Cloud Applications Architecture

Course 8 - PaaS & Managed Services

What is PaaS?

IaaS, PaaS, SaaS - What We Manage

On-Premises	IaaS	PaaS	SaaS
Application	Application	Application	
Data	Data	Data	
Runtime	Runtime		
Middleware	Middleware		
OS	OS		
Virtualization			
Server			
Storage			
Networking			

IaaS, PaaS, SaaS - What We Manage

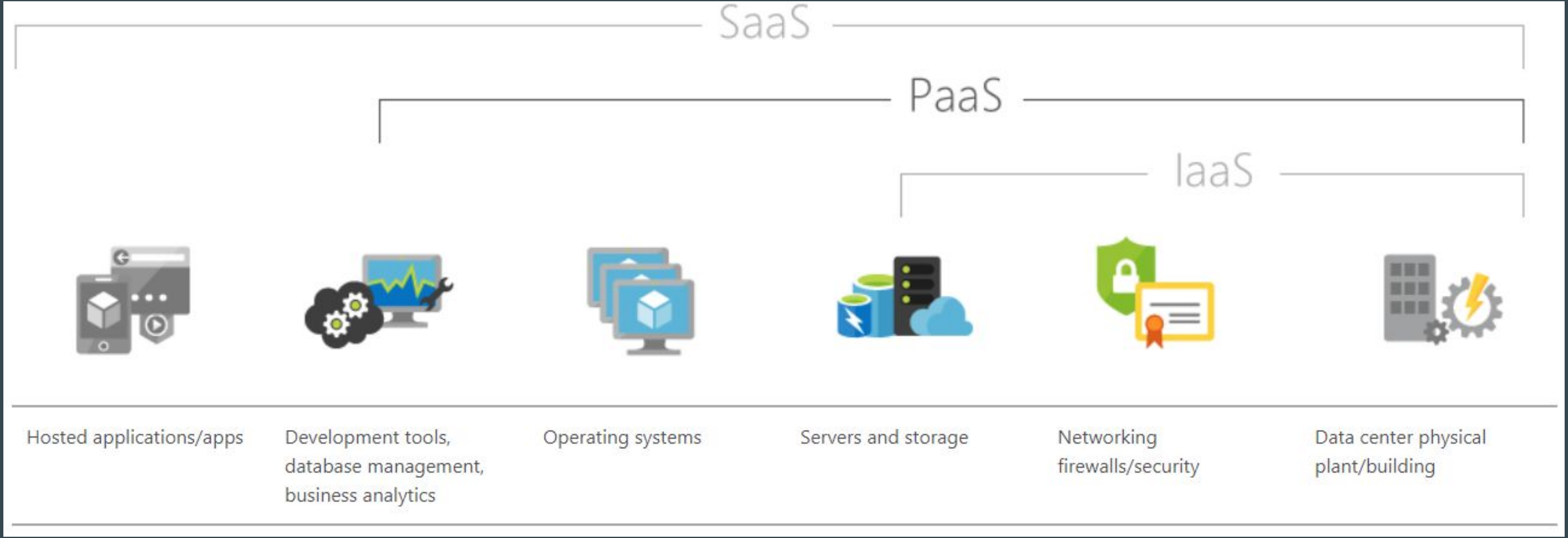
SaaS
PaaS
IaaS
Hosted applications/apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking firewalls/security
Data center physical plant/building

Image from Azure

What is PaaS?

- Based on IaaS
- Services for common challenges/requirements (e.g. DB)
- Managed services
- W**ide** range of services
  - Compute
  - Data storage
  - Monitoring
  - User management
  - Encryption
  - Combination of some or all

Pros of PaaS

- Less to worry about => focus on business and actual features
- Dedicated/specialized teams manage them
- Different billing
- Can be cheaper
  - Billing might be more favorable
  - No administration effort/cost

Cons of PaaS

- Opinionated, you have to work with what the provider gives you
- Can be more costly to run
- Vendor lock-in

When to Use PaaS

- Fast prototyping/product validation
- When the upsides outweigh the downsides
- When working with a small team
- When you understand and accept the lock-in to a decent degree

When Not to Use PaaS

- When building “exotic” products
    - E.g. banking systems
- When building ultra-high performance systems
    - Especially if the latency is decisive
    - E.g. high-frequency trading
- When control/security is a must
    - Most PaaS will provide better security than we can achieve, however we cannot completely audit it.
- When working with “special” licenses
    - Some require dedicated hardware

Examples - General/Compute

Google App Engine

Azure App Service

AWS Elasticbeanstalk

Cloud Foundry

Examples - Databases

AWS RDS

Azure CosmosDB

MongoDB Atlas

Environment Parity

“It works on my machine”

Environment Parity

One of the 12 factors.

Local, dev, stage, …, prod should be as similar as possible.

More and more relevant as we start using more services.

Achieved by using the right tools.

Containers

“Lightweight VMs”

Unit containing the application and all its dependencies.

Useful for:

- Creating identical/similar environments
- Quickly provisioning dependencies locally (DBs, caches, queues etc)
- Keeping your computer clean

Docker’s explanation

Docker

One container implementation.

Leverages **Linux** tools for creating the “lightweight VMs”.

- **Namespaces**
  - Makes the container look like a complete VM
  - Isolates it from other containers/processes
- **Control Groups**
  - Limits resources
- **UnionFS**
  - Keep the size small
  - Copy on write

Docker docs

Docker - Terminology

Images

- Similar to OOP classes that make heavy use of inheritance
- Stored locally and/or on container registries (e.g. Docker Hub, GCR, ECR, etc.)

Containers

- Processes running based on given images. Similar to OOP objects.
- Can be created, started, stopped, removed.
- Allows us to execute commands within it (similar to a VM).

Services

- Manage, scale and load balance containers across multiple hosts

Docker - Terminology

Volumes

- Needed since containers are ephemeral and isolated
- A way to persist data separate of the container’s lifecycle

Networks

- Enable communication between multiple containers.
- Allow us to control network isolation separately

Demo

Useful tutorials:

- The official getting started guide from Docker

Docker Compose

Useful for running and managing multiple containers (on the same host)

Has a more declarative approach - *docker-compose.yml*

Kubernetes

Container orchestrator

Works with Docker and other container engines (link)

Useful for:

- Running (heterogeneous) containers in a cluster (and handling communication)
- Optimizing resource utilization
- Large/variable workloads
- Rolling updates

Offers what cloud providers already offers, but in a cloud/provider-agnostic way.

Kubernetes - Terminology

Cluster

- The underlying infrastructure (can be any IaaS and on-premises)

Nodes

- VMs available to kubernetes (i.e. having kubelet installed)

Pods

- Smallest deployment unit.
- Runs one or more (related) containers
- Multiple pods can be deployed on a node
- Pods are usually managed by other components
    - Deployment, StatefulSet, DaemonSet

Kubernetes - Terminology

Services

- Pods cannot be reached by default
- Services expose pods to other pods or to the outside world
- Multiple types: ClusterIP, NodePort, LoadBalancer (this will create the corresponding service offered by the cloud provider)

PersistentVolumes

- Represent a physical storage volume (e.g. EBS on AWS)
- Consumed by Pods through PersistentVolumeClaims

Relevant Resources

Tools

- **Helm** ~ package manager for kubernetes
- **Skaffold** ~ streamlines local development (e.g. hot-reload) and deployment

Tutorials

- The official kubernetes tutorials
  - E.g. Deploy Wordpress