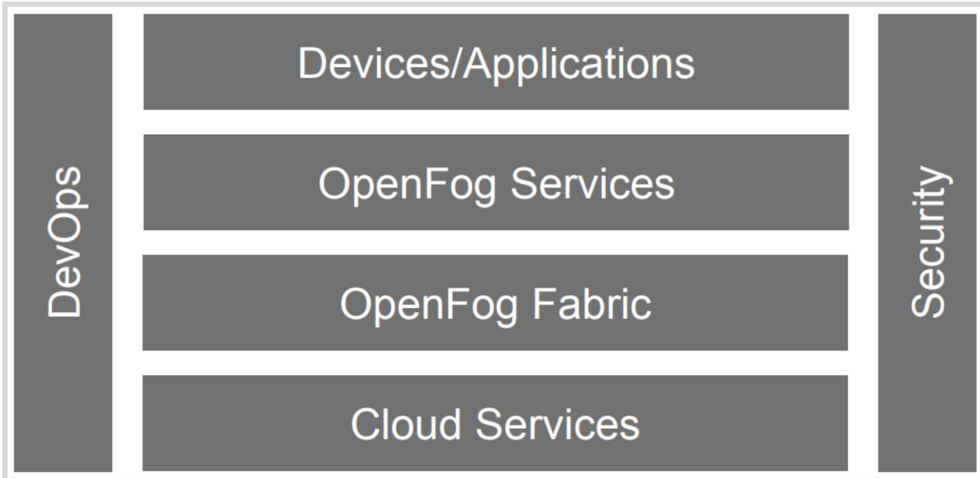
FOG COMPUTING
From Cloud to Fog

1. Cloud Computing Characteristics:
    a. On-Demand Self-Service.
    b. Broad Network Access.
    c. Resource Pooling.
    d. Rapid Elasticity.
    e. Measured Service.
2. Why is Cloud Computing not Enough?
    a. Requires continuous connectivity, which is not feasible in many real world situations.
    b. Latency might be too high for specific use cases.
    c. Bandwidth limitations.
    d. Regulations / privacy requirements.
3. Edge Computing:
    a. Data is created at the edge (and in the cloud) and needs to be synced everywhere. Applications run at the edge.
4. What is Fog Computing?
    a. Fog computing combines cloud resources with edge devices and potential intermediary nodes in the network in between.
    b. Fog Computing provides the ability to analyze data near the Edge for improved efficiency (regarding delay or bandwidth), or to operate while disconnected from a larger network (autonomy).
    c. At the same time, cloud services can be used for tasks that require more resources or elasticity.
5. Fog Computing Characteristics:
    a. Runs required computations near the end-user and data to avoid latency, network, and other migration costs (including bandwidth).
    b. Uses lower latency storage at or near the Edge.
    c. Uses low latency communication at or near the Edge rather than requiring all communications to be routed and synchronized through the backbone network.
    d. Implements elements of management at or near the Edge rather than being primarily controlled through the Cloud.
    e. Uses the Cloud for strategic tasks that require a large data context or huge amount of storage/compute power.
    f. Multi-tenancy on a massive scale is required for some use cases.
6. Fog Computing is Geo-Distributed:
    a. The physical location is significant as an application that needs to run close to its users needs to be in the “right part” of the Fog.
    b. Entire pool of sites is dynamic as the physical separation comes with unreliable connections in between sites.
    c. Sites are remote and potentially requires administration via the network.
    d. “Fog nodes” deployed at a site can have various sizes and scales, e.g., from single device to complete data center.
7. Fog Computing Benefits:
    a. Data Collection, Analytics & Privacy:
        i. Sending data over limited network connections to a centralized analytics system is counterproductive.
        ii. Pre-analyzing data near the Edge can save bandwidth.
        iii. Also enables anonymization of sensitive data before it is sent to the

Cloud.

b. Security:
    i. IoT devices are often vulnerable to attacks.
    ii. Moving security closer to these devices enables higher performance security applications.
    iii. Also adds additional layers of security to protect the core against breaches and risks.
c. Compliance Requirements:
    i. Broad range, e.g., geofencing, data sovereignty, and copyright enforcement.
    ii. Restricting data access based on geography and political boundaries.
    iii. Limiting data streams depending on copyright limitations.
    iv. Storing data in places with specific regulations.
d. Real-time
    i. AR/VR, connected cars, telemedicine, industry 4.0, or smart city applications can be very sensitive to latency or jitter.
    ii. Doing necessary computations closer to the Edge helps to reduce latencies.

