patterns & practices

proven practices for predictable results

This poster depicts common problems in designing cloud applications (below) and patterns that offer guidance (right). The information applies to Microsoft Azure as well as other cloud platforms. The icons at the top of each item represent the problem areas that the pattern relates to. Patterns that include code samples are indicated by this icon: </>

Visit http://aka.ms/Cloud-Design-Patterns-Sample to download.

Problem areas in the cloud

Availability

Availability defines the proportion of time that the system is functional and working. It will be affected by system errors, infrastructure problems, malicious attacks, and system load. It is usually measured as a percentage of uptime. Cloud applications typically provide users with a service level agreement (SLA), which means that applications must be designed and implemented in a way that maximizes availability.

http://aka.ms/Availability-Patterns

Data Management

Data management is the key element of cloud applications, and influences most of the quality attributes. Data is typically hosted in different locations and across multiple servers for reasons such as performance, scalability or availability, and this can present a range of challenges. For example, data consistency must be maintained, and data will typically need to be synchronized across different locations.

http://aka.ms/DataManagement-Patterns

Design and Implementation

Good design encompasses factors such as consistency and coherence in component design and deployment, maintainability to simplify administration and development, and reusability to allow components and subsystems to be used in other applications and in other scenarios. Decisions made during the design and implementation phase have a huge impact on the quality and the total cost of ownership of cloud hosted applications and services.

http://aka.ms/Design-and-Implementation-Patterns

Messaging

The distributed nature of cloud applications requires a messaging infrastructure that connects the components and services, ideally in a loosely coupled manner in order to maximize scalability. Asynchronous messaging is widely used, and provides many benefits, but also brings challenges such as the ordering of messages, poison message management, idempotency, and more.

http://aka.ms/Messaging-Patterns

Management and Monitoring

Cloud applications run in a remote datacenter where you do not have full control of the infrastructure or, in some cases, the operating system. This can make management and monitoring more difficult than an on-premises deployment. Applications must expose runtime information that administrators and operators can use to manage and monitor the system, as well as supporting changing business requirements without requiring the application to be stopped or redeployed.

http://aka.ms/Management-and-Monitoring-Patterns

Performance and Scalability

Performance is an indication of the responsiveness of a system, while scalability is the ability to gracefully handle increases in load, perhaps through an increase in available resources. Cloud applications, especially in multi-tenant scenarios, typically encounter variable workloads and unpredictable activity peaks and should be able to scale out within limits to meet demand, and scale in when demand decreases. Scalability concerns not just compute instances, but other items such as data storage, messaging infrastructure, and more.

http://aka.ms/Performance-and-Scalability-Patterns

Resiliency

Resiliency is the ability of a system to gracefully handle and recover from failures. The nature of cloud hosting, where applications are often multi-tenant, use shared platform services, compete for resources and bandwidth, communicate over the Internet, and run on commodity hardware means there is an increased likelihood that both transient and more permanent faults will arise. Detecting failures, and recovering quickly and efficiently, is necessary to maintain resiliency.

http://aka.ms/Resiliency-Patterns

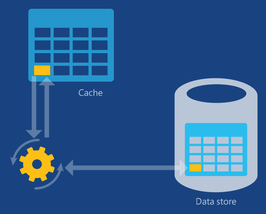
Security

Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. Cloud applications are exposed on the Internet outside trusted on-premises boundaries, are often open to the public, and may serve untrusted users. Applications must be designed and deployed in a way that protects them from malicious attacks, restricts access to only approved users, and protects sensitive data.

http://aka.ms/Security-Patterns

Cache-aside

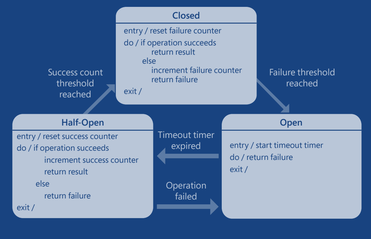
Load data on demand into a cache from a data store. This pattern can improve performance and also helps to maintain consistency between data held in the cache and the data in the underlying data store.



For more info, see http://aka.ms/Cache-Aside-Pattern

Circuit Breaker

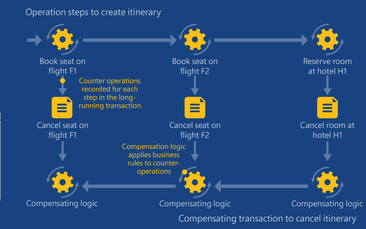
Handle faults that may take a variable amount of time to rectify when connecting to a remote service or resource. This pattern can improve the stability and resiliency of an application.



For more info, see http://aka.ms/Circuit-Breaker-Pattern

Compensating Transaction

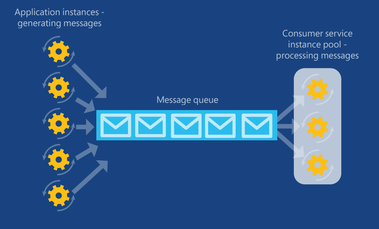
Undo the work performed by a series of steps, which together define an eventually consistent operation, if one or more of the operations fails. Operations that follow the eventual consistency model are commonly found in cloud-hosted applications that implement complex business processes and workflows.



