in Object Storage, and how it differs from the more traditional storage types such as

File and Block Storage. The first thing to note about Object Storage is that you do not connect it to a particular compute node in order to use it.

Instead, you provision an Object Storage service instance and use an API (or Application Program

Interface) to upload, download, and manage your data. This means you can directly use Object Storage with anything that can call an API and you

don’t need an underlying compute node.

The second thing to note about Object Storage is that it’s less expensive that other cloud

storage options.

It’s per gigabyte cost is typically a couple of US cents per month and in some cases, even

less, depending on the storage tier used.

More on storage tiers later.

The third and possibly most important thing to note about Object Storage is that it’s

effectively infinite.

With file and block storage, you specify the size of the storage you want in gigabytes

or terabytes and then pay a fee based on the size you provisioned.

With Object Storage, you just consume the storage you need and pay per gigabyte cost

for what you use.

You can keep uploading files and the storage will never run out.

So, when would you use Object Storage?

Well, Object Storage is great for storing large amounts of unstructured data.

By unstructured this means that the data is not stored in any kind of hierarchical folder

or directory structure – Object Storage uses ‘buckets’, and objects are stored

within these buckets in a structurally flat way.

A bucket is a bit like a folder, in the sense that you can give them meaningful names, and

of course have different buckets for different object-types but you cannot place a bucket

within a bucket.

When an object is placed in a bucket, it also has some metadata (data about the data) added

to it, such as an object ID.

This metadata helps applications to both locate and access the object, as well as provide

information on the time that the data was stored or last accessed.

When you create a bucket, you don’t need to provide or define any sizing information—the

bucket will just hold the data that you place inside it and the service provider ensures

that there is sufficient storage capacity available.

Buckets can hold as little as a few bytes of data, right up to multiple petabytes and

you can build up the amount of data stored as slowly or quickly as you like—as well

as shrink it back down again.

The service provider also takes care of resilience and making sure that the Object Storage solution

is highly available.

Some cloud providers offer different types of buckets with different levels of resilience.

For example, they offer buckets which are resilient, but the data is only stored in

one data centre.

This is a good option where data needs to reside in a particular geographical location

or in situations where high availability is less of an issue.

They will then offer buckets which are highly available across regions, where the data is

stored multiple times in different datacentres (or zones) in the same region or even in multiple

regions.

These options usually cost more but they provide both the highest level of resilience as well

as availability for your data.

Object Storage has a very ‘flat’ storage structure, which we’ll explain in the next

lesson.

This data can be anything from text files to audio and video files, from IOT data to

virtual machine images, from backup files to data archives.

Pretty much any data which is static and where fast read and write speeds are not necessary

would make a good fit for object storage.

**Object Storage would, however, not be suitable for running operating systems, nor applications**

such as databases or anything else where the contents of the files changes.

So, to summarize what we have learned in this lesson:

Object Storage is used to store files—or Objects—which are static.

The data that you can store using Object Storage can be anything from text files to audio and

video files, from IOT data to virtual machine images, from backup files to data archives.

You cannot run operating systems or other applications such as databases using Object

Storage.

Objects are stored in Buckets.

You can have multiple buckets, but you cannot place buckets within buckets.

You do not need to specify a size for a bucket, you can just use as little or as much space

as you need.

Many providers offer different types of buckets with different charges for each.

Some are based on resilience and availability, while others are based on the frequency at

which the objects inside are accessed.

Storage APIs.

Object Storage buckets also have storage ‘tiers’ or ‘classes’ associated with them and

these tiers are based on how frequently the data is accessed.

A standard tier bucket is where you would store objects that are frequently accessed.

This tier tends to have the highest per gigabyte cost associated with it.

**A ‘vault’ or ‘archive’ tier is where you might store documents that are only accessed**

perhaps only once or twice a month or less, and this will be offered at a lower storage

cost, whereas there may also be ‘cold vault’ tier, where you would store data that is typically

accessed only once or twice a year.

This storage often costs just a fraction of a US cent per gigabyte per month.

Often, you can also set up automatic archiving rules for your data, meaning that if an object

isn’t accessed for a period of time, it will automatically be moved to a cheaper storage

tier.

The rule uses some of the object’s metadata to determine when it should be archived.

Note that Object Storage does not come with IOPS options.

