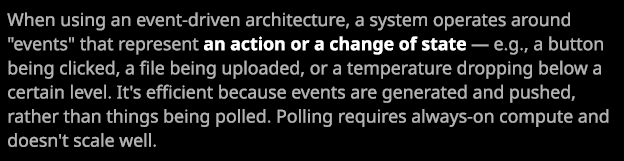
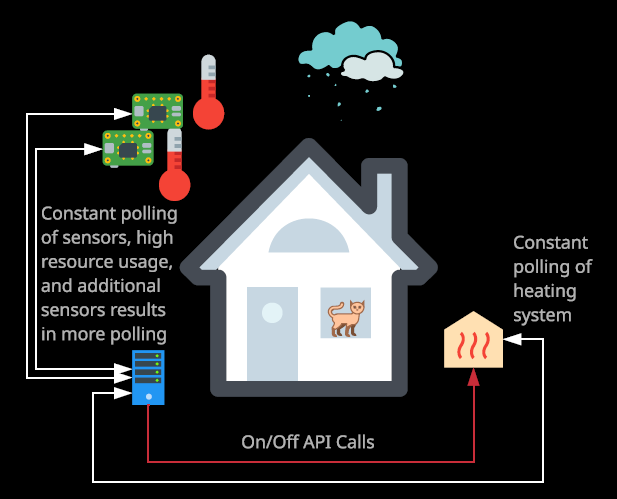
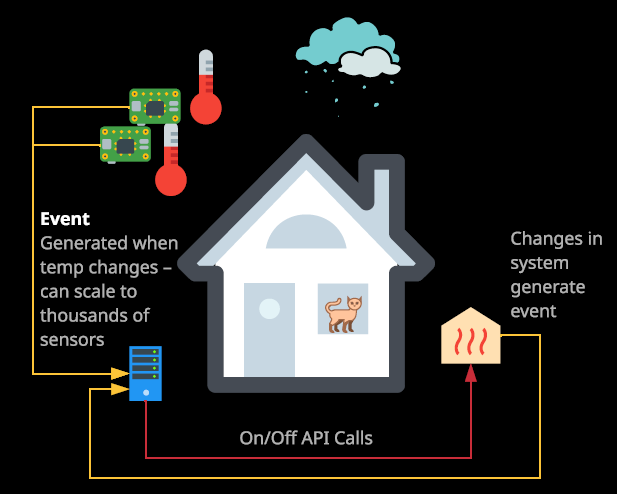
**Event driven architecture and Serverless architecture**

Event driven architecture is one that operates around events and events represent actions or a change of state. Examples of events might be a button being clicked, a file being uploaded, or a temperature dropping below a certain level. It's efficient because events have generated and pushed the other entities rather than things being polled. Polling requires compute that is always switched on, so it's always costing you money, and it doesn't scale well. 

Now, to illustrate the difference between traditional and event driven architectures, it's probably easier to run through a quick example scenario.



