- for example, let's say I have an operation that I need to support. I'll be searching for customer records. Then, after searching for customer records, I need to process the latest five transactions. Then, after processing the latest five transactions, I would need to get the top sales rep that has helped this particular customer. And then, after getting that information, I need to send an email to those top five employees for a prize or something.

As you can see with that, it's a little bit complicated. If I try to do that with SuiteTalk, then that will be a little bit problematic because SuiteTalk does not have that on the contract. SuiteTalk has several pieces to support that, but not that one particular operation, because that operation is kind of unique, if you think about it.

So what you can do is that, for example, let's start with the fetching of customers. So in SuiteTalk you have to perform a search, get that information, process that on your client, and then perform another operation. So maybe you're going to load the record. So again, send the information to the server. Process that on your client. And then maybe you need to perform another search for the transaction. So go back to the server, process that. And then, after that, you're going to look for your sales reps. So you go to the server, process that. And then you send an email. Actually, you can't send an email to SuiteTalk, but let's just say you can. Again, you're going to go to the server and then go back to the client.

So, as you can see, that one, single process would have to take multiple calls to and from the server. And then all of the processes, all of the heavy lifting, would be done on your client. So that would be fine, again, if you have a fast computer and a fast internet connection. But what if you have a slow computer and a slow internet connection? What's going to happen there?

So here, if you're dealing with a RESTlet, what you could do is that you can create a script that would support that particular scenario. So on the server, on the script, you can perform a search. We already know how to perform a search. We can perform a load on transactions. We already know record.load. And then we can also send email. So we have the email module. And again, everything is happening on the server.

On the client side, all the client needs to do is that, OK, this is the customer that I need. Process it. So it'll be sent to the server. The server would be doing the heavy lifting because all of the processing will be happening on the script that's running on the server. And once everything is done, the server would be then throwing the information back to the client and say, OK, this is the information that you need, and it was successful and everything's OK.

So as you can see, it is faster because you don't need to keep going back to the server, using up your internet connection. And then you don't need to worry about your client doing the heavy lifting or processing your information because the processing happens on the server. All you need to do is just send the request and wait for the request to be finished. And that's the advantage of using a RESTlet.

Now the question is, when are you going to use a RESTlet or when are you going to use SuiteTalk for integration? If you want to use a full-blown integration, like maybe you have a system that you want to connect to the server, I would recommend that you would use SuiteTalk. You always want to start with SuiteTalk. The reason you want to start with SuiteTalk is because you already have all of those contracts, or you already have everything that you need for integration. SuiteTalk is built for integration.

If there's anything that SuiteTalk cannot support-- especially for the process that I've mentioned, it's a very complicated process and it would take a while for you to do that on the client side or in SuiteTalk-- what you can do is that you can augment your script, kind of like your SuiteFlow and SuiteScript comparison. It really depends on what you need to do. If you want to have a full integration, then just do SuiteTalk. And then anything that's not supported directly by SuiteTalk, you can put that on a RESTlet. So those actually work together.

Or, if you have a smaller, specialized integration-- so maybe you just need to call the server for one specific process, and this process does a lot of things in the background. So if that's the case, then just create one RESTlet script and then have your client call that RESTlet script. Because it's just one operation. So again, that's a consideration that you need to think of-- SuiteTalk for full-blown integration, then RESTlets for more of create-your-own web service.

In this walk-through, what we're going to do is that we're going to create a RESTlet and then push the validation of our employee code on the client-side script to our RESTlet. The reason why you want to do this is that you sometimes want to hide the implementation. Maybe you have a very important validation on your script and you want to make sure that your script is secure, or any data that you have on your validation is not exposed to the user. Because if a user would be experienced in JavaScript-- let's say they know how to debug your client-side script-- they can just open that script and discover what is happening on that script.

So here, by moving the implementation on the server, even if they try to open the debugger, they would not be able to see the information. So that's what we want to do here. What we want to learn in this walk-through is how we can call a RESTlet from a client side-script.

