Cloud-based Software

Where we've been (Recently)

- Software architecture
  - System quality attributes
  - Design issues and trade-offs
  - System decomposition
  - Service-oriented architectures (SOA)
- Distribution architecture
  - Decision-making
    - Choosing an architecture, tech
  - HTTP and REST, MVC pattern

Software organized by **components**, their **relationships** to each other & environment, and **principles** guiding design/evolution

Arrangement of **servers** for system, the **components** allocated to each, how they **interact** to provide service to users

Outline

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making

The cloud

- Very large number of remote servers
  - Offered for rent by companies that own them
  - Each may have multiple “virtual servers”
    - implemented in software rather than hardware
  - Generally thought of as on Internet
- May hear term “private cloud”
  - Means servers dedicated to one customer
  - May be physically located on-site

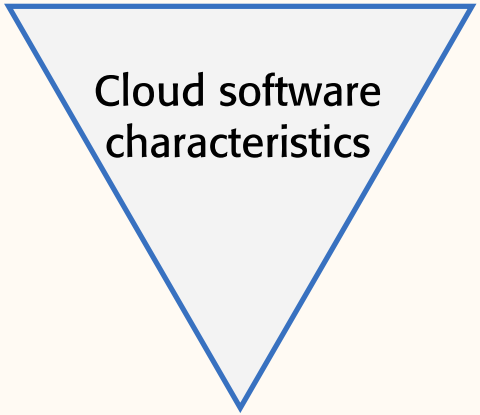
Benefits

- Convenience
  - Startup time—No wait for hardware delivery
  - Server choice—Quick/easy to upgrade/downgrade
  - Distributed dev.—everyone can contribute remotely
- Flexibility, speed in responding to changes in demand
- Cost savings—initial purchases and ongoing costs
- Disaster recovery—secondary/backup sites built-in

Cloud software characteristics

Scaleability
Maintain performance as load increases

Elasticity
Adapt the server configuration to changing demands



Resilience
Maintain service in the event of server failure

Scalability

Ability to increase/decrease IT resources as needed to meet changing demand

- Key driver of popularity with businesses
- How? Provision existing cloud infrastructure:
  - Data storage capacity
  - Processing power
  - Networking bandwidth
- Flexibility considered over long-term (Baseline)

Elasticity

Ability to grow/shrink ***dynamically*** in response to sudden change in demand

- Example: sudden spike in web traffic
- Automatic adaptation to match resources with demand in real time
- Scaling happens without disruption or down-time
- Important for businesses with unpredictable workloads
- Flexibility considered over short-term (Variations)

Resilience

Ability to remain functional even when failures are encountered

- Not about avoiding failure completely
- About **expecting failure**, and constructing cloud-native services to **respond appropriately**
- Goal: return to fully-functioning state ASAP
  - Could be immediate if have secondary resource active

