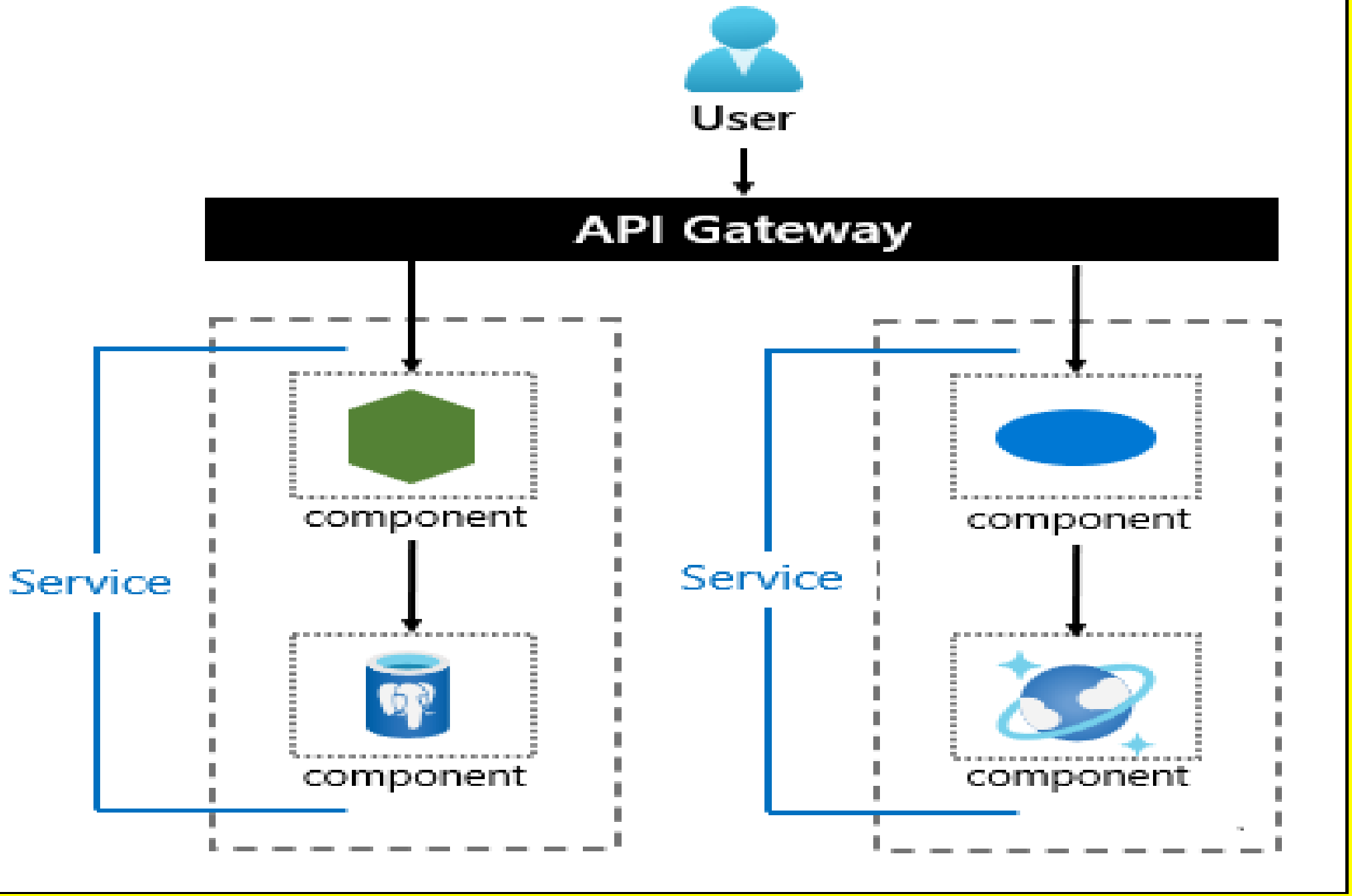
22AIE305: CLOUD COMPUTING

SERVICE ORIENTED ARCHITECTURE (SOA)

The term *service* represents a collection of components that collectively deliver specific, workload-oriented functionality to your cloud-native application.

Each service is part of a larger system that forms an application.

For example, a smart refrigerator application might have an inventory service, an ordering service, a payment service, and a management interface, each with its own, independent set of technology choices.


User
API Gateway
Service
component
component
Service
component
component

FLEXIBILITY OF SERVICES

Each service can be independently upgraded, deployed, scaled, and restarted without affecting other services' availability or performance.

This is possible because each service is assigned a virtual IP address that is visible only locally. This info is kept as Metadata.

DYNAMIC-SCALING

The loose coupling of services allows you to change the underlying technology without forcing significant code rewrites of the entire application.

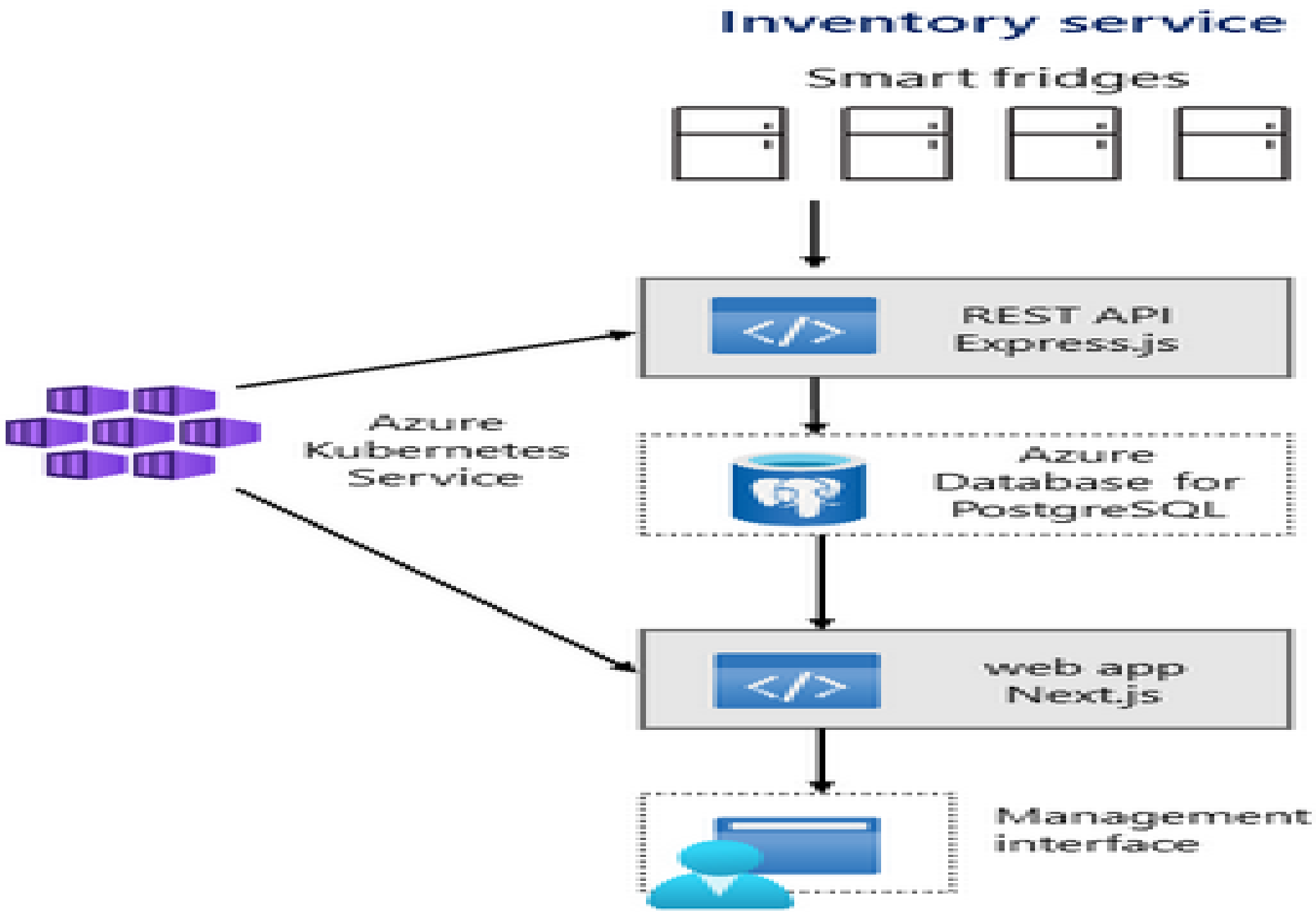
For example, the Express.js backend service could be containerized and deployed to a Kubernetes cluster, allowing it to scale dynamically (auto-scaling) based on demand.

