CLOUD ARCHITECTURE PATTERNS

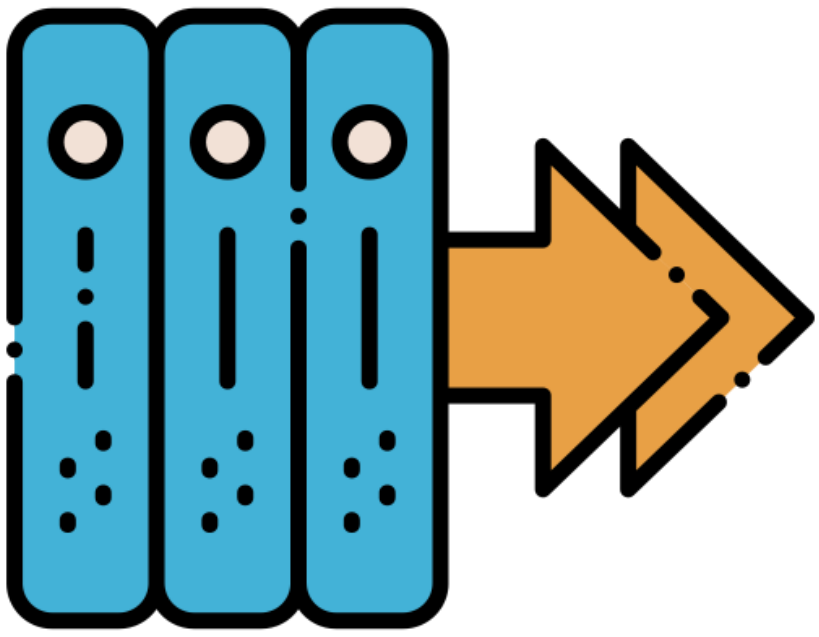
A short essay about Software Architecture in the Cloud

This post contains a short summary of a few cloud architecture patterns.

Cloud architecture pattern is a tested architectural approach to solve a set of problems (or only one) in cloud-based applications. Cloud patterns applied to cloud-native applications. In plain English, a cloud-native application is an application that is meant to run in a cloud (AWS, Azure, private).

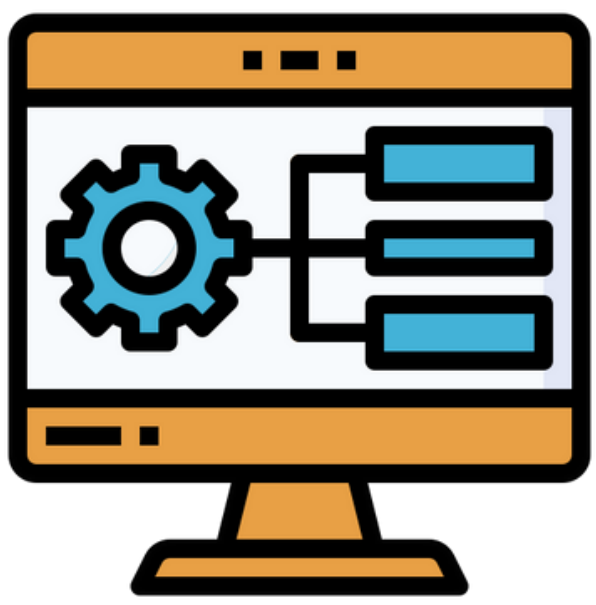
HORIZONTALLY SCALING COMPUTE PATTERN

This simple pattern implies that an application ready to handle auto-of-the-box increases or decreases in compute capacity. The Horizontal Scaling Compute Pattern architecturally aligns applications with the most cloud-native approach for resource allocation. There are many potential benefits for applications, including high scalability, high availability, and cost optimisation, all while maintaining a robust user experience. User state management should be handled without sticky sessions in the web tier. Keeping nodes stateless makes them interchangeable so that we can add nodes at any time without getting the workloads out of balance and can lose nodes without losing customer state.



