

ELLYPUNK

Mastery

WEEKLY DIGEST / M401

21-27 September 2026



This week

21-27 September 2026

Available archived editions, in their original date order.

MONDAY / 21 September 2026

Agent Tool Authorization: Bind Approval to the Exact Action

3

[Open original online](#)

TUESDAY / 22 September 2026

API Rate Limits: Design Fair Budgets and Safe Backoff

8

[Open original online](#)

WEDNESDAY / 23 September 2026

API Key Rotation: Overlap Credentials and Audit Revocation

13

[Open original online](#)

THURSDAY / 24 September 2026

Recovery Drills: Prove a Database Backup Can Restore

18

[Open original online](#)

FRIDAY / 25 September 2026

Database Schema Rollouts: Expand, Migrate, Contract

23

[Open original online](#)

Only retained editions are included; gaps have not been filled.

Copyright Ellypunk. Contains AI-generated artwork.

ellypunk.com

AGENT TOOL AUTHORIZATION

BIND APPROVAL TO THE EXACT ACTION



GOOD SYSTEMS RESPECT BOUNDARIES

- ☒ CLEAR ACTIONS
- ☒ EXPLICIT APPROVAL
- ☒ ENFORCED LIMITS
- ☒ SAFER OUTCOMES

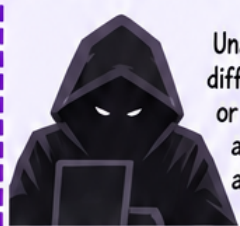
AGENT SYSTEMS
SECURITY BY DESIGN
HUMAN OVERSIGHT
RELIABLE OUTCOMES

OBJECTIVE



OBJECTIVE:
Make approval valid only for the action the user actually reviewed.

THREAT BOUNDARY

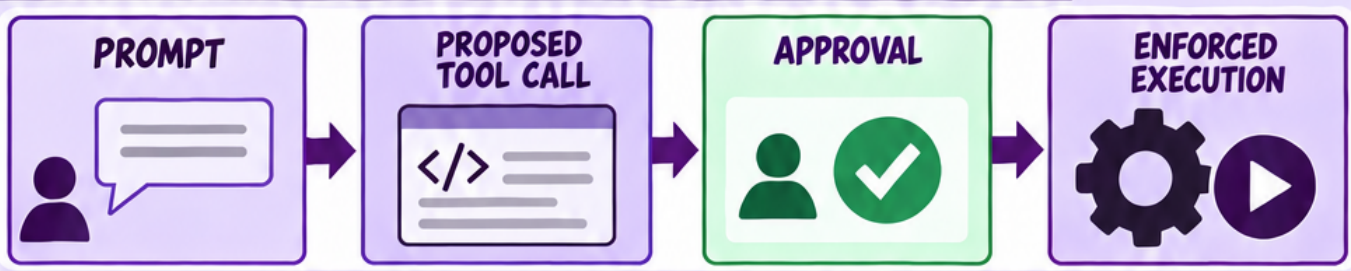


Unapproved tools, different parameters, or modified actions are outside the approval boundary.



NOT APPROVED = NOT ALLOWED

APPROVAL-TO-EXECUTION FLOW



BIND THE WHOLE REQUEST

An approval is only valid for one exact action. These five fields define it completely.

Each field removes ambiguity. Bind them all.

Precise
Actions
Safer
Systems
Brighter
People

SECURITY
SYSTEMS
TRUST
GOOD DESIGN

Better
Questions
Better
Systems

</>

1

ACTOR

Who may act?

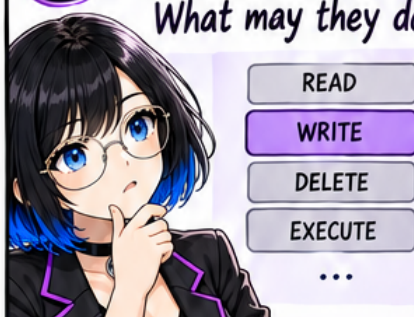


- User or service identity
- Authenticated and authorized
- No wildcard actors

2

OPERATION

What may they do?



- Specific, named operation
- One action, not a batch
- No broader privileges

3

RESOURCE

Which exact object?



- Exact, canonical identifier
- No wildcards or patterns
- Single, well-defined object

4

NORMALIZED ARGUMENTS

What values are enforced?



- Canonicalized and validated
- Expected keys and types
- No extra or ignored parameters

5

TIME + NONCE

When, and only once?



- Expiry time (time-bound)
- Cryptographic nonce (one-time use)
- Reject if expired or nonce already used

APPROVAL RECORD

APPROVAL RECORD =
actor + operation + resource +
normalized arguments +
expiry + nonce

Same action.
Same record.
No surprises.

