CLOUD ARCHITECTURE CHECKLIST

WHITEPAPER

TABLE OF CONTENTS

CLOUD ARCHITECTURE CHECKLIST

Cloud computing has substantially advanced the state of the art in technology. In many ways it has simplified software architecture and practices, however, it has also complicated them. Successful cloud architectures must take into account resilience to faults, scalability and maintainability. There are key practices that make those objectives possible.

This e-book looks at possible cloud architecture issues and can serve as a guide to selecting the right infrastructure, software and data architecture to ensure success. A system checklist is also included to guarantee that your cloud architecture leverages best practices.

GENERAL CLOUD ARCHITECTURE CONCERNS

Cloud architecture has some trade-offs. The first trade-off is flexibility versus complexity. If services are too coarsely grained, an application cannot easily be modified or redeployed without a significant loss in service quality or even outages. With extremely fine-grained services, it is easy to alter small bits of the application, but complexity will slow development and result in performance issues.

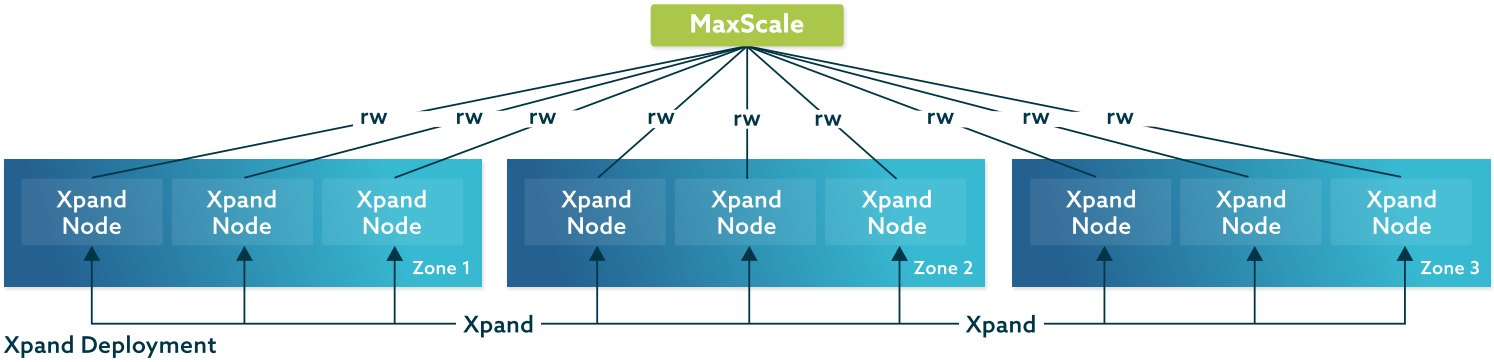
Proper cloud architecture is resilient to faults, secure, scalable to meet demand and maintainable. Achieving these goals requires making the right trade-offs and choosing components that guarantee resilience and distribution while ensuring adequate performance and controlling costs.

RESILIENCE TO FAULTS

Modern users demand ultimate reliability. Users do not remember all of the time a service was available when they needed it. However, they do remember that time it was not. Meanwhile, deploying infrastructure to meet their needs, especially at scale, means dealing with faults and failures all of the time without the user knowing it. In order to accomplish this, the system should provide multiple layers of redundancy.

Zone outages

The most common infrastructure outages are that of a single zone. Each cloud data center or "region" distributed around the world typically has three to four zones. These can be thought of as virtual "racks" in the parlance of yesteryear. **To avoid an outage each service of the system should be deployed on more than one zone per region**.



Xpand Performance Topology – Multi-Zone

Region outages

Regional outages are exceedingly rare at least in the most well-developed parts of the cloud (i.e., U.S. and EU). Partial outages are more common. During a partial outage, the region is up and running but there is some kind of intermittent reliability, connectivity or performance issue. These are equivalent to a data center outage or connectivity issue. **To recover from a regional outage, the system should be deployed in more than one region and applications must be able to switch between them.**

Service outages

Service outages or intermittent service problems are more common. A service is a dependency provided by a third party or cloud service provider. The failure rate of these services differs by vendor and complexity of the service provided, but these failures seem to happen at the worst possible time. They also tend to snowball when cloud provider services are affected.

Architects should balance the probable reliability of self-hosted solutions or the coupling to redundant solutions. It is unlikely that an application developer will be able to host a more reliable messaging service than Amazon does. Creating a coupling to allow failover to a separate messaging service complicates the software and creates another part that could go wrong.

