Serverless

- The cloud providers offer that enable programmers to create, deploy, and scale applications quickly, easily, and at low cost.
- It explains why such facilities have gained popularity and can reduce costs for a cloud customer that offers a service

Traditional Client-Server Architecture

To understand the advantages of serverless computing, one must first understand the traditional client-server architecture that network applications use.

When application programs communicate over a network, they follow the *client server paradigm*, which divides applications into two categories based on how they initiate communication.

One of the two programs must be designed to start first and remain ready to be contacted.

The other application must be designed to initiate contact.

We use the terms *client* and *server* to describe the two categories:

Server: an application that runs first and waits for contact

Client: an application the contacts a server

The categories only refer to the initial contact between the two: once network communication has been established, data can flow in either direction.

In the simplest case, two computers attach to a network or to the Internet.

A client application running on computer 1 contacts a server application running on computer 2.

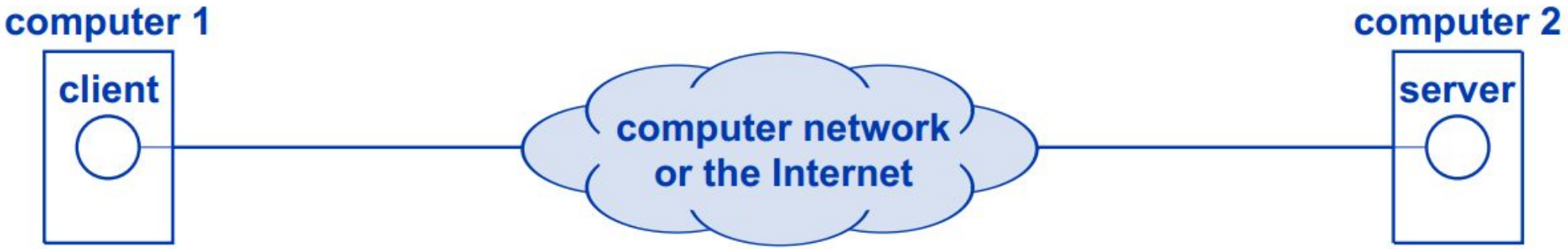

Figure 14.1 An example of traditional client-server communication.

Scaling a Traditional Server to Handle Multiple Clients

- In a traditional client-server architecture, a server scales by using concurrent execution to handle multiple clients at the same time.
- That is, the server does not handle one client and then go on to the next. Instead, the server repeatedly waits for a client to initiate communication and then creates a concurrent process to handle the client.
- Thus, at any time, *N + 1* server processes are running: a *master* server that clients contact, and *N* instances of the server that are each handling an active client.
- To handle multiple clients, the computer on which a server runs must have more memory and processing power than a computer that runs a client.

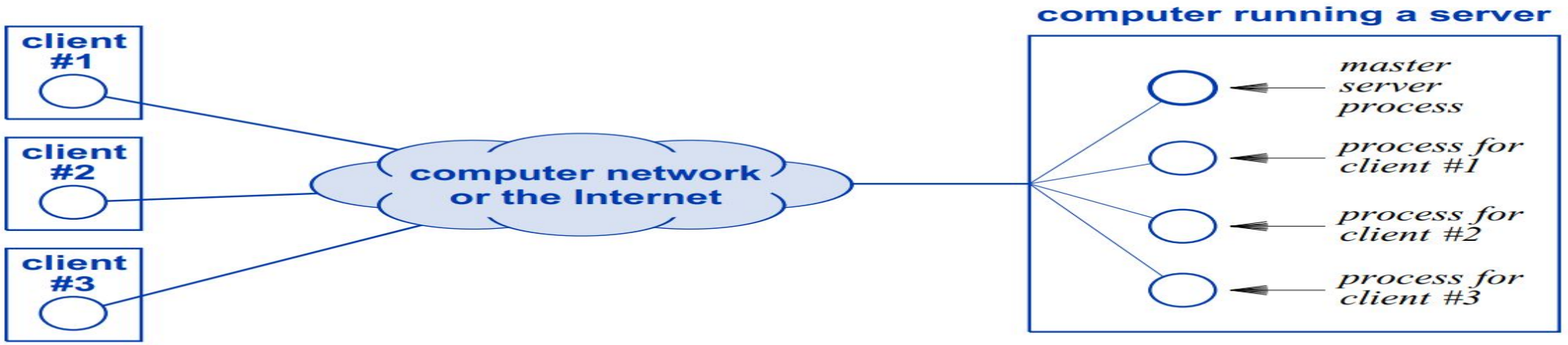

Figure 14.2 An example of traditional client-server communication.

