

Cloud and Software Design Patterns: A Layman's Guide & Bonus AI Agent Patterns

Intro

Cloud architecture for applications and software development can feel like a maze of technical jargon. But behind every scalable, resilient system lies a set of tried-and-tested design patterns — reusable solutions to common problems. Think of them as recipes for building robust cloud applications.

In this guide, we'll decode each pattern with a simple analogy and highlight how a cloud provider can help you implement it.

I've used Azure and AWS as an example for resources that I've used in the past to implement these patterns, however they can be implemented in every provider in a similar fashion.

I've also included some not-so-common uses of different functionalities of other common tools to help people think a bit more "creatively" when it comes to maximising usage of technology.

Bonus! Once you have some understanding of design patterns, I've included a more advanced use case on how AI agents work and their "special" patterns which are usually a combination of the basic ones.

Cloud Patterns

Ambassador Pattern

Analogy: Like a personal assistant making calls for you so you can focus on your work.

What it is: A helper service that makes network requests on behalf of another service.

Tools are intermediaries to handle requests, add logging, or enforce security before reaching backend services.

Azure: Azure API Management or Azure Functions

AWS: Amazon API Gateway, AWS Lambda

Anti-Corruption Layer

Analogy: Like a translator between two people who speak different languages.

What it is: A protective layer between a modern app and a legacy system. Tools are used to transform and sanitize data between systems.

Azure: Use Azure Logic Apps or Azure Functions

AWS: AWS Lambda, AWS Step Functions

(Bonus) Microsoft Fabric / Azure Synapse / AWS Redshift: Dataflows for Data Transformation / Ingestion

Asynchronous Request-Reply

Analogy: Like ordering food and getting a buzzer instead of waiting at the counter.

What it is: Decouples frontend from backend when backend processing is slow. Tools are used to queue requests and notify when done.

Azure: Use Azure Service Bus or Azure Event Grid

AWS: Amazon SQS + SNS, AWS EventBridge

Backends for Frontends

Analogy: Like different menus for kids and adults at a restaurant.

What it is: Tailors backend services for each frontend (mobile, web, etc.). Tools are used to create specialized APIs per client type.

Note: this is not popular lately since GraphQL and similar APIs are available

Azure: Use Azure Functions or App Services, API Management with custom integrations

AWS: AWS Lambda, Amazon API Gateway with custom integrations

Bulkhead

Analogy: Like watertight compartments in a submarine.

What it is: Isolates parts of an app so one failure doesn't sink the whole ship.

Azure: Use Azure Kubernetes Service (AKS) with pod-level isolation or Azure App Service deployment slots.

AWS: Amazon ECS/EKS with task isolation, AWS Lambda concurrency controls

(Bonus) Terraform, DevOps Pipelines can be used to deploy a split version of the full application and database for extra resilience

Cache-Aside

Analogy: Like checking your fridge before going to the store.

What it is: Loads data into cache only when needed. Tools are used to store frequently accessed data and load it from a cache

Azure: Use Azure Cache for Redis

Amazon ElastiCache (Redis or Memcached)

Choreography

Analogy: Like dancers following music instead of a choreographer.

What it is: Services coordinate themselves without a central controller. Tools trigger workflows across services.

Azure: Use Azure Event Grid or Service Bus

AWS: AWS EventBridge, SNS

Circuit Breaker

Analogy: Like a fuse box cutting power during a surge.

What it is: Stops calls to failing services to prevent cascading failures. Tools are used to implement retry and circuit break the logic if multiple failures occur.

Azure: Use Polly in .NET with Azure Functions or App Services

AWS: SDK retries, custom logic with AWS Lambda or Step Functions

(Bonus): Using event queues to monitor failures and a polling app to check when they reach a certain amount and turn the app off and notify centrally

Claim Check

Analogy: Like leaving your coat at a cloakroom and getting a ticket.

What it is: Splits large messages into a reference and payload.

Azure: Store payload in Azure Blob Storage and pass reference via Azure Service Bus.

AWS: Amazon S3 store + SQS

Compensating Transaction

Analogy: Like returning items after a failed shopping spree.

What it is: Reverses steps when a multi-step process fails.

Note: Try never to use this as reversal is generally difficult and design the system to not needed where possible.

Azure: Use Durable Functions to orchestrate workflows with rollback logic.