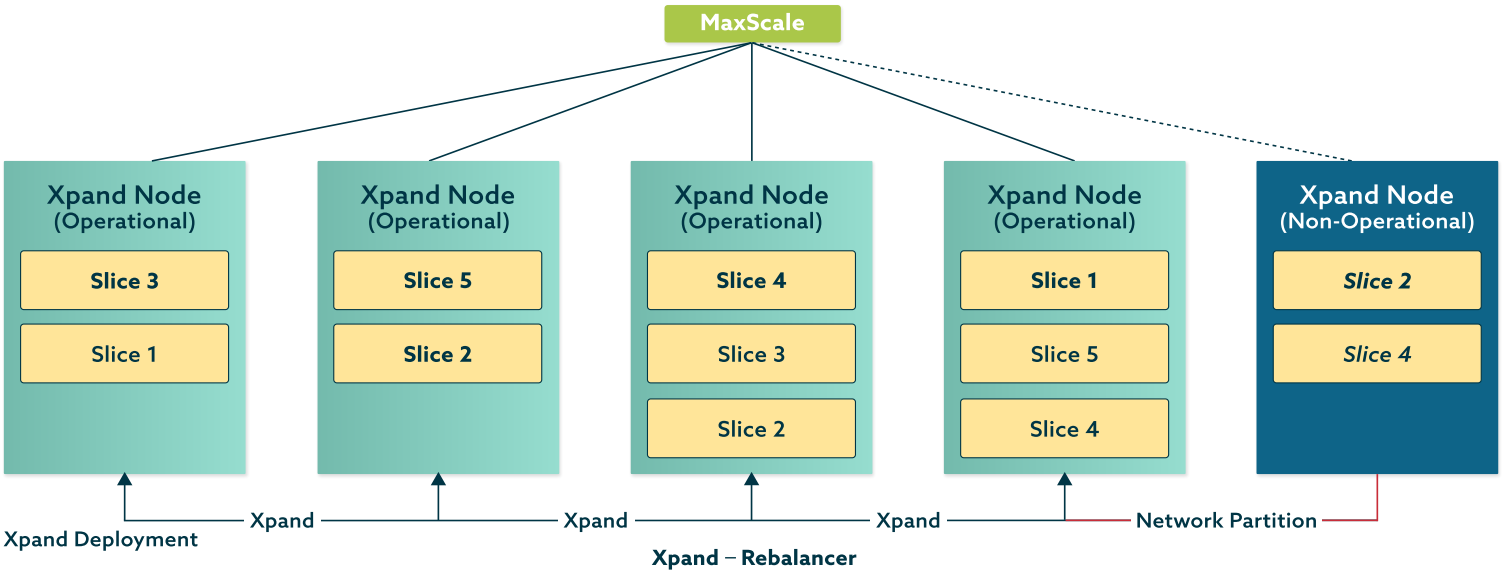
An alternative is to code to a standard or open source API. For instance, in the case of Kinesis, Kafka is an acceptable alternative and is provided by multiple third parties and can even be self-hosted. In a disaster scenario, a different provider or self-hosted version could be selected and deployed in a reasonable amount of time. In other words, **to ensure service dependability use standard APIs or open source solutions available from multiple providers.**

Cloud provider and multi-region outages

True cloud provider outages and multi-region outages are uncommon in the most developed parts of the cloud (i.e., U.S. and EU). In fact, some cloud providers claim to have never lost an entire datacenter and other providers have lost one at some point in time. Thus far, the most catastrophic scenarios (i.e., "a cloud provider outage has brought the internet down") have been due to multiple service providers with a single point of failure on one availability zone (AZ) or due to a service outage. Globally, however, there are still regions with a single availability zone along with infrastructure where natural disasters are more common. Cloud architects should balance complexity before running on multiple clouds concurrently. And to ensure business continuity, applications should use standard and open source APIs to avoid single cloud provider dependencies.