The figure makes it appear that four independent application programs are running on the server computer. In fact, they are all copies of one server application. When a connection arrives from a new client, the master server copy calls *fork* (or the equivalent) to create a new process to handle the client

- Conceptually, the application for a concurrent server integrates two aspects of a
  server into a single application program:
  Fulfilling the service being offered
  Replicating and scaling the server

Fulfilling the service: A client contacts a server to access a service.

- Typically, a client sends a request to which the server sends a reply.
- The chief function of a server lies in interacting with clients to fulfill clients' requests.

Replicating and scaling: The second aspect of a server arises from the techniques used to replicate and scale the server.

A traditional concurrent server uses contact from a new client to trigger the creation of a separate process to handle the client.

Scaling a Server in a Cloud Environment

- A traditional concurrent server replicates copies of itself automatically to handle multiple clients simultaneously.
- The approach does not work well in a cloud environment because cloud systems achieve large scale by replicating instances across many physical machines.
- The systems handle instance management by deploying copies as needed and must also handle network communication by arranging to forward traffic from each client through a proxy to the correct instance.

The Economics Of Servers In The Cloud

A large set of management technologies exist that can be used to deploy and operate services, including orchestration systems, proxies, load balancers, and service mesh management software.

In terms of cost, many of the software technologies follow the open source model, making them free.

Thus, it may seem that open source software allows a cloud customer to port existing servers to the cloud and then deploy and scale the servers at little extra cost.

That is, a customer only needs to have basic server software, lease a set of VMs, and use open source software to handle deployment and scaling

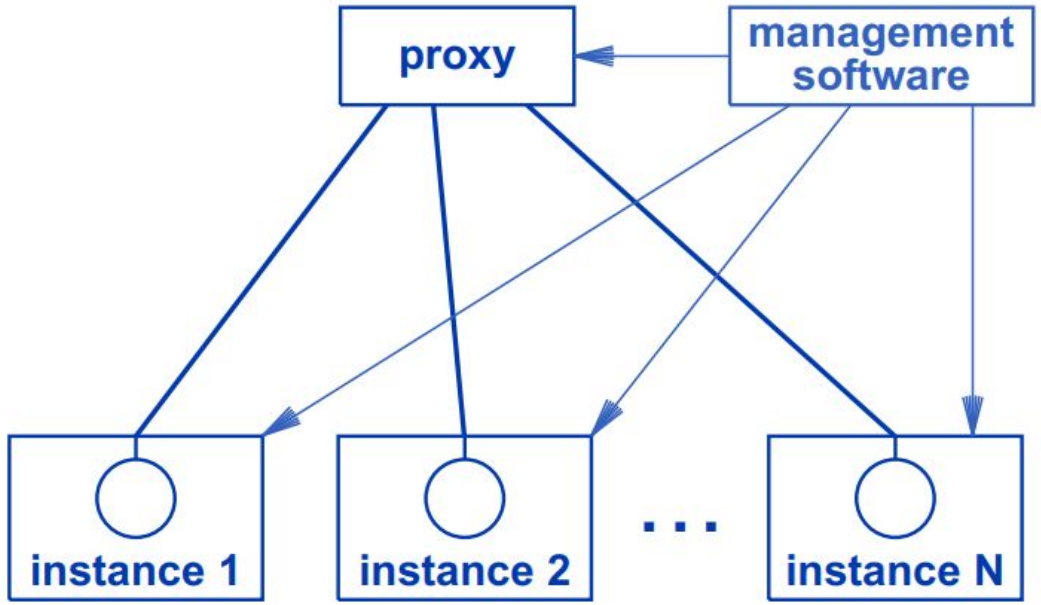

Figure 14.3 Management software controlling a proxy and replicas of a server on multiple physical machines.

- Unfortunately, two costs can be significant:
  Unused capacity
  Expertise and training

Unused capacity: Because setting up a VM takes time, a customer must plan ahead by allocating sufficient VMs to handle the expected peak load.

As a result, the customer must pay for VMs that remain idle during off-peak times.

Expertise and training: Although open source management systems are free, using such technologies effectively and safely requires a significant amount of expertise that lies outside the typical expertise of a software engineer.

Thus, a customer must hire experts.

A customer that uses multi-cloud, must have expertise for each cloud system. Furthermore, because cloud technologies continue to evolve, the customer must pay for training to keep their staff up to date.

The Serverless Computing Approach

In response to the need for server management, cloud providers created a way that a cloud customer can pay to have the provider handle all tasks associated with running the customer's servers.