Outline

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making

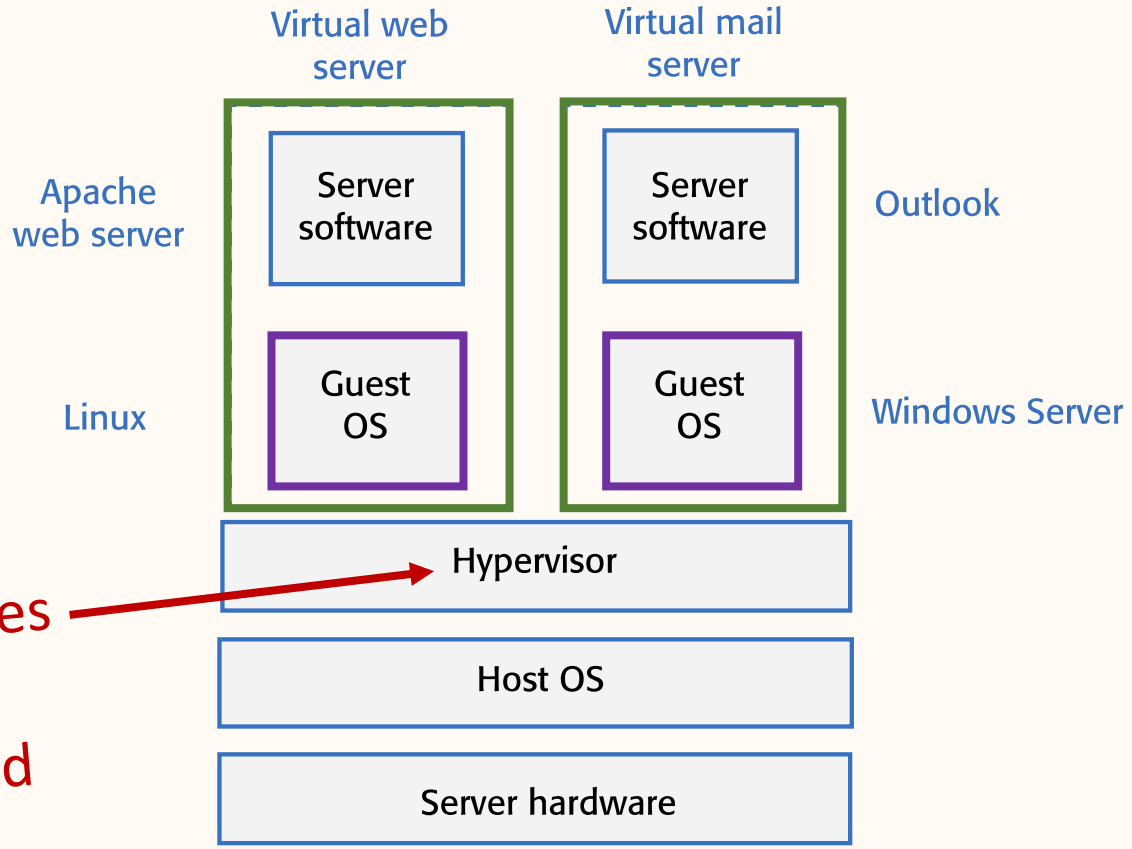
Virtual server

- Stand-alone software system that can run on any hardware in the cloud (“**run anywhere**”)
- Runs on underlying physical computer (server)
- Consists of
  - An **operating system** (OS) plus
  - Set of **software packages** that provide required functionality
  - No **external dependencies** (all dependencies included)

Virtual machines (VMs)

Emulation of a full computer system

- Run on physical server hardware
- Used to implement **virtual servers**



Allocates physical resources like CPU, memory to individual VMs as required

Main drawback to VMs

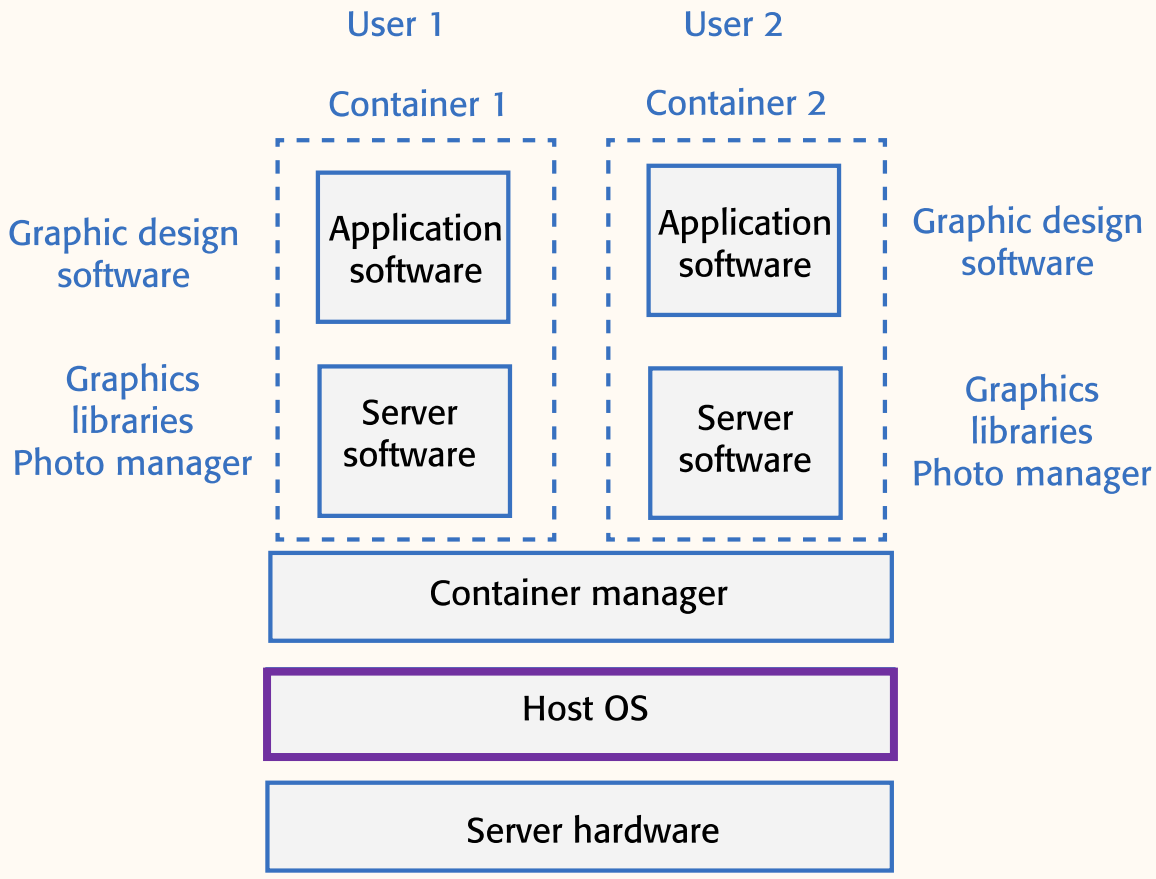
- VM creation involves loading & starting up a large and complex OS
- Excessive when have **multiple VMs** on the same physical machine with **same OS**
- Often don't need generality of a "heavy" VM
- A simpler, lightweight, virtualization technology may be used…