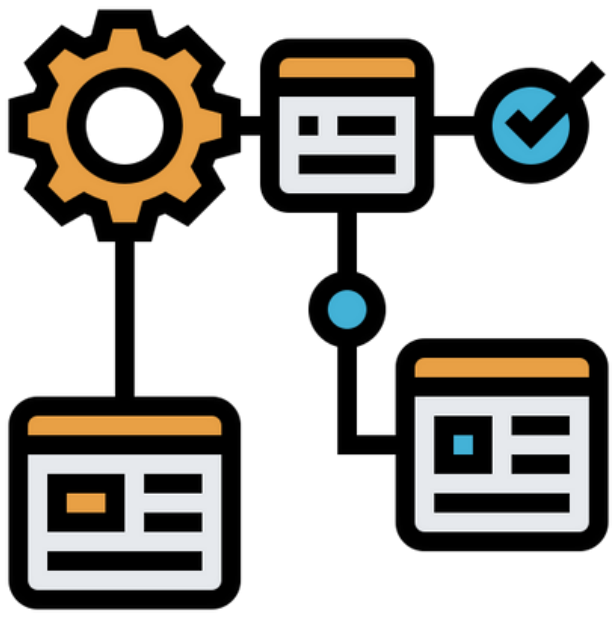
QUEUE-CENTRIC WORKFLOW PATTERN.

Please use queues/exchanges/streams to decouple components and increase elasticity. You can have multiple queuer-s and de-queuer-s to work with queues that don't know anything about each other and can horizontally scale. This pattern is for decoupling tiers of your application, especially between the web (user interface) tier and a service tier that does business processing. It is not useful for routine, read-only page requests. Communication is in one direction, from the web tier to the service tier, and is handled by adding messages onto a queue. Reliable cloud queue services simplify implementation. A decoupled web tier can be more responsive and reliable, providing a better user experience. Concern-independent scaling also allows each tier to be provisioned with the ideal level of resources for that tier.



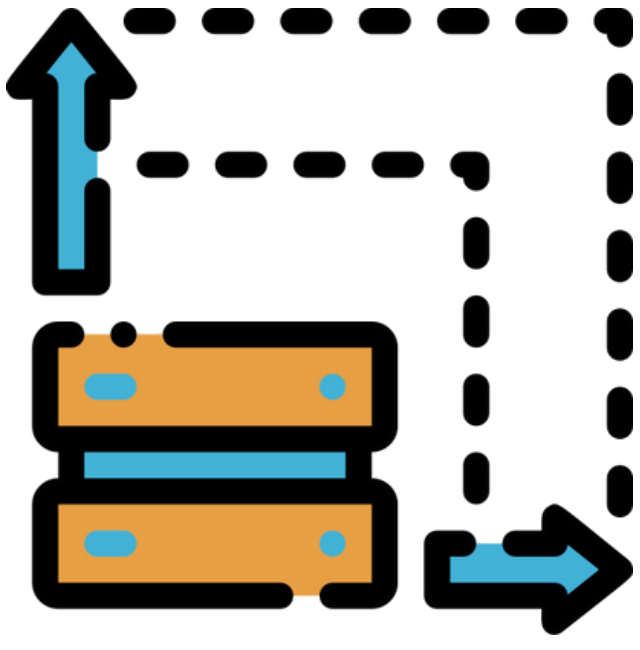
CLOUD ARCHITECTURE PATTERN

Is a tested architectural approach to solve a set of problems (or only one) in cloud-based applications. Cloud patterns applied to cloud-native applications. In plain English, a cloud-native application is an application that is meant to run in a cloud (AWS, Azure, private).