The industry has adopted the name *serverless computing* to refer to the approach.

To an outsider, the name seems inappropriate because the underlying system does indeed run servers and uses the client-server model of interaction.
However, providers assert that from a customer's point of view, the facilities are "serverless" in the sense that the customer can build and run software to fulfill users' requests without thinking about the deployment and replication of servers, without configuring network names and addresses for servers, and without leasing VMs to run the servers. To help eliminate confusion, the industry sometimes uses the alternative name *Function as a Service* (*FaaS*).

The name arises because a cloud customer writes code that performs the main function of the server and allows the provider to handle deployment.

Function as a Service (FaaS)

Also known as Function as a Service (FaaS)*, the serverless computing approach allows a cloud customer to avoid dealing with servers by paying a cloud provider to deploy and scale the customer's servers.*

Why would a customer pay a provider to deploy servers?

The customer can reduce overall cost.

First, the customer can avoid charges for unused capacity because the provider only charges fees for the time a server is being used.

That is, instead of creating VMs, the provider only runs a copy of a server when needed.

Second, a provider amortizes the cost of experts and their training over all customers, making it less expensive for each customer.

The reduction in local expertise can be especially significant for customers that follow the multi-cloud approach of using more than one provider.

The point is: *Because a provider amortizes the cost of expertise among all customers and only charges for the time a server is used, the serverless approach can cost a customer less than running servers themselves.*
What happens if no one uses a given server?

In some cases, a provider imposes a minimum monthly fee.

However, other providers offer service without a minimum fee, meaning that if no one uses a server in a given period, the customer pays nothing.

Industry uses the term *scale to zero* to refer to such services.
In terms of accommodating large numbers of clients, the serverless approach offers great flexibility.

Unlike a self-managed set of servers that require a customer to plan for a peak load, serverless computing allows arbitrary scale.

Industry uses the term *scale to infinity* to characterize a serverless system that does not place a limit on scale.

In practice, of course, a provider must set some limits, but the idea is that the limits lie beyond what a typical customer will ever need.

Stateless Servers and Containers

How can a provider offer serverless computing that scales from zero to infinity and
achieves both efficiency and low cost?

To make it feasible to deploy servers quickly, serverless technologies use containers.

To deploy and manage a server, serverless systems use orchestration, and to handle scale out, a serverless system uses the controller-based approach.
Despite building on extant technologies, the serverless approach introduces two key features that distinguish it from the traditional server approach:
The use of stateless servers
Adherence to an event-driven paradigm

- *The use of stateless servers:* We use the term *state* to refer to data related to clients that a server stores internally.
- The term *stateful server* refers to a server that stores state information; and the term *stateless server* refers to a server that does not store state information.
- Stateful servers store information for two reasons. First, keeping information related to a given client allows a server to provide continuity across multiple contacts by the client.
- Second, keeping state information allows a server to share information among multiple clients.
- A stateful approach works well for a traditional server because the server runs on a single computer.
- Therefore, a traditional server can use mechanisms such as shared memory to allow all instances of the server to access and update the state information.
- In particular, a design that uses threads allows all instances to share global variables.

- Furthermore, because a traditional server has a long lifetime, state information usually persists across many client contacts.
  The stateful approach does not work well for a server that runs in a data center and handles large scale.
- To achieve scale, orchestration systems deploy instances on multiple physical computers, making shared memory impossible.
- To handle microservices, containers are designed with a short lifetime: the container starts, performs one function,
  and exits.
- Thus, state information does not persist for more than one client connection.
- To capture the idea that serverless computing focuses on running a single, stateless function in each container (i.e., FaaS), some engineers say that serverless computing runs *stateless functions*.
- *Because it uses containers and can run on multiple physical servers, a serverless computing system requires server code to be stateless.*

- It is important to distinguish between a stateful server and a server that keeps data in a database or persistent storage.
- Statefulness only refers to the information a server keeps in memory while the server runs.
- When a server exits, state information disappears.
- In contrast, data stored on an external storage system (e.g., a database, or an object store) persists after the server exits; a later section contains an example.
- A server can use external storage to achieve the same effect as statefulness — the server saves a copy of internal data before exiting and reads back the data when it starts
  running.
- Stored data does not count as state information because the data is not lost when the server exits.
- The point is: *Although serverless computing requires servers to follow a stateless design, a server may store and retrieve data from a database or persistent storage.*

- Serverless computing adopts the paradigm, and generalizes it.
- The underlying cloud system generates events when changes occur (e.g., when a physical server fails).
- In addition, serverless systems count each server access as an event.
- For example, in addition to a REST interface that accepts contact via HTTP, some serverless systems provide *management interface* or other *programmatic interface* components that allow a human user to interact with the system or a computer program to use a protocol other than HTTP.
- A contact from any of the interfaces counts as an event.