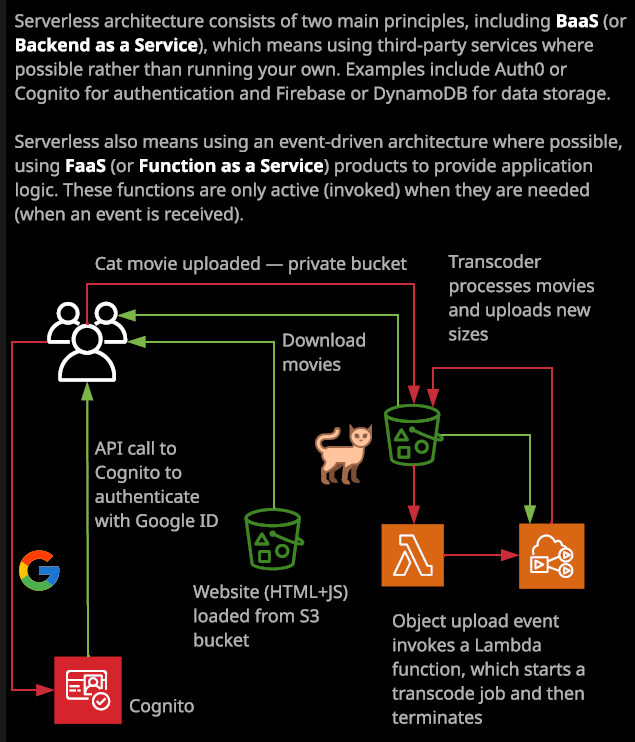
So let's say that we have a house. It's a smart house and so it has environmental sensors in every room. These are going to be monitoring things such as temperature, humidity, light levels, and air quality. So these censors are located in every area of the house and to help us control the temperature of the rooms we've got a smart heating system, which has exits in each room of the house and is able to either pump out heat or pumping cold air on demand and that's controlled by a smart home automation server and for this demonstration, let's say this is running on a Raspberry Pi doesn't matter the exact hardware. But for the sake of this example, let's say that it doesn't have a huge amount of compute capacity. Now, without using an event driven architecture, this server would need to constantly poll these sensors. It would need to do so constantly, potentially multiple times a second. It would be asking for their temperature, the humidity, the air quality, and most of the time, they would be reporting the exact same details as the previous time that the home automation server polled them.   
In addition to that, this home automation server is also going to be constantly polling the state of the cooling and heating to maintain an accurate status. This is important because the heating and cooling solution uses quite a lot of power and so the home automation server needs to make sure that it's constantly aware of whether it's switched on or off on whether it's heating or cooling certain rooms. Now, this example might seem pretty trivial because you've only got two sensors and one heating system. But what if you were managing 100 times the sensors, 1,000 times, or even a 1,000,000? The overhead of this polling style architecture would be substantial for the sensors 90, 95, or even 99% of the time there would be reporting the same temperature. So, in effect, most of the polls to all of the temperature sensors is going to yield no useful data, no state change, no difference. Nothing that the home automation system can act on. So, in effect, it's useless data transfer and useless compute.



**With the event driven version of this, I could configure the temperature sensors to report any state changes to the server. So every time there was a one degree change or more these sensors would generate an event and send that event to the server. The server would see this event and then send any control messages that are required to the heating and cooling system to increase or decrease the temperature in the various rooms of the house. In addition to that, the heating and cooling system would also generate any events for any major state changes. So if there are any power on or off events, any warnings, any system efficiency changes, anything that the home automation server needed to be aware of would be generated as events by the heating and cooling system and then sent to the server.** Now this simple change between the traditional architecture of polling and the event driven architecture, which uses events being generated and pushed to the various different entities. **This** **simple change would significantly reduce the overhead of the system rather than this home automation server having to constantly poll the various different devices around the house.** The devices themselves would report any state changes to the home automation server, so the only time any compute resources would be consumed is when these events were being generated and received by the different entities. So the base level of compute that's required for this type of architecture, as simple as it is, is significantly reduced.

Serverless architecture has two main components.

**The first is BaaS, known as backend as a service on the second is FaaS known as function as a service.**



Now, **backend as a service is actually pretty easy to understand. It's essentially the process whereby you as a consumer use all the service is for functionality rather than managing long running compute processes of your own.** So examples of this might be to use third party services authenticate your customers so maybe auth0 or Cognito, which use third party ID providers such as Twitter or Google and by doing this you don't need to manage your own authentication systems or manage your own store of identities. **Another example is that you could utilize third party database providers such as Fir Base or Dynamodb to store and retrieve data and by doing so, you obviously don't need to run your own database infrastructure.** **So BaaS is an architecture or service model that prioritizes the consumption of third party services and you do that, rather than running your own services. By utilizing BaaS, you remove a lot of the long running infrastructure that your platform or application needs. It removes a lot of the management overhead, a lot of the risks, and alot of the costs that you would otherwise incur by running your own infrastructure.**

Now the other part of serverless is **FaaS or function as a service, and this is a service model where you utilize short running or temporary compute rather than long running computer services like EC2 instances or containers.** With FaaS, you work with functions you give a function, which is a bit of code to a service, and it's executed as required. Now this execution could be based on the schedule, so certain times of day or certain days of week or based on certain events happening. Remember, we just talked about event driven architecture and that's how it supports serverless by supporting function as a service **with function as a service products you only pay for the time that your function is executing.**

With EC2 instances, you are paying for the EC2 instance to run for a certain amount of time but you're doing that regardless of whether any of your functions or any of your code are consuming the compute resources of that instance. **With FaaS products, you only actually pay for the amount of time that your code is executing.** **So it is truly on demand compute. With FaaS, you don't have any other costs. So no infrastructure to manage. No installation, no configuration, no maintenance and no patching.** You give a function to a FaaS product, and it's executed when you want. Now functions can be used for API, and they could be run whenever a request comes into the API functions can also be run nightly toward alternate tasks in your infrastructure. Functions can be run on events maybe when an EC2 instances stopped, started, or terminated.