Object Storage tends to be very slow in comparison with file or block storage, where downloads

typically take seconds, if not longer, to complete.

Where providers offer ‘cold vault’ buckets, data retrieval from these tiers can sometimes

even take hours because the storage is kept off-line.

If your application needs fast access to files, then Object Storage may not be a good option.

We’ve mentioned that Object Storage is priced per gigabyte used but there can also be other

costs related to retrieval of the data.

These costs are similarly low but access charges can be higher for data that is in vault or

cold vault tiers, so it is important to ensure that the data is in the correct tier, based

on its frequency of access.

Object Storage does not need to be attached to a compute node for you to access it, rather

you access Object Storage through an Application Program Interface, or API.

The most common API for object storage is called the ‘S3’ API, which is a standard

based on the S3 Object Storage offered by AWS.

Many providers offer APIs to their Object Storage which is S3 compatible, which is useful

because it means developers can write code which is able to access multiple vendor’s

Object Storage.

The API itself is an HTTP-based RESTful API or RESTful Web service.

The API call allows applications to manage object storage and buckets as well as PUT

(upload) or GET (download) objects to and from them,

Object storage is not just for new applications but can be used to meet requirements for existing

ones.

It can also be used as an effective solution for backup and disaster recovery as a replacement

for offsite, tape-based solutions, reducing the time to restore data.

Many backup packages now include the ability to back data up into the cloud, using Object

Storage.

Object storage is more efficient than tape backup solutions, which require tapes that

need to be physically loaded into, and removed from, tape drives, and moved off-site for

geographic redundancy.

So, to summarize what we have learned in this lesson:

Object Storage has different tiers, with different charges for each.

Some are based on the frequency at which the objects inside are accessed.

Object Storage is priced per gigabyte of storage used per month, plus some charges for data

retrieval.

Object Storage is much cheaper than file or block storage.

Object Storage is very slow in comparison with File and Block Storage.

You can often create rules which allow the automatic ‘archiving’ of objects to cheaper

tiers when they are infrequently accessed.

Object Storage is accessed using an API.

Many Object Storage providers have an ‘S3 Compatible’ API, which means developers

can create code that will work against multiple-vendors Object Storage solutions

Object storage in the Cloud offeres an effective Backup and Disaster Recovery solution.

In the next video, we will be covering Content Delivery Network (CDN), which is driven by

Object Storage.

**Serverless Computing:**

Serverless is an approach to computing that offloads responsibility for common infrastructure

management tasks such as scaling, scheduling, patching, and provisioning application stacks

to cloud providers, allowing developers to focus their time and effort on the code and

business logic specific to their applications or process.

Serverless doesn’t mean there are no servers; only that the management of the underlying

physical or virtual servers is removed from their users.

The serverless computing environment allocates resources as needed for the applications.

Let’s look at some key attributes that distinguish serverless computing from other compute models.

The serverless model requires no provisioning of servers, installation of application stacks

and software, or operation of the infrastructure by the developer.

Serverless computing runs code only on-demand on a per-request basis, scaling transparently

with the number of requests being served.

Serverless enables end users to pay only for resources being used, never paying for idle

capacity, which is unlike virtual servers on the cloud—where end users pay for VMs

as long as they are running even if idle.

Effectively, serverless abstracts the infrastructure away from developers.

Code is executed as individual functions where each function runs inside a stateless container.

No prior execution context is required to serve a request; and with each new request,

a new instance of the function is invoked.

Let’s look at a scenario.

You could, for example, have a serverless platform between the front-end of your website

and your storage layer, running individual functions.

The serverless app could be translating text files and storing it in a cloud-based storage

service.

Using the front-end of your website, you send text files to a serverless app; the app creates

translations in different languages, and then stores these translated files in cloud storage,

and sends their links back to you.

Some of the key serverless computing services today include IBM Cloud Functions (which is

based on Apache OpenWhisk), AWS Lamb-da, and Microsoft Azure Functions.

It is important to note that serverless may not be the best fit for all applications or

scenarios.

You need to evaluate application characteristics and ensure that the application is aligned

to serverless architecture patterns.

Applications that qualify for a serverless architecture include some of the following

characteristics: Short-running stateless functions (seconds

or minutes).