The Architecture Of A Serverless Infrastructure

The chief components include an event queue, a set of interface components that insert events into the queue, and a dispatcher that repeatedly extracts an event and assigns a worker node to process the event. In the case of serverless computing, worker nodes run the server code.

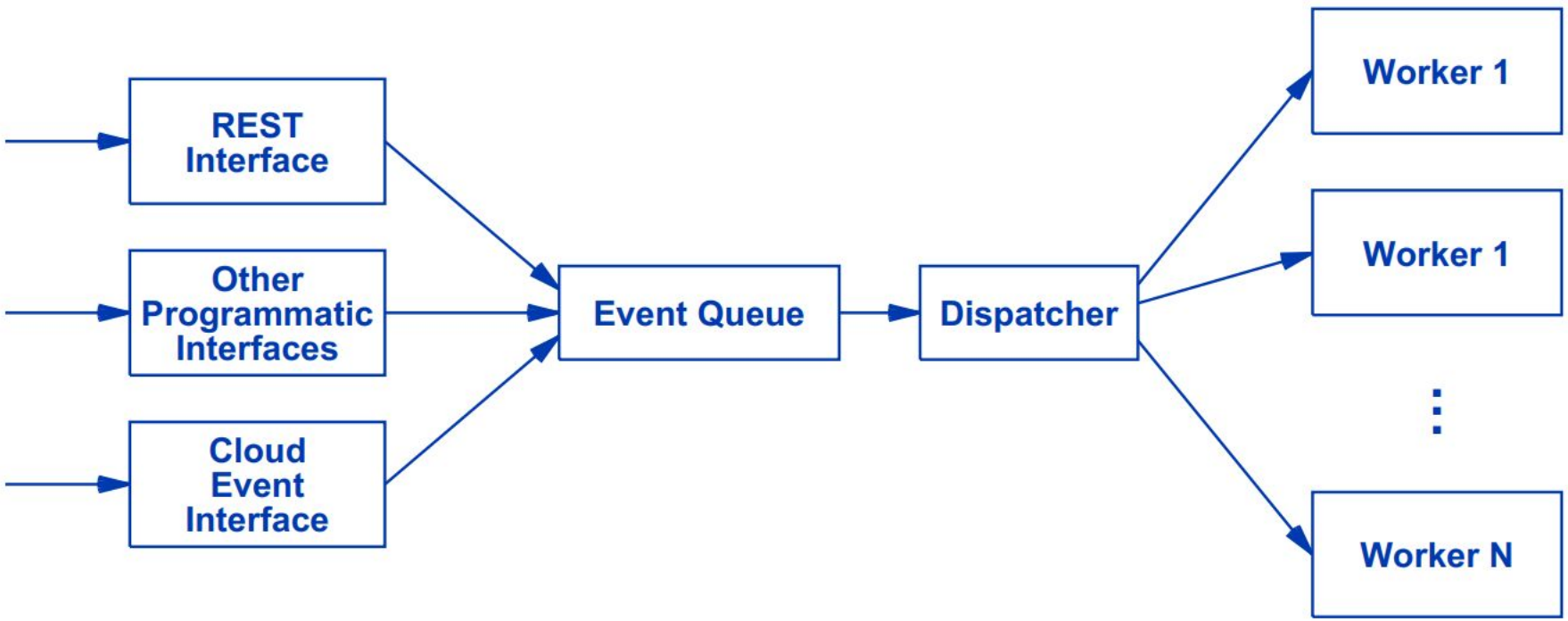

Figure 14.4 The architecture that providers use for serverless computing.

An Example of Serverless Processing

- Netflix uses the AWS *Lambda* event-driven facility for *video transcoding*, a step taken to prepare each new video for customers to download.
- The arrangement has become a canonical example of serverless computing.
- **Action Taken**
  1. A content provider uploads a new video
  2. A serverless function divides the new video into 5-minute segments
  3. Each segment is given to a separate serverless function for processing
  4. The processed segments are collected and the video is available for customers to access

The basic steps Netflix uses to transcode a video.
Events trigger each of the serverless processing steps.

When a content provider uploads a new video, the system places the new video in an Amazon S3 bucket.

The S3 object storage system generates an event that triggers a serverless function to divide the video into segments that are each five minutes long.