AWS: AWS Step Functions with rollback logic

(Bonus) PowerAutomate and/or Dataverse in Microsoft Dynamics or PowerApps have a compensating transaction connector that does not write in until everything is completed so a workaround using the table store there can be used.

Competing Consumers

Analogy: Like multiple cashiers serving customers from one line.

What it is: Multiple services process messages from the same queue/topic.

Azure: Use Azure Service Bus with multiple consumers for parallel processing. You can use topics to distribute to separate queues which is normally the optimal approach

AWS: Amazon SQS with multiple Lambda consumers or EC2 workers - similar with topics

Compute Resource Consolidation

Analogy: Like carpooling to save fuel.

What it is: Combines tasks into fewer compute units.

Azure: Use Azure Batch or Azure Functions to consolidate workloads.

AWS: AWS Batch, AWS Lambda

CQRS (Command Query Responsibility Segregation)

Analogy: Like having different counters for returns and purchases.

What it is: Separates read and write operations.

Azure: Use Azure Cosmos DB for reads and Azure Functions or LogicApps for writes.

AWS: Amazon DynamoDB + Lambda

(Bonus) Use Synapse / Databricks / Redshift to store data as Parquet/DeltaLake for reads directly from the data lakehouse as it naturally covers the eventual consistency / sync with no extra resources required

Deployment Stamps

Analogy: Like having identical stores in different cities.

What it is: Deploys multiple isolated copies of an app.

Azure: Use Azure Resource Manager templates to deploy multiple environments.

AWS Equivalent: AWS CloudFormation, CDK

(Bonus) Terraform should allow any cloud provider as well as CI/CD

Event Sourcing

Analogy: Like keeping a diary instead of just the latest entry.

What it is: Stores all changes as events.

Azure: Use Azure Event Hubs or Cosmos DB with change feed.

AWS: Amazon Kinesis, DynamoDB Streams

External Configuration Store

Analogy: Like storing your Wi-Fi password in a notebook instead of hardcoding it into your phone.

What it is: Keeps configuration settings outside the app code.

Azure: Use Azure App Configuration or Azure Key Vault to manage settings securely and centrally.

AWS: AWS Systems Manager Parameter Store, AWS Secrets Manager

Federated Identity

Analogy: Like using your passport to access multiple countries.

What it is: Allows users to log in using external identity providers.

Azure: Use Azure Active Directory B2C or B2B / EntraID to integrate Google, Facebook, or Microsoft accounts for authentication.

AWS: Amazon Cognito

Gatekeeper

Analogy: Like a bouncer checking IDs before letting people into a club.

What it is: A service that validates requests before they reach your app. Use tools to inspect and filter traffic.

Azure: Use Azure API Management or Azure Front Door with WAF

AWS: Amazon API Gateway + AWS WAF

Geodes

Analogy: Like having local branches of a bank in every city.

What it is: Deploys app instances close to users for low latency.

Azure: Use Azure Traffic Manager and Azure Front Door to route users to the nearest regional deployment.

AWS: Amazon Route 53 + Global Accelerator

(Bonus) use Edge compute with cache

Health Endpoint Monitoring

Analogy: Like a doctor doing regular check-ups.

What it is: Checks if services are alive and healthy.

Azure: Use Azure Monitor and Application Insights to track health endpoints and alert on failures.

AWS: Amazon CloudWatch + Route 53 health checks

(Bonus) External ping from other tools to check cache availability / edge availability and using Always Online setups will maximise uptime

Hexagonal Architecture Pattern

Analogy: Like a castle with gates on all sides — the core stays protected, and you can swap out the drawbridge, moat, or guards without changing the castle itself.

What it is: A design that separates the core logic of your app from external systems like databases, APIs, or user interfaces.

Note: Not a base pattern, but widely used today

Azure: Build your domain logic in Azure Functions or .NET services, then connect external systems via adapters using Azure API Management, Event Grid, or Cosmos DB.

AWS: Use AWS Lambda or ECS for core logic, with adapters for S3, DynamoDB, or API Gateway to handle external interactions.

Index Table

Analogy: Like a library index card to locate books.

What it is: A lookup table to quickly find data. Tools use indexed fields for fast retrieval.

Azure: Use Azure Table Storage or Cosmos DB for Vectors

AWS: DynamoDB with Global Secondary Indexes

(Bonus) Pinecone / PGVector for AI applications naturally index like this

Leader Election

Analogy: Like electing a team captain.