Seasonal workloads with varying off-peak and peaks.

Production volumetric data that shows too much idle time.

Event-based processing or asynchronous request processing for implementing use cases.

Microservices that can be built as functions that are stateless.

Serverless architectures are well-suited for use cases around data and event processing,

IoT, microservices, and mobile backends.

Given its inherent and automatic scaling, rapid provisioning, and a pricing model that

does not charge for idle time, supporting microservices architecture has become one

of the most common use cases of serverless computing today.

Serverless is well-suited to working with structured text, audio, image, and video data

around tasks such as data enrichment, transformation, validation and cleansing, PDF processing,

audio normalization, thumbnail generation, and video transcoding.

Parallel tasks such as data search and processing, and genome processing, are also well-suited

to be run on a serverless runtime.

Serverless is also well-suited for working with all sorts of data stream ingestions,

including business data streams, IoT sensor data, log data, and financial market data.

And finally, let’s look at some challenges worth considering about serverless.

Serverless workloads are designed to scale up and down in response to workload, but for

workloads characterized by long-running processes managing a traditional server environment

might be simpler and more cost-effective.

The serverless application architecture can be vendor dependent, and so there is a potential

for vendor lock-in, particularly involving platform capabilities such as authentication,

scaling, monitoring, or configuration management.

Because serverless architectures scale up and down in response to workload, they also

sometimes need to start up from zero to serve a new request.

For certain applications, this delay isn’t much of an impact, but for something like

**a low-latency financial application, this delay wouldn’t be acceptable.**

**Cloud Native Applications:**

Simply put, a cloud native application is

an application developed from the outset

**to work only in the cloud environment, or**

an existing app that has been refactored

and reconfigured with cloud native

principles. A cloud native application

consists of microservices working

together as a whole to comprise an

application, yet each can be

independently scaled and iterated

through automation and orchestration

processes. These microservices are often

packaged in containers, which are

executable units of software in which

the application code is packaged along

with its libraries and dependencies so

that it can be run from anywhere. This

independence enables frequent, iterative

improvement of cloud native applications,

without disrupting the experience of

end-users. Cloud native applications are

unlike traditional, or monolithic

applications, that are built out of one

huge piece of software; applications that

tightly couple the user interface,

business-logic layer, and data layer.

Let's take the example of how a cloud

native application might be used on a

travel website. Each topic covered by the

site - flights, hotels, cars, specials - is its

own microservice. Each microservice may

roll out new features independent of the

other microservices. Specials and

discounts can also scale out

independently. While the travel site is

presented to customers as a whole, each

microservice remains independent and

can be scaled or updated as needed

without affecting other services. Whether

creating a new cloud native application

or modernizing an existing application,

developers adhere to a consistent set of

development principles: Follow the

microservices architecture approach by

breaking applications down to single-

function microservices. Rely on

containers for maximum flexibility,

scalability, and portability.

Adopt Agile methods that speed the

creation and improvement process through

quick iterative updates based on user

feedback. In this video, we'll take a

closer look at the key concepts of cloud

native, its benefits and use cases. Hi. I'm

Andrea Crawford and I'm with IBM Cloud.

Today we're going to talk about cloud

native apps. In the heritage world, we

have our lumpy, monolithic apps. And in

the new world, we have our microservices

living on the cloud. If we take a look at

this diagram here, we see we have cloud

infrastructure. This is your private, your

public, and your enterprise

infrastructure. Cloud native apps apply

to hybrid and multicloud situations. We

also have our scheduling and

orchestration layer. This layer is all

about control planes, like our kubernetes.

We also have our application and data

services layer. This layer is all about

backing services, and being able to

integrate our application code with

existing services that may be available

on other clouds, or even on-premise. We

have our application runtimes, these are

what we're traditionally, or

conventionally, known as middleware. And

over here, well, that's where we have our

cloud native apps. This is the sweet spot

right up here. So our application code is

actually designed, built, and delivered

very differently for cloud native, than

it would be for conventional, monolithic,

lumpy apps over here. Let's talk a

little bit about why cloud native apps

can actually leverage benefits like:

enabling innovation,

business agility, and most importantly -

from a technology perspective - the

commoditization of this solution stack

over here. As time has progressed and

technologies have matured and emerged, a

lot of the services are actually being

refactored lower down in this stack. This

means that core services are starting to

have a lower center of gravity, freeing

up innovation at this level over here.