So let's go ahead and build our script by first going back to our client-side script. So let me close all those and look for my client-side script here. And what we want to do is we want to move the validation of our save record function and look for that. So here, what we're trying to do is we're trying to validate that save record. And we kind of want to hide that x value.

So maybe that x value is a very important value. If you think about it, this can be like a coupon code. Maybe it's going to give a customer 80% discount. And we don't want to expose that on the client-side script because anybody who can open the debugger can just look at that and say, oh, OK, it looks like this is something that I can use. And I'll just copy that value and then use that coupon code, and then get 80% discount on everything. So we want to hide that validation.

So what I'm going to do here is that I'm going to create a RESTlet. I can use a Suitelet or RESTlet. Doesn't really matter. But in this case, let's just use a RESTlet so we can see how that's used. So let me open this RESTlet. And I'll name this sdr\_rl-- for RESTlet-- \_validate\_emp\_code. I don't need any modules here because I'm simply just getting the validation from our save record.

Now, because we have also added save record and validate field for the same thing, I'm going to comment out my validate field so that it doesn't conflict with my save record. For this one, I'll just grab that and then put that in my doGet function. So the doGet function will be performing my GET operation.

So here, what we're going to do is that we're going to get the parameters, so I have my requestParams object. Now, for this requestParams object I need to get my employee code value. And I will be passing some values from my client-side script to my RESTlet, and then I'm going to pass that employee code value for validation. So I can get that from requestParams. So here requestParams. Or maybe I can just make that shorter. I'll just use params. OK. Params.

And then we don't have the parameter yet. Maybe I'll just call this sdr\_emp\_code. Again, we haven't decided on that yet, but maybe let's just use this. We haven't passed any of the values yet.

So anyway, so we have our empCode here. So if empCode is equal to x, which is our invalid value-- of course, I'm not going to put in an alert statement. So we're going to keep that alert statement there. But what we're going to do is that we're going to just return a value. So if the employee code is x, we're going to say that the value is invalid. Again, we want to hide that implementation or that particular data so that you only have either valid or invalid in your client-side script.

And I'll pretty much do the same thing here. I'll use invalid. I can use any value that I want. I can return an actual JSON object, like an error message, if I want to. I can use success or fail, similar to what we did in our workflow. It's really up to you and how you want to do this.

So anyway, as you can see, our RESTlet is very, very easy. And what is a little bit different here is how you can access our RESTlet from the client-side script. Or, if you want to access that on a user event script, that's also something that you can do. But at least for this particular use case, you don't really need to put this on a user event script because if it's already on the user event script, it's already hidden from your user, so it's not really a big problem.

But anyway, looking at the implementation of your RESTlet, there's really nothing fancy here. This is something that you can easily do. The only thing that's different is just make sure you know where to get the parameter values.

Now anyway, let me upload that and go to my record here. And let me create my script record. We opened that. I don't need that anymore. And open my RESTlet here. Create my script. And again, I'll just use the same naming convention, SuiteDreams RL Validate Employee Code. And then for the ID, pretty much the same, \_sdr\_rl\_validate\_emp\_code. For the deployment, I'll just use the same title here and also the same ID.

Like what we talked about in the previous discussion, our RESTlet is very similar to a Suitelet. And as you can see, the configuration is the same. You have your title. You have your ID. If you click on Save and if you open your deployment record, you have the same thing here where you have your URL. So in a RESTlet you have two URLs. You have an internal URL, and you also have an external URL. With a Suitelet, you have to enable the external URL for you to get the external URL. So it's not default. You can still get an external URL if you want to.

One thing that I noticed, by the way, going back here, is that it looks like it didn't comment out the functions that we don't need. So I'm going to comment out POST, DELETE, and PUT. But I'm going to go back to my deployment record because I need the URLs. So I'm going back here. I'll just comment out these definitions. I don't need those. OK, so that's it. Let me upload this to make the update.

And I'm going to go back to my client-side script. And this is the key to our automation. What I'm going to do is that once I get my employee code value, I'm going to use the HTTPS module so that I can request for information from my web service or from my RESTlet. Now, going to our definition here, I'm going to go all the way up and then add my HTTPS module, so here N/https.