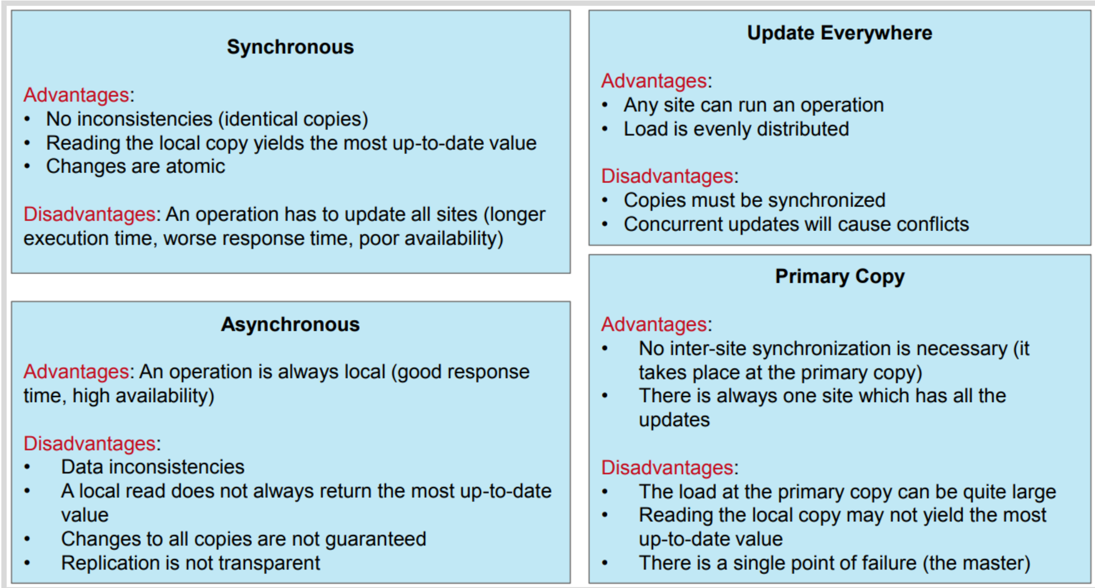
8. Inherent Obstacles for Fog Computing Adoption:
    a. No on-demand Edge infrastructures available (yet).
    b. Fog application providers currently have to build, manage, and run their own physical “Edge machines” and setup Cloud integration.
    c. Edge machines come in a variety of flavors (heterogeneity).
    d. Software stacks need to be adapted / have to run everywhere.
    e. Application architecture need to be modularized so that they can deliver some or all service features depending on the available resources.
    f. To cover the “entire Edge”, many Fog nodes are needed.
    g. Issues such as network latencies, message reordering, message loss, network partitioning, or byzantine failures.
    h. At the same time, Fog application domains such as IoT or autonomous driving have stronger quality requirements.
9. External Obstacles for Fog Computing Adoption:
    a. Traditional approaches such as onsite security staff is not possible for small and geo-distributed sites.
    b. Attaching hardware on the top of a street light pole instead of on eye level.
    c. Protecting hardware with a fire-resistant coating to counter vandalism.
    d. Attackers can easily gain a physical attack vector into software systems.
    e. Some application domains such as health care require data to be held in certain physical locations.
    f. Cloud data centers can be certified if they exist in certain regions.
    g. Liquid Fog-based applications might have trouble fulfilling certain aspects of privacy regulations such as transparency about the storage location of personal data.

10. 

11. Cloud Services: Take advantage of the Cloud for computational work that needs to operate on a larger data scope.
12. OpenFog Fabric: Allows the construction of a homogenous computational infrastructure.
13. OpenFog Services: Most likely a micro-service architecture.
14. Devices/Applications: Edge sensors, actuators, and applications.

