AIC Summary

***Warning**: Use this summary at your own risk and with caution. I may have left out things or not covered them completely.*

4 Evolutions: Infrastructure/Software/Processes/Teamwork:
Dependencies between systems no longer fixed and predetermined → need abilities to deal with context changes and unanticipated events and people, IoT supports active objects.

Service-oriented approaches: Apache River, OSGi, UPnP

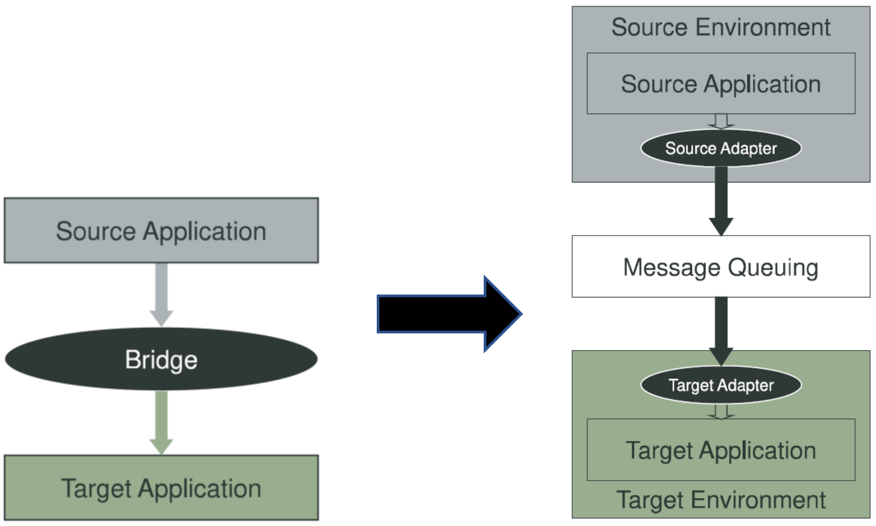
Components vs. services

Components	Services
Tight Coupling	Loose Coupling
Client/Server	P2P
Extendable	Composable
Stateless	Context independent
Fast	Some overhead
Small-medium granularity	Medium-coarse granularity

Common Object Request Broker Architecture (CORBA) vs. Web Services (WS)

Feature	CORBA	WS
Data Model	Object Model	SOAP model
Client-Server Coupling	Tight	Loose
Type System	IDL, InterfaceDefinitionLang	XML
Location Transparency	Object references	URL
Parameter passing	By reference/value	By value
Type checking	Static and runtime	Runtime
Service discovery	Naming/Trading service	UDDI
Serialization	Built into ORB	Chosen by user

Enterprise Application Integration



Bridge and Adapters to enable applications working together.

- Bridge: hooks into source app, transforms message, invokes target app
    - Problems: heterogeneity, distributed systems, QoS
- Adapters = splitted bridge.
    - Source adapter: hooks into source app, puts data in queue of target adapter
    - Target adapter: transforms messages into data for target app and invokes it
    - Problem: Each target app needs separate adapters to different source message formats → n*n complexity
        - First Improvement: neutral message format (n+n complexity)
- Further improvement → Message Broker
  on top of queueing, identifies + transforms messages, routes to target → simplifies adapter creation.

Message oriented middleware (MOM)
infrastructure that involves passing data between apps using common communication channel, carries self-contained messages.

- Messages sent and received asynchronously.
- Provides:
    - Event-driven processing (PubSub-model)
    - Reliability + serialization of messages
    - Subject-based names and attributes
    - Multiple communication protocols
- Messaging system (Integration broker) responsible for managing connection points and multiple channels between points.

Integration broker, used for message process flow, is a A2A middleware service capable of 1:n, n:1 and n:n message distribution, records & manages contracts between Pubs and Subs. It provides:

- Message transformation
- Business rules processing
- Routing service
- Naming services
- Adapter services
- Repository services
- Events & Alerts

Service oriented computing

For programmatic interactions between autonomous systems, with:

- Loose coupling of systems through self-contained SW-entities (web services)
- Virtualization
- Agile development through composition
- Services provided everywhere

Services

Characteristics:
- Standardized interface
- Self-contained (no dependencies to other services)
- Coarse-grained
- Context-independent
- Allows for Service Composition
- Little integration need

- Available

Types:
- Informational:
  relatively simple, provide access to content or expose business applications to others, f.e. weather or financial info
- Complex:
  involve assembly & invocation of pre-existing services, f.e. supply-chain-application

Properties
- Functional → operational characteristics that define overall behavior
- Non-Functional → quality attributes, f.e. performance, scalability, availability, security

State:
- Stateless → can be invoked repeatedly without having to maintain context or state
- Stateful → require that context is preserved between invocations

Granularity
- Simple → fine granularity, discrete, exhibit request/reply-mode
- Complex → Coarse-grained, interactions with other services in single or multiple sessions

