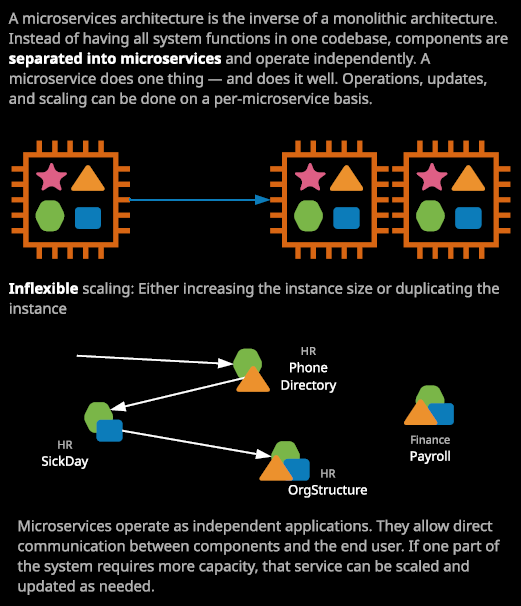
**Microservices and APIs**



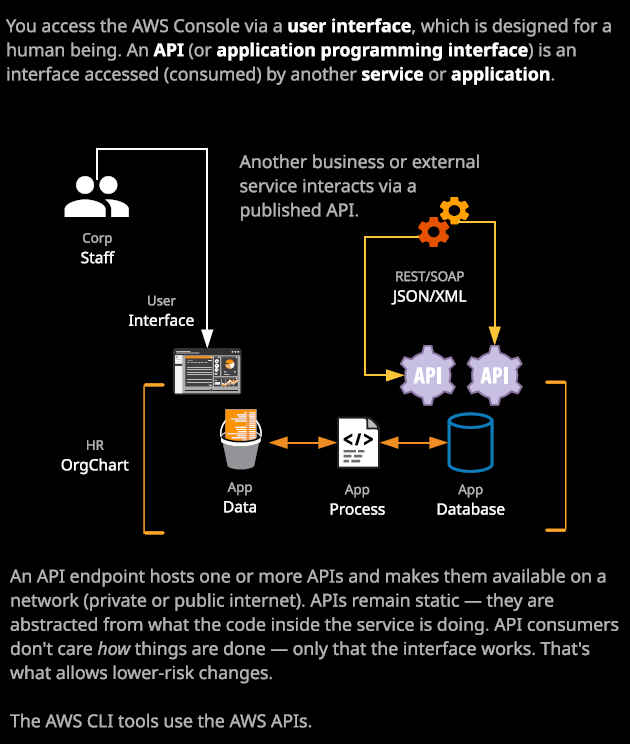
Monolithic applications, which I mentioned earlier in the course are generally a collection of individual bits of functionality together with shared supporting components and examples of this are web servers and database servers. The problem is that because of the monolithic nature, all of these individual components, they require each other and rely on each other, and so you can't really separate them and can't individually scale them. The only option we've got and I've tried to illustrate this visually on the right of my screen is either to **vertically scale the compute resource that's running the monolithic app so that means increasing the size of the instance or, if it's compatible, to horizontally scale it, which means to add additional compute resources but each of these compute resources will be running the same exact copy of all of these components.**

It's a monolithic application, and so you can't separate them. Now, in addition to scaling, having a monolithic application also makes updates and maintenance significantly harder, because each small change to one part of the code could have implications to other parts and so it adds significant business risk. Now I want to use an example to make this easier to illustrate. Let's assume that we've got a personnel management system, something that you might find within a large enterprise for managing staff, finance, and security. It includes functionality for human resources, for finance, payroll, and even physical building access control. All of those individual bits in one large growing blob of code, together with supporting components such as web services to present this application to its users. Now large applications like this are great candidates to implement using a microservices architecture.