DETAILS
MAKE
TRUST

LEAST PRIVILEGE
EXACT ACTIONS
AUDIT TRAILS
SAFER TOMORROW



ENFORCE AT EXECUTION

Same approval.
Exact execution.
No exceptions.

EXACT
ACTIONS
SAFER
SYSTEMS
BRIGHTER
PEOPLE
♡



Policies
Mean More
When Enforced
♡

SECURE SYSTEMS
TRUSTED TOOLS
REAL OUTCOMES

1 • NORMALIZE

Resolve the exact
resource and argument
types before approval.

REQUEST
tool: get_file
path: "/docs/a.txt"
limit: 100
(user input)

NORMALIZED CANONICAL FORM

tool: get_file
path: /docs/a.txt
limit: integer(100)

2 • COMPARE

Execution must match
the approved record
exactly.

APPROVED RECORD

tool: get_file
path: /docs/a.txt
limit: 100
nonce: 7f3a...
expires: 2026-09-21

EXECUTION ATTEMPT

tool: get_file
path: /docs/a.txt
limit: 100
nonce: 7f3a...
expires: 2026-09-21

=

ALL FIELDS MUST MATCH

3 • RECHECK

Policy, resource identity,
expiry, and nonce are
checked at the
enforcement point.

- Policy still allows this action
- Resource identity matches
- Not expired
- Nonce not used before

ENFORCEMENT POINT
Revalidate now,
with current state.

4 • DENY

Changed, expired, or
replayed requests
do not run.

- Arguments changed
- Different resource
- Expired request
- Replay (nonce used)
- Policy no longer allows
- REQUEST BLOCKED**

TOCTOU GUARDRAIL:
Revalidate mutable facts as close
to execution as possible.

Same Action
Same Context
Same Safety
♡



WORKED EXAMPLE

PROBLEM:

A delete tool can cause irreversible loss.

SAME APPROVAL
≠
DIFFERENT TARGET

EXACT ACTIONS
KEEP PEOPLE
AND DATA SAFE

TOOLS

AUTHORIZATION

TARGET BINDING

SAFER SYSTEMS

APPROVED



delete
report/2026/Q3.csv

Approved to delete
this exact file.

CHANGED



delete
reports/*

The target was changed
from a specific file
to a wildcard.

ENFORCED RESULT



RESULT: DENY

The operation and resource
no longer match the approval.

IMPROVEMENT

IMPROVEMENT: Show
the normalized target,
bind it into the approval
record, and request new
approval for any change.



Never broaden a
single-file approval
into a wildcard.



AGENT TOOL AUTHORIZATION BIND APPROVAL TO THE EXACT ACTION

RECAP: WHAT WE BUILT



PRECISE
APPROVAL



POLICY
ENFORCEMENT



EXACT
ACTION



AUDIT
TRAIL

PRODUCTION CHECKLIST



Change one argument
→ **DENY**



Change the resource
→ **DENY**



Replay the nonce
→ **DENY**



Execute after expiry
→ **DENY**



Revoke policy before
execution
→ **DENY**



AUDIT



AUDIT: Record the proposal,
approval identity, normalized action,
decision, execution result, and time.

ELLY'S FINAL TIP

Approval is safe only when
the enforcement point can prove:
**this exact actor, this exact action,
now.**

Secure approvals.
Real safety.
That's mastery!

Better
Questions
Safer Agents
Brighter People
♡

PEOPLE
PROCESS
TECHNOLOGY
TRUST

API RATE LIMITS

DESIGN FAIR BUDGETS AND SAFE BACKOFF



OBJECTIVE



OBJECTIVE:

Share capacity fairly and recover without a retry storm.

WHAT ARE API RATE LIMITS?



Rate limits cap how many requests you can make in a given time window so resources are shared fairly.

They protect stability, ensure fairness, and keep services available for everyone.

KEY PRACTICES

1. BUDGET PER CLIENT



Set clear budgets or quotas per client, user tier, or feature to share capacity fairly.

2. HANDLE 429 RESPONSES

429

Too Many Requests

Retry-After: 120

When you receive 429, respect the optional Retry-After header to know when to try again.

3. BOUNDED RETRIES WITH JITTER



Retry a limited number of times with exponential backoff and jitter to avoid a retry storm.

4. MEASURE AND IMPROVE



Track rejection rates (429s) and request latency. Use the data to tune budgets, backoff, and user experience.



CAPACITY AND BURST

A TOKEN BUCKET HAS A CAPACITY AND A REFILL RATE

A token bucket lets you allow bursts while keeping a steady long-term rate.

