Cloud Organisation guide
V0.1 (beta)

KORITSU AI

Intro

Raphael Yoshiga

15+ Developer experience
10+ Cloud usage

Microsoft CERTIFIED
AZURE SOLUTIONS ARCHITECT
EXPERT
CERTIFIED PROFESSIONAL MICROSOFT
Microsoft
MCSE
Cloud Platform and Infrastructure
Earned 2018
CERTIFIED PROFESSIONAL MICROSOFT
Microsoft
MCSD
App Builder
Earned 2016
CERTIFIED kubernetes APPLICATION DEVELOPER
aws certified
Developer
ASSOCIATE

Managed APIs that:

- £2 billion/year – Delivery API Asos.com
- Processed 4,000 requests/sec at VeInteractive
- Online banking dev team, £500 million in savings

Highest ROI Possible

“Run workloads as effectively as possible”

KORITSU AI

Why This Document Exists

The cloud's fundamental promise, unprecedented agility, scalability, and innovation, can swiftly become a source of complexity, risk, and unexpected cost if not met with intentional design. The very ease of spinning up resources can lead to an environment that is opaque, insecure, and financially unpredictable. Success in the cloud is not merely about using it, but about organizing it.

This guide serves as a strategic framework for building a cloud environment that is inherently secure, scalable, and financially accountable.

It is not a rigid, one-size-fits-all prescription. Instead, we present foundational principles, proven patterns, and critical decision points across four core architectural and operational disciplines:

What to Expect from this Guide

Our Three Strategic Pillars

Security by Design: We use "Accounts" as physical boundaries to ensure that a localized issue never becomes an organizational catastrophe.

Financial Optimization: We move from "estimated billing" to "precise accounting," ensuring we capture every available discount and track every penny.

Operational Organization: We eliminate the "Wild West" of random resource names, replacing them with a standardized taxonomy that allows for 100% visibility and automation.

The Language of the Cloud

Before diving into the architecture, it is important to bridge the gap between different providers. An "Isolated Container" has different names depending on who you buy it from, but they serve the same purpose.

Provider	Equivalent Term
AWS	Account
Google Cloud (GCP)	Project
Microsoft Azure	Subscription

For the remainder of this guide, we will be referring as “Accounts” for this logical grouping.

KORITSU AI

Intro – Why standards matter

Transitioning to the cloud without a framework is like building a city without a zoning plan. In the beginning, it’s fast, but eventually, the traffic becomes unmanageable, and the infrastructure collapses under its own weight.

Establishing these standards now, before the first workload is deployed, is critical for three primary reasons:

1. Security
2. Operational Maintenance
3. Financial Accountability

Security By Default

KORITSU AI

Security & Environment Isolation

In the cloud, the strongest security boundary we have is the Account. By physically separating Production from everything else, we ensure that a mistake or a breach in a testing area cannot "bleed" into our live business operations.

The Production / Non-Production Split. We maintain a strict "Air-Gap" between environments. This means:

- Unique Identities: Credentials used in a Development account simply do not exist in the Production account.
- Network Isolation: There is no default routing between a Non-Prod VPC and a Prod VPC.
- Data Protection: Production data should never be pulled down into Non-Prod accounts for testing without rigorous masking and anonymization.

This split besides providing a better Security, it will also bring advantages into our **FinOps** capabilities later.

Least Privilege principle

Access to the Production Account should be treated as a high-risk exception, not a daily right.

- Zero-Standing Access: On a day-to-day basis, nobody, including senior engineers should have permanent "Write" or "Delete" access to Production.
- Handful of Gatekeepers: Only a very small group of "Emergency Admins" should have the rights to bypass standard deployment pipelines.

Just-In-Time (JIT) Access & PIM

To balance security with the need to fix issues, we implement Privileged Identity Management (PIM) or "Request-Based" access.

1. The Request: When an engineer needs to troubleshoot a live issue, they submit a PIM request.
2. The Approval: A manager or a peer must approve the request (Multi-party authorization).
3. The Window: Access is granted for a limited time (e.g., 2 hours).
4. The Audit: Once the time expires, access is automatically revoked, and a full log of every action taken during that window is saved for audit.

Infra Manager Tip: This approach eliminates the risk of "Orphaned Permissions" where an ex-employee or a compromised account retains access to your most sensitive data.

KORITSU AI

Operational Maintenance

Accounts granularity - Goldilocks Principle

When deciding how to slice the organization into accounts, we must account for the **Multiplication Effect**. Every new level we add multiplies the total number of accounts our team must manage.

Account sprawling
If we create an account for every Team across every Environment, the numbers escalate quickly:

10 Teams × 4 Environments (Dev/Test/Stage/Prod) = 40 Accounts.

For a small-to-mid-sized Infra team, managing 40 separate security perimeters, network VPCs, access controls, and billing alerts is often unsustainable.

The “just right” balancing act, depending on your organization size. Larger companies can have granularities by department, even then there might be over 40 accounts, but it can be managed by the large DevOps team.

Deciding between splitting Dev/Test/Staging is also a lever you can use, depending on requirements, you can save on operational maintenance of configuring access to each account.