Container

- Lightweight virtualization technology for running applications in the cloud
- Allows independent servers to share same OS
- Particularly useful for providing isolated application services where each **user** sees **own version of an application**

Containers vs. VMs

- Containers use the **host OS** instead of each getting own OS



Using VMs and containers

- VM is best option:
  - Application depends on a large, shared database providing continuous service
- Containers particularly effective:
  - Running small applications such as stand-alone services
- VMs & containers can coexist on the same physical system (containers running on VM)

Benefits of containers (1/4)

Solve the problem of **software dependencies**

- No concerns about different libraries, other software on application server vs. dev. server
- Can ship container including all support software product needs (rather than rely on installation of stand-alone software on existing OS)

Benefits of containers (2/4)

Provide a mechanism for **software portability** across different clouds

- Docker containers can run on any system or cloud provider where the Docker daemon is available

Benefits of containers (3/4)

Provide an efficient mechanism for **implementing software services**

- Easy to replicate the same service to scale out
- Support the development of service-oriented architectures (Chapter 6)

Benefits of containers (4/4)

Simplify **adoption of DevOps** (Chapter 10)

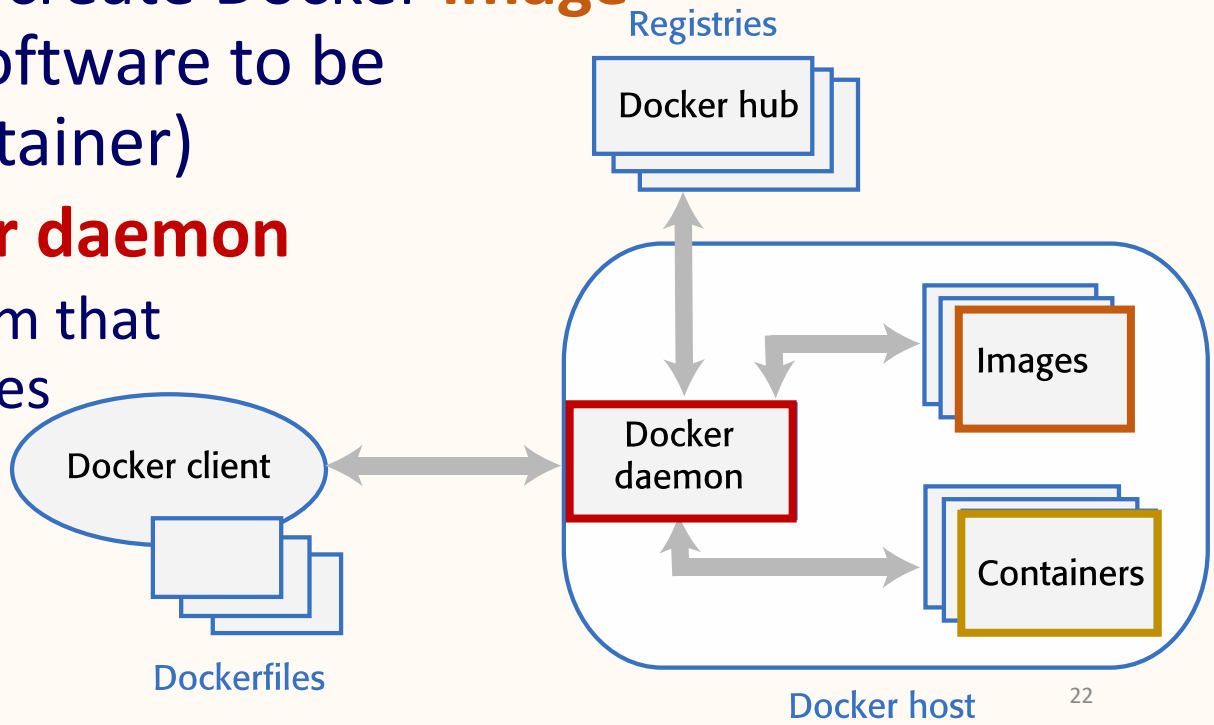
- Approach to software support where the same team responsible for both dev. and support
- (More on this later)

Outline

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making

Docker

- Container management system
  - Allows users to create Docker **image** (collection of software to be included in container)
  - Includes **Docker daemon**
    - Run-time system that creates/manages **containers** according to Docker **images**