Synchronicity
- Synchronous/RPC-style: express request as method call with arguments, returns response with return value
- Asynchronous/message (or document)-style: requests are entire documents rather than set of parameters

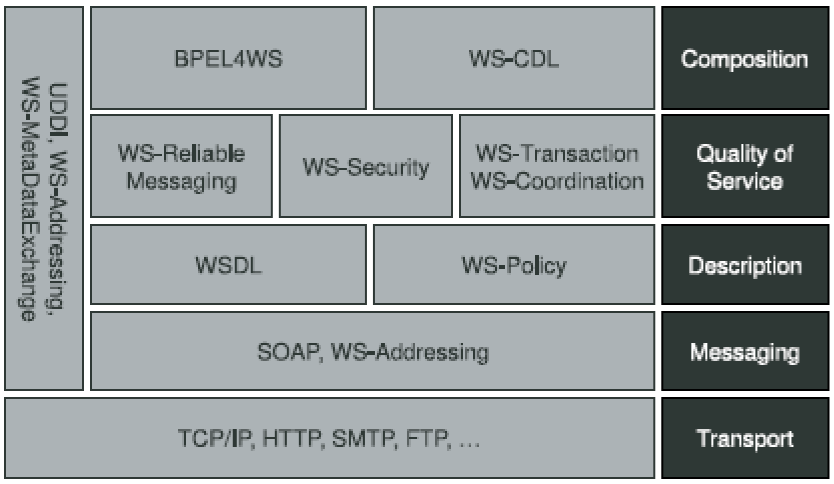
Loose Coupling
Low degree of dependency between two systems, not need to know behavior or implementation

Well-definedness
Define rules for interfacing and interacting, WSDL allows that.

Service interface
defines service functionality visible to public and provides means to access it.

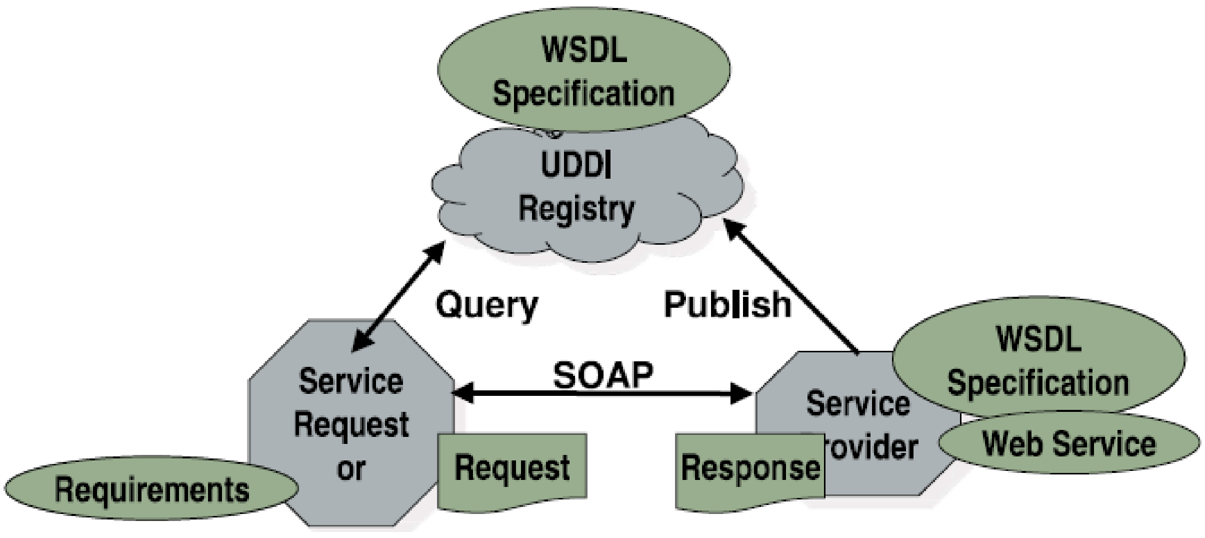
Service implementation
realizes service interface, whose implementation details are hidden.

SOA – Web Service Framework



BPEL4WS – Orchestration (how a services work)
WS-CDL – Choreography (how services work together)
WS-ReliableMessaging (In-order-, at least once-, at most once delivery)
WS-Security (Security Framework)
WS-Transaction (-AtomicTransaction for short, -BusinessActivity for long duration activities)

WS-Coordination (coordinating multi-party, multi-message WS tasks)
WSDL (web service description language, XML vocabulary to describe WS, extensible and adaptable, two parts: abstract and concrete)
WS-Policy (define constraints, conditions, service-level requirements and assurances
SOAP (Simple Object Access Protocol, XML-based messaging protocol, defines enveloping mechanism, processing model for messages, extensibility scheme, binding mechanism for transport)
WS-Addressing (interoperable way of identifying message senders and receivers)
UDDI (Universal description, discovery & integration, flexible directory/registry service for WS, which are described by WSDL and bounded/invoked using SOAP)
WS-MetaDataExchange (dynamic MDT, like Policy)



Defined for step-by-step adoption, already in widespread use (UDDI partially)

Business Process
An activity, consisting of related tasks, performed in specified sequence, according to business rule, that produces outcome. Short (minutes, hours)- or Long-Lived. Internally or between business partners.