DYNAMIC CHANGING

Similarly, the backend service to connect to a different database of choice can be modified, potentially relying on the same set of libraries.

Most common object-relational mapper libraries available with Node.js, such as Prisma or Sequelize, support a range of relational database products.

These include PostgreSQL, MariaDB, and Microsoft SQL Server.


Inventory service
Smart fridges
REST API
Express.js
Azure
Kubernetes
Service
Azure
Database for
PostgreSQL
web app
Next.js
Management
interface

	AWS	Azure	Google
Resource	aws.amazon.com/products/	azure.microsoft.com	cloud.google.com/products/
Compute	EC2 (instances), Lambda micro services (containers). Various size instances, types of images (Windows, Linux, applications), management options, LightSail private servers. Instance-based including SSD (i.e., DAS as part of the instance).	VM and container services. Various size instances, types of images (Linux, Windows, applications), management options. Instance-based including SSD (i.e., DAS as part of the instance).	Various compute engine granularities, sizes, and types (Windows, Linux) of instances and services. Instance-based including SSD (i.e., DAS as part of the instance).
Database	Various services from tables to SQL and NoSQL compatible. Database instance images also for EC2.	SQL databases, data warehouse, tables, DocumentDB, redistributed database on Azure VM..	Cloud SQL, Cloud Bigtable, NoSQL cloud datastore.
Block storage	EBS HDD and SSD	Disk (HDD and SSD)	Persistent (SSD)
Bulk Big data Object Content Archive Log and Snapshot Storage	S3 standard, RR (reduced redundancy, lower cost), IA (infrequent access, lower cost, slower access). Glacier (cold, very slow access [hours], low cost).	Blobs, various regional and geographical protection schemes. Various big data services, resources, and tools.	Object multiregional; regional, near-line, and cold are off-line. Various big data services, resources, and tools.
File storage	EFS (within AWS)	Azure File within Azure, external Windows SMB 3	
Networking	Route 53 DNS, load balancing, direct connect, VPC (virtual private cloud), Edge locations	Virtual networking, CDN, load balancer, gateways, VPN, DNS, ExpressRoute (direct connect), management	Networking, CDN, load balancing, virtual network, DNS, and interconnect

DOCKER

In a cloud-native application built with Node.js and AWS, each service is typically containerized using Docker and deployed to ECS or EKS.

AWS Lambda can also be used to run serverless functions that are triggered by specific events, such as an HTTP request or a change in a backend database.

DEPLOYMENT

To deploy an application, you can use platforms like Heroku, AWS Elastic Beanstalk, or Google Cloud Run, which support Node.js out of the box.

Make sure to set environment variables (like PORT) as needed for your cloud environment.