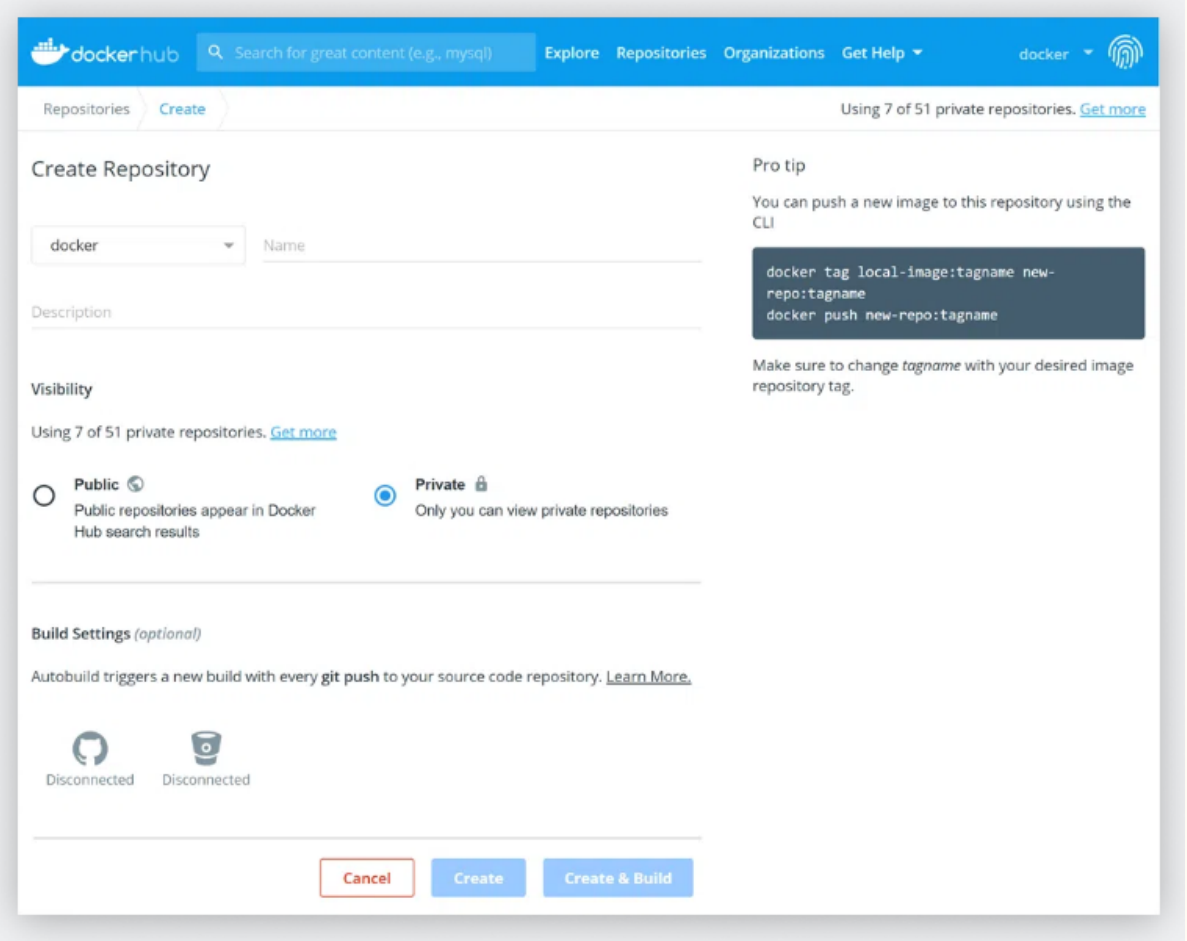


Images and containers

- Docker images—Directories that can be run, archived, & shared on different Docker hosts
  - Includes everything needed to run—binaries, libraries, system tools, etc.
- **Container**—Running instance of an image
- Client(s) can access the running container via the **Docker daemon**

Docker Hub

Repository service provided by Docker for finding/sharing container **images** with team



Docker Hub key features

- **Private repositories**—Push/pull **images**
- **Automated builds**—Automatically build **images** from GitHub, BitBucket, push to Hub
- **Teams & orgs**—Manage access to **private repositories**

