Cloud Design Patterns

Article • 12/12/2024

Architects design workloads by combining platform services, functionality, and code to fulfill functional and nonfunctional requirements in workloads. Designing workloads requires understanding of those workload requirements and then choosing topologies and approaches the solve for the challenges presented by the constraints of the workload. Cloud design patterns that address many common challenges.

Systems design is heavily steeped in design patterns. Infrastructure, code, and distributed systems are all designed around a combination of design patterns. These design patterns are useful for building reliable, secure, cost optimized, operationally sound, and performant applications in the cloud.

These design patterns aren't specific to any technology and are relevant to any distributed system, whether hosted on Azure, other cloud platforms, and some can even extend to on-premises or hybrid workloads.

Cloud design patterns help the design process

Cloud workloads are prone to the fallacies of distributed computing. Some examples of cloud design fallacies are:

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There's one administrator
- Component versioning is simple
- Observability implementation can be delayed

Design patterns don't eliminate notions such as these but can help bring awareness, compensations, and mitigations of them. Each cloud pattern has its own trade-offs. You need to pay attention more to why you're choosing a certain pattern than to how to implement it.

A well-architected workload considers how these industry-wide design patterns should be used as the core building blocks for workload design. Every Azure Well-Architected pillar is represented in these design patterns, often with the design pattern introducing tradeoffs with the goals of other pillars.

Catalog of patterns

Each pattern in this catalog describes the problem that the pattern addresses, considerations for applying the pattern, and an example based on Microsoft Azure. Some patterns include code samples or snippets that show how to implement the pattern on Azure.

Expand table

Pattern	Summary	Azure Well-Architected Framework pillars
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.	• Reliability • Security
Anti-Corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.	• Operational Excellence
Asynchronous Request-Reply	Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, but the frontend still needs a clear response.	• Performance Efficiency
Backends for Frontends	Create separate backend services to be used by specific frontend applications or interfaces.	• Reliability • Security • Performance Efficiency
Bulkhead	Isolate elements of an application into pools so that if one fails, the others continue to function.	• Reliability • Security • Performance Efficiency
Cache-Aside	Load data on demand into a cache from a data store.	• Reliability • Performance Efficiency
Choreography	Let each service decide when and how a business operation is processed, instead of depending on a central orchestrator.	• Operational Excellence • Performance Efficiency
Circuit Breaker	Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.	• Reliability • Performance Efficiency

Pattern	Summary	Azure Well-Architected Framework pillars
Claim Check	Split a large message into a claim check and a payload to avoid overwhelming a message bus.	• Reliability • Security • Cost Optimization • Performance Efficiency
Compensating Transaction	Undo the work performed by a series of steps, which together define an eventually consistent operation.	• Reliability
Competing Consumers	Enable multiple concurrent consumers to process messages received on the same messaging channel.	• Reliability • Cost Optimization • Performance Efficiency
Compute Resource Consolidation	Consolidate multiple tasks or operations into a single computational unit.	• Cost Optimization • Operational Excellence • Performance Efficiency
CQRS	Segregate operations that read data from operations that update data by using separate interfaces.	• Performance Efficiency
Deployment Stamps	Deploy multiple independent copies of application components, including data stores.	• Operational Excellence • Performance Efficiency
Edge Workload Configuration	Centralize configuration to address the challenge of configuring multiple systems and devices on the shop floor.	
Event Sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.	• Reliability • Performance Efficiency
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.	• Operational Excellence
Federated Identity	Delegate authentication to an external identity provider.	• Reliability • Security • Performance Efficiency

Pattern	Summary	Azure Well-Architected Framework pillars
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.	• Security • Performance Efficiency
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.	• Reliability • Security • Operational Excellence • Performance Efficiency
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.	• Reliability • Security • Cost Optimization • Operational Excellence • Performance Efficiency
Gateway Routing	Route requests to multiple services using a single endpoint.	• Reliability • Operational Excellence • Performance Efficiency
Geode	Deploy backend services into a set of geographical nodes, each of which can service any client request in any region.	• Reliability • Performance Efficiency
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.	• Reliability • Operational Excellence • Performance Efficiency
Index Table	Create indexes over the fields in data stores that are frequently referenced by queries.	• Reliability • Performance Efficiency
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.	• Reliability
Materialized View	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for required query operations.	• Performance Efficiency

Pattern	Summary	Azure Well-Architected Framework pillars
Messaging Bridge	Build an intermediary to enable communication between messaging systems that are otherwise incompatible because of protocol or format.	• Cost Optimization • Operational Excellence
Pipes and Filters	Break down a task that performs complex processing into a series of separate elements that can be reused.	• Reliability
Priority Queue	Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than those with a lower priority.	• Reliability • Performance Efficiency
Publisher/Subscriber	Enable an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers.	• Reliability • Security • Cost Optimization • Operational Excellence • Performance Efficiency
Quarantine	Ensure external assets meet a team-agreed quality level before being authorized to consume them in the workload.	• Security • Operational Excellence
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.	• Reliability • Cost Optimization • Performance Efficiency
Rate Limit Pattern	Limiting pattern to help you avoid or minimize throttling errors related to these throttling limits and to help you more accurately predict throughput.	• Reliability
Retry	Enable an application to handle anticipated, temporary failures when it tries to connect to a service or network resource by transparently retrying an operation that's previously failed.	• Reliability
Saga	Manage data consistency across microservices in distributed transaction scenarios. A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step.	• Reliability

Pattern	Summary	Azure Well-Architected Framework pillars
Scheduler Agent Supervisor	Coordinate a set of actions across a distributed set of services and other remote resources.	• Reliability • Performance Efficiency
Sequential Convoy	Process a set of related messages in a defined order, without blocking processing of other groups of messages.	• Reliability
Sharding	Divide a data store into a set of horizontal partitions or shards.	• Reliability • Cost Optimization
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.	• Security • Operational Excellence
Static Content Hosting	Deploy static content to a cloud-based storage service that can deliver them directly to the client.	• Cost Optimization
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.	• Reliability • Cost Optimization • Operational Excellence
Throttling	Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service.	• Reliability • Security • Cost Optimization • Performance Efficiency
Valet Key	Use a token or key that provides clients with restricted direct access to a specific resource or service.	• Security • Cost Optimization • Performance Efficiency

Next step

Review the design patterns from the perspective of the Azure Well-Architected Pillar that the pattern seeks to optimize.

- Design patterns to support the Reliability pillar
- Design patterns to support the Security pillar
- Design patterns to support the Cost Optimization pillar
- Design patterns to support the Operational Excellence pillar
- Design patterns to support the Performance Efficiency pillar

Feedback

Was this page helpful? Yes No