For more info, see http://aka.ms/Compensating-Transaction-Pattern

Competing Consumers

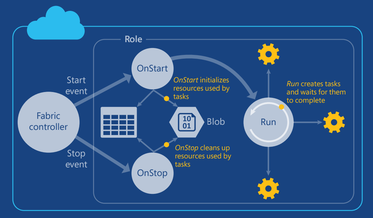
Enable multiple concurrent consumers to process messages received on the same messaging channel. This pattern enables a system to process multiple messages concurrently to optimize throughput, to improve scalability and availability, and to balance the workload.



For more info, see http://aka.ms/Competing-Consumers-Pattern

Compute Resource Consolidation

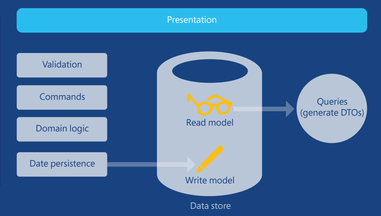
Consolidate multiple tasks or operations into a single computational unit. This pattern can increase compute resource utilization, and reduce the costs and management overhead associated with performing compute processing in cloud-hosted applications.



For more info, see http://aka.ms/Compute-Resource-Consolidation-Pattern

Command and Query Responsibility Segregation (CQRS)

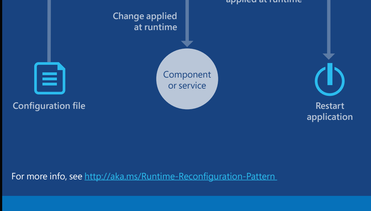
Segregate operations that read data from operations that update data by using separate interfaces. This pattern can maximize performance, scalability, and security; support evolution of the system over time through higher flexibility; and prevent update commands from causing merge conflicts at the domain level.



For more info, see http://aka.ms/CQRS-Pattern

Event Sourcing

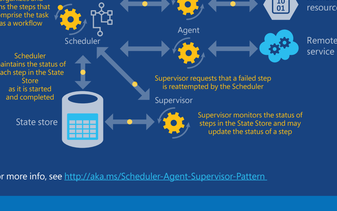
Use an append-only store to record actions taken on data, rather than the current state, and use the store to materialize the domain objects. In complex domains this can avoid synchronizing the data model and the business domain; improve performance, scalability, and responsiveness; provide consistency; and provide audit history to enable compensating actions.



For more info, see http://aka.ms/Event-Sourcing-Pattern

External Configuration Store

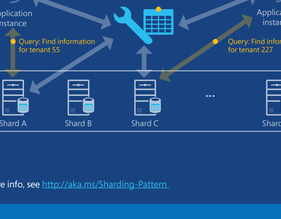
Move configuration information out of the application deployment package to a centralized location. This pattern can provide opportunities for easier management and control of configuration data, and for sharing configuration data across applications and application instances.



For more info, see http://aka.ms/External-Configuration-Store-Pattern

Federated Identity

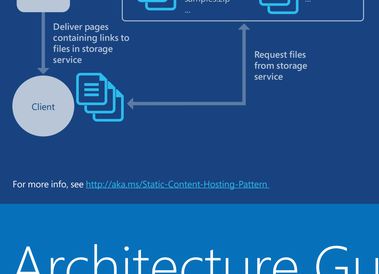
Delegate authentication to an external identity provider. This pattern can simplify development, minimize the requirement for user administration, and improve the user experience of the application.



For more info, see http://aka.ms/Federated-Identity-Pattern

Gatekeeper

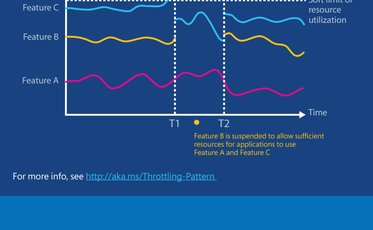
Protect applications and services by using a dedicated host instance that acts as a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them. This pattern can provide an additional layer of security, and limit the attack surface of the system.



For more info, see http://aka.ms/Gatekeeper-Pattern

Health Endpoint Monitoring

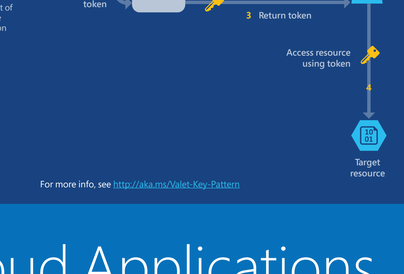
Implement functional checks within an application that external tools can access through exposed endpoints at regular intervals. This pattern can help to verify that applications and services are performing correctly.



For more info, see http://aka.ms/Health-Endpoint-Monitoring-Pattern

Index Table

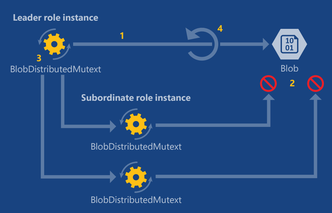
Create indexes over the fields in data stores that are frequently referenced by query criteria. This pattern can improve query performance by allowing applications to more quickly locate the data to retrieve from a data store.