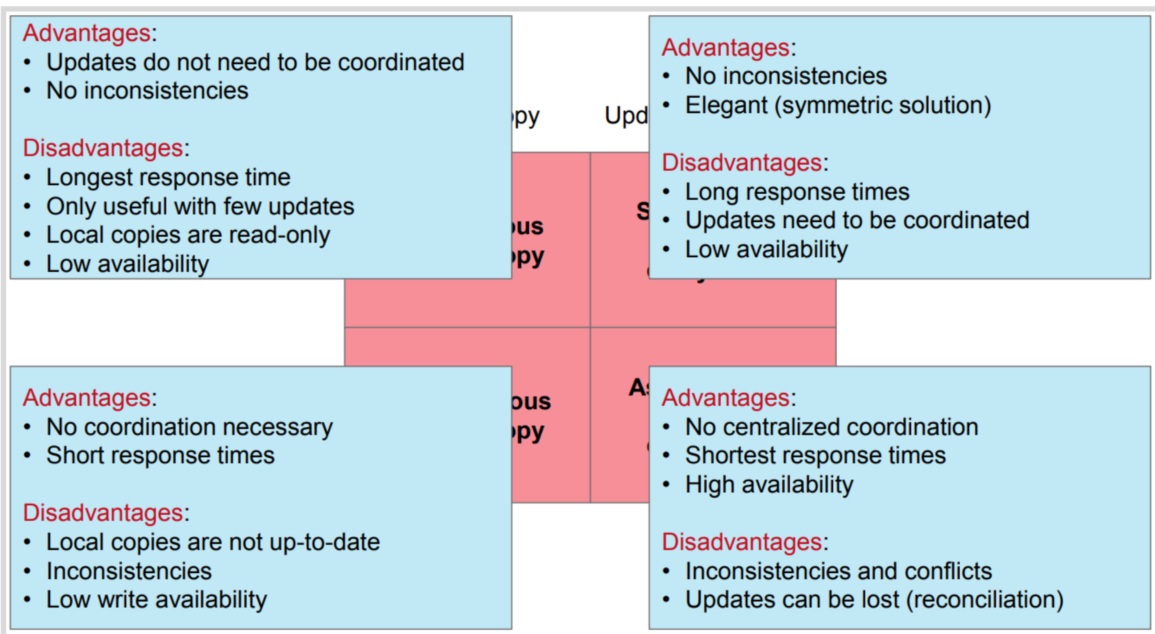
Data Management

1. Replication
   a. The main idea is to use and to maintain multiple copies of an entity (data, process, file, etc) – called replicas – on multiple servers for better availability and performance.
   b. Keeping replicas consistent in face of updates can be costly (and may even negatively impact performance).
2. Why replication?
   a. System availability / Fault-tolerance
   b. Performance / Scalability
3. Data-centric consistency models:
   a. Sequential Consistency: All replicas execute all updates in the same order.
   b. Causal Consistency: All replicas execute causally-related updates in the same order; concurrent requests are executed in arbitrary order.
   c. Eventual Consistency: In the absence of updates and failures, all replicas converge towards the same state.
   d. Monotonic Reads: A read will never return older values than previously returned to the same client.
   e. Read Your Writes: A read will never return older values than previously written by the same client.
   f. Write Follows Reads: A client that reads version X and then updates the same data item, will only update replicas that have at least version X.
   g. Monotonic Writes: Two updates of the same client will always be serialized corresponding to the chronological order of their submission.



4.

5.



6. Quorums:
   a. Updates are propagated asynchronously, but the original operation does not commit until a majority of replicas has acknowledged update success.
   b. Reads can no longer contact a single replica to avoid stale reads – quorum sizes must be set in a way to preclude concurrent updates and to assert intersection of read and write quorums.
   c. No stale reads: $R+W > N$
   d. No concurrent updates: $W > N/2$
7. Replicas Placement Strategies:
   a. Global Mapping:
      i. Control replica placement in a single centralized component.
      ii. Supports arbitrary complex and intelligent replica placement decisions.
      iii. Single point of failure.
      iv. All control flow needs to pass through a centralized component.
   b. Hashing:

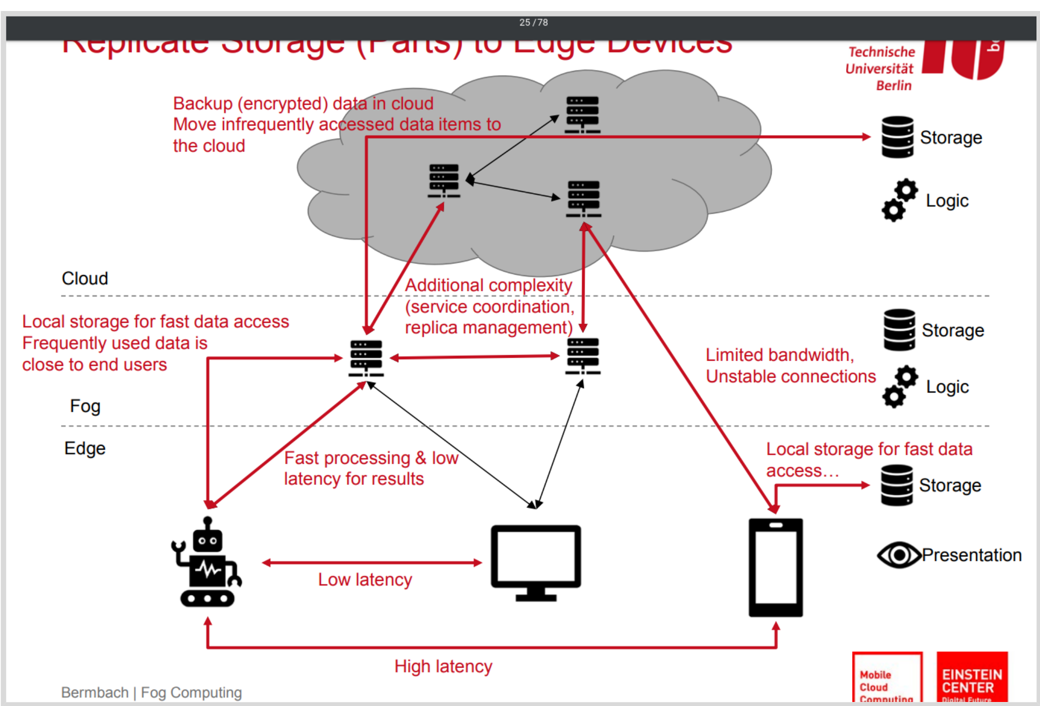
        i. In hashing-based replica placement, a hash value – usually of the data item's key – is used to deterministically identify a set of machines which will then store the corresponding data item.
        ii. Scales very well as replica placement and selection are decentralized.
        iii. Not a good fit for Fog deployments as the full determinism of the static hash function.
        iv. Does not allow to place data close to actual access locations based on current demand.
        v. Makes it hard to consider underlying network topologies in replica placement.
    c. Chaining:
        i. In chaining-based replica placement, additional replicas are created (deterministically) on adjacent machines of a primary replica selected through some other replica placement strategy.
        ii. Makes it possible to control where chaining replicas should reside
        iii. Is relatively static, so not equipped well for dynamic replica movement
    d. Scattering:
        i. Scattering creates a pseudorandomized but deterministic distribution of replicas across machines.
        ii. Can be used for good placement in geo-distributed deployments.
        iii. Poorly equipped to deal with end user mobility and resulting access pattern variances across system nodes.

8. IPFS
    a. As IPFS is peer-to-peer, no nodes are privileged and all IPFS nodes stored IPFS objects in local storage.
    b. Nodes connect to each other to transfer objects. Objects can be any data structure such as files or directories.
    c. Objects are not identified by their location (such as an URL/IP address), but by their content (in form of a hash)
9. Benefits of Using Merkle Trees:
    a. All content is uniquely identified by its cryptographic hash (including links).
    b. Sharing the address (hash) of a root directory is enough to allow another person to find all related data.
    c. All content is verified with its checksum (Tamper-proof).
    d. Tampered or corrupted data is detected by IPFS as the hash changes.
    e. Objects that hold the exact same content are equal (No Duplicates).
    f. IPFS stores these only once (No Duplicates).
10. The Global Data Plan (GDP):
    a. The GDP is a data-centric abstraction focused around the distribution, preservation, and protection of information.
    b. GDP builds upon append-only, single-writer logs:
        i. Lightweight and durable.
        ii. Multiple simultaneous readers.
        iii. No fixed location, migrated as necessary.