Process Model → describes real world steps.
Workflow Model → describes technology interactions that support/interact with/implement real-world processes.

Composition
Combine and link existing atomic or composed WS,
can occur at design time (static) / at deployment time / at runtime (dynamic).
Dynamic could allow service discovery during runtime based on QoS parameters/requirements, but services may not answer, send wrong state information or doesn't meet (non-)functional requirements.

Requirements
Flexible integration, Recursive composition, Independent of QoS (SoC), Recoverability

Challenges
Discovery, interoperability and efficiency of WS; Execution and monitoring of process.

Orchestration	**Choreography**
Composition for business processes	Composition for business collaboration
Define composite services	P2P model
Reuse existing WS	Define how parties collaborate
Part-of Composition	Sequencing Composition
Private	Public

Select most suitable WS → QoS required: exhibited when invoked, characterized by operational metrics.
Application are combinations of services, which can be composed by other services (recursive composition).
Approaches: Process-based, Requirements driven, AI-based
Protocol for:

- Orchestration (centralized cooperation of services) → BPEL
- Choreography (decentralized cooperation of services) → WS-CDL
- Event-based federation (event-based approach applied to SOA) → WS-Eventing

BPEL

Definition:

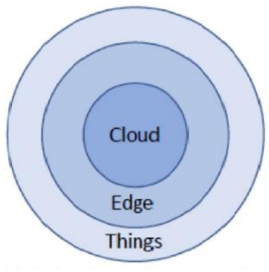
- Process-oriented composition language to define concrete and abstract processes
  - Concrete: Implementation of business process
  - Abstract: Describe externally visible interactions
- Used to specify business collaborations + to implement them as composite WS
- Mainly intended for orchestration
- Relies on WSDL
- Block-structured language
- Allows exception handling and compensation
- BPEL processes are exposed as WSDL services interact with services through their WSDLs
- Basic component: primitive or structured (order in which activities are executed) activity
- Instance-relevant data: containers
- Defines an executable process by specifying
  - Activities
  - Partners
  - Data
  - Messages
  - Fault handling

Elasticity (Resilience)

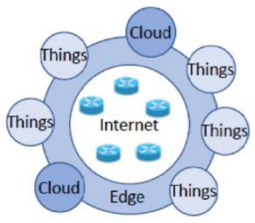
- Resource elasticity: SW/human based computing elements multiple clouds
- Quality elasticity: of non-functional parameters (performance, availability, …)
- Costs & benefit elasticity: of rewards and incentives
- Elasticity requirements specified with SYBL
- Pathway functions: to describe elasticity behaviour from general/particular view
- Space functions: to determine if service is in elasticity behaviour

Three Perspectives on IoT: Edge, Cloud, Internet

Edge-perspective → 1:1, 1:N, N:1 mapping of IoT- to Edge-device(s)

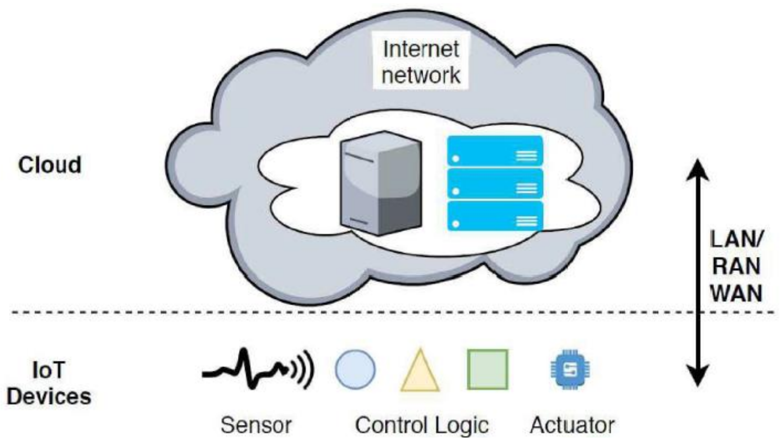


Cloud-perspective → cloud provides core services, edge computers support IoT devices, use cloud servers for big computation, data storage, security/privacy, communication between IoT-systems
Missing: cloud is at network edge, not surrounded by edge. Edge computers don't have to depend on cloud, could operate autonomous and collaborate with each other without help of cloud