Network infrastructure fault awareness

Load balancers, Domain Name System (DNS), network infrastructure and even application clients must be fault aware. When a failure occurs the load balancer must know to remove the faulty infrastructure from the group. DNS should also change if a region or AZ is missing. However, the application client should not assume these things will definitely happen. The application should know how to failover to redundant DNS servers, IPs or data centers when these failures happen. In other words, **each layer of infrastructure and the application should be aware of failures and know how to find a redundant service.**



Data layer faults

One of the most difficult challenges is when an availability zone or region causes a fault at the data layer. Traditional database infrastructure requires replicating the entire database to at least one other node in one other AZ or ideally an additional region. Distributed database architectures allow using many different nodes across multiple AZs and can replicate to more than one region. With traditional database architectures, failures may temporarily disable writes during a failover. With some distributed database architectures, writes can happen on any node in a region. This is the equivalent of having one or two primaries or potentially a dozen or more primary nodes. For ultimate reliability, **the system should use a distributed database that supports multiple node failures without a loss of read or write functionality.**

A rarely mentioned issue is when a distributed database fails over, it often leaves the client to retry transactions in flight. Application code to catch database faults and retry transactions is difficult to write and maintain. Retry code is a cross-cutting concern that tends to permeate every layer of an application's architecture. Even a small mistake results in intermittent and difficult to detect data loss during faults. To avoid complexity and the possibility of data loss, **use a database proxy that retries transactions automatically during failovers.**

SCALABILITY

Scalability in the cloud is a measure of latency, capacity and cost. A cloud service must meet user expectations in terms of latency but must also have the capacity to meet the needs of all of the system's users. In the cloud, or really any modern business, utilization can be unpredictable so scalability must be achieved quickly. However, excess capacity can be as big of a problem as inadequate capacity.

Cost control

According to Gartner, "Cloud services can initially be more expensive than running on-premises data centers. Cloud services can become cost-effective over time if organizations learn to use and operate them more efficiently." The key to running efficiently is cost control. There are now software and services, known as FinOps, used for reporting and managing cloud costs. For most transactional databases, the majority of the cost is storage and IOPS rather than CPU/instance prices. **At the database layer, ensure IOPS are provisioned and controlled to achieve reasonable performance at a sensible price point.**

Service architecture

A cloud system should be able to deploy more instances of a service and add them to the load balancer without downtime. Services should, where possible, be stateless. This is not to say that services do not store state, but that they do not contain interim states that prevent a user or request from being handled by a different instance. More specifically, **services do not maintain context between requests and are not tied to a user session.** Modern cloud architecture deploys and manages these services via Kubernetes. **As one instance of a service goes down another replaces it.**

Data architecture

Scale requires the ability to use more resources as traffic demands. From a data architecture standpoint, **the database must be able to scale linearly both out and back.** Persistent state, storage in general, and databases, in particular, require different handling than other cloud services. For a truly scale out and back architecture, a distributed database is required. Most critical applications should use a distributed database that assures true consistency, such as a distributed SQL database. **The database should be able to drop a node, add a node and automatically rebalance without data loss or transactional integrity issues.**

MAINTAINABILITY

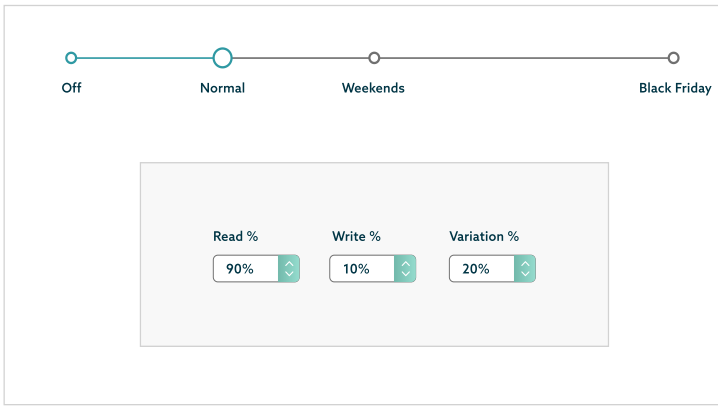
Maintainability and observability

According to industry expert David Linthicum, "Of all the barriers to the cloud moving forward, what scares me most is that we'll build too much complexity with no clear way to manage it at scale." Manageability and observability are critical to maintaining peak uptime and performance in a cloud environment.

For a database, this means being able to control components of the system and inspect operation at a deep enough level. Author and distributed systems expert Cindy Sridharan says, "An observable system is one that exposes enough data about itself so that generating information (finding answers to questions yet to be formulated) and easily accessing this information becomes simple." However, cloud environments are infamous for giving dashboards of mostly irrelevant information and no easy way to get the event log details. **The system should provide enough information from each layer in a well-organized manner to make common problems obvious and less common problems discoverable.**

Cloud databases should provide multiple layers of observability. First, there should be a single-pane monitoring console that shows whether there is a problem with reads, IIOPS, memory, disk – all the usuals. Second, it should allow for direct inspection of events. An administrator sees writes are failing in the dashboard. A few clicks later, they see that there are disk errors. Hopefully, the system fails over quickly.