- ☒ CAPACITY FOR BURSTS
- ☒ STEADY REFILL RATE
- ☒ FAIR TO EVERYONE
- ☒ PROTECT REAL CAPACITY

Good Limits Build Better Products

DESIGN WITH DATA

API RESILIENCE

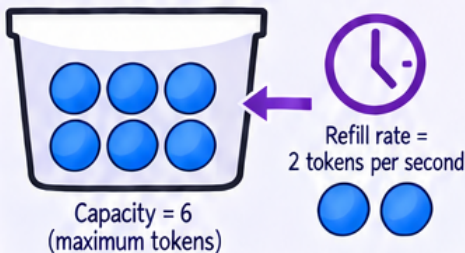
FAIR USAGE

REAL IMPACT

1 CAPACITY HOLDS TOKENS

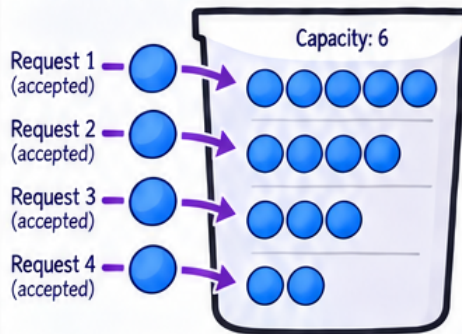
A token bucket has a capacity and a refill rate.

Capacity: 6 tokens
Refill rate: 2 requests per second



2 EACH REQUEST USES ONE TOKEN

Each accepted request uses one token.



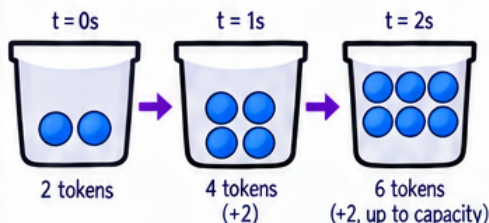
3 WHEN EMPTY: 429

When the bucket is empty, the API may respond 429 (Too Many Requests).



4 REFILL OVER TIME

Tokens are added back at the refill rate, up to the capacity. Bursts are allowed, but only up to the capacity.



5 CHOOSE LIMITS WISELY

Choose limits to protect your real capacity and keep the system healthy.

- ☒ Set a capacity that allows useful short bursts.
- ☒ Set a refill rate that matches your real capacity.
- ☒ Consider all clients and use cases to keep things fair.

Good limits protect users and the system.

6 KEY TAKEAWAY



KEY TAKEAWAY:

A token bucket has a capacity and a refill rate. Each accepted request uses one token. When empty, the API may respond 429. Choose limits to protect real capacity.

Right limits keep APIs fast, fair and reliable for everyone.

PLAN CAPACITY

OBSERVE USAGE

PROTECT USERS

BUILD FOR TOMORROW

Better APIs Brighter Tomorrow

Fair Limits Brighter Tomorrows



FAIRNESS NEEDS A SCOPE

**SET BUDGETS BY CLIENT OR TENANT,
NOT JUST GLOBALLY.**

Fairness works best when you set budgets by client or tenant, not just globally.

RIGHT SCOPE FAIRER OUTCOMES

- ☒ CLIENT OR TENANT BUDGETS
- ☒ SHARED COUNTERS WHEN NEEDED
- ☒ CLEAR 429 GUIDANCE
- ☒ BOUNDED RETRIES WITH JITTER
- ☒ MONITOR AND IMPROVE

Fair APIs
Build Trust

SCOPE FOR FAIRNESS

DESIGN FOR REAL USE

OBSERVE AND IMPROVE

USERS COME FIRST

1 SCOPE BUDGETS PROPERLY

Set budgets by client or tenant, not just a single global limit.

Global Limit
(less fair)



One shared limit
(all clients compete)

Per-Client or Per-Tenant
(more fair)



Each client or tenant gets its own budget and refill rate.

2 SHARED COUNTERS WHEN APPROPRIATE

Use shared counters when multiple instances must enforce the same limit (e.g., across servers) to ensure consistent and fair usage.



- ☒ One limit enforced everywhere
- ☒ Keeps usage fair
- ☒ Prevents easy bypass
- ☒ Works across multiple instances

3 RETURN CLEAR 429 GUIDANCE

When a request is rate limited, return a clear 429 response. Include a helpful message and optionally a Retry-After header.

HTTP 429 Too Many Requests

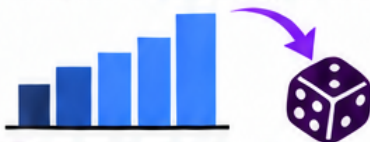
Message: "Rate limit exceeded.