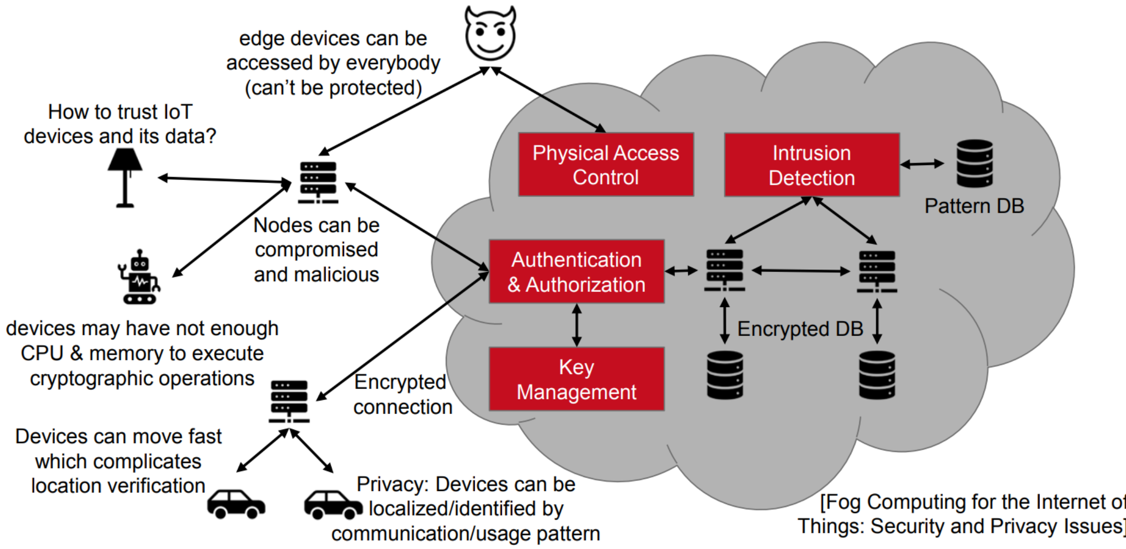
Application Engineering

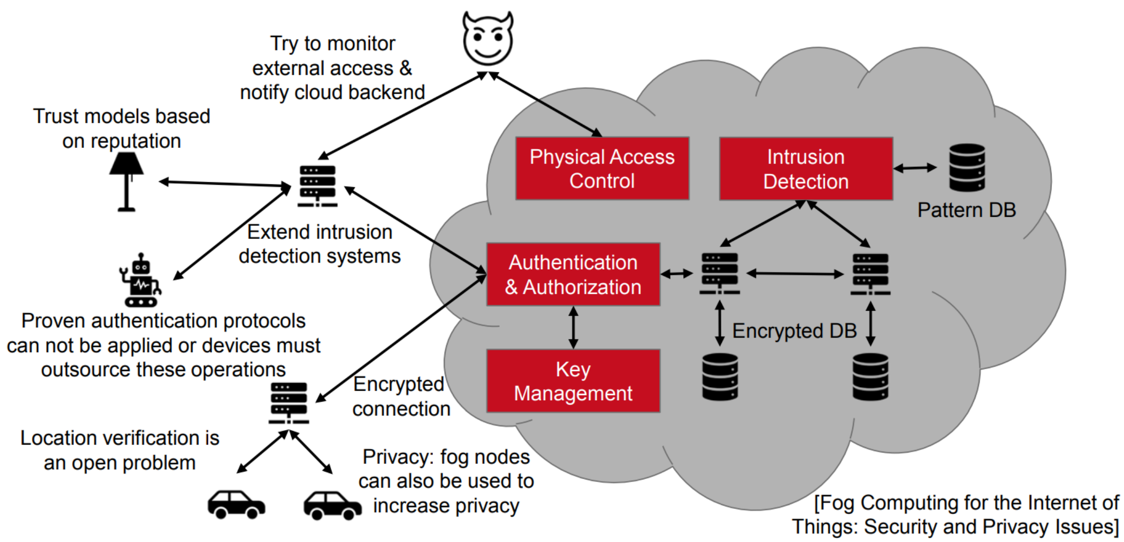
1. Characteristics of Microservices

    a. Componentization via Services
    b. Organized around Business Capabilities
    c. Products not Projects
    d. Smart endpoints and dumb pipes
    e. Decentralized Governance
    f. Decentralized Data Management
    g. Infrastructure Automation
    h. Design for failure
    i. Evolutionary Design
2. Microservice design principles combine well-known modular software design guidelines (e.g. Unix design maxims) and best-practice experience in building SOA at scale.
3. A microservice architecture is domain-specific, i.e. bounded contexts of microservices must be chosen depending on data models.
4. Microservices are aligned to (individual) product ownership responsibilities.
5. Microservice architectures adopt (shared) platforms in their development, deployment and runtime environment.
6. Fog Architecture: Microservices:
    a. Pros:
        i. Communication via interface calls.
        ii. No shared memory.
        iii. Design for failure.
        iv. Variable / unstable environment.
        v. *Loose coupling*
    b. Cons:
        i. Limited bandwidth.
        ii. Higher latency between nodes.
        iii. Unreliable connections.
        iv. Difficulty of infrastructure automation.
        v. Geographic distribution.
7. Cloud Architecture vs. Fog Architecture:
    a. Data is geo-replicated (full data set in the cloud, subset near the edge).
    b. Application logic is broken down into parts, replicated, and distributed across nodes.



8. Synchronous Communication:
    a. Sender waits for an answer (will not send another message until it receives an answer from the Receiver).
    b. Code sends a message (or calls a function) and blocks until an answer (or return value) is received.
9. Asynchronous Messaging (PTP):
    a. Sender doesn’t waits but specifies an event-trigger for the answer.
    b. Code sends a message, a defined function eventually handles the answer.
10. Asynchronous Messaging (PubSub):
    a. Sender publishes to a topic, subscribers receive the messages.
    b. Code either defines a topic and publishes to it; and or subscribes to (multiple) topics and handles incoming messages.