So, what are our use cases for when to build

a cloud native app? Everything!

Everything that lives in the cloud

should have a cloud native app design

and approach. This means our application

code needs to be instrumented with

things like: standardized logging,

standardized events, and being able to

match those logging and events to a

standard catalog, that multiple microservices

and cloud native apps can use.

The last thing we want to do is have our

development squads have to figure out

what their log and event messages should

be. Let's standardize that, because we

want to be able to commoditize that as

well. We also need to have things like

distributed tracing. When we get over

into the microservices world over here,

we have a lot of moving parts. This means

we're going to need to leverage services

core to the system, like: load balancing,

service discovery, and routing. These are

the kinds of things that are

commoditized in this layer here, with

things like Istio, and with the emergence

of newer projects, like Knative. And so,

if we were to recognize the benefits for

cloud native apps and to sum it all up,

we are all about enterprise and

engineering at scale.

**DevOps in Cloud**

Development teams need to design, develop, deliver and run software as reliably and efficiently

as possible.

Operations teams need to identify and resolve problems as soon as possible by monitoring,

predicting failure, managing the environment, and fixing issues.

Combining development and operations with the ability to monitor and analyze and optimize

bottlenecks gives us DevOps—a collaborative approach where business owners and the development,

operations, and quality assurance teams collaborate to continuously deliver software.

A DevOps approach applies agile and lean thinking principles to all stakeholders in an organization

who develop, operate, or benefit from the business’s software systems, including customers,

suppliers, partners.

By extending lean principles across the software supply chain, DevOps capabilities improve

productivity through accelerated customer feedback cycles, unified measurements and

collaboration across an enterprise, and reduced overhead, duplication, and rework.

Using the DevOps approach, developers can produce software in short iterations

on a continuous delivery schedule of new features and bug fixes in rapid cycles;

and businesses can seize market opportunities and reduce time to include customer feedback

in their products.

The DevOps process involves: Continuous Delivery, which is about delivering

small, well-designed, high-quality, increments of software to customers.

Continuous Integration; creating packaged builds of the code changes released as immutable

images; where immutable implies that when modifications are needed, the entire component

is replaced with an upgraded version.

Continuous Deployment, which involves progressing each new packaged build through the deployment

lifecycle as rapidly as possible.

Continuous Monitoring; with tools that help developers understand the performance and

availability of their applications, even before they are deployed to production.

Delivery Pipeline; which is an automated sequence of steps that involves the stages of

Ideation, Coding, Building, Deploying, Managing, and Continuous Improvement; which loops back

to the Ideation phase in the delivery pipeline.

While DevOps can apply to applications anywhere, there is especially a compelling case for

DevOps when it comes to cloud-ready, and cloud-native applications.

DevOps and Cloud share a symbiotic relationship.

With its near limitless compute power and available data and application services, cloud

computing platforms come with their own risks and challenges.

DevOps’ tools, practices, and processes are helping tackle some of the complexities

and challenges posed by the cloud and allowing solutions to be delivered—quickly and reliably.

Let’s look at some core capabilities that DevOps provides to help building and running

applications in the cloud a lot more manageable: DevOps best practices make it possible to

programmatically provision servers, build middleware, install application code, and

fully automate the installation process in a way that is documented, repeatable, verifiable,

and traceable.

Application deployments often involve considerable complexity.

The DevOps’ practices of continuous integration and continuous deployment help create a fully

automated deployment pipeline, which is important all through the application development lifecycle.

Cloud native applications form a complex distributed system with multiple moving parts, independent

tech stacks, and rapid release cycles.

DevOps principles are essential to define how people work together to build, deploy,

and manage applications in a cloud native approach.

With the DevOps best practices of automated provisioning and continuous deployment, developers,

quality professionals, and other stakeholders can test in low-cost, production-like test

environments that were previously not available—enhancing both productivity and quality.

When systems are compromised or struggling to recover from natural disasters, DevOps

best practices make it possible to rebuild these systems quickly and reliably.

DevOps provides a powerful set of principles, practices, and tools to realize the full potential

of cloud-native computing, as well as for modernizing existing applications to leverage

cloud benefits.