* More apex integration
* SOAP webservices
  + Simple object access protocol
  + Xml based communication architecture
  + Use a WSDL
    - Web service description language
    - Xml based language to describe how to communicate or interact with the service
  + Salesforce gives us the ability to expose our own code/customizations by creating a SOAP API
  + Salesforce gives us the ability to communicate with other SOAP APIs
  + When we make a call to a webservice, it is called a callout
  + Exposing our code as a SOAP webservice
    - We can expose apex classes as a SOAP webservice
    - The class must be global
    - Any exposed methods must be static
      * Must be prepended with the webservice keyword in place of an access modifier
    - Once we finish writing our class, we can go to **Setup | Custom code | Apex class**
    - From there, we can generate the WEDL from our exposed class to give to anyone needing to communicate with us
    - Any method exposed as a SOAP webservice will run in system mode if we do not enforce permissions/security using apex security
    - Testing is no different than testing a regular apex class
    - Is not used as much
    - Works better for security
    - How do we consume a SOAP webservice
      * Means we communicate with someone elses API
      * Salesforce offers us a tool that will automatically generate an apex class to communicate with the webservice as long as we have the WSDL
      * We can do this by using the WSDL to apex tool in apex classes
      * To make a request to any external site, we need to whitelist it
    - Testing callouts to SOAP webservices
      * Apex tests do not allow callouts to external services
      * We have to create a mock response that would be the result of the request being sent
      * For SOAP, we create another test class that implements the webservicemock interface
      * Inside the test, we let the system know to direct test callouts to this mock by setting test.setMock(instanceOfMockingClass, classWeAreMocking) with the instance of te class before a callout is made in each test method
      * This mocked class must be global or public and must define the do invoke() that has a void return type and is public or global
      * doInvoke() takes 9 parameters
        + Type object

Commonly called stub

Instance of the class generated by WSDL2Apex

* + - * + Type object

Called either soapRequest or request

Hold the SOAP request being made

* + - * + Type map<String, Object>

Is called response or responseMap

Contains key-value pairs that represent an imitation of the data we expect from the service

* + - * + Type string

Called endpoint

Contains URL that the request would be sent to

* + - * + Type string

…

* + - * + Don’t need to know them
      * We must populate the response\_x jet of the response map with an instance of the class indicated by responseType
    - The built-in salesforce SOAP API
      * We can use this API to find duplicate records, execute queries, send out-bound messages, empty the recycle bin, perform DML and do other basic things
      * You can use this from an external service
      * Only available in production and dev editions
  + REST API
    - Representation state transfer
    - Less of a protocol and more of a set of guidelines for API architecture
    - Includes having a uniform interface
    - The service itself should be stateless
    - How to consume
      * Create an instance of the HTTP object
      * Create an instance of the HTTPRequest object
      * Set the request endpoint
        + Req.setEndpoint(‘http://api/address/here’);
      * Set the method
        + Req.setMethod(‘GET’);
      * Send the request
        + httpResponse res = http.send(req);
      * Check for status code 200
        + res.getStatusCode() == 200
        + This means everything is ok
      * Start parsing JSON
        + JSON.deserializeUntyped(res.getBody());
      * You will also need to set remote site settings to allow interaction within salesforce
    - Testing
      * We need to mock the response
      * 2 ways to test REST callouts
        + Creating a mock class

Needs to implement HTTPCalloutMock interface

Have to implement the respond method

Public HttpResponse respond(HttpRequest req)

Define a response

Set a header

Set the body

In the test method

Set Test.setMock(httpcalloutmock.class, new RestCalloutTest());

* + - * + Using a static resource to hold your JSON information that would get in your response

Upload the JSON to static resources

Instantiate StaticResourceCalloutMock

SrcMock.setStaticResource(‘nameOfResource’);

SrcMock.setStatusCode(200);

Set the header

Test.setMock(httpCalloutMock, srcMock);

* + - * Exposing apex code as Restful webservice
        + Create a class
        + Must be global
        + Must have @RestResource above class declaration
        + @RestResource(url=’endpoint’)
        + Each method in the class that you want to expose must be annotated with the @HttpMethod
        + Use @HttpGet above the get methods in your class
        + They run in system mode
      * Salesforce built in REST API
        + Can execute queries get information on objects and fields, perform DML operations
  + Other OOB salesforce APIs
    - BULK API
      * Is a REST API
      * Will take JSON, XML or CSV
      * Lets us import or delete bulk amounts of records
    - Chatter REST API
      * Restful API
      * Lets us post to chatter feeds
      * Read chatter feeds if needed
      * Uses JSON or XML when treansmitted
    - User interface API
      * Restful API
      * Lets us mimic the salesforce UI
      * Communicates with JSON
    - Analytics REST API
      * Allows us to synchronously interact with Einstein analytics
    - Tooling API
      * Can be REST but we can also use SOAP
      * Designed to help us create custom dev tools like ideas
      * Allows us to access org metadata
    - Metadata API
      * Based in SOAP
      * Asynchronously communicates using XML to migrate metadata customizations
      * Used with SFDX
    - Streaming API
      * Based on Bayeux protocol
      * Allows us to subscribe to notifications about changes to salesforce records or execution of custom events
      * Uses JSON
* Advanced org customizations
  + Custom permissions
    - These are a way of creating more granular permissions
    - They do this by allowing you to create a permission and then perform checks on your tools on those permissions
    - These permissions that you create become global variables that you can check for in things like validation rules, formula fields, flows etc
    - Add these through profile settings
  + Custom metadata types
    - Really powerful and important when working with your own packages or multiple instances of salesforce
    - Allows you to create, package, deploy and upgrade your own custom metadata
    - Used to hold configuration information for applications
    - Global constants