Cloud-specific software architecture patterns

**Packaged configuration**
- Configuration is packaged with deployment artefacts
- Simplify system, increase resilience by removing runtime dependency on configuration service
- Configuration is managed in configuration repository, CI/CD combines generic application artefact with stage/tenant-specific configuration and deploys it
- Multiple stages / tenants, build pipeline is flexible
- No runtime update of configuration, configuration changes require redeployment

**Natural multi-tenancy**
- Each logical tenant and/or stage is an isolated installation
- Simplify system by designing only for single tenant use
- Each tenant runs in a dedicated and isolated environment
- Groups of users requiring isolated setups
- Problematic when users have access to multiple tenants

**Swarm uptime**
- Combining uptime of multiple, unreliable service instances into high application uptime
- Lower cost by using cheap, volatile instances
- The cloud runtime sends tasks to running service instances, but not to failing instances. Failing instances are restarted automatically.
- App instances don’t rely on internal state, or that state can be restored easily. App instances boot quickly.
- Containerised applications must be able to handle arbitrary restarts

**Cloud-specific software architecture patterns**

**Automated maintenance**
- What: Offload maintenance tasks to runtime platform
- Motivation: Simplify application & project design
- How: The cloud runtime handles maintenance tasks transparently
- When: Periodic instance restarts to combat latent resource leaks, clean filesystem, backup, store logs, produce metrics
- Watch out: Can’t handle overly specific tasks

**Outscale caching**
- What: Achieve high application performance
- Motivation: Simplify application design by avoiding the complexity of caching
- How: Scale service instances instead of caching data
- When: Caching (and cache invalidation) are overly complex for the domain, service instances don’t rely on internal state, service invocation cascades are shallow, latency requirements can be met
- Watch out: Potentially resource-hungry

**Service discovery**
- What: Broker between service instances
- Motivation: Simplify application design by avoiding the complexity of service discovery
- How: The cloud platform injects service location information as part of “Packaged configuration” and routes requests dynamically at runtime
- When: Always
- Watch out: None

Cloud-specific software architecture patterns

Security and access control

❓ What
- Secure application easily

💪 Motivation
- Simplify application design by delegating (more) access control to the cloud runtime
- Less infrastructure is exposed, access is controlled by the platform

👉 How
- Always
- Reduced flexibility

🕒 When
- Always

❕ Watch out!
- Hard to get used to, slows down development, won’t work on non-cloud-native infrastructure

Infrastructure as code

❓ What
- Configure infrastructure through (versioned & audited) code

💪 Motivation
- Simplify change management

👉 How
- Infrastructure as code allows defining entire systems through declarative scripts

🕒 When
- In a cloud environment

❕ Watch out!
- Hard to get used to, slows down development, won’t work on non-cloud-native infrastructure