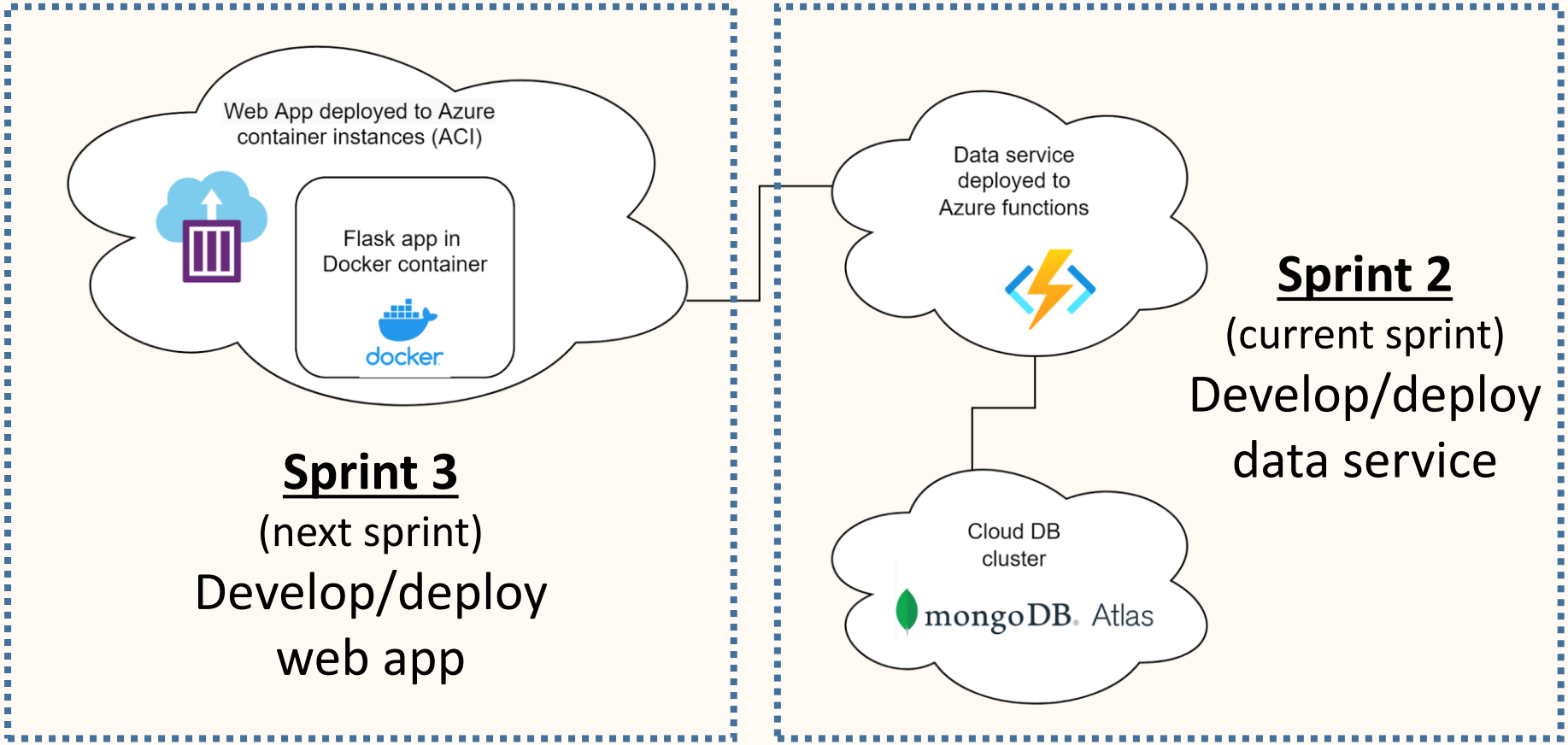
Docker Hub key features (cont'd)

- **Official images**—Pull/use high-quality **images** provided by Docker
- **Publisher images**—Pull/use high-quality **images** from external vendors
- **Webhooks**—Trigger actions in other services after successful push to **repository**

Bridge network

- Docker mechanism that enables containers to communicate with each other
- Can create systems made of communicating components, each running in own container
- Can deploy many communicating containers to implement complex distributed system
  - Use a management system such as **Kubernetes** to manage the set of deployed containers

The Cloud and your project



Outline

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making

Everything as a service (XaaS)

- Describes general category of services related to cloud computing & remote access
  - IaaS—Infrastructure
  - PaaS—Platform
  - SaaS—Software
  - FaaS—Functions
- Recognizes vast number of products now delivered to users as service over the internet

Some aaS's

- On-Site
  - Manage all your own infrastructure
- Each level involves more mgmt. by service provider
  - Infrastructure
  - Platform/OS
  - Software

Infrastructure as a Service (IaaS)

- 1 step away from on-premises infrastructure
- Pay-as-you-go service: 3rd party provides services like storage & virtualization via cloud
  - Service purchaser is responsible for OS, any data, applications, middleware, etc.
  - No need to maintain/update own on-site datacenter because the provider does this
  - Purchaser controls infrastructure via API or dashboard

Platform as a Service (PaaS)

- Provider hosts own hardware, software
  - Delivers this platform through internet
  - Primarily useful for developers and programmers who write the code, build, and manage apps
  - Environment to build, deploy helps users avoid
    - Having to build/maintain the platform
    - Software updates, hardware maintenance
- PaaS is like IaaS, with pre-installed OS
  - IaaS involves VMs; PaaS involves containers

Software as a Service (SaaS)

- AKA cloud application services
- Most comprehensive form of cloud service
  - Delivers entire application managed by provider
  - **End-user SaaS** examples: Outlook, Gmail
    - Users can log in to access service from anywhere
  - **Developer SaaS** examples: Google Maps API, OpenAI API
    - Software services accessed by other services using REST APIs

Functions as a Service (SaaS)