For more info, see http://aka.ms/Index-Table-Pattern

Leader Election

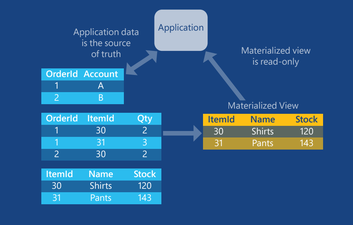
Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader that assumes responsibility for managing the other instances. This pattern can help to ensure that task instances do not conflict with each other, cause contention for shared resources, or inadvertently interfere with the work that other task instances are performing.



For more info, see http://aka.ms/Leader-Election-Pattern

Materialized View

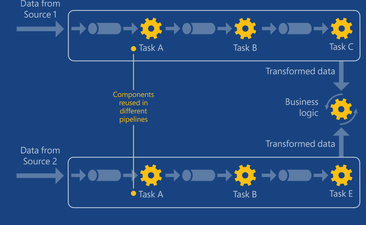
Generate pre-populated views over the data in one or more data stores when the data is formatted in a way that does not favor the required query operations. This pattern can help to support efficient querying and data extraction, and improve performance.



For more info, see http://aka.ms/Materialized-View-Pattern

Pipes and Filters

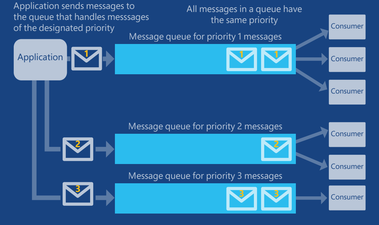
Decompose a task that performs complex processing into a series of discrete elements that can be reused. This pattern can improve performance, scalability, and reusability by allowing task elements that perform the processing to be deployed and scaled independently.



For more info, see http://aka.ms/Pipes-and-Filters-Pattern

Priority Queue

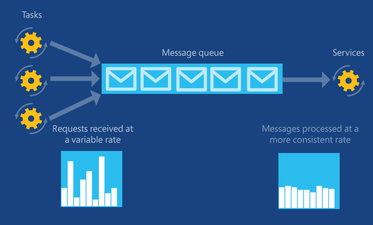
Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those of a lower priority. This pattern is useful in applications that offer different service level guarantees to individual clients.



For more info, see http://aka.ms/Priority-Queue-Pattern

Queue-Based Load Leveling

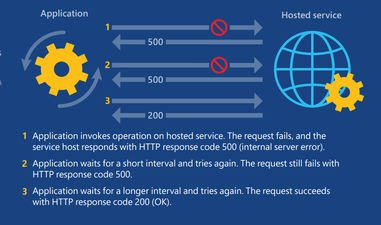
Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads that may otherwise cause the service to fail or the task to time out. This pattern can help to minimize the impact of peaks in demand on availability and responsiveness for both the task and the service.



For more info, see http://aka.ms/Queue-Based-Load-Leveling-Pattern

Retry

Enable an application to handle anticipated, temporary failures when it attempts to connect to a service or network resource by transparently retrying an operation that has previously failed in the expectation that the cause of the failure is transient. This pattern can improve the stability of the application.



1 Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).

2 Application waits for a short interval and tries again. The request still fails with HTTP response code 500.

3 Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

For more info, see http://aka.ms/Retry-Pattern

Runtime Reconfiguration

Design an application so that it can be reconfigured without requiring redeployment or restarting the application. This helps to maintain availability and minimize downtime.

For more info, see http://aka.ms/Runtime-Reconfiguration-Pattern

Scheduler Agent Supervisor

Coordinate a set of actions across a distributed set of services and other remote resources, attempt to transparently handle faults if any of these actions fail, or undo the effects of the work performed if the system cannot recover from a fault. This pattern can add resiliency to a distributed system by enabling it to recover and retry actions that fail due to transient exceptions, long-lasting faults, and process failures.

For more info, see http://aka.ms/Scheduler-Agent-Supervisor-Pattern

Sharding

Divide a data store into a set of horizontal partitions or shards. This pattern can improve scalability when storing and accessing large volumes of data.

For more info, see http://aka.ms/Sharding-Pattern

Static Content Hosting

Deploy static content to a cloud-based storage service that can deliver these directly to the client. This pattern can reduce the requirement for potentially expensive compute instances.

For more info, see http://aka.ms/Static-Content-Hosting-Pattern

Throttling

Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service. This pattern can allow the system to continue to function and meet service level agreements, even when an increase in demand places an extreme load on resources.

For more info, see http://aka.ms/Throttling-Pattern

Valet Key

Use a token or key that provides clients with restricted direct access to a specific resource or service in order to offload data transfer operations from the application code. This pattern is particularly useful in applications that use cloud-hosted storage systems or queues, and can minimize cost and maximize scalability and performance.

For more info, see http://aka.ms/Valet-Key-Pattern

Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications

Like it? Get it.

http://aka.ms/Cloud-Design-Patterns

Microsoft

pagdoc@microsoft.com