A maintainable database should be capable of some degree of "self-healing," restarting processes that might fail due to temporary upstream or downstream errors. It should also allow for human intervention in the event of serious faults.



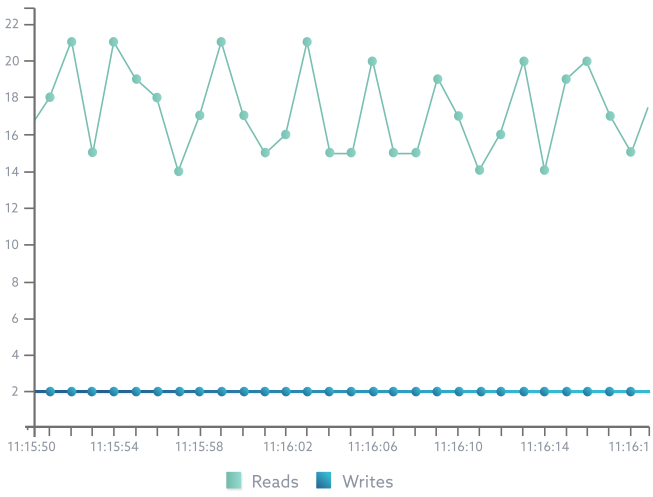
Queries per second

20

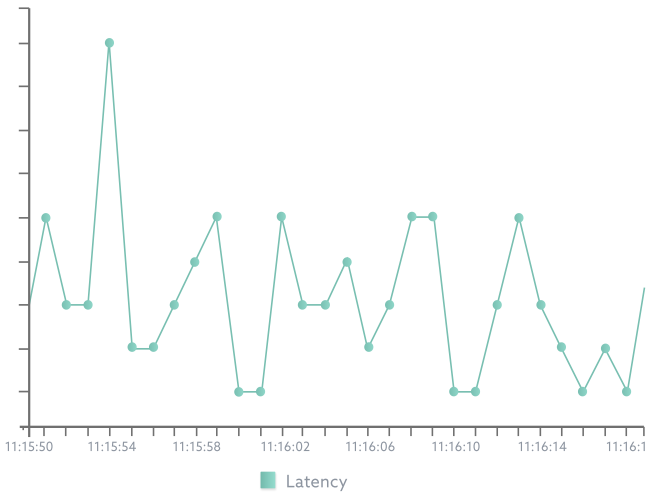
Average latency

39.11 ms

Reads and Writes per second



Transaction latency per second



Online database changes

Critical to maintainability is a database that can enable online changes and atomic schema changes. As applications are changed and deployed, the database will often have to be altered. Changes can fail, in which case they should be completely rolled back and diagnosed. The data architecture should allow the database to be changed without application downtime. In order to do that the **database should be replicable to a secondary cluster and a proxy should be able to redirect application traffic.**

SYSTEM CHECKLIST

The following is a system checklist to inspect your proposed cloud architecture for conformance with best practices.

Resilience to faults	
Each service of the system should be deployed on more than one zone per region	☐
The system should be deployed in more than one region and applications must be able to switch between them	☐
Use standard APIs or open source solutions available from multiple providers	☐
Each layer of infrastructure and the application should be aware of failures and know how to find a redundant service	☐
The system should use a distributed database that supports multiple node failures without a loss of read or write functionality	☐
Use a database proxy that retries transactions automatically during failovers	☐

Scalability	
At the database layer, ensure IOPS are provisioned and controlled to achieve reasonable performance at a sensible price point	☐
Services do not maintain context between requests and are not tied to a user session	☐
As one instance of a service goes down another replaces it	☐
The database must be able to scale linearly both out and back	☐
The database should be able to drop a node, add a node and automatically rebalance without data loss or transactional integrity issues	☐

Maintainability	
The system should provide enough information from each layer in a well-organized manner to make common problems obvious and less common problems discoverable	☐
Cloud databases should provide multiple layers of observability	☐
The database should be capable of some degree of "self-healing," restarting processes that might fail due to temporary upstream or downstream errors	☐

SKYSQL PROVIDES A SCALABLE DATA ARCHITECTURE

A lot of cloud architecture depends on having a resilient and scalable cloud-native database. MariaDB SkySQL meets all of these requirements in an easy-to-deploy DBaaS offering on AWS and GCP. SkySQL supports transactional, distributed SQL, and analytical workloads in one easy-to-use service. Better yet, SkySQL can be used in a hybrid format with MariaDB Enterprise. Both SkySQL and MariaDB Enterprise support the MySQL/MariaDB ecosystem.

Get started with SkySQL today.