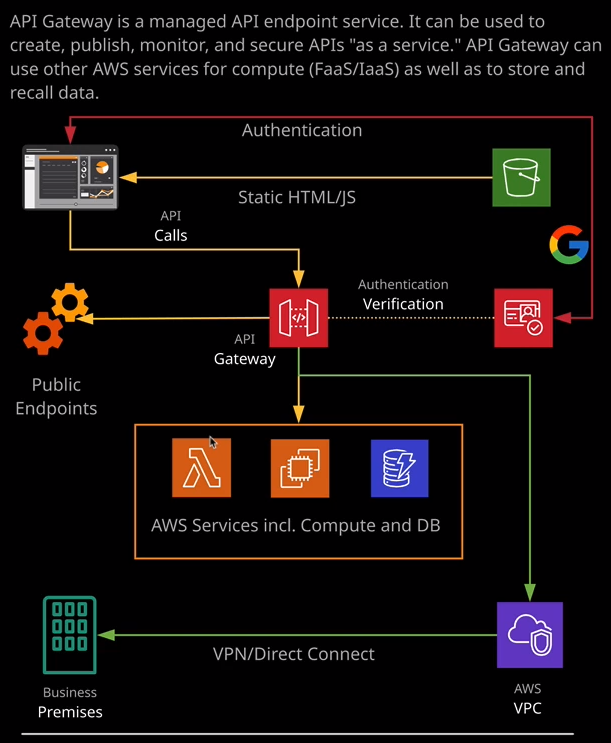
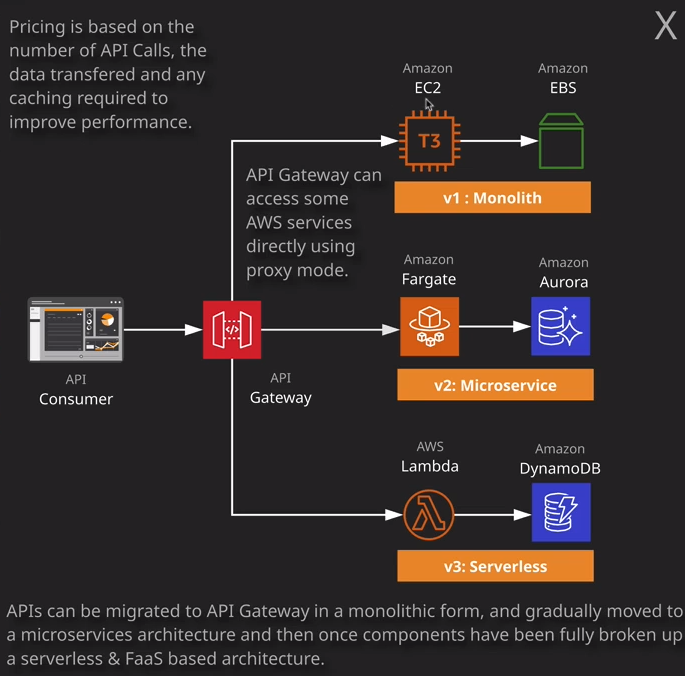
**API gateway**

**API gateways are a product in AWS, which allows you to create, publish, monitor, and secure APIs**. So you can develop the APIs and then use the API Gateway product host, secure, and manage them. It's a managed service where AWS will handle the high availability and scaling and many of the other areas traditionally required to manage APIs, and it'll let you focus on using the APIs and integrating them with other AWS services.

In the last lesson I demonstrated how you can use a Lambda function to resize images. So it's an event driven architecture invoked when an object was uploaded to S3 and it was used to resize those images into thumbnail versions, which were uploaded to a separate S3 bucket but **Lambda can also be used as part of creating an API**. Remember APIs at their core are just pieces of code and **by utilizing API gateway together with Lambda or other AWS services, you can generate production highly performing** **APIs**.