Create a file named server.js and add the following code:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const port = process.env.PORT || 3000;
```

CNA

Cloud native application development uses an open-source stack to deploy applications as microservices, package each part into its own container, and dynamically orchestrate the container to optimize resource utilization.

5 ARCHITECTURAL PRINCIPLES IN CNA

Containerisation

Microservices

Dynamic management

Automation

Orchestration

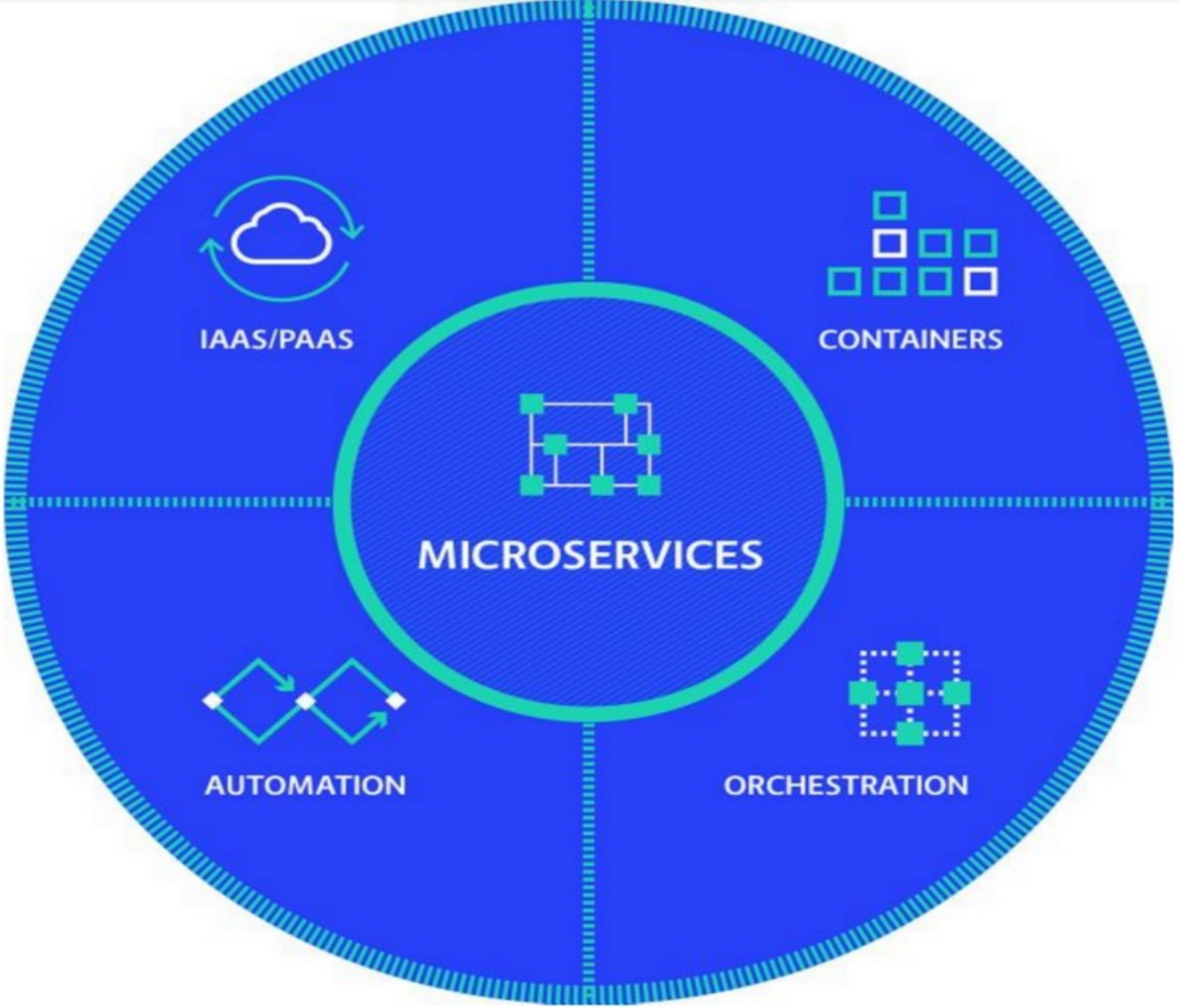


Figure 1-1. The relationship diagram between the five principles of cloud native architecture

MICROSERVICES

Microservice based cloud development uses a collection of small, decoupled component services.

Each microservice can be deployed, upgraded dynamically, scaled (using multiple instances) and restarted (when it is stuck or goes in loop) independent of other microservices used in an application with no impact on the application performance.

CONTAINER ENGINE

Cloud service providers use a **Container Engine** to manage millions of containers in the memory, orchestrate them, and optimally allocate available resources (disk-space, number of cores, RAM, Cache memory, communication bandwidth, etc) for delivery to users.

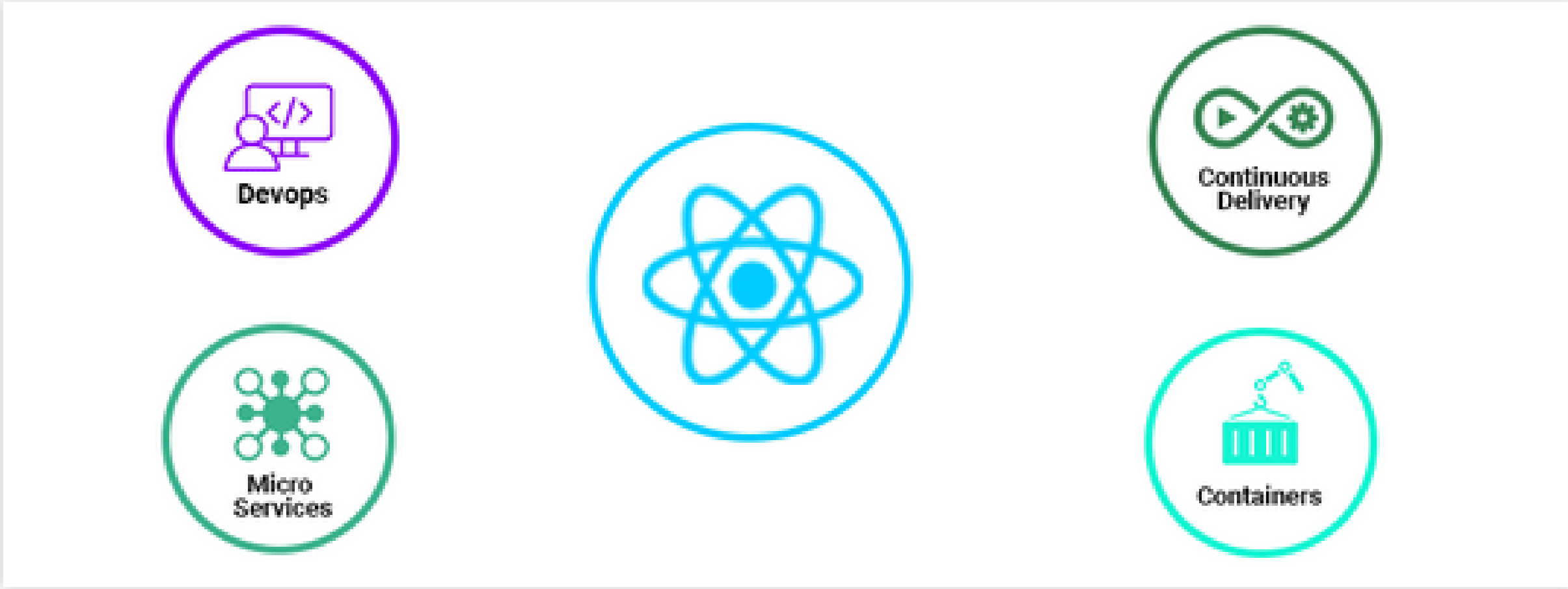
Cloud-native patterns and best practices

- While the tools and architectural standards of a cloud-native environment are important, using them effectively requires following commonly accepted best practices and patterns.
- Many of these practices fall under the umbrella of DevOps, a software development approach that emphasizes continuous improvement, automation, collaboration and shared ownership in order to improve the speed, quality and reliability of resulting products.

Architectural Patterns

- Two architectural patterns that serve as the basis for building and optimizing Cloud services are
- 1) Domain-driven design (DDD) and
- 2) Command and Query Responsibility Segregation (CQRS)
- There exist many other architectural patterns

Using React to Build Cloud-Native Applications



To build CNA with React, you will need to:

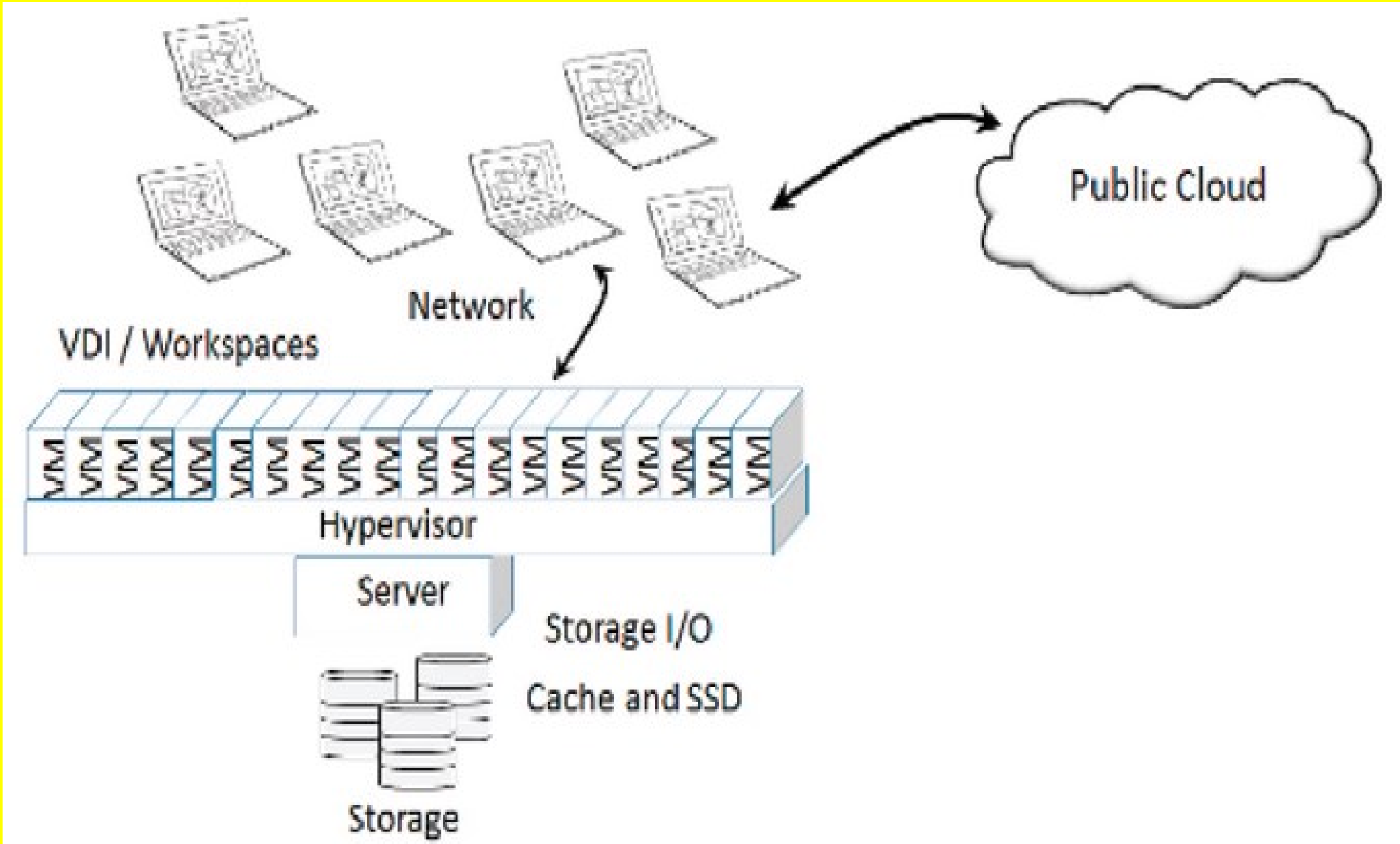
- Choose a cloud platform. There are a number of different cloud platforms available, such as AWS, Azure, and Google Cloud Platform.
- Choose a container orchestration platform. A container orchestration platform, such as Kubernetes, can help you to deploy and manage your cloud-native application.
- Create a React application. You can use a variety of tools and frameworks to create a React application, such as Create React App and Next.js.
- Deploy your React application to your cloud platform. You can use a variety of tools and services to deploy your React application to your cloud platform, such as AWS Amplify and Netlify.

Using react to build CNA

- Choose a cloud platform. There different cloud platforms available, such as AWS, Azure, and Google Cloud Platform.
- Choose a container orchestration platform. A container orchestration platform, such as Kubernetes, can help you to deploy and manage your cloud-native application.
- Create a React application. You can use a variety of tools and frameworks to create a React application, such as Create React App and Next.js.
- Deploy your React application to your cloud platform. You can use a variety of tools and services to deploy your React application to your cloud platform, such as AWS Amplify and Netlify.

