Microservices are a particular software architecture. A microservice architecture breaks an application up into a collection of small, loosely coupled services. This is different from traditional architecture, which can be called a monolithic architecture. In a monolithic architecture, all features and services part of an application are part of one large executable. They are all kept together as one giant entity.

Microservices, however, are small pieces broken up into their own individually executable portions, so each microservice implements only a small part of an application, overall functionality. And importantly, microservices are loosely coupled. This means that microservices interact with each other and the holistic application using stable and well defined APIs. In short, it means microservices are independent of one another.

Microservices can be easily maintained if you ensure the APIs are maintained and that any changes made to microservices will not disrupt the rest of the application.

Let us picture what this looks like visually. A monolith would contain all the different parts of an application, so an application needs to handle authentication. This particular application may have customer information databases. It may have inventory databases. It may have functionality for processing payments, and those are all part of one single executable process.

Microservices, however, are split up into multiple pieces. So in this diagram, we have a UI, which is its own separate entity, and it is communicating with an authentication service, a customer info service, an inventory service, and a payment service. These services are all separate, and they don't necessarily even need to exist in the same place. They don't need to be built using the same technologies. They just communicate with each other over standardized APIs. Microservices are all about breaking up large applications into individual functional components.

There are many different ways to structure and organize a microservice architecture. How you specifically decide to break up the functionality of an application into microservices could change depending on what type of application you're building or what makes sense at that time.

But here's an example (change it), let's say that we have a pet shop application, and this is an application where you can go, and you can search and find pets that are available for adoption and perhaps fill out of form in order to apply to adopt a pet and potentially process payments to purchase things from the pet store,

process payments for any goods or services that customers are purchasing through the web app. Each one of these services would be built within its own codebase and would be a separate running process or processes because, of course, we could run any of these services on multiple nodes, and

each of these pieces of the app can be coded, built, deployed and scaled separately from the other portions of the app.

Here is an example of how a microservice is in use. Let's say there is a toy store application. This application allows the customer to search and find a specific toy, and if they want to purchase the toy, they will fill out a purchase form then move to a payment service. There would also be an inventory management service that ensures the toy is in stock, and another that stores the customers account if they create an account for their next purchase. This customer account will have an authentication service so the customer can log in with all their saved credentials.

So why would you want to use microservices? Well, one significant advantage of microservices is that they are very good at supporting modularity. They encourage you to build modular apps in monolithic applications. Individual pieces become tightly coupled, and over time can be very complex. Over several years, eventually, the application gets so large that it is challenging to maintain. And it's complicated to change anything without breaking something.

If the team makes the slightest change, a colossal regression testing effort is required to ensure that change didn't break something. That slows down your ability to deliver new features on functionality and makes that application a nightmare.

We could create each of those pieces in its own separately maintainable and much more straightforward entity with microservices. Microservices also offer technological flexibility because each element of the application is separate from the others. They communicate with each other over standardized APIs. Each piece of the application does not necessarily need to be built using the same programming languages and technologies as the different parts of the application. That means for any individual section of the app, you can use the tool that is best for that specific job.

Having this ability allows you to adopt new technologies as the lifetime of your application goes forward. You can build new functionality on top of new languages and new technologies, and they'll work just fine with the older pieces of the application.

Microservices also offer optimized scalability. That means that you can scale individual parts of the app based on resource usage and load. In contrast, if you want to scale the app with a monolithic application, you will dedicate more resources or build additional nodes to support it. You have to devote those resources to every single feature of the app, as the app is one giant executable. However, it may only be certain portions of the applications overall functionality that require that additional resource.

With microservices, you can scale each of these individual elements on its own. Let us return to our toyshop example. A lot more people will probably look at the toys versus the customers that will end

up buying a toy. So you can scale your toy information service up to meet that additional load while the customer information service continues to use fewer resources.

These benefits are exceptional, but microservices are imperative to remember that they aren't always the best choice. If you have a smaller, more straightforward app, sometimes a monolith can be easier to maintain and manage. But microservices, in almost all cases, will be the best option once you have an extensive, complex application with many distinct parts and pieces to it.

A quick summary of microservices is that microservices are essential to DevOps when it comes to managing large and complex applications that can often have distinct differences. As each microservice is its own piece, it can be deployed and scaled separately. Microservices are great at supporting that fast speed of delivery, coupled with stability.