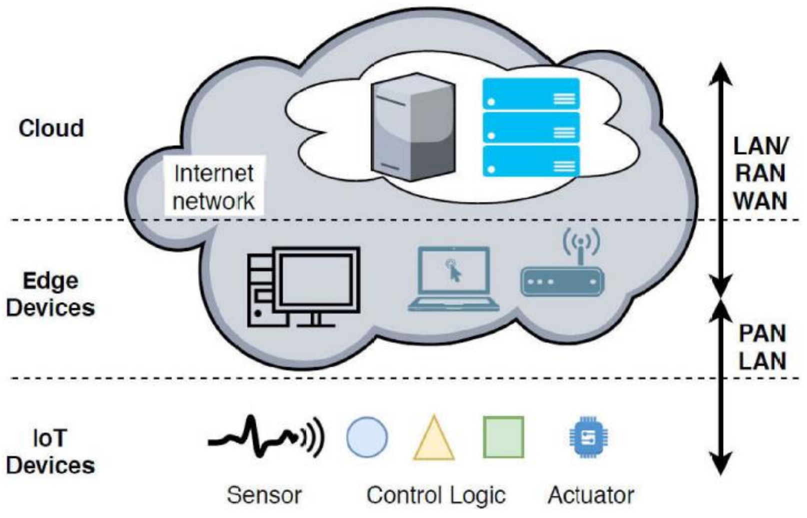


Internet-perspective → internet is center of IoT architecture, edge devices are gateways to internet for IoT devices, each LAN organizable around edge devices autonomously, so local devices do not depend on cloud.
Therefore: Devices belong to subsystems rather than to a cloud directly. Remote IoT systems can be connected directly, not via cloud. IoT system can act autonomously.

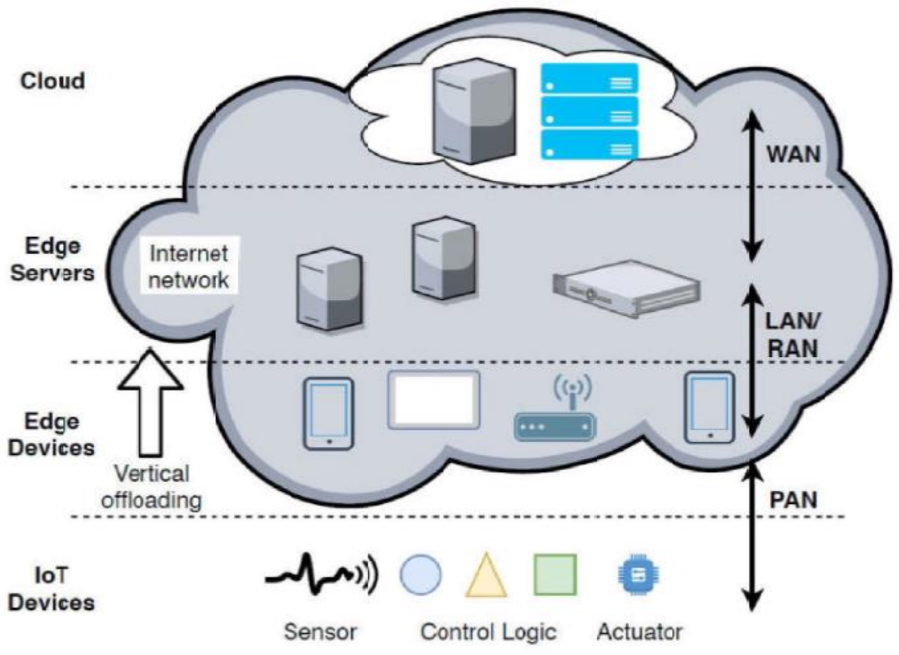
Cloud-IoT



Edge/Cloud hybrid IoT (=Horizontal Edge Architecture)



Vertical Edge Architecture



IoT Data Sources

Representation (Structure and represent data)
→ Translation (Interpret data from one modality to another)
→ Alignment (Identify relations among modalities)
→ Fusion (Fuse information from different modalities)
→ Co-learning (Transfer knowledge among modalities)

IoT Computational Units

1. Micro-Services: implement specific functionalities, can be deployed on containerized infrastructure
2. Micro-Data: encodes contextual information about
   a. Devices and resources it needs to collect/send data from/to
   b. Type of data it needs to process
   c. Data manipulate operations like storing data + results and forwarding data
3. Micro-Computing: executing specific types of computational tasks, realized by using variety of data storage and analytics programming models
4. Micro-Actuator: implementing programming interfaces for changing or controlling states in IoT environment

ICT-Views on Smart City

- Traditional: focus on optimizing physical/digital infrastructure, not society
- Societal: Active involvement of individuals to achieve collective benefits
- Holistic: Include all stakeholders into active management, Integrated management of infrastructure → new values

Cyber-Human Smart City Values

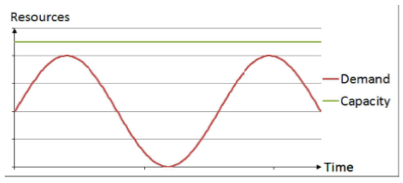
- Infrastructural: Traditional management of city-owned infrastructure, integral management of privately owned IoT devices for common benefits
- Societal: inclusion and empowerment of citizens, direct democracy, formation of ad-hoc teams for collective activities

- Business: New work models by management mechanisms, New business models by augmenting overall city infrastructure, microtransactions and dynamic crowdsourced workforce.