**In AWS FaaS is provided by Lambda.** So it's serverless both FaaS on BaaS are being used. It's called serverless, not because there aren't any servers, which would be silly. **It's called serverless because you don't manage those servers and don't have to worry about them. BaaS handles the backend by using third party services for databases, for authentication, and for much more and FaaS handles the logic of your application and only runs when needed.**

Now let's run through a quick, theoretical example. So I got this architectural diagram on the right of my screen and before I talk about this architectural diagram I want to set the scene that if you serverless for this architecture before any users access this system, there is no compute running none. The only thing that you were built for is the storage in S3. Now, let's assume for now that this architectural represents a movie upload site similar to YouTube but dedicated to cats. So you as the customer, browse the website in your browser, and you directed to an S3 bucket. Now I haven't called this yet, **but S3 can be used to host static HTML files and other files so it can operate as a simple web server.** So you browse to this S3 bucket and inside this S3 bucket is HTML files and Java script files, which is essentially a login page. So your web browser downloads this login page and runs it, and it's a simple login form. Now, at this stage, just as a reminder, you still have no compute running, so you've got no computer services in this entire architecture. So as the user, you type in your login details or click a button to use Google ID and the Java script that was downloaded from this S3 bucket runs in the browser and communicates with third party authentication service such as Cognito or Auth0. In either case, the service verifies your login and gives you a token which Cognito will exchange for temporary security credentials within AWS. Now that we logged in, we can safely use the site functionality. Let's say that my cat just did something incredibly entertaining and we want to upload a 4K version of that video to an S3 bucket. So we'll do that. We'll use these temporary security credentials that we've now got to upload this movie to an S3 bucket. Once the movie arrives in the S3 bucket we're going to have configured this S3 bucket to generate an event and that event is going to happen whenever any new object is uploaded into this bucket so still, we've not run any compute that we manage in this entire architecture and all that we've been billed for so far is storage and transfer, both of which cost very little. You won't even notice them on most AWS bills. **Once this object arrives, it generates an event, and that event is used as a trigger for a Lambda function. So this Lambda function is invoked based on the object being uploaded into this bucket. Now the Lambda function does one thing and one thing only, and that's to start an elastic transcode job.**

**It's essentially a service that can transcode various different media formats.** In this case, we're going to use it to generate some different size videos. At this point, the elastic transcode service begins its work. It takes the original 4K version of the movie, and it generates a number of different sizes. Maybe one in 1080p, one for tablets, one for mobile, and even one sized appropriately for your TV and it uploads those into the S3 bucket and again, **you're only billed for the time that this elastic transcode job is running and so again, you've not got any long term compute.** Once these different sizes of movies are uploaded to the bucket then you inside the application can see them and choose to share them with everyone because, of course, everyone wants to see cats doing entertaining things, and that's a simple work flow through this application but there are a few things I want to draw specific attention to. **It's all event driven end to end. Nothing is running constantly waiting for things to happen. It's all based on events: logins, clicks, uploads. There's no base cost of this infrastructure because you have everything being based on events because you're consuming services and those services are only consumed when things are happening. You're only charged when those things happen. So these Lambda functions only cost when they're actually executing. Elastic transcode only has any charges when it's being run. You've got no compute infrastructure that's running consistently 24/7 365 and not only that, but you've got no compute infrastructure that you need to manage.** Everything in this architecture is delivered as a service, and that's why the serverless architecture is so powerful. Not because there are no servers, because we all know servers are used in provisioning all AWS services. **Serverless architecture is powerful because there are no servers that you as a customer need to manage because there are no servers because there is no long term compute an environment such as this has little to no base costs, but it's the number of paying customers of this architecture increases your income increases based on that increased usage, your usage of the services increases and so your costs increase but because it's an on demand model that's billed based on the amount of the service that you can choose your costs for this environment generally increase at the same rate as you're paying customers. So, it's easy to manage the cost of an application that's implemented using a serverless architecture.**

Now **serverless architectures are capable of scaling to near infinite levels because you're not managing the compute services you can just consume more and more of these services delivered by the third parties**. Whether that's AWS providing Lambda or Elastic Transcoder whether it's AWS providing Cognito or Auth0 providing authentication services generally as you increase your consumption of the services the vendors are increasing the back and provisions so you don't even have to worry about scaling.