Strategy	**Pros**	**Cons**
Low Granularity (One big 'Prod' account)	Simplest to network; lowest admin overhead.	Poor "Blast Radius"; impossible to tell which team is spending the most.
High Granularity (Account per Team/Project)	Absolute isolation; perfect cost tracking per team.	**Administrative Nightmare.** Complex cross-account networking and massive overhead.
Balanced (Account per Dept/Env)	**The Best Practice.** Clear ownership and security without the "Sprawl."	Requires a well-defined naming and tagging convention (covered next).

KORITSU AI

Preventing the "Wild West" – Naming Standards

In an environment with hundreds of engineers, infrastructure can quickly become chaotic. A lack of naming standards leads to "Shadow IT," where resources are forgotten, security holes are missed, and costs spiral because "Account-123-Test" doesn't tell anyone who is responsible for the bill.

The goal of the central team is not to dictate, but to **facilitate**. We define the standard template, with input from the dev teams, and then use automation to help them follow it.

Naming Accounts

The account name is the first thing an engineer sees. It should follow a strict hierarchy so it's clear where they are logged in.

Example Template: [Org]-[Dept]-[Env], resulting:

- acme-finance-prod
- acme-marketing-dev

Naming Resources

Inside those accounts, every resource (Servers, Databases, Storage) must follow a predictable "slug." We use hyphens (kebab-case) because they are the most universally accepted character across all cloud providers.

Example Template: [Project]-[Env]-[Region]-[Resource]

- crm-stg-use1-db
- crm-prd-use1-db

Segment	**Purpose**	**Example**
Project	Which app is this?	crm
Env	Which stage?	stg
Region	Physical location	use1 (US-East-1)
Resource	Type of service	vm, db, lb

KORITSU AI

Preventing the "Wild West" – Naming Standards

Defining a naming standard is not just an administrative task; it is an application of core software engineering principles to our physical infrastructure. When resources are named correctly, the infrastructure becomes "readable" code.

In software, we avoid naming variables x or data1 because it forces the next developer to guess their purpose. In the cloud, the same rule applies.

A well-named resource tells a story: "I am a Production Database for the Finance ERP in the UK region." This reduces "cognitive load" for your engineers, they don't need a manual to understand what they are looking at.

Good engineering avoids redundancy. We want our names to be clear but concise.

- Bad: finance-production-dept-finance-database-server (Too much redundant info).
- Good: fin-prd-erp-db (Clean, distinct, and informative). We only include the information necessary to distinguish the resource within its specific context.

Adapting to Resource Constraints

While we strive for a universal standard, we must account for the "Physical Laws" of the cloud providers. Different resource types have different naming limitations:

- Global Uniqueness: Some resources (like S3 Buckets in AWS or Storage Accounts in Azure) must have a name that is unique across the entire world, not just within our company.
- Character Restrictions: Certain resources only allow lowercase letters and numbers, with no hyphens or underscores allowed (e.g., Azure Storage Accounts are limited to 24 characters, lowercase only).
- Length Limits: Some legacy cloud services truncate names after 15 or 63 characters.
- Our Strategy: We define a Base Standard, but we provide a "Constraint Map" for specific resource types. If a resource doesn't allow hyphens, we simply remove them while keeping the segments in the same order. This ensures the "DNA" of the name remains recognizable even when the format changes.

KORITSU AI

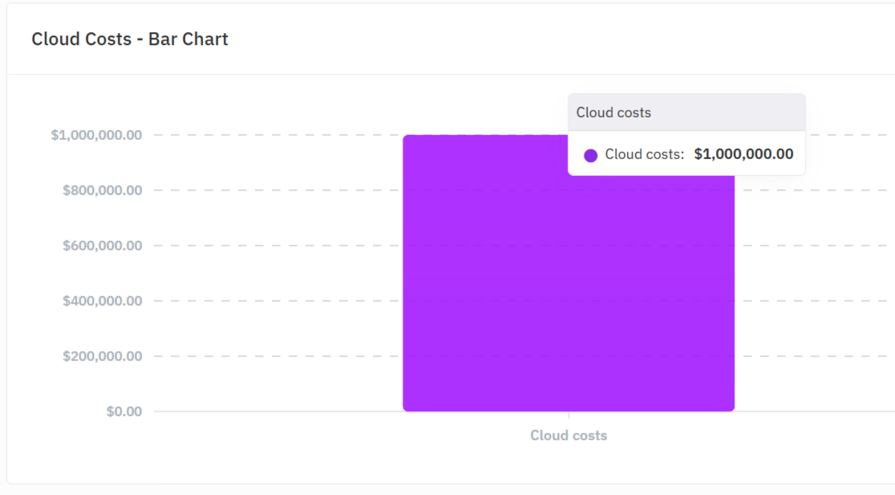
Financial Optimization

Cost Allocations

It’s not uncommon the monthly cloud bill remains a frustrating "black box." You see a single, mounting line item for the entire engineering organisation, but the trail goes cold when you try to connect those dollars to specific business outcomes.