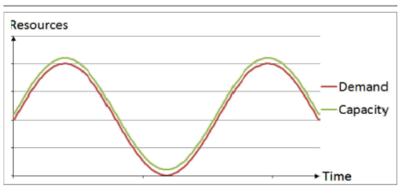
Cloud computing

Demand for services varies with time and is unknown in advance.
But: Overprovisioning → unnecessary costs, Underprovisioning → lost revenue and customers

Traditional datacenter



Virtual datacenter in the cloud



Three cloud service models:

- IaaS - (Cloud) Infrastructure as a Service → Deliver infrastructure as service: VMs, storage (Amazon EC2, S3)
  Highest customization possibilities
- PaaS - (Cloud) Platform as a Service →Deliver computing platform and solution stack as service (Google App Engine)
  PaaS apps faster developed than IaaS, Vendor lock-In, cheaper to run than IaaS
- SaaS - (Cloud) Software as a Service → Deliver piece of software as service: Google Docs
  Is ready very fast, but not fully customizable to needs, Vendor lock-in

Cloud system may provide any or all of these through external APIs.

Virtualization

Abstract view on resources, possible on platform-, memory-, storage- and network-level.

Benefits:

- Higher degree of capacity utilization
- Consolidation
- Fault Tolerance

Two types of Virtualization

- Hardware-level: Emulating virtual computer hardware → VMs
  Hypervisors = VMMs, Type 1 (HW-) & 2 (OS-level) ← distinction not always clear
  Techniques:
  - Full-Virtualization
    Completely abstracted from HW, Hypervisor translates all OS calls on the fly
  - Native (HW-Assisted)-Virtualization
    Efficient full virtualization with help of HW capabilities (primarily host-CPU → VT-x)
  - Paravirtualization
    Efficient and lightweight technique with near-native performance, Hypervisors provides API ← Guest OS calls, does not require CPU extensions (like VT-x) but kernel support and drivers
- OS-level: OS-Kernel manages coexistence of multiple isolated spaces → Containers, have small overhead, nearly native performance, more elastic than hypervisors but not secure as VMs

Four cloud deployment models

- **Public Cloud** – Open to public, owned by organization selling cloud services
  Advantages:
  - Lower costs (pay per use) + no upfront costs for hw/sw
  - No maintenance
  - High scalability
  - High reliability
- **Private Cloud** – Operated solely for one organization
  Advantages
  - Flexibility – customized environment to meet needs
  - Security – Tighter control possible
  - Compliance – with regulations
  - Self-reliance
- **Community Cloud** – Shared by several organizations
- **Hybrid Cloud** – Composition of the other models
  Advantages:
  - Control
  - Flexibility/Scalability
  - Cost-effective

Top 10 Obstacles and Opportunities for Cloud Computing

Obstacles	**Opportunities**
Availability/Business Continuity	Use multiple cloud providers
Data Lock-in	Standardize APIs, Enable Hybrid C2
Data Confidentiality & Auditability	Deploy Encryption, VLANs, Firewalls
Data Transfer Bottlenecks	FedExing Disks; Higher BW Switches
Performance Unpredictability	Better VM support; Flash Memory
Scalable Storage	Invent it
Bugs in large distributed systems	Invent debugger for distributed VMs
Scaling Quickly	Invent Auto-Scaler based on ML
Reputation Fate Sharing	Reputation guarding services like for mail
Software Licensing	Pay-for-use license

AWS Offerings

Elastics Computing Cloud (EC2) = VMs with different capabilities (different VM resources/configs, vCPU, regions and availability zones), 4 types of billing:

- On-Demand (pay per use)
  Flexible, not long-term → dynamics scaling, good for unpredictable workloads or dev/testing
- Reserved Instances
  Cheaper, but long-term. Good for known, steady workload
- Spot Instances
  Bidding of spare EC2 capacity → big discounts, good for apps flexible start/end-times that require cheap run.
- Dedicated Hosts
  Dedicated physical server for meeting compliance targets.

Three types of storage services in AWS

1. Simple storage service (S3) – Object-based, persistent
2. Elastic block store (EBS) – behave like raw, unformatted; can used as FS or DBMS, accessible for single EC2, transient by default → persisted via snapshots in S3
3. Elastic file system (EFS) – Standard file system → persistent, mountable. Accessible by multiple EC2 instances; auto-scaling.