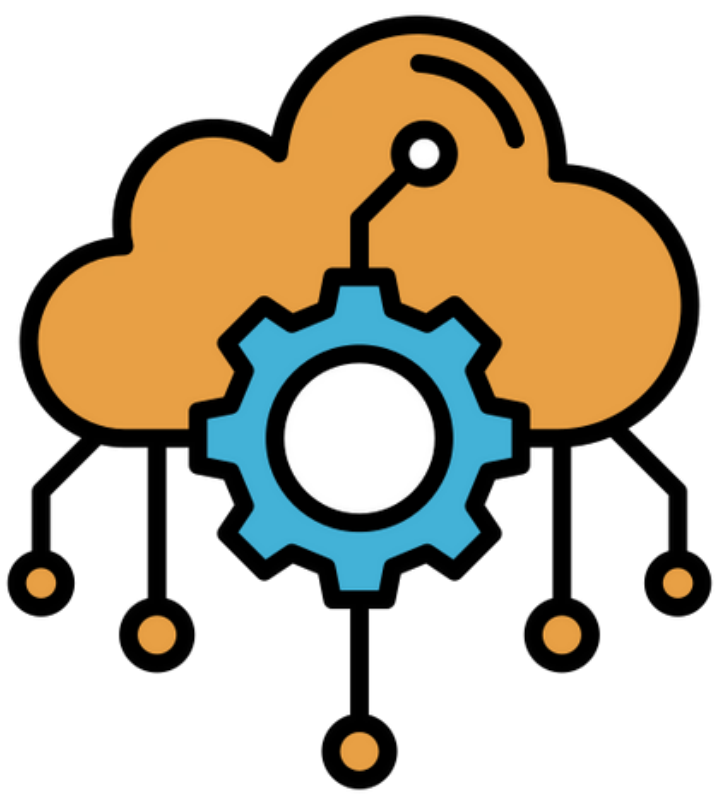
AUTO-SCALING PATTERN

The Auto-Scaling Pattern is an essential operations pattern for automating cloud administration. By automating routine scaling activities, cost optimization becomes more efficient with less effort. Cloud-native applications gracefully handle the dynamic increases or decreases in resource levels. The cloud makes it easy to plug into cloud monitoring and scaling services, with self-hosted options also available.



EVENTUAL CONSISTENCY

The CAP Theorem provides the theoretical basis that explains why we cannot guarantee both consistency and availability in a distributed database. A useful compromise is to allow for eventual consistency in favor of better scalability. Determining if your application data is a suitable candidate for eventual consistency is a business decision. The choice is between displaying stale data and scaling more efficiently.

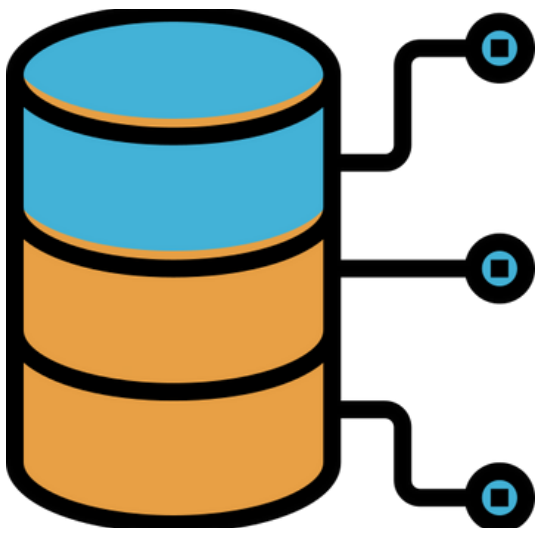


MAP REDUCE PATTERN

The MapReduce Pattern provides simple tools to efficiently process arbitrary amounts of data. There are abundant examples of common use that are not economically viable using traditional means. The Hadoop ecosystem provides higher-level libraries that simplify creation and execution of sophisticated maps and reduce functions. Hadoop also makes it easy to integrate MapReduce output with other tools, such as Excel and BI tools.

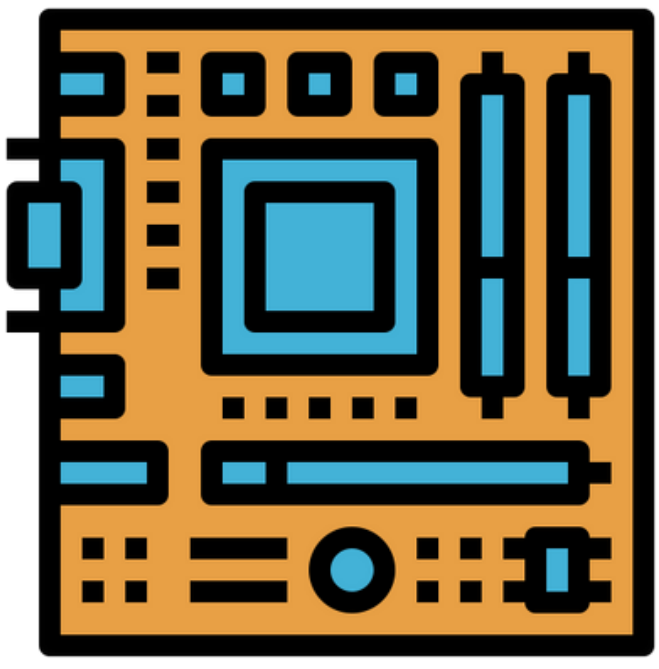
DATABASE SHARDING

When using the Database Sharding Pattern, workloads can be distributed over many database nodes rather than concentrated in one. This helps overcome size, query performance, and transaction throughput limits of the traditional single-node database. The economics of sharding a database become favorable with managed sharding support, such as found in some cloud database services. The data model must be able to support sharding, a possible barrier for some applications not designed with this in mind. Cross-shard operations can be more complex.



MULTI-TENANCY AND COMMODITY HARDWARE

Cloud platform vendors make choices around cost-efficiency that directly impact the architecture of applications. Architecting to deal with failure is part of what distinguishes a cloud-native application from a traditional application. Rather than attempting to shield the application from all failures, dealing with failure is a shared responsibility between the cloud platform and the application.