And we also need the URL module. Now, we're going to talk about what that URL module is in a bit. But for our RESTlet, we just need the HTTPS module. There are two modules you can use, either the HTTP or the HTTPS module, depending on the web service that you're accessing. If you're trying to access a Suitelet or a RESTlet, it's always going to be HTTPS. If you're accessing other web services-- for example, a non-NetSuite web service-- then you might be using the HTTP module. It's really up to you.

So anyway, so we have our runtime module. Store that in the HTTPS. I added an m instead of comma. And then I also have my URL module here. So here I'll add my annotation, so https, which I also named that https. And @param. I have my URL module, which I also named url.

Let's jump back to our save record function so that we can continue what we're doing. So if you want to access our operations, so here I'm going to get the response from my request operation. So I'm going to use the HTTPS module. And then, for my HTTPS module, I have four different methods that I use. I can use GET, POST, PUT, and DELETE. Now, in this case, I'm going to use the GET request. Again, it's up to you in which request you want to use. You can use GET. You can use POST. It's pretty much the same.

So on my Get request, I have one parameter, which is the URL of the request or the web service that I want the request for. Now, we already have the URL, so let me grab my URL here and open that. And I can get this URL and copy that over here. And by the way, if you are performing a request using HTTPS, you should be able to use, as far as I can remember, this kind of URL because this is an internal URL and this would work with NetSuite. It wouldn't work outside of NetSuite. This would only work if you're calling from a SuiteScript. This is an internal URL, so that should be fine.

Also, I need to add my employee code to my URL. Remember, I'm using a GET request. Since I'm using a GET request, I need to pass my parameter values inside the URL. So all I need to do is just add in a plus sign here, then add in my key. So add in an and sign. Remember, that's how you would add a request on your GET operation.

The parameter ID that I need to make sure that I use here should match the parameter that I have on my RESTlet because I'm grabbing this from my URL parameter. So what I'm going to do is that I'm just going to copy that, put that there. And let's say my sdr\_emp\_code is equal to the employee code value, so empCode. And that's about it.

So here, what we're going to do now is that once we issue this https.get command, they'll give you the response object, so kind of the same response object that you would get on a Suitelet. And we're going to use that response object for our validation. So here, response. And then to get the contents of that response, I need to extract that from the body property of your response. So here, I want to make sure that I get that so that's invalid. So if my response.body is invalid, that means that there's something wrong with my request. That's it.

So as you can see here, it's very simple. All you need to do is just call that https.get. Pass in your parameter. So in this case, I'm using GET and passing that parameter value in the URL.

So let's go ahead and verify if this is working. I'll open an existing employee record so it'll be faster for us to test. And let's open, say, Cameron Murdoch. And then, on Cameron Murdoch's account, I will put in an invalid employee code value, which is x. And if I click on Save, if everything is working, as you can see, I have a validation saying that invalid employee code value. Please try again.

Now, if I try to look at my codes-- for example, if I open the debugger, then on my debugger I have my require statement here. So, similar to what you've seen on your exercise, if you want to load your script, you can open that script in the file cabinet. Let me just close that first. And let me get path for the script file that I want to debug. So I want to debug my client-side script. So I'll just copy that. SuiteScripts, go back here to my debugging session.

And if I open my require statement, I'll replace this with my debugging statement there, or my script path. So I hope this is something that you did in one of our previous exercises. And I believe this is what we want to debug. Let me open that again. Paste that there, and remove the JavaScript extension. And if I refresh my form-- let me make this a little bit smaller. Let me refresh this.

So here in my require statement-- so again, let's try to see if this validates properly. Yeah, I've saved. And, yeah, so it's still validates properly. And let's look at the require statement here. Look like it's loading. There you go. Let me open this. And if I look at my save record function, let me open the implementation of this. You can see here-- now let's make it bigger.

There we go. If I try to look at the implementation again, even if I know JavaScript and even if I can open the source code on the client side, all I'm seeing is that it looks like I'm just accessing our web server. And I don't really know the value. So this would be very important if you're dealing with things that you need to hide from experienced developers, or from power users, or anybody who has experience with JavaScript. So that's pretty much it.