Simple Queue Service (SQS): message queue aiming at scalability manage by amazon → can scale transparently, components can fail safely, can delay delivery of messages. Either LIFO or FIFO.

QoS = Quality of Service

Measure for (technical) quality of a (web or cloud) service, f.e.:

- Performance
- Availability
- Failure rate
- Security
- Trust
- Compliance
- Costs

Instance-level, Performance-related QoS metrics:

- Round-trip Time & Response Time
  Time from request is issued at client to moment server's response is received by client
  - RTT: First byte out – last byte in
  - RT: Last byte out – first byte in
- Latency, time a message spends in transport medium
- Processing Time, time needed executing requested operation
- Wrapping Time, time needed by client or service to (un)marshal messages
- Execution Time = Processing + Wrapping

Aggregated QoS metrics:

- Throughput → max. number of requests processed by service in given timeframe.
- Availability → probability that service is operative at any time = uptime / downtime + uptime
  - Redundancy example with 1 service interface (availability 0.99), 3 web servers (each 0.9 availability) and 5 DB instances (each 0.85 availability):
    → 0.99 * (1 – 0.1^3) * (1 – 0.15^5) = 0.9889

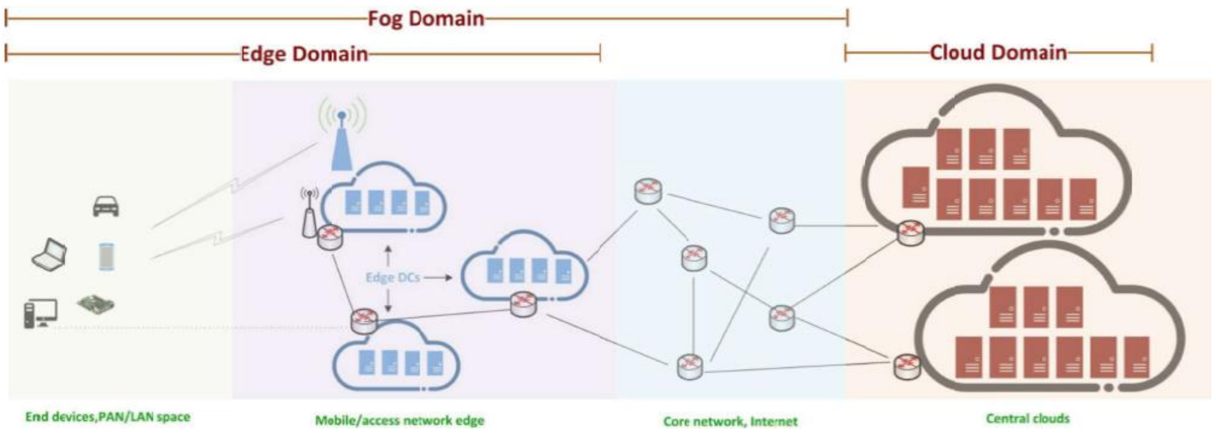
SLAs

=specific agreements between client and service on QoS terms.
Mostly relevant for B2B interactions, contain:

- Concrete SLOs
- Metrics and their definitions
- Concrete target values
- Penalties for non-achievement
- Validity period
- Responsible monitory entity

IoT-Continuum

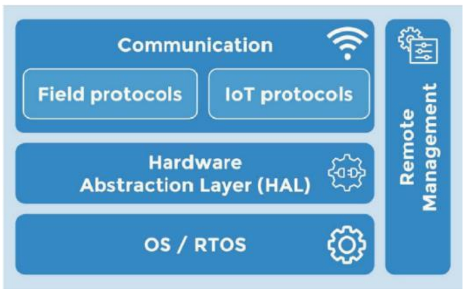
Gartner: IoT is a network of physical object that contain embedded technology to communicate and sense or interact with their internal states or the external environment.



Networking techniques:

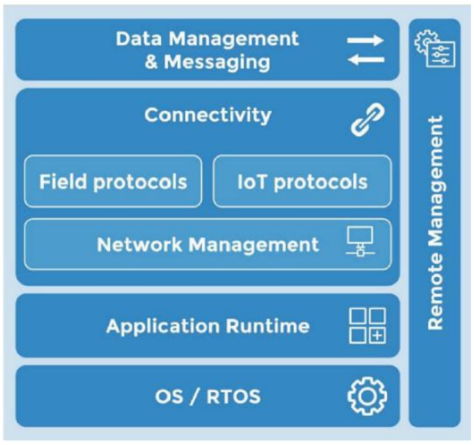
- Wireless P-/LAN for connectivity at local/personal space: Bluetooth, Wi-Fi, ZigBee
  → High(er) throughput, lower coverage, local gateway as aggregation point
- WAN for direct access to "remote" gateway: 4G, 5G
- LPWANs use event-driven or periodic transmission, long-range wireless connectivity, operate at very low power, very large number of devices → networking HW should be cheap:
  - LoRa – proprietary, low cost, unlicensed spectrum
  - SigFox – proprietary, very low bandwidth, unlicensed spectrum
  - LTE-M, NB-IoT – Used by operators, can reuse some 4G HW
- LoRa and SigFox quite similar, NB-IoT better in Scalability, QoS and Latency Performance, but not so good in coverage, deployment and cost efficiency.