BUSY SIGNAL PATTERN

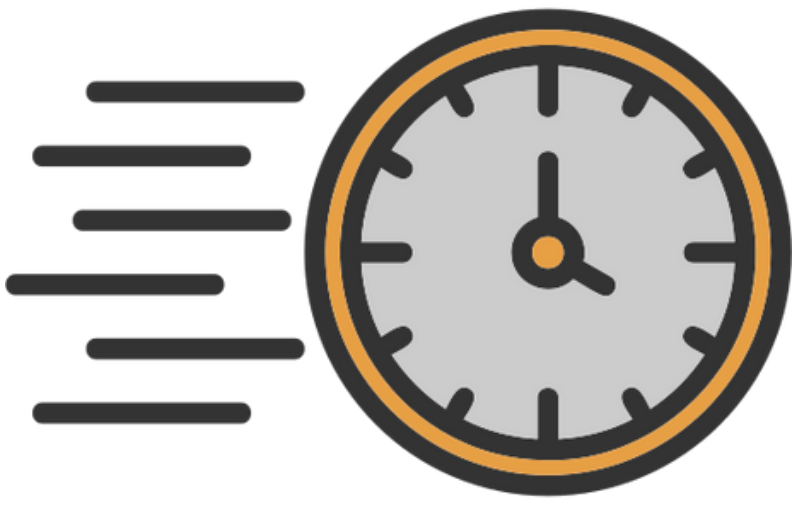
Handling transient failures is essential for building reliable cloud-native applications. Using the Busy Signal Pattern, your application can detect and handle transient failures appropriately. Further, your approach can be tuned for batch or interactive user scenarios. It may be difficult to test your application's response to transient failure conditions if running on non-cloud hardware or with an unrealistically light load.

NODE FAILURE PATTERN

Basically, node failure will happen, so be ready ;) Failure in the cloud is commonplace, though downtime is rare for cloud-native applications. Using the Node Failure Pattern helps your application prepare for, gracefully handle, and recover from occasional interruptions and failures of compute nodes on which it is running. Many scenarios are handled with the same implementation. Cloud applications that do not account for node failure scenarios will be unreliable: user experience will suffer and data can be lost.

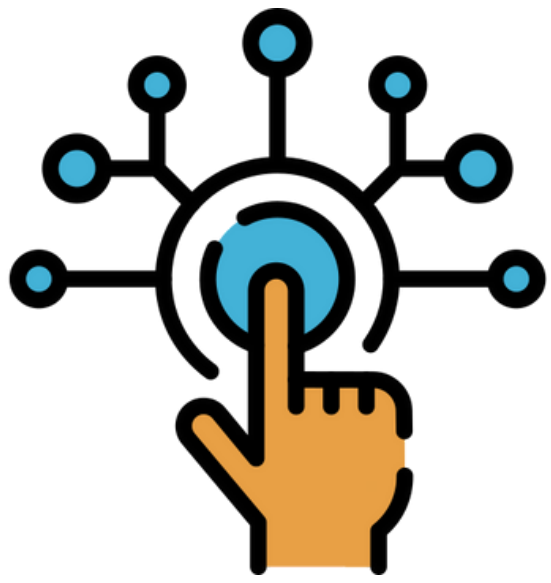
NETWORK LATENCY PATTERN

A comprehensive strategy for dealing with network latency will use multiple strategies. One set of strategies focuses on reducing the perceived network latency. Another set of strategies focuses on actually reducing network latency by shortening the distance between users and the instances of our application.



COLOCATE PATTERN

The simplest way to get started in the cloud is to colocate nodes, usually all in a single data center. This is appropriate for many applications, and should be the usual configuration. Only deviate for good reason, and avoid the mistake of accidental deployment across more than one data center, including for storage of operational data.