If you can’t see exactly which product, team, or department is driving spend, you aren't just missing data; you're missing the ability to make informed strategic pivots.

You can imagine your $1,000,000 monthly cloud bill arrives. It’s a single, monolithic bar labelled "Cloud Infrastructure: $1,000,000." To a CFO, this is a "Black Box." You know the money is gone, but you don't know if it was spent growing the business or just keeping the lights on.

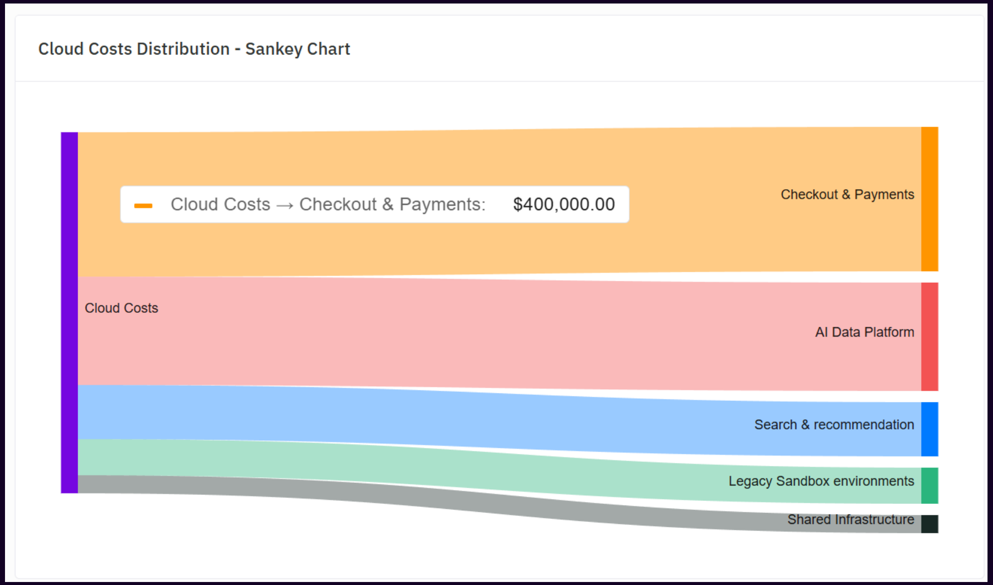


Cost Allocations

Now, imagine transforming that single bar into this Chart for an example e-commerce platform.

Suddenly, that $1,000,000 starts to tell a story:

- Product Attribution: You see that $400,000 went to the "Checkout & Payments" engine. If revenue is up 20% this month, that’s a healthy investment.
- Team Accountability: You see $150,000 flowing into the "Search & Recommendation" team. Now you can measure the ROI of that team's latest AI deployment.
- Waste Identification: You notice $100,000 flowing into "Legacy Sandbox Environments" that haven't been touched in six months. That’s an immediate, data-backed saving.
- Operational Health: You see $50,000 in "Shared Infrastructure" (like networking or security) prorated across all departments, giving you a true cost-to-serve for every customer order.



Cost Allocations

How to do this in practice?

Tagging. We can define a standard set of metadata for every resource, by for example:

- Team
- Product
- Department

Each business have different requirements and cost center structures that you can leverage. The key is to have the standard and enforce it.

How to enforce?

Each cloud provider supports having Resource Policies, so that new resources can’t created unless the right tags are in place.

References

- Azure policy definitions for tag compliance

KORITSU AI

Environment Segregation

The primary reason we enforce a hard split between Production and Non-Production accounts is that they serve two completely different business purposes. Trying to manage them with the same rules results in either stifled innovation (too much red tape in Dev) or unacceptable risk (too much freedom in Prod).

Production accounts

The Production account is the "Revenue Engine." Its primary requirements are Stability and Trust.

- Service Level Agreements (SLA): This environment requires high-availability configurations. Downtime here has a direct financial cost.
- Tighter Security & Compliance: This account holds "Live" customer data. It must be hardened to meet regulatory standards (GDPR, PCI, SOC2).
- Restricted Access: Human access is the #1 cause of downtime. Production access is strictly controlled, audited, and limited to a handful of vetted personnel using temporary credentials.

Non-Production accounts

The Non-Production accounts are "Innovation Labs." Their primary requirement is Velocity.

- Flexibility over SLA: Dev and Test environments do not need 99.99% uptime. If a dev server goes down at 2:00 AM, the business doesn't lose money.
- Test Data Only: By policy, these accounts contain no sensitive customer data. This lower risk profile allows for more relaxed security rules, giving developers the freedom to experiment without risking a major data breach.
- Developer Autonomy: Developers have broader permissions here to build, break, and rebuild. This ensures that the "messy" phase of creation stays far away from the "stable" phase of operation.

From a FinOps perspective, this gives us clear expectations of costs, given "Marketing Dev“ should generally be significantly cheaper than "Marketing Prod.“

Raphael Yoshiga - Founder
raphael.yoshiga@koritsu.ai
+44 7342 635234

KORITSU AI