What it is: Chooses one service instance to act as the leader.

Note: this is much better implemented using event messaging where possible

Azure: Use Azure Kubernetes Service with leader election logic in pods or use Azure Service Fabric.

AWS: Amazon ECS/EKS with leader election logic, DynamoDB for coordination

Materialized View

Analogy: Like printing a report instead of recalculating it every time.

What it is: Precomputes and stores query results.

Azure: Use Azure Synapse Analytics or Cosmos DB with pre-aggregated views.

AWS: Amazon Redshift, DynamoDB with precomputed views

(Bonus) Databricks / Snowflake can be used for this

Pipes and Filters

Analogy: Like an assembly line in a factory.

What it is: Breaks processing into steps that can be chained.

Note: another older approach that's now ideally replaced - this time by microservices where one service does one step based on event messaging or topics for filtering

Azure: Use Azure Data Factory or Durable Functions to build step-by-step workflows

AWS: AWS Glue, Step Functions

Priority Queue

Analogy: Like giving emergency patients priority in a hospital.

What it is: Processes high-priority tasks first.

Azure: Use Azure Service Bus with message properties to prioritize processing.

AWS: Amazon SQS with message attributes

Retry

Analogy: Like redialling a number if the call doesn't go through.

What it is: Automatically retries failed operations.

Azure: Use Azure SDKs with built-in retry policies or Polly in .NET

AWS: AWS SDK retries, Lambda retry settings

(Bonus) Logic Apps / Power Automate / Step Functions and similar connectors for low-code usually have an in-built retry policy that can be configured for fixed, linear or exponential retry

Scheduler Agent Supervisor

What it is: Manages scheduled tasks and monitors their execution.

Analogy: Like a manager assigning tasks and checking if they're done.

Azure: Use Azure Logic Apps or Azure Durable Functions with timers and monitoring.

AWS: Amazon EventBridge Scheduler, Step Functions

(Bonus) Create an orchestrator in any platform - to manage and schedule tasks to run on a schedule to guarantee ordering when no events are available

Sharding

Analogy: Like dividing a big book into volumes.

What it is: Splits data across multiple databases.

Azure: Use Azure SQL Database with elastic pools or Cosmos DB partitioning

AWS: Amazon Aurora with partitioning, DynamoDB with partition keys

(Bonus) Synapse / Databricks / Snowflake / Redshift - all support partitioning and when storing as parquet/delta lake format it usually naturally partitions

Sidecar

Analogy: Like a sidekick riding next to a superhero.

What it is: A helper service deployed alongside the main app.

Azure: Azure Kubernetes Service with sidecar containers for logging, monitoring, or proxying.

AWS: Amazon EKS with sidecars (e.g., Envoy, Fluentd)

Static Content Hosting

Analogy: Like displaying posters instead of live performances.

What it is: Serves files that don't change often.

Azure: Use Azure Blob Storage with static website hosting or Azure CDN

AWS Equivalent: Amazon S3 static websites, CloudFront

Strangler Fig

Analogy: Like a new tree growing around and eventually replacing the old one

What it is: Gradually replaces legacy systems with new ones

Note: there are multiple ways of replacing legacy systems and there is no ideal, however creating an API to route requests is usually

Azure: Use Azure API Management to route traffic between old and new services.

AWS: Amazon API Gateway with staged deployments

Throttling

Analogy: Like restricting water flow to avoid flooding

What it is: Limits resource usage to prevent overload

Azure: Use Azure API Management or Azure Functions with rate limits and quotas.

AWS: API Gateway rate limits, Lambda concurrency limits

Valet Key

Analogy: Like giving a valet access to your car but not your house.

What it is: Gives limited access to a resource.

Azure: Use Shared Access Signatures (SAS) in Azure Blob Storage to grant scoped access.

AWS: Pre-signed URLs for S3

Watchdog

Analogy: Like a lifeguard watching swimmers and jumping in when someone's in trouble.

What it is: Monitors services and restarts them if needed.

Azure Example: Use Azure Monitor with alerts and Azure Automation or AKS liveness probes.

AWS: CloudWatch + Lambda, ECS/EKS health checks

AI Agent Patterns

Now that the general patterns are understood, the below is a similar format of more advanced patterns that combine general ones to achieve AI Agents

Orchestration Pattern

Analogy: Like a conductor guiding different musicians in a symphony.