But one other thing that can be improved upon here is that this URL is not really good. If you look at it, we don't want to hard-code URLs. Because if you hard-code the URL, what's going to happen is that this can potentially change if you move from one account to another. For example, if you're on a sandbox account then you move to a production account, then this would produce an error because your ID, that script=98, that will change.

To prevent that, we want this to be dynamic. This is already slightly dynamic, because you can use this both on the debugger domain and also the production domain. Or also on the sandbox domain this would also work, as long as it's using the right ID. If it changes the ID, then that's going to be problematic.

Now, what we can do is that we can use the URL module. So if you remember, we added the URL module earlier. So here, what we can do is that we can just say restletUrl. And then, using the URL module, there's a method called the resolveScript. And what that resolveScript method does is that it grabs the URL for your script if you provide the script ID and also the deployment ID.

So let's go ahead and do that. Let me grab the script ID and deployment ID. Let me go to my script record here. I'll copy my script ID. Just copy that, go back to my code, and put that in. Then also, I want to get my deployment ID. Copy that, and then paste that here. And instead of hard-coding that, I'll just get the restletUrl and replace this with the restletUrl.

That's it. Now this would be more dynamic. So even if I used this on the sandbox account and if I transfer to a production account, or even if I am a partner and I'm selling this particular Suitelet to my customers-- it's the same production account but different companies-- this would still work because this would always dynamically get your proper URL.

So let's verify if this works. Let me upload this. And let's go to my employee record. Again, let me refresh this by pressing Control-F5 to make sure that we are getting the latest copy of our script. Let me go down here to our employee code value, type in an invalid employee code value, which is x. Click on Save. And let's see if this works. And as you can see, we still get the validation. So again, this is a very good way to hide your implementation to the server.

Or if you are worried about any cross-browser platform compatibility, like maybe this is working on Chrome and it's not working on Safari, or anything like that, then what you can do is that you can just push the processing to the server to make sure that any important business processes would be handled on one environment. Because if it's running on the server, you would know that it would still work regardless of the browser because it's not dependent on the browser and any client-side script, but it would be dependent on the server.

So that is about it for this walk-through. Some things to remember here-- just make sure you use Suitelets or RESTlets if you want to execute client-side business logic. So here, the advantage of using a Suitelet or a RESTlet is that you only have a single JavaScript implementation. So remember, if you have every process is on the client side, you have to think about am I running on Chrome, or Firefox, or Internet Explorer. We have Safari. So I think there are a lot of options that you have to think of. And sometimes moving from one browser to another can affect the behavior of your script.

But if you are going to put that on a Suitelet or a RESTlet, you only have one environment, the server environment. So you can be guaranteed regardless of what browser and what operating system they're using, it's always going to give you the same behavior because it's not running on the browser. It's actually running on the server.

Also, it allows you to secure the business logic. Remember, like what we saw in our walk-through, we're pushing all of the implementation on the server. And even if the user knows what they're doing-- like they know how to access a script, they know how to debug a script-- then they still wouldn't be able to see validation information. Or not just validation, but any kind of secure information that is on the script, because it's hidden away, tucked away in the server.

Also, try to make sure you use SuiteTalk if you're going to deal with integrations. Start with SuiteTalk, especially for full-blown integration. But if it's not a full-blown integration-- like, OK, I just want to make this one call, it's always just going to be this one call, and it's a specialized call-- then just go ahead and create the RESTlet. But if you want to synchronize the systems, or be doing CRUD operations-- Create, Retrieve, Update, and Delete operations-- I will be doing searching, validation, I have my authentication, username and password and all that, then you want to start with SuiteTalk, because SuiteTalk already supports that. You don't have to build anything on the server. And anything that is not directly supported by SuiteTalk you can augment that using a RESTlet.

So again, we do have a five-day integration course. So if you want to, check that out. So please try to check that out if you are interested in SuiteTalk and integration in general.

Now it's your turn. Please complete the following activity.

This concludes module 13, Web Services.