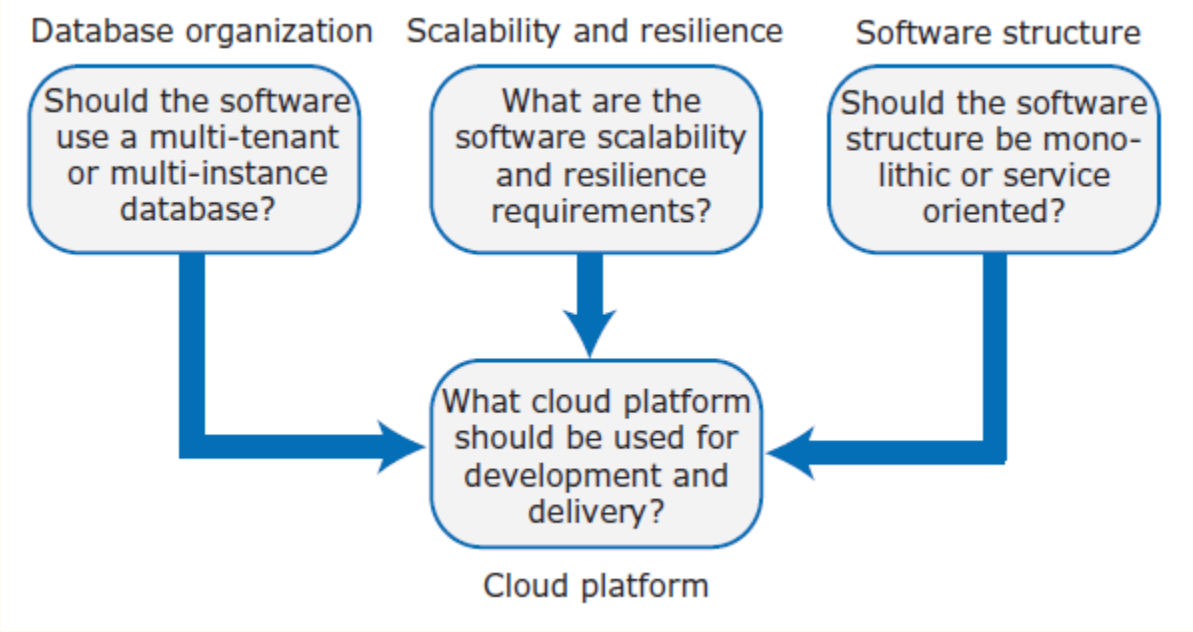
- AKA “Serverless”
- User can execute code in response to events
  - Without managing any server/infrastructure
  - Cloud provider takes care of infrastructure
  - Examples: Amazon Lambda or Azure Functions.
- User pays only for time function is executing
  - Not renting underlying server
  - Leads to large savings for on-demand services (such as recovery) that don’t run continuously

Outline

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making

More decision making

- Database organization
- Scalability and resilience
- Cloud platform
- Software structure/architecture (revisited)



Multi-tenant vs. multi-instance

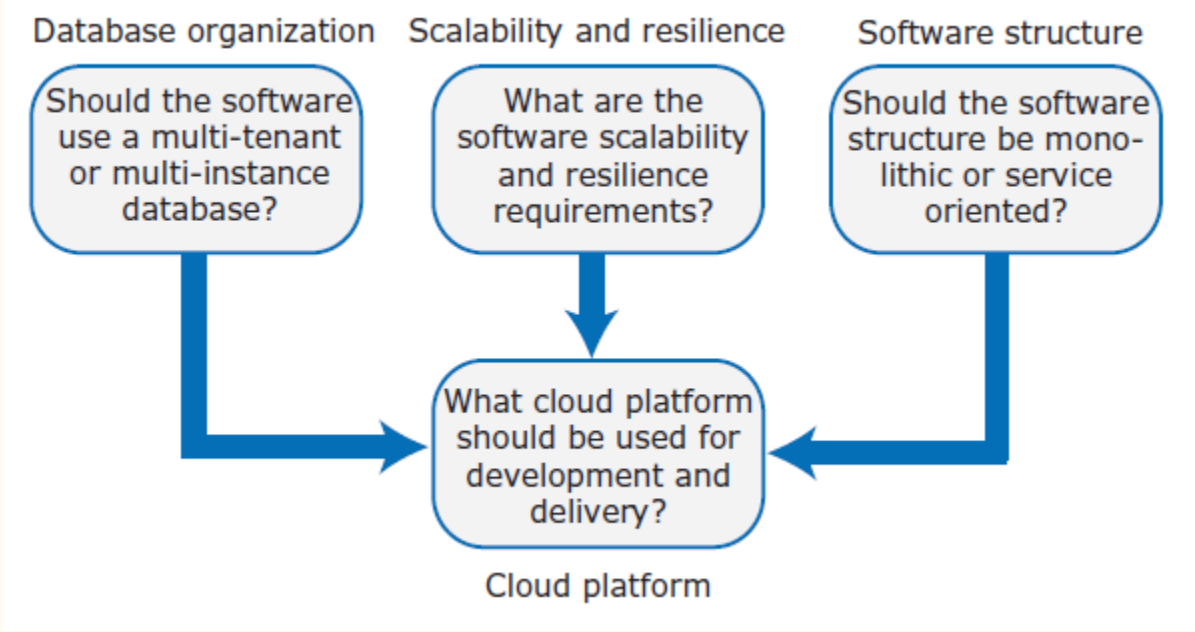
- **Multi-tenant systems**
  - All customers served by single system instance of the system and one multi-tenant database
  - Database data partitioned so that customer companies can store/access only their own data
- **Multi-instance systems**
  - Provide a separate copy of the system and database for each customer