Device



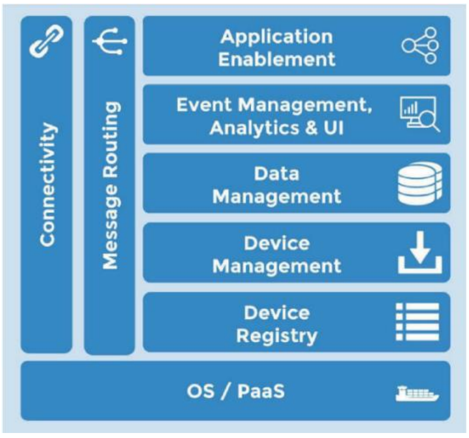
- Typically resource constrained
- Special-purpose embedded SW or RTOS for MCU-based devs
- General purpose OS might be possible
- HAL: access to HW resources, sensors
- SW for communication
- SW to remotely control device

Gateway



- More powerful, general-purpose OS
- 1+ runtime environments for different apps
- >= 2 different NW interfaces/technologies
- Storage + processing capabilities at edge
- App-level protocols for pushing device messages to cloud

IoT cloud platform



- Runs on cloud →scalability, large volumes of events and data
- Supports multiple app-level protocols, data formats, device types
- App-enablement: Expose well-defined APIs
- Message-routing to different subs
- Centralized device + gateway registration

Key design principles for IoT stacks

- Loose coupling
- Modularity
- Platform independence
- Open standards
- Well-defined APIs

Microservices

Another incarnation of SOA, popular recently, set of application principles added to SOA fundamentals → Many independent components, consumed via services.

Characteristics:

- Independently deployable small components
- Communicates through well-defined APIs
- Performs a small, business-oriented functionality
- Not externally exposed
- Developed by a small-2-pizza-team

Advantages:

- Small understandable functionality units focused on business needs
- Lightweight → quick to deploy and scale
- Good fault isolation
- Possibly independent teams
- Easy CI/CD

Drawbacks:

- Not a panacea
- Much inter-service communication and coordination
- Duplication of data
- CI/CD requires tooling support

Serverless computing aka FaaS cloud model

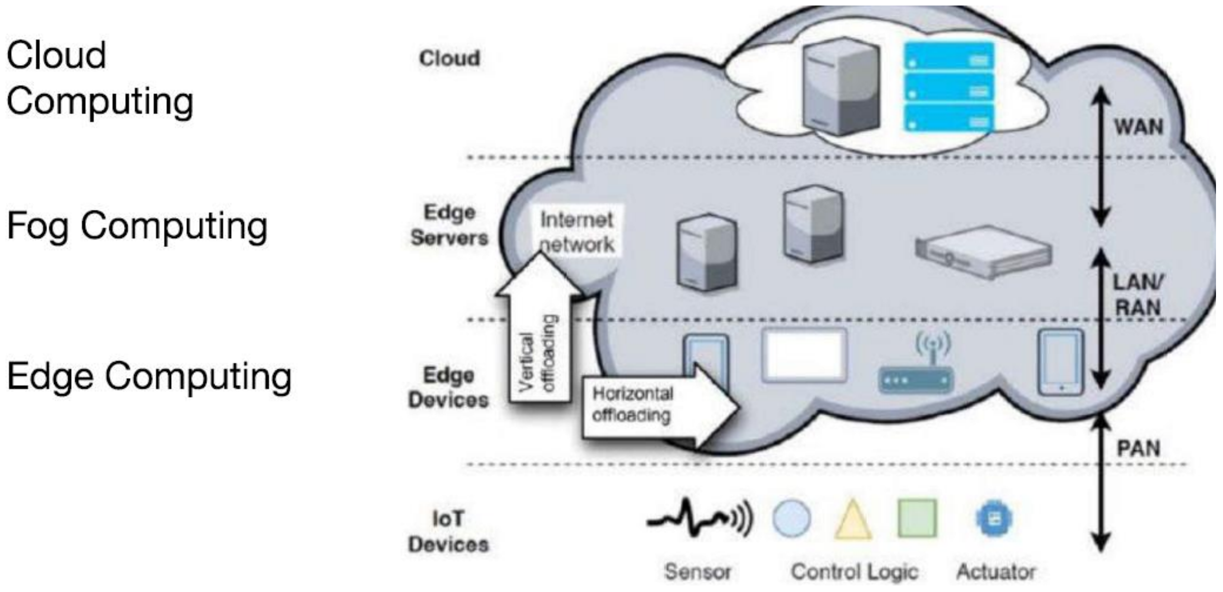
A piece of executable business logic uploaded as a function to cloud provider. May be simpler and cheaper is use case is right.