So this represents a fairly standard architecture that you might encounter with API gateway. **You got the HTML and Java script that hosted in the S3 bucket. They're loaded in a browser and that could be used to make API calls to the API gateway, which can in turn, communicate with other AWS services such as Lambda, EC2, Dynamodb, and even private services using a VPC, business premises by using VPN or Direct Connect, and API gateway can even use public internet endpoint**. So there's a lot of flexibility, but essentially you can make calls to the API endpoints hosted by API gateway and it will use underlying services on your behalf.



Now you might already have APIs within your business, and they're probably going to be **monolithic**. That means they're probably running on compute resources such a virtual machines that operate 24/7 365 and that's fine. API gateways can also run APIs in this way. In fact, it can **act as a front end for existing APIs directing these requests through VPCs to your private networks.** You can benefit from the cutting edge availability, security, and scale of API gateway, by having it pointing at existing monolithic resources but over time you can use it to evolve your API. So instead of running on these monolithic services, you might decide to break your API up into more manageable pieces and move these into a microservices style architecture using something like containers and this might mean using a service such as **Amazon Fargate**.

**That's a container compute based service, and that might utilize Aurora which is a highly performant database as a service product available from AWS, and it might use that for these backend database services but you could also go further. You could move from a version two microservice style architecture by breaking things up even further and having individual APIs performing one discreet functions or have Lambda functions backing API gateway performing one thing and doing it well and Lambda could use something like Dynamodb for data storage and retrieval services. So you can move from a monolithic to a micro service to a serverless and API gateway supports whatever configuration you need across all of these three different architectures.**

Now, **API gateway is even capable of accessing some services directly such as Dynamodb without the need for any intermediate compute**. So if you need an API, which just allows access to data stored in DynamoDB, then you might not even need to use Lambda and keep that in mind for the exam because it tends to feature as an exam question.

Okay, so I'm going to do a demo to highlight how API gateway works. Now this demo is something that's freely available from the AWS documentation site. I'll make sure that it is linked but rather than explain the theory of API gateway, I thought it might make a really good example to step through. For this demo, I'm going to make an API, which is capable of doing some simple calculations, nothing complicated. It'll just do simple two value addition, subtraction, division, and multiplication. Now, the API that I'm going to create is going to utilize API gateway but it'll use Lambda for its backend compute. So step one is to create a Lambda function. So I'm going to move across the Lambda console and I'm going to author a Lambda function from scratch. So I'll click on Create Function. I'm going to call my function count and this time, because the code is from AWS, it's going to be based on the node.js 10.x runtime. So I'm going to leave this as default and then for the execution role as the previous lesson, I'm going to change this drop down to create a new role with basic Lambda permissions. The only thing that's Lambda function is going to be able to do as a basis is interact with CloudWatch logs. So that's all I need to set. I'll go ahead and click on Create Function. Now, while this is creating, I'm going to move across to my code editor and the code for this Lambda function is available inside the lesson files for this lesson. Essentially, it's a pretty simple function. It just accepts two values. So that's two different numbers together with an operator. So you can basically add those two values together, subtract them, multiply them, or divide them and anything else generates an error. So it's a pretty simple Lambda function, so I'm going to copy it into my clipboard and don't worry, if you don't understand Java script, it is not a core component of this lesson. So I'm going to move back to the AWS console and as with the previous lesson, we've given this Hello World style example. I'm going to delete that and paste in the code that I just copied. I'm going to go ahead and save this function. Just going to hit set up and then what's also in the lesson files is this test event.json. Now this allows us to test out the Lambda function, make sure that it's operating successfully. So I'm going to copy this into my clipboard and move back to the console. I'm going to click Test. I'm recall the event name Cal and I'm going to paste in this test event. So this is what the Lambda function expects to receive two different numbers and then an operator. So I'm going to put two for the first number, three for the second number, and then for the operator, I'm going to put plus and obviously we expect this to add two and three and the results should be five. So I'll hit Create to test the Lambda function and just make sure it generates the output that we expect. So if expand details there, we go two three plus and the result is five. So that indicates the Lambda function is operating as we expect. So that's a Lambda function. It's operating as we expect but in order to do anything useful with it, we need to make it available as an API and we're going to use API gateway to do that. So that's the next step.