Compute and Cloud storage Services

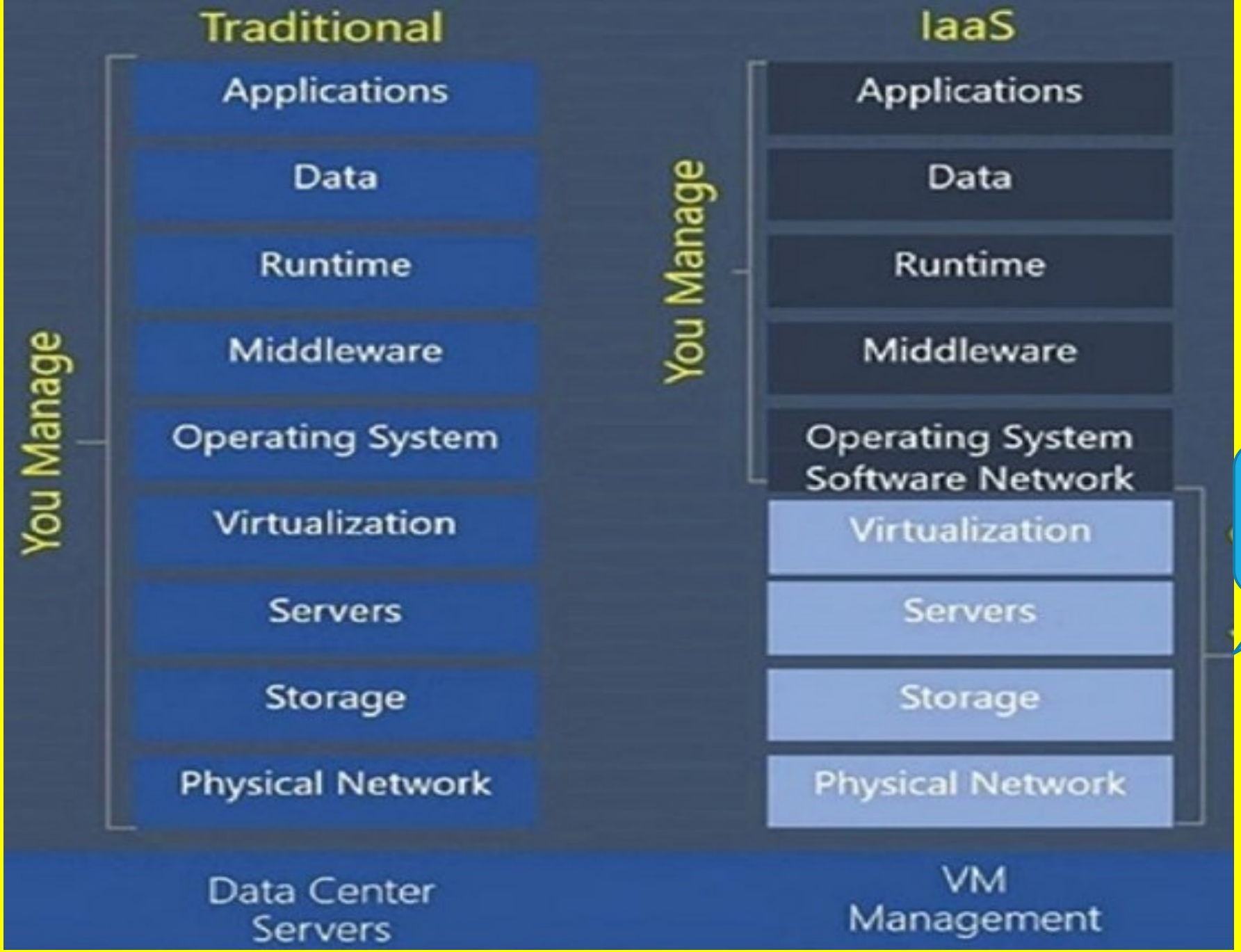
- Cloud storage can be either unmanaged or managed.
- *Unmanaged storage* is presented to a user as if it is a ready-to-use (formatted) disk drive with a specific label.
- Most user-oriented software such as file-sharing and backup consume unmanaged cloud storage.
- Applications using unmanaged cloud storage are Software as a Service (SaaS) Web services.

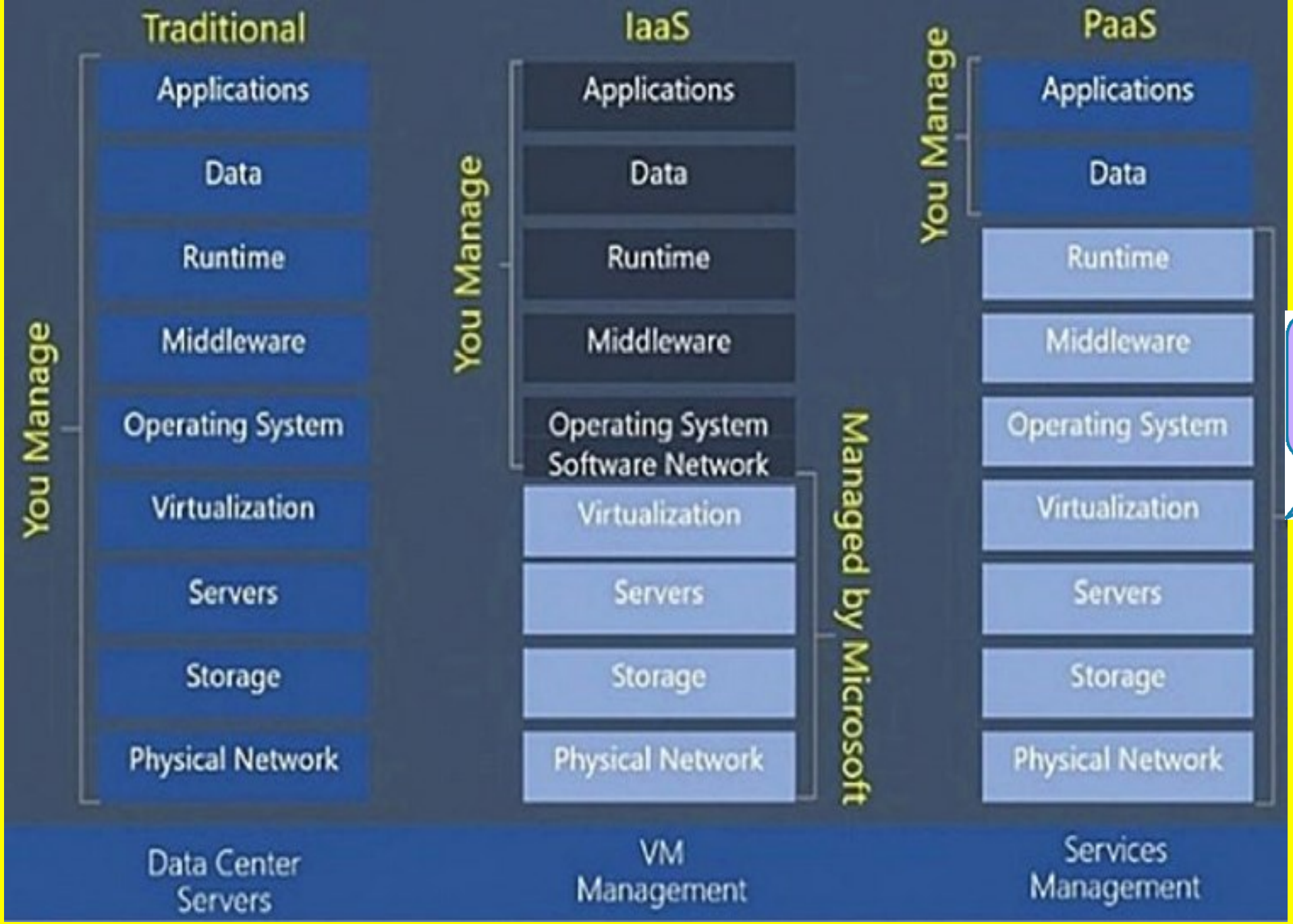

Public Cloud
Network
VDI / Workspaces
VM
Hypervisor
Server
Storage I/O
Cache and SSD
Storage

Storage devices may be broadly categorized as either block storage devices or file storage devices.
A block storage device exposes its storage to clients as Raw storage that can be partitioned to create volumes.
It is up to the OS to create and manage the file system; from the standpoint of the storage device, data are transferred in blocks.
Block storage devices offer faster data transfers, but impose additional overhead on clients.

Network Attached Storage

- The alternative type of storage is a file server, most often in the form of a Network Attached Storage (NAS) device or Virtual Hard Disk (VHD).
- NAS exposes its storage to clients in the form of files, maintaining its own file system.
- File-oriented storage devices are generally slower (with the exception of large file-streaming applications), but require less overhead from attached clients.


Traditional
IaaS
Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Physical Network
You Manage
You Manage
Applications
Data
Runtime
Middleware
Operating System
Software Network
Virtualization
Servers
Storage
Physical Network
Managed by Cloud Service Provider
Data Center Servers
VM Management


Traditional
IaaS
PaaS
You Manage
Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Physical Network
You Manage
Applications
Data
Runtime
Middleware
Operating System
Software Network
Virtualization
Servers
Storage
Physical Network
Managed by Microsoft
You Manage
Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Physical Network
Managed by Cloud Service Provider
Data Center Servers
VM Management
Services Management

Data and Application Migration

- P2V Physical to Virtual storage
- P2C Physical to Cloud
- V2V Virtual to Virtual
- V2C Virtual to Cloud
- C2C Cloud to Cloud
- V2P Virtual to Physical

Cloud Infrastructure

- The key component of an IaaS cloud architecture is the cloud OS which manages the physical and virtual infrastructures and controls the provisioning of virtual resources according to the needs of user services.
- A cloud OS's role is to efficiently manage datacenter resources to deliver a flexible, secure, and isolated multitenant execution environment for user services that abstracts the underlying physical infrastructure and offers different interfaces and APIs for interacting with the cloud.
- While local users and administrators can interact with the cloud using local interfaces and administrative tools that offer rich functionality for managing, controlling, and monitoring the virtual and physical infrastructure, remote cloud users employ public cloud interfaces that usually provide more limited functionality.

The Cloud OS

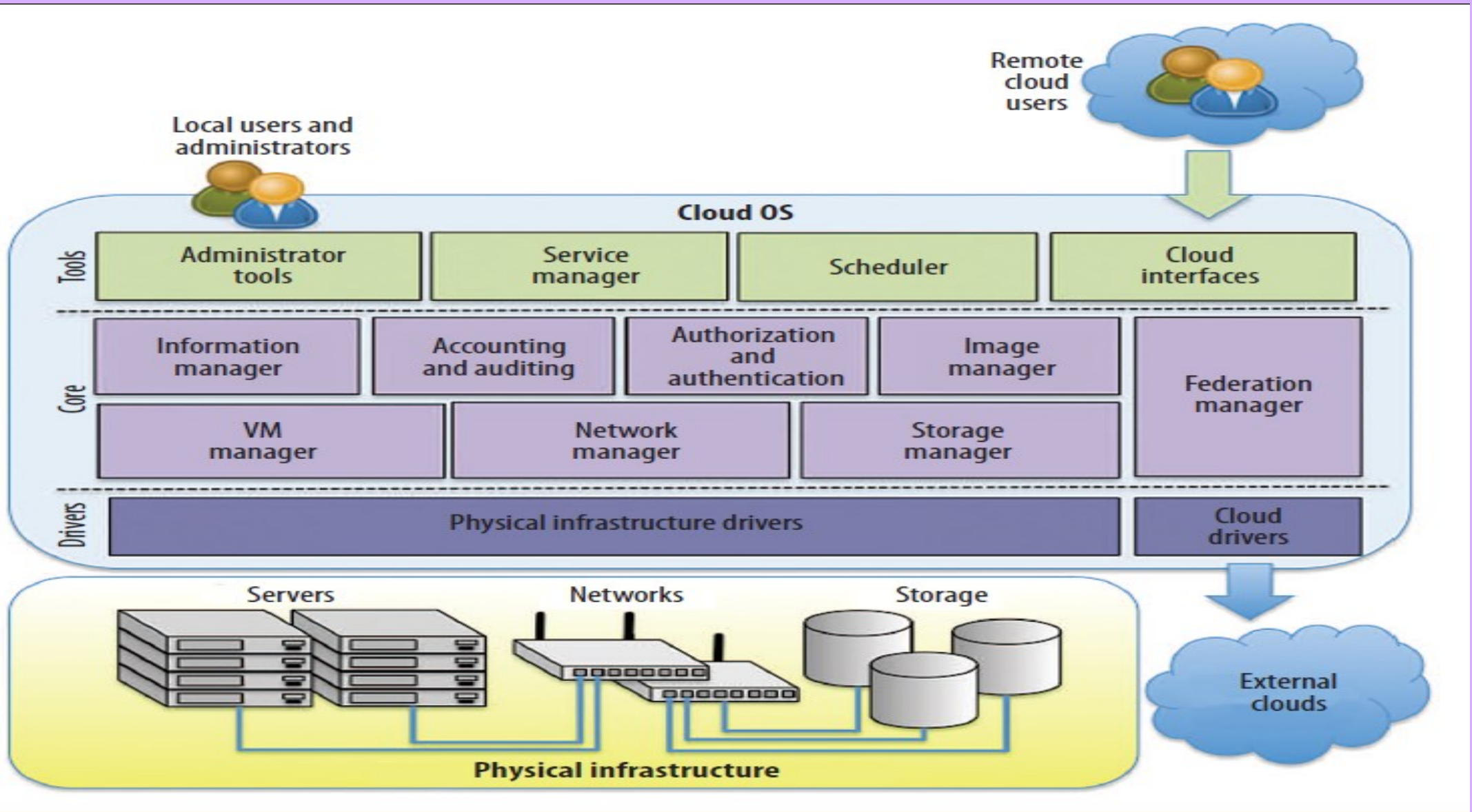


Figure 1. The cloud OS, the main component of an IaaS cloud architecture, is organized in three layers: drivers, core components, and high-level tools.

The Cloud OS