**A microservice is a small component of a service which performs one or a small number of functions. Each microservice is given everything that it needs to communicate with the consumer of that service.** So a web service or a data storage or database service, they're all built in to the microservice itself. The idea is that the microservice is self contained and can operate independently for whatever functionality it provides. Now I've got some examples, and I've tried to illustrate this. We've got an HR sick day service, which, given a specific employee ID would indicate if he or she is in the office today or on a specific day in the past. We've got an HR phone directory, which would be able to pull email and phone records for a given staff member. A HR organizational structure service, which could identify immediate reports for line managers for a given staff member and then maybe even a financial payroll service, which could indicate the date of the next payroll for a given staff member. Now, a large enterprise application might be built from hundreds or even thousands of microservices but because they freely communicate, it operates like one big application and that they can work together to achieve a bigger goal. With this example, let's say you're looking up a phone number for a given member of staff. So you're using the HR phone directory microservice, well, this might consult the sick day microservice to check if the staff member is in the office and if not, it might consult the HR org structure service to locate alternative staff members in the same business unit **because these are microservices, and because they operate as independent applications, they can be scaled independently, updated independently, and maintained independently.** The challenge is how to make these individual microservices available to our organization.

So how do you allow these microservices to be consumed and that's where the next topic that I want to discuss becomes useful and the next topic is **APIs or application programming interfaces**. Now we need to consider the scenario of when you as a human being interact with an application, maybe the application that you're looking at right now to watch this video. When you do that, you're interacting with what's known as a user interface. The user interface has been optimized for interactions with real human beings, so it looks nice. It has things to click, boxes to type in, and user elements to move around. Now while this user interface is efficient, and it's been optimized for human beings. It's pretty inefficient for applications, for scripts, for SDKs, or for microservices to consume the services offered by an application. For that, **we need a type of interface, which is optimized for the exchange of data. An API or an application programming interface is an interface published by a service or application, which is designed to be used or consumed by another service or application.**

Now they're different types of API and I'm going to be demonstrating some of those in this topic of the course but some examples include rest, sub, and web sockets, and each of these has their pros and cons. **APIs, though, have data transfer formats.** For example, rest APIs generally used JSON and JSON is a format that you've already had experience of using AWS within this course. It's the format that a lot of AWS products and services used to exchange data. So IAM policies, for example, use JSON. In the first part of this lesson I talked about breaking an application up into microservices and APIs how those microservices can talk to each other. **APIs are published in a way that allows a microservice to interact with another service so an API defines what interactions can occur with that microservice. It defines what data gets supplied and returned, and it defines any authentication or authorization that's used to interact with that microservice.**



Now as a human being, you actually use APIs all the time, but you do so indirectly. If you use a Linux Academy application on your phone, it uses the APIs to communicate with the Linux Academy platform to show you content. So, for example, if we've got staff and they're accessing a user interface this user interface generally uses APIs to communicate with a microservice. When using AWS when we've been interacting with the AWS console behind the scenes that's actually using APIs to communicate with the backend AWS services and features of those services. So without even realizing it, you're already using APIs. If you use Microsoft Word to store files in to One Drive that's an API. If you used the Netflix application on your phone, tablet, or TV that's not connecting to the user interface that you might use if you go to netflix.com. Instead, it's using an API to efficiently communicate with the platform. When you use the AWS CLI tools, they're actually communicating with AWS using the APIs. This process uses access keys to authenticate and IAM policies to authorize its one of IAMs primary functions to control access to the AWS API and now that you understand what an API is and what a microservice is you can start to understand that AWS is a product is actually made up of lots of APIs and microservices that provide each of the different product and product features.

Now, the last concept that want to introduce before we going to finish up this lesson and move on is that of an API endpoint. **An API endpoint is how you actually access an API.** Just like you as a human being can browse to a user interface of an application, whether that's the AWS console or netflix.com an **API also has an endpoint and is provided by the product which hosts the API an API and on its own does nothing. It's just a piece of code and an agreement of how to communicate with that code in order to do something it needs to be hosted. It needs an endpoint.**

Now traditionally API endpoints have come with a management overhead. There's no avoiding it. You'll be running these API endpoints on physical or virtual servers, or even EC2 instances. So you'll still need to install and manage the operating system as well as worrying about some form of web server to host the API.

So this is the architecture of microservices. **There's the code and the compute needed to run that code. There's the API and the API endpoint needed to present that code to you and both of those together are what are required to make a microservice. A microservice is designed to be self sufficient. You can scale it independently, manage it independently, and update it independently.**