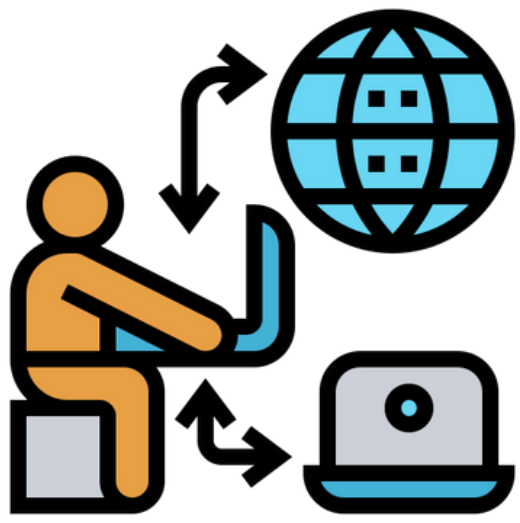


VALET KEY PATTERN

Use of this pattern should be considered anywhere it can be safely applied. The ability to manage temporary access for reading and writing makes this a broadly usable pattern. The most common troublesome use case will be an upload directly from a web browser, but reading is well supported, and writing from more flexible clients such as mobile apps is also well supported. When used with storage containers that support this pattern, applications can avoid having a web page or web service act as a security proxy to read or write data stored in a secure container. This reduces load on the web tier because it is not acting as a middleman for data transfer. Also, the code for implementing all the variants of data passing through is replaced by the far simpler generation and issuing of temporary access URLs, while the upload code is offloaded to the client. The client code should utilize existing helper libraries where available to minimize complexity. The end result is that applications scale better and the user experience is improved.

CDN PATTERN

Adding CDN support to a cloud application is a great example of a low-friction adoption of a cloud service. Enabling a CDN can be accomplished either programmatically or through a one-time manual configuration via the cloud vendor's web-hosted management tool. This is substantially easier to get started with than traditional CDNs due to the degree of convenience and integration. Once enabled, this is a great technique for reducing the load on web servers, distributing load across many servers (there are many more CDN locations than data centers), while decreasing network latency. All this helps to both improve scalability and improve user experience.



MULTISITE DEPLOYMENT PATTERN

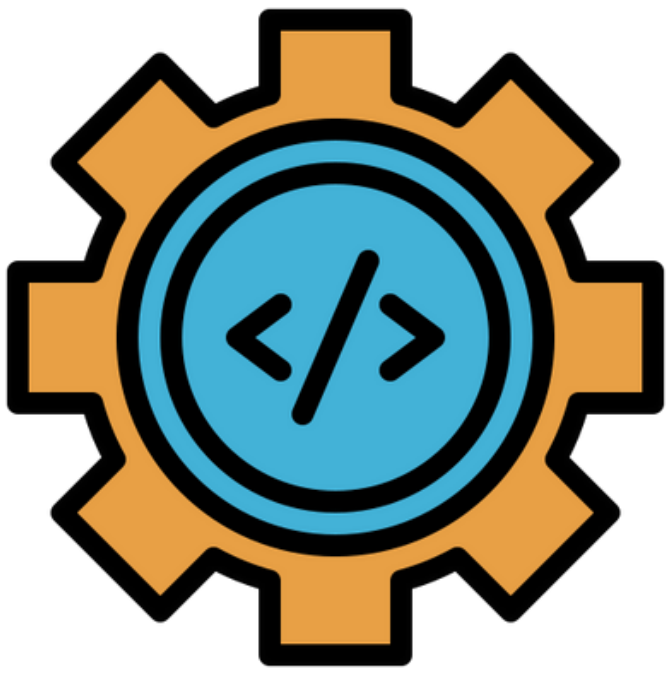
Using the Multisite Deployment Pattern primarily helps improve the user experience for a geographically distributed user base. Users need not be all over the world, but at least distributed such that more than one data center provides sufficient value if the goal is to improve performance. This pattern is also useful for applications requiring a failover strategy in case one data center becomes unavailable. This is a complex subject, but many of the components in this chapter will get you started. Because the use of this pattern will result in a more complex and more expensive application than a single data center solution, the business value needs to be assessed and compared with the cost.

DYNAMIC DNS ROUTING PATTERN

This pattern gives flexibility to faster reconfigure DNS Routing for all above mentioned scenarios and more. F.e. load-balancing, direct access, switch between resources etc.

AUTOMATED CONFIGURATION PATTERN OR INFRASTRUCTURE AS CODE (IAC)

This pattern implies fully automated, scripted and versioned cloud architecture, which enables fast redeployment of the whole architecture for many purposes of scalability and development. When your scripts are versioned in a Git you can track history of your architecture, as well as, fully automate cloud environment provisioning and configuration. Example of this pattern is AWS CloudFormation.



AUTOMATED API MONITORING PATTERN

Using this pattern you can have a full understanding of the performance, control, scalability, consumption patterns and monitoring. Examples are 3scale platform, CloudWatch.

CONTINUES DEPLOYMENT INTO CLOUD PATTERN

This is an application of a standard software development practice applied to cloud environments, when the code is pushed automatically to many QA, Test, Staging/PreProd and other environments.
To sum up, Cloud gives us the possibility to develop highly scalable applications much faster, easier and cheaper.