As a key component in a modern datacenter, the ***cloud operating system*** is responsible for:

1. managing the physical and virtual infrastructure,
2. orchestrating and commanding service provisioning and deployment,
3. providing federation capabilities for accessing and deploying virtual resources in remote cloud infrastructures
4. Most popular is Redhat Linux Enterprise 64 bit edn

Cloud infrastructure management

- Cloud infrastructure management comprises the processes and tools needed to effectively allocate and deliver key resources when and where they are required. The UI, or dashboard, is a good example of such a tool; it acts as a control panel for provisioning, configuring and managing cloud infrastructure. Cloud infrastructure management is useful in delivering cloud services to both:
- Internal users, such as developers or any other roles that consume cloud resources.
- External users, such as customers and business partners

Infrastructure and Cloud Drivers

- To provide an abstraction of the underlying infrastructure, technology, the cloud OS can use adapters or drivers to interact with a variety of virtualization technologies. These include hypervisor, network, storage, and information drivers.
- The core cloud OS components, including the virtual machine (VM) manager, network manager, storage manager, and information manager, rely on these infrastructure drivers to deploy, manage, and monitor the virtualized infrastructures.

In addition to the infrastructure drivers, the cloud OS can include different cloud drivers to enable access to remote providers.

OpenNebula (http://opennebula.org) is an example of an open cloud OS platform focused on datacenter virtualization that fits with the architecture proposed in Figure 1.

Other open cloud managers, such as OpenStack (http://openstack.org) and Eucalyptus (www.eucalyptus.com), primarily focus on public cloud features.

Unmanaged Cloud Storage

Service	Site	Storage Size	Maximum File Size	Direct Access	Remote Upload	Developer API
4Shared	http://www.4shared.com/	10GB free t0 100GB paid	200MB	Yes	Yes	WebDAV
Adrive	http://www.adrive.com/	50GB free to 1 TB paid	2GB	No, through a Web page	Yes	WebDAV
Badongo	http://www.badongo.com/	Unlimited	1GB	No, through Captcha	Only for paid users	
Box.net	http://www.box.net/	1GB free, 5 – 15GB paid	25MB free, 1GB paid	Yes		
Dropbox	https://www.dropbox.com/	2GB free, up to 8GB	Unlimited	Yes	Yes	No
Drop.io	http://drop.io/	100MB free	100MB	Yes	Yes	
eSnips	http://www.esnips.com/	5GB		Yes	Yes	
Freedrive	http://www.freedrive.com/	1GB		Yes	Yes	
FileFront	http://www.filefront.com/	Unlimited	600MB	Yes		
FilesAnywhere	http://file-sanywhere.com/	1GB, more can be purchased		Yes	Yes	FA API
Hotfile	http://hotfile.com/	Unlimited	400MB free	No, through Captcha	FTP	

Service	Site	Storage Size	File Size	Access	Upload	API
Humyo	http://www.humyo.com/	10GB free	None	Yes		
iDisk	http://www.apple.com/mobileme/	20/40/60GB	1GB	Yes	Yes	WebDAV
Google Docs Storage	http://docs.google.com/#	1GB free, more can be purchased	250MB			
MagicVortex	http://magicvortex.com/	2GB	2GB	Yes	Yes	
MediaFire	http://www.mediafire.com/	Unlimited	200MB	Yes	Only for paid	
Megaupload	http://www.megaupload.com/	200GB free, Unlimited paid	2GB	No, through Captcha	Only for paid	
RapidShare	http://www.rapidshare.com/	20GB	200MB free, 2GB paid	No, imposed wait time	Yes	Yes
sendspace	http://www.sendspace.com/		300MB free, 1.5GB paid	Yes	Yes, through a wizard	