So I'm going to open the API gateway console and the basic entity available inside API Gateway is the API. So because I've not used the service before, I'll go ahead and click on Get Started. I'll just close this example dialogue by clicking on OK. Now with API gateway, **you're able to create two forms of API. You can create a rest API and a web socket API. A rest API is simpler and has a wider compatibility but for high throughput APIs for things that need lots and lots of transactions, web sockets could be much more efficient.**

With any API, there are methods used to communicate to the consumer of the API. **Rest uses a request and response style exchange, and it's stateless. Every time you make a request to a rest API it's considered a brand new communication.** **When you use web sockets and connection architectures, you use a open connection and then you communicate in a constant fashion between the API and the consumer whenever there's any data to be delivered. A good analogy here is push versus pull.** Imagine you have 1000 consumers of an API using a rest API all 1000 will be constantly or as much as needed making API request to check for updates. The API would be overloaded as it scaled. With web sockets, because it's a connection architecture all of the consumers would have an open connection. The push of information could happen once to all of the consumers and the requirements on the API are substantially reduced. **Web sockets has much better performance with larger usage or higher frequency of transactions.**

So the basic entity inside API gateway is the API. So I'm going to go ahead and create a new rest API. So I'll make sure that rest is selected and select new API. For the API name, I'm going to call it Lambda cal. I'll the description blank, and I'll set the endpoint type to be regional. Now there are different types of APIs that you can use but at an associate level, you only really need to be aware of regional, which is the default selection. You can create APIs that utilize a high performance edge infrastructure as AWS but that's well beyond what you need to know at an associate level. So I'm just going to continue and select the regional API endpoint type and hit Create API.

**So that's the API and you can think of that as a container. So next we need to create something inside the API which could be consumed or used, and that's known as a resource.** So to create a resource, I'm going to go to actions and create resource. This is the thing that will be used by any consumers of your API. So I'm going to call mine cal. Additionally, in certain circumstances, **you could enable API gateway CORS support or cross origin resource sharing. If you got a website which uses a certain domain name and this consumes an API using a different domain name then in order for that to work without any security errors you'll need to enable CORS**. I'm not going to bother selecting it but keep in mind, especially for the exam, that if you do have any security related errors when using different domain names for APIs and websites, **you need to enable CORS support** on the specific resource of the API and everything looks good. It's called cal. So I'm going to go ahead and click on Create Resource. Once we got the resource, **we need to configure a way to interact with that resource, and that's known as a method.** So we're going to go ahead and create a get method, and that's how we're going to interact with this API. So I'm going to go to actions and create method, and I need to make sure that I've got the calc resource selected when I do so, because methods are created on a per resource basis. So I click on Create Method. There's different types of method available, so you've got get, head, options, patch, post, and port but I'll be utilizing get for this demonstration so I'm going to select Get and then click on the tick to go ahead and create that method. Now it's on the method that you'll configure the method of integration. So what does this API resource actually do? Well, we want it to integrate with the Lambda functions so the integration type needs to be Lambda function. It's a per regional service. You'll need to pick the particular region and then I'm going to go ahead and start typing calc and it should all to present any available Lambda function. So I'm going to select the calc Lambda function so that I'll configure the integration between this Lambda function and the particular method and the particular resource and that's what we want. We want this resource so /cal to use Lambda function for its logic whenever it's accessed. So this is the integration that we configuring so once have selected the Lambda function to use and click save**, I'll need to allow API gateway to invoke the Lambda function. So every Lambda function, in addition to its execution policy, has another policy which defines who can invoke that Lambda function. That's something that's actually hidden in the UI but it is something that could be set by any services that use Lambda. So in this particular case, it's asking me to give API gateway permission to invoke the Lambda function.** So I'm going to do that by clicking on OK.