- ➔ Define mappings for inputs and outputs (cloud-based storage/DBs, messages/queues)
- ➔ Define metadata (triggers, runtime parameters, failure strategy)
- ➔ When triggers fires, function is executed
- ➔ Pay-per-use
- ➔ Scaling per invocation

Cloud, Fog, Edge

Typical cloud designs: collect and process data centralized, but through emerging number of IoT applications more data is generated, and more processing power is available at Edge → usable?

Edge & Fog Computing → bring computation close to end-devices.



Edge vs. Fog

Fog	Edge
System-level horizontal architecture that distributes resources and services anywhere along continuum from Cloud to Things	Distributed computing performed near the boundary between pertinent digital and physical entities (=edge)
Wider scope	
Deeply hierarchical, multi-layer architecture	Spans up to edge of operator’s NW
Computation anywhere among collaborating entities along continuum	Computation on-device or 1 hop away

Multi-access Edge computing (MEC)

Narrower but more clearly defined case of edge computing.

- Architecture s and interfaces to standardize → deployment of 3rd party apps to edge
- Tailored to NW-operator controlled edge infrastructure
- Consumer-oriented, Operator and 3rd party-oriented services
- Critical enabler for 5G →Enables URLLC and eMBB services.
- Difference from traditional clouds → end-user context available to edge apps

Typical workflow

Prepare app → Onboard app to MEC system → Instantiate app → App instance subs to MEC services it wants to consume

Intelligence at edge

Massive amounts of data generated at edge → valuable input for ML tasks. Traditional way: send data to cloud, train at cloud, distribute new ML model to end devices.

Challenges for traditional way:

Performance - Huge amount of data to be communicated, but low bandwidth.
Privacy - sensitive data is generated at the edge
→ Due to increased power and storage on edge (AI chips, runtime environments on small-factor devices) and new regulations (data minimization principle), train models directly on remote devices without revealing the data, only send outcomes to create a global model with multiple local models = Federated learning

Security – a cross-cutting concern

CANAI – Confidentiality, Availability, Non-repudiation, Authentication, Integrity

Security within IoT-Cloud continuum:

- Heterogenous devices which may belong to different domains and (dis)appear
- Loosely coupled SW components
- Fragile full stack solutions
- Wide attack surface
- Operation in unknown or untrusted environments
- Advantage of malicious actors because of:
    - Out-of-date FW/SW
    - Massive IoT numbers
    - Physical access

Key Principles:

- Assume hostile edge
- Test for scale
- Expect isolation, exploit autonomy
- Protect uniformly
- Limit what you can
- Support full lifecycle

Relevant exam questions

Name and briefly describe the benefits of using Web services. At least 3 technical and 2 business benefits.

What is an Enterprise Application Integration adapter, what are its disadvantages, how can you make it better?

Sketch the SOA triangle (including descriptions)

Describe technologies and interactions (WSDL, UDDI, SOAP?)

The empty fields of Web Services framework are given. Fill them up properly.

Describe the difference between functional and non-functional properties

Meaning of loosely coupled and stateless web services?

Challenges of autonomous service composition (at least 3)

Describe the difference between orchestration or choreography

Process vs. Workflow model

What is BPEL? what is used for? Describe language characteristics, what 2 types of activities does it offer. How does BPEL achieve recursive composition with WSDL?

Describe 3 elasticities discussed in the lecture

Describe these 3 intrinsic cloud characteristics Resource Pooling, Service metering, Elasticity?

Public / private / hybrid cloud: Explain in 1 sentence respectively and give 2 advantages per cloud model

Private Cloud vs Public Cloud

From a cloud centric perspective: Sketch and explain: Edge/cloud hybrid infrastructure with IoT devices

Explain: Horizontal and Vertical offloading (you can use the sketch from subquestion above)

Why would you as a provider use virtual machines instead of actual hardware machines?

Example given (volatile computation demand for video surveillance, minimize cost), which cloud products would you use and why?

Calculate availability value (no numbers, formula has to be written down) for 1 load balancer, 3 web servers, 5 db servers

Example given (DNA analysis that will take >1 year, minimize cost), would you use EC2 On-demand, Reserved, Spot or Dedicated instances?

Single choice questions Yes/No

- Is availability an aggregated, serverside QoS attribute for web services?
- Workflow models implement real-world process models
- Web Services are by design tightly coupled
- BPEL, WSDL and SOAP are encoded/transmitted with JSON
- Do BPEL-scopes define variable visibility within the process?
- Response-time is a server-side web service QoS characteristic
- BPEL is mainly intended for choreography
- Web services almost always represent result in JSON
- QoS monitoring is only possible at server side