Multi-instance systems

- Each customer has own system that adapted to their needs (database, security controls…)
- Two types of multi-instance systems:
  - **VM-based**
    - Software and database in VM per customer
  - **Container-based**
    - Software, database running in isolated set of container
    - Generally uses microservices architecture, each service running in a container and managing its own database

More decision making

- Database organization
- Scalability and resilience
- Cloud platform
- Software structure/architecture (revisited)



Achieving scalability

- Scalability achieved by either
  - Adding new virtual servers (**scaling out**) or
  - Increasing power of a system server (**scaling up**) in response to increasing load
- In cloud-based systems, scaling out is the approach most commonly used

Considerations when scaling out

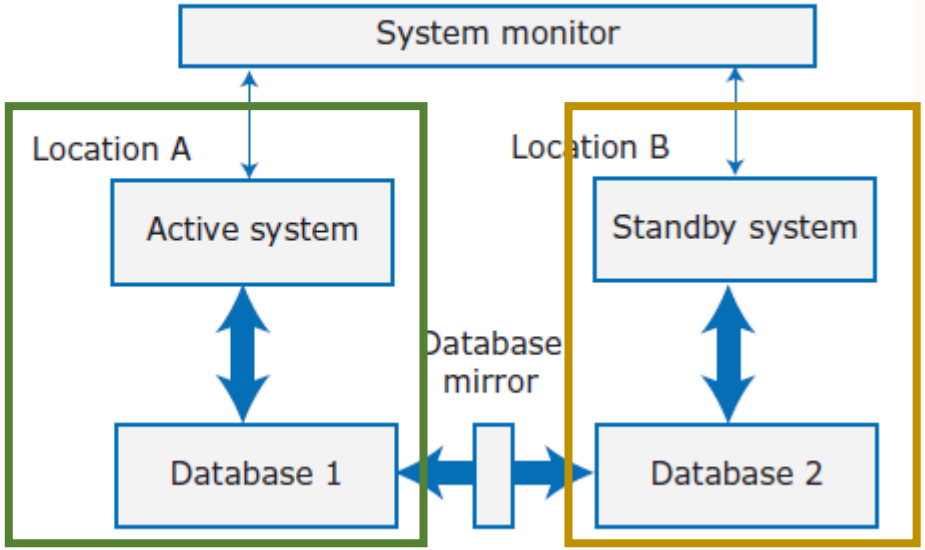
- Software design
  - Organized as individual components that can be replicated, run in parallel
- Load-balancing hardware or software
  - Used to direct requests to different instances of these components
  - When software developed using the cloud provider's PaaS support, it can automatically scale your software as demand increases

Achieving resilience

- Copies of software, data kept in different locations
- Database updates are mirrored
  - **Standby database** is working copy of **operational database**
- System monitor continually checks the status
  - Switches to **standby system** automatically if **operational system** fails