11. Geo-Awareness in the Cloud:
    a. Limited to large regions (e.g., counties)
    b. High latency if the closest data center is quite far.
12. Geo-Awareness in Fog (Fog Nodes):
    a. Fast connection to nearby fog nodes but limited bandwidth to cloud.
    b. Access point(s) of mobile devices must be adapted based on its location.
13. Fog Geo-Awareness:
    a. Must to be aware of ist deployment location.
    b. Needs to handle client movement (handover to other edge device).
    c. Must be prepared to move components elsewhere (=> stateless application logic).
    d. Must move data when necessary.
    e. May not rely on the availability of remote components.
14. Fault-Tolerance in Cloud Applications:
    a. Redundant Servers.
    b. Retry-on-error principle.
    c. Monitoring services and its workload, auto-scaling.
    d. Chaos-Monkey which randomly shuts down services to check if the system adapts and catches the outage.
15. Fault-Tolerance in Fog Applications:
    a. Systems and/or its components fail continuously.
    b. Connecting infrastructure fails or operates with reduced quality.
    c. Buffer messages until its receiver is available again.
    d. Expect data staleness and ordering issues.
    e. Cache data aggressively.
    f. Compress data items as much as possible on unreliable connections.
    g. Plan with incompatibility, constantly monitor software versions on devices.

16. 

17. 

18. DevOps:
    a. Overall establishing development practices that leverage frequent code commits, automated verification and builds, and early problem detection.
19. Cloud Integration Tests:
    a. Setup (virtual) testbed and check whatever is required.
    b. Mock services, data, devices.
    c. Evaluate corner-cases which usually should not exist in production.
20. Fog Integration Tests:
    a. Testbed creation is difficult because physical infrastructure and devices cannot be duplicated.
    b. (Partial) Solution: virtualize & emulate fog environment in the cloud.
21. Principles of IaC:
    a. Reproducibility.
    b. Consistency.
    c. Repeatability.
    d. Service Continuity.
    e. Self-testing systems.
    f. Self-documenting systems.
    g. Version all the things.