SkyDrive	http://skydrive.live.com/	25GB	50MB	Yes	Yes	WebDAV
Steek	http://www.steek.com/	1GB		Yes	Yes, through DriveDrive	
Wu.ala	http://wua.la/	1GB free, additional paid	None	Yes	Yes	
ZumoDrive	http://www.zumodrive.com/	2GB free, 10 – 500GB	None	Yes	Yes	

Example providers of unmanaged cloud

- iDrive (http://www.driveway.com/),
- FreeDrive (http://www.freedrive.com/)
- OmniDrive (gonzo.com),
- XDrive (kaput)

Managed storage

- *Managed storage* involves the provisioning of raw virtualized disk and the use of that disk to support applications that use cloud-based storage.
- Storage options involved in formatting, partitioning, replicating data, and other options are available for managed storage.
- Applications using managed cloud storage are Infrastructure as a Service (IaaS) Web services

- Managed cloud storage is mainly meant for developers and to support applications built using Web services.
- Managed cloud storage is provisioned and provided as a raw disk. It is up to the user to partition and format the disk, attach or mount the disk, and make the storage assets available to applications and other users.

EC2

- A typical example of utility computing
- functionality:
  - launch instances with a variety of operating systems (windows/linux)
  - load them with your custom application environment (customized AMI)
  - Full root access to a blank Linux machine
  - manage your network’s access permissions
  - run your image using as many or few systems as you desire (scaling up/down)

Amazon Machine Images

- **Public AMIs:** Use pre-configured, template AMIs to get up and running immediately. Choose from Fedora, Movable Type, Ubuntu configurations, and more
- **Private AMIs:** Create an Amazon Machine Image (AMI) containing your applications, libraries, data and associated configuration settings
- **Paid AMIs:** Set a price for your AMI and let others purchase and use it (Single payment and/or per hour)
  - AMIs with commercial DBMS

Normal way to use EC2

- For web applications
  - Run your base system in minimum # of VMs
  - Monitoring the system load (user traffic)
  - Load is distributed to VMs
  - If over some threshold → increase # of VMs
  - If lower than some thresholds → decrease # of VMs
- For data intensive analysis
  - Estimate the optimal number of nodes (tricky!)
  - Load data
  - Start processing

Tools (most are for web apps)

- Elastic Block Store: network-attached persistent storage, can be attached to each VM instance
- Elastic IP address: programmatically remap public IP to any instance
- Virtual private cloud: bridge private cloud and AWS resources
- CloudWatch: monitoring EC2 resouces
- Auto Scaling: conditional scaling
- Elastic load balancing: automatically distribute incoming traffic across instances

Type of instances

- Standard instances (micro, small, large, extra)
  - E.g., small: 2GB Memory, 1EC2 Compute Unit (Xeon processor?), some GBs of instance/EBS storage (i.e., root volume)
- High-CPU instances
  - More CPU with same amount of memory

Amazon machine images(AMIs)

- Virtual machine images
- When users ask for a specific system
  - AWS loads the corresponding AMI and creates the virtual machine (VM)
- The user access the VM instance, no different from accessing a real remote server

- AMIs cover most common systems
  - Linux distributions
  - Windows server

AMIs with special software

- IBM DB2, Informix Dynamic Server, Lotus Web Content Management, WebSphere Portal Server