What it is: Coordinates multiple AI components to complete a task.

Azure: Use Azure AI Orchestrator or Azure Logic Apps to manage LLMs, search, and tools.

AWS: Use AWS Step Functions or Amazon Bedrock Agents to orchestrate models and services.

Tool Use Pattern

Analogy: Like asking a calculator to solve math while you focus on writing.

What it is: AI agents invoke external tools to complete tasks.

Azure: Azure AI Orchestrator with tool calling (e.g., search, code execution, plugins).

AWS: Amazon Bedrock Agents with tool integrations or Lambda functions.

Retrieval-Augmented Generation (RAG) Pattern

Analogy: Like checking a textbook before answering a question.

What it is: Combines LLMs with external knowledge sources for accurate answers.

Note: most AI Agents today are built to use this as a base

Azure: Azure AI Search + OpenAI + Vector DB (e.g., Azure Cosmos DB or Azure PostgreSQL with pgvector).

AWS: Amazon Kendra + Bedrock + OpenSearch or DynamoDB with vector search.


Agent Memory Pattern

Analogy: Like remembering someone's coffee order after a few visits.

What it is: Stores and recalls past interactions to personalize responses.

Note: a lot of agents today also store chats in memory to allow you to continue and refer back

Azure: Azure Cosmos DB or Azure Table Storage for memory; integrated with Azure OpenAI.

AWS: Amazon DynamoDB or S3 for memory; integrated with Bedrock or SageMaker. 
Learn more

Planning Pattern

Analogy: Like planning a trip with flights, hotels, and activities.

What it is: Breaks complex tasks into smaller steps and executes them sequentially.

Azure: Azure AI Orchestrator with planner modules or LangChain agents.

AWS: Amazon Bedrock Agents with planning logic or custom workflows via Step Functions.

Self-Healing Pattern

Analogy: Like a robot that retries or switches tools when it hits a wall.

What it is: Detects and fixes errors during agent execution.

Azure: Azure AI Orchestrator with retry logic, fallback chains, or monitoring via Azure Monitor.

AWS: Bedrock Agents with fallback tools or Step Functions with error handling.

Multi-Agent Collaboration Pattern

Analogy: Like a team of specialists — one for writing, one for coding, one for research.

What it is: Multiple agents work together to solve a problem.

Azure: Azure AI Orchestrator coordinating multiple OpenAI agents with shared memory.

AWS: Bedrock Agents or SageMaker endpoints communicating via shared state or queues.

Guardrails Pattern

Analogy: Like bumpers in a bowling alley to keep the ball on track.

What it is: Enforces safety, compliance, and boundaries in agent behaviour.

Azure: Azure AI Content Safety, Prompt Flow with validation, Azure Policy for governance.

AWS: Amazon Guardrails (preview), Bedrock safety filters, custom moderation via Comprehend.

Human-in-the-Loop Pattern

Analogy: Like a pilot taking control when autopilot fails.

What it is: Involves humans to validate or guide AI decisions.

Azure: Azure Machine Learning with labeling tools, Azure OpenAI with feedback loops.

AWS: Amazon SageMaker Ground Truth, Bedrock Agents with escalation logic.

Agent Persona Pattern

Analogy: Like asking the same question to a teacher vs. a comedian — different style responses.

What it is: Customizes agent behaviour and tone based on role or audience.

Azure: Azure OpenAI system prompts, Prompt Flow with persona templates.

AWS: Bedrock Agents with prompt engineering or role-specific configurations.

Conclusion

Design patterns aren't just fancy diagrams or buzzwords — they're the building blocks of reliable, scalable software. Whether you're designing a cloud-native app, an AI agent, or a multi-service platform, these patterns help you avoid reinventing the wheel and steer clear of common pitfalls.

Think of them like recipes in a cookbook:

You don't need to invent a new way to bake bread — just follow the proven steps.

And if you're switching kitchens (Azure, AWS, etc.), the ingredients might change, but the method stays the same.

By understanding these patterns in plain terms — like bouncers, librarians, or symphony conductors — you can better communicate with teams, make smarter architectural choices, and build systems that don't just work... they thrive.

So next time you're staring at a whiteboard or pitching a solution, reach for these patterns.

They're your cheat codes to building software that's not just functional, but future-proof.

Sources:

Azure: <https://learn.microsoft.com/en-us/azure/architecture/patterns>

AWS - <https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/introduction.html>