When a segment arrives in an
S3 bucket, another event triggers a serverless function that processes and transcodes the segment.

Thus, transcoding can proceed in parallel.

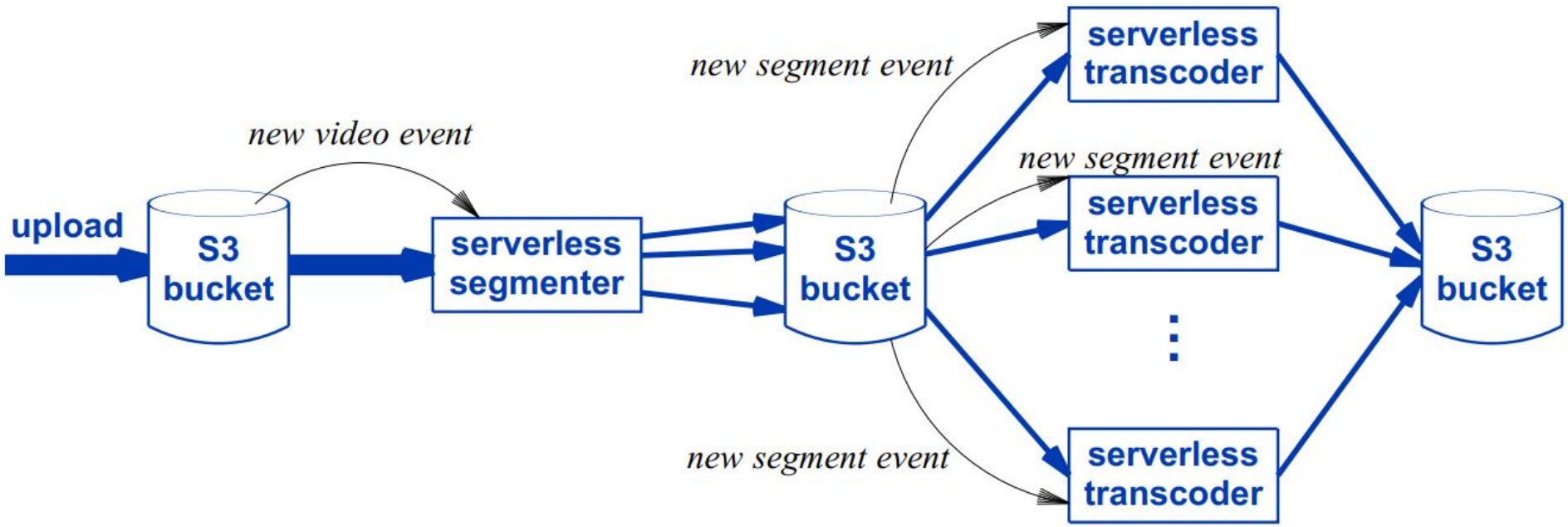

Figure 14.6 The Netflix transcoding system. A new video event causes the video to be divided into segments; a new segment event causes the segment to be transcoded.

Potential Disadvantages Of Serverless Computing

It may seem that compared to traditional server designs, serverless computing offers three unbeatable advantages: the ability to scale arbitrarily, no need to manage
servers, and lower overall cost. Despite its advantages, serverless computing does have
potential disadvantages.

First, serverless systems introduce latency. A traditional server

Summary

Serverless computing follows the traditional client-server paradigm in which one or more clients initiate contact with a server.

Unlike a traditional concurrent server that is limited to one computer, serverless computing uses cloud technologies that allow it to scale arbitrarily.

Also unlike traditional server designs, serverless computing separates server management from the core function of a server, allowing cloud providers to offer services that deploy and operate servers for their customers.

To achieve maximum flexibility, providers offer arbitrary scaling with no minimum charge, known as *zero to infinity* scaling.

The chief motivation for serverless computing lies in its economic benefits. Because a cloud provider handles the details of managing server deployment and scaling, a customer does not need to maintain staff with expertise.

Because a cloud provider only charges for the computation actually used, a customer does not pay for idle VMs or servers.

- In terms of implementation, serverless computing adopts and extends the architecture used for Kubernetes controller-based systems.
- Serverless systems use an event based paradigm in which each change in the cloud system and each contact from a client becomes an event that is added to a queue.
- A dispatcher repeatedly extracts events and assigns them to worker nodes to handle.
- Despite all the advantages, serverless computing has potential disadvantages. Unlike a conventional server that starts before clients initiate contact, serverless computing creates servers on demand, leading to a small delay.
- Unexpectedly high costs can arise from cascades of events and from microservices that divide computation onto small functions.