- MS SQL Server, IIS/Asp.Net
- Hadoop
- Open MPI
- Apache web server
- MySQL
- Oracale 11g
- …

Access methods

- Web interface
- Command line
- Programming Interface
  - E.g., boto python library

Simple Storage Service (S3)

- Write,read,delete **objects** 0byte-5TB, single PUT <5GB
- Namespace: two levels: buckets, keys
- Accessible using URLs

Once subscribed to a data product, you can use the AWS Data Exchange API to load data directly into Amazon S3 and then analyze it with a wide variety of AWS analytics and machine learning services. For example, property insurers can subscribe to data to analyze historical weather patterns to calibrate insurance coverage requirements in different geographies; restaurants can subscribe to population and location data to identify optimal regions for expansion; academic researchers can conduct studies on climate change by subscribing to data on carbon dioxide emissions; and healthcare professionals can subscribe to aggregated data from historical clinical trials to accelerate their research activities.

AWS DATA PIPELINE

AWS Data Pipeline is a web service that helps you reliably process and move data between different AWS compute and storage services, as well as on-premises data sources, at specified intervals. With AWS Data Pipeline, you can regularly access your data where it's stored, transform and process it at scale, and efficiently transfer the results to AWS services such as Amazon S3 (p. 74), Amazon RDS (p. 28), Amazon DynamoDB (p. 26), and Amazon EMR (p. 11).

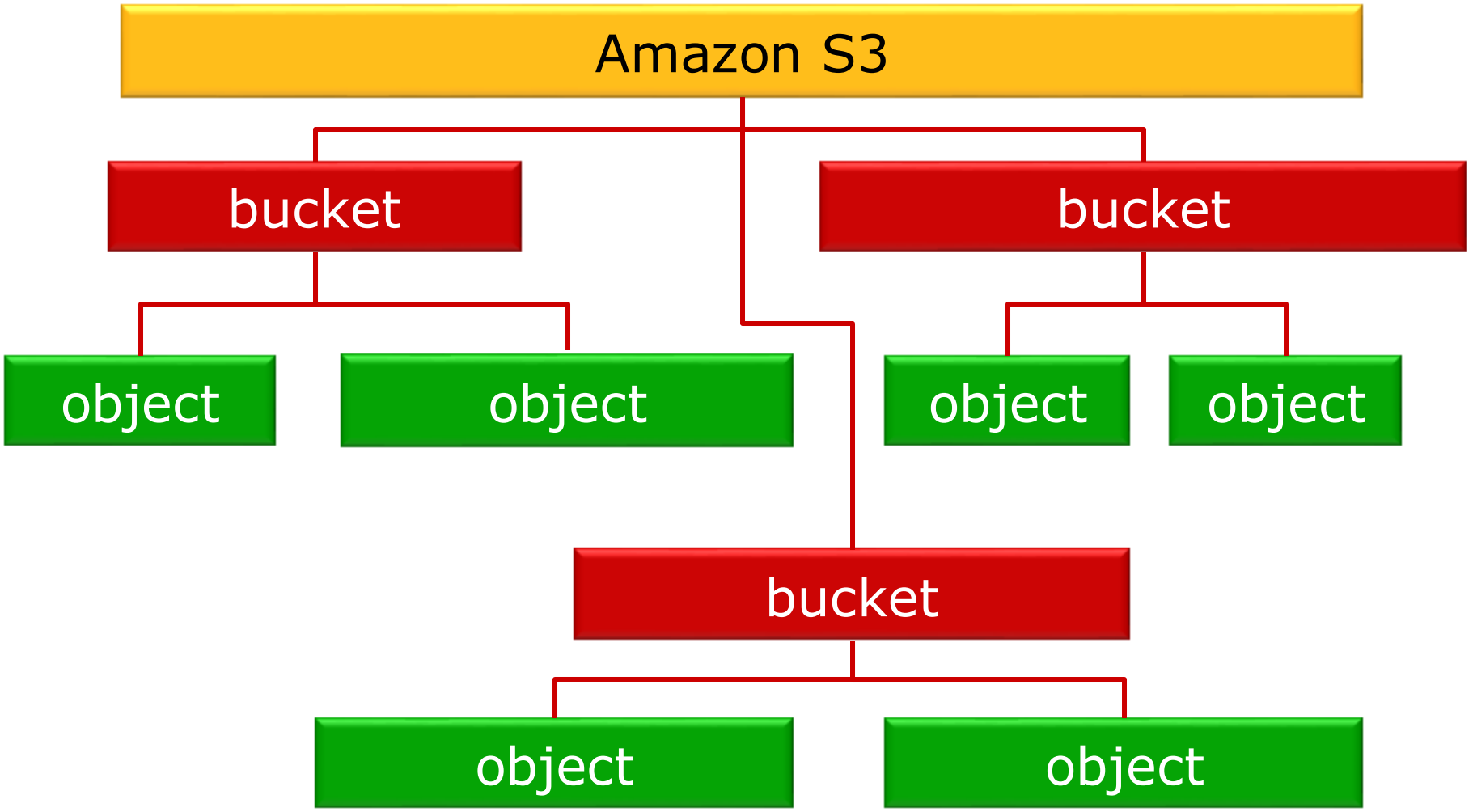
AWS Data Pipeline helps you easily create complex data processing workloads that are fault tolerant, repeatable, and highly available. You don't have to worry about ensuring resource availability, managing inter-task dependencies, retrying transient failures or timeouts in individual tasks, or creating a failure notification system. AWS Data Pipeline also allows you to move and process data that was previously locked up in on-premises data silos.

Durability

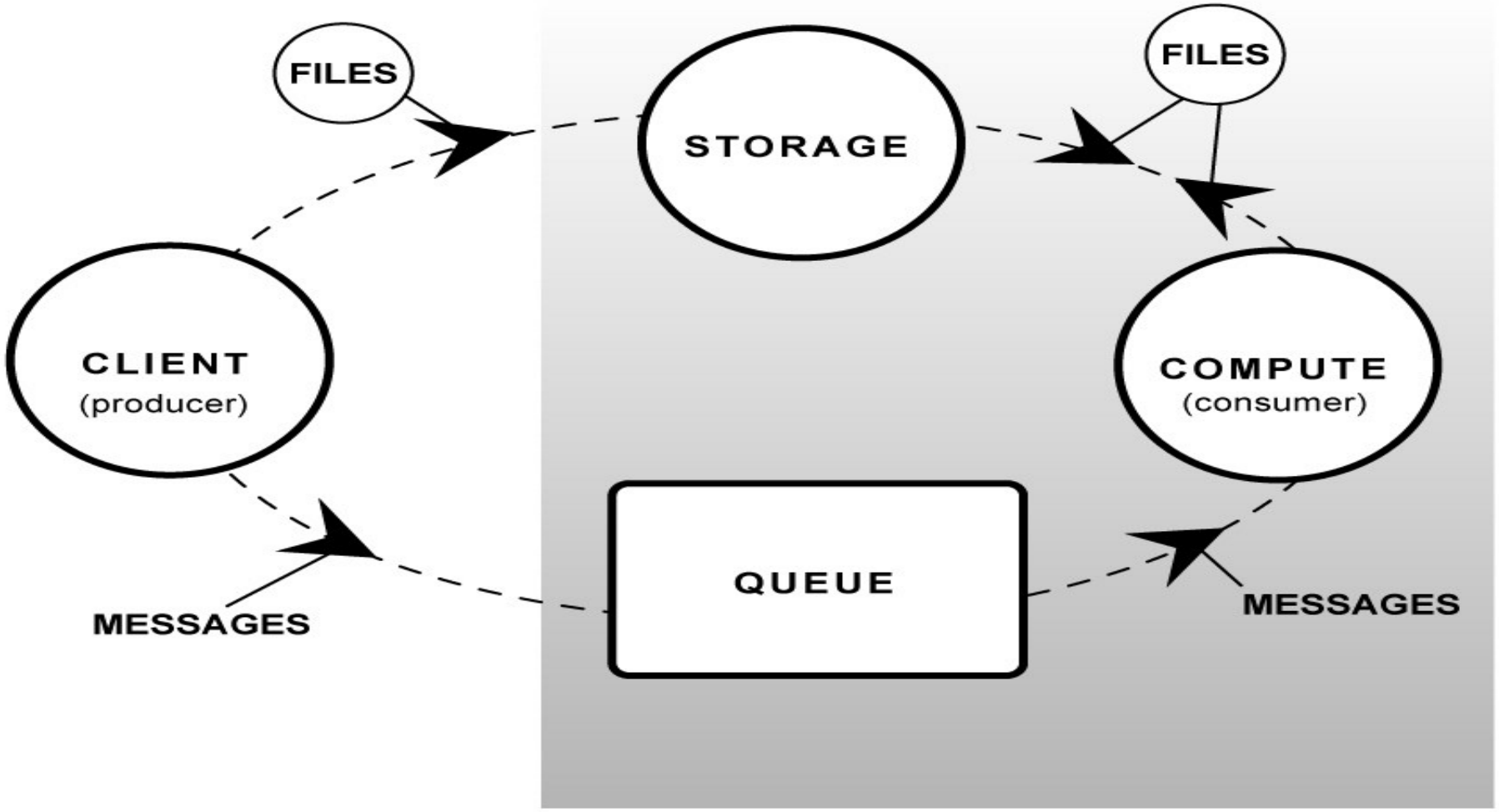
- Amazon claims about S3:
  - Amazon S3 is designed to sustain the concurrent loss of data in two facilities, e.g. 3+ copies across multiple available domains.
  - 99.999999999% durability of objects over a given year.
- Amazon claims about EBS:
  - Amazon EBS volume data is replicated across multiple servers in an Availability Zone to prevent the loss of data from the failure of any single component.
  - Volumes <20GB modified data since last snapshot have an annual failure rate of 0.1% - 0.5%, resulting in complete loss of the volume.
  - Commodity hard disks have an AFR of about 4%.
- Amazon claims about Glacier is the same as S3:
  - Amazon S3 is designed to sustain the concurrent loss of data in two facilities, e.g. 3+ copies across multiple available domains PLUS periodic internal integrity checks.
  - 99.999999999% durability of objects over a given year.

- **Beware of oversimplified arguments about low-probability events!**

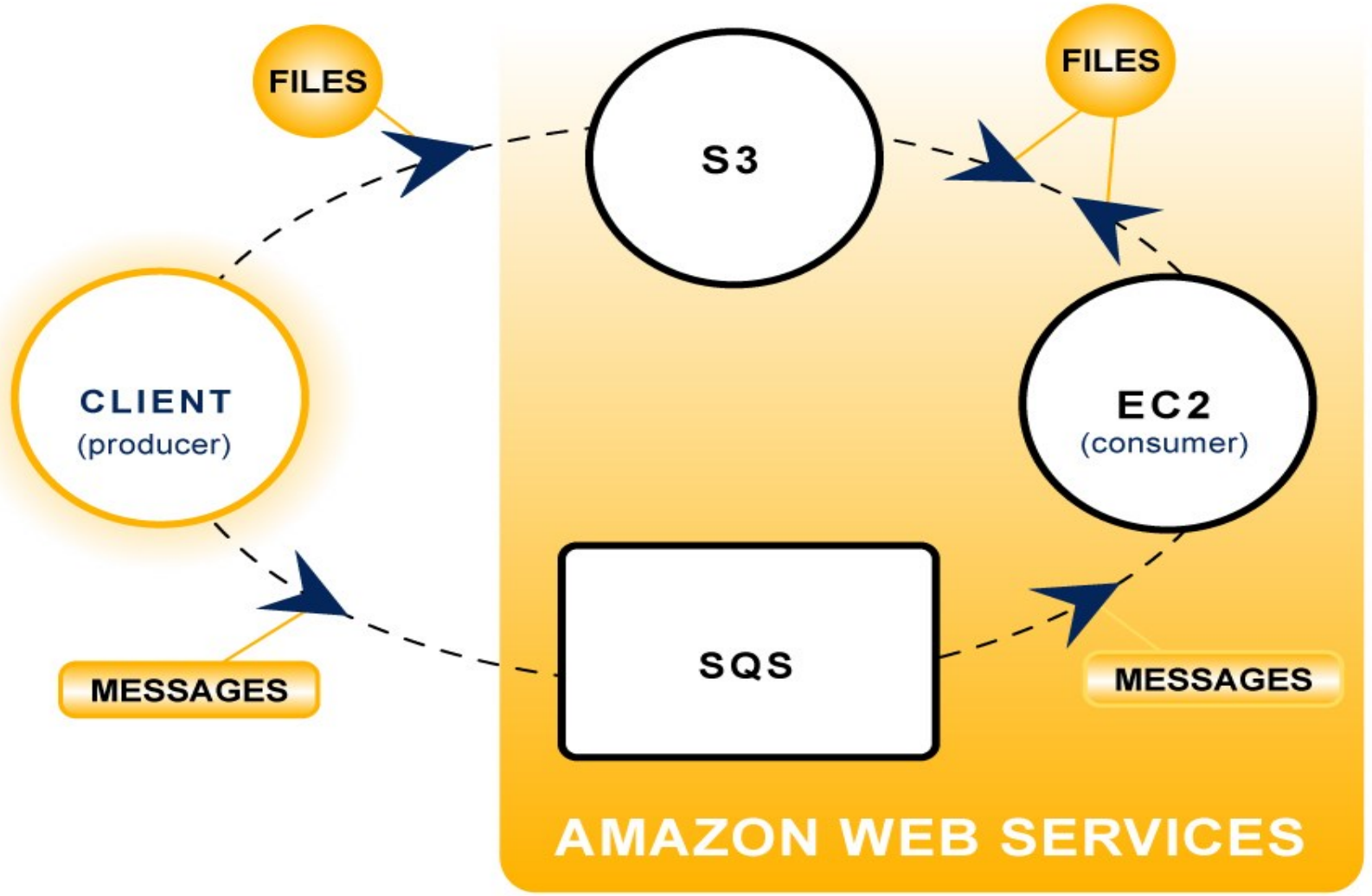
S3 namespace



A typical workflow



Workflow with AWS



CLOUD-NATIVE EXAMPLE

This example creates a simple “To-Do” task list

You have to do lots of things daily (read newspapers and technical magazines, update new happenings in information sciences, check emails, etc.)

A “to-do” list can help in organizing the things to be done on a daily basis

CREATE A DIRECTORY TO STORE FILES