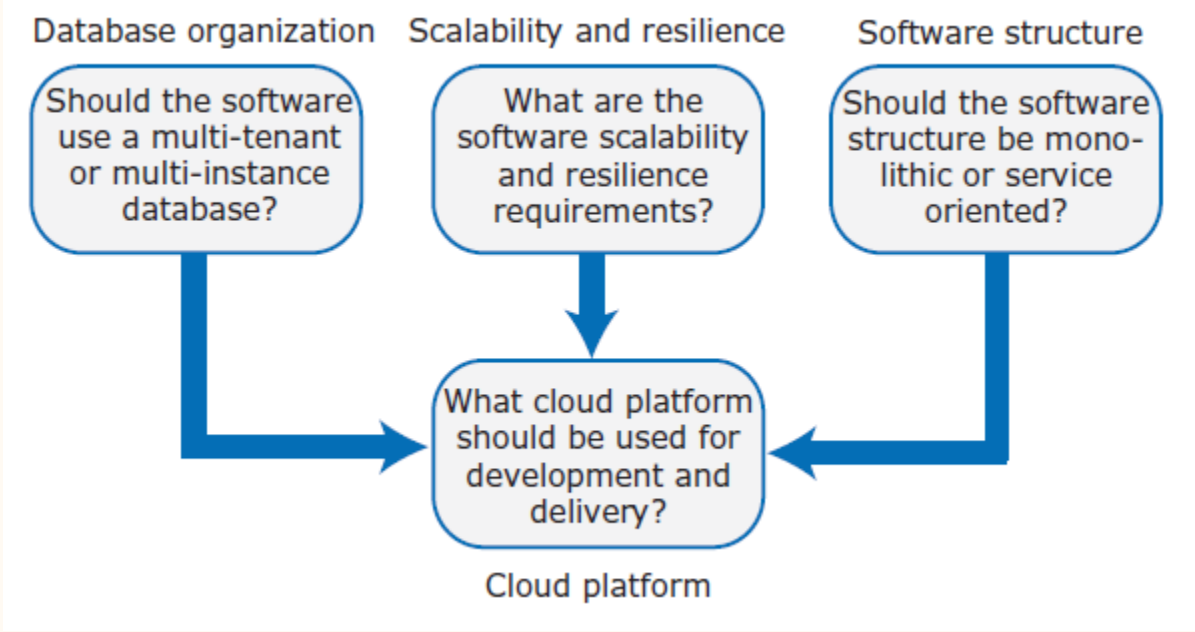
Achieving resilience

Figure 5.15 Using a standby system to provide resilience



More decision making

- Database organization
- Scalability and resilience
- Cloud platform
- Software structure/architecture (revisited)



Choosing a cloud platform

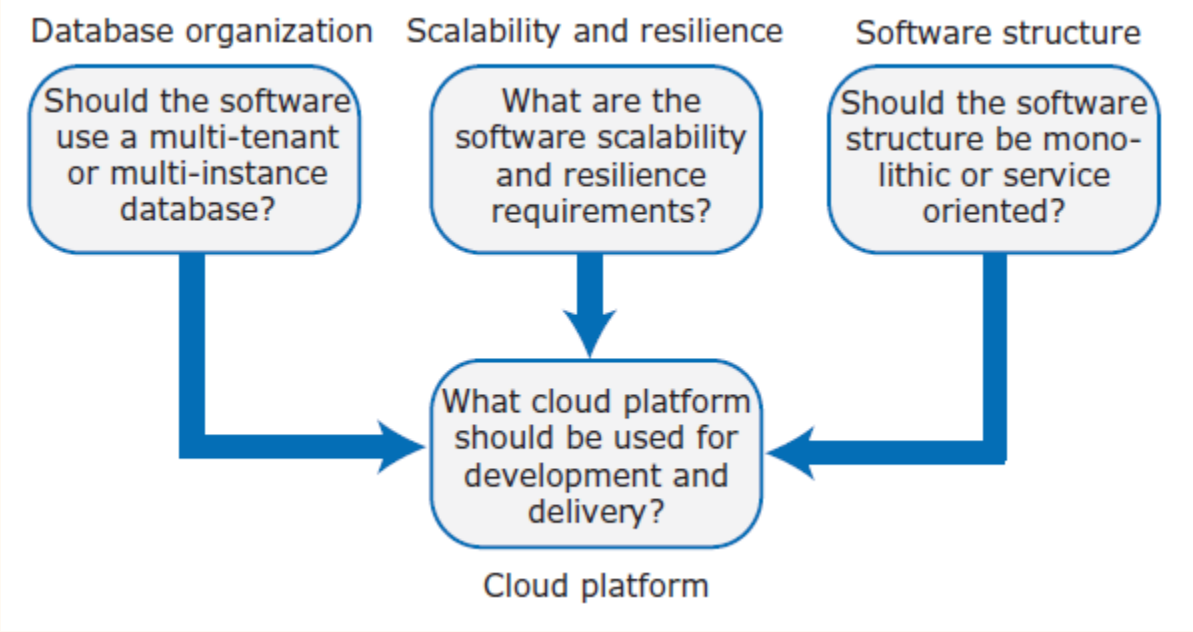
- Involves both technical and business issues
- Technical issues—what platform supports
  - Expected load and load predictability
  - Resilience
  - Supported cloud services
  - Privacy and data protection

Choosing a cloud platform

- Involves both technical and business issues
- Technical issues—what platform supports
- Business issues
  - Developer experience
  - Cost
  - Target customers
  - Portability and cloud migration
  - Service-level agreements

More decision making

- Database organization
- Scalability and resilience
- Cloud platform
- Software structure/architecture (revisited)



Monolithic vs. service-oriented

- Monolithic application: self-contained and independent from other applications
- Service-oriented: system decomposed into fine-grain, stateless services that interact
  - Independent, stateless services can be replicated, distributed, and migrated between servers
  - Well-suited to cloud-based software with services deployed in containers
  - (more on this in the next chapter!)

Easier to get started

Better for large deployments

Summary

- The cloud
- Virtualization & containers
- Docker container management system
  - The cloud & your project
- Everything/anything as a service (XaaS)
- More decision-making