```
mkdir todo-api
cd todo-api
npm init –y
npm install express body-parser cors
```

Create a file named server.js and define constants:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const app = express();
const port = process.env.PORT || 3000;
```

```
app.use(cors());
app.use(bodyParser.json());
let todos = [
  { id: 1, task: 'Learn Node.js', completed: false },
  { id: 2, task: 'Build a cloud-native app', completed: false },
];
// Get all todos
app.get('/todos', (req, res) => { res.json(todos); });
```

CREATE A NEW TODO

```
app.post('/todos', (req, res) => {
const newTodo = { id: Date.now(),
...req.body };
todos.push(newTodo);
res.status(201).json(newTodo);
});
```

UPDATE A TODO

```
app.put('/todos/:id', (req, res) => {
  const { id } = req.params;
  const index = todos.findIndex(todo => todo.id == id);
  if (index !== -1) {
    todos[index] = { ...todos[index], ...req.body };
    res.json(todos[index]);
  } else {
    res.status(404).send('Todo not found');
  }
});
```

DELETE A TODO

```
app.delete('/todos/:id', (req, res) => {
  const { id } = req.params;
  todos = todos.filter(todo => todo.id != id);
  res.status(204).send();
});

app.listen(port, () => {
  console.log(`Server is running on
http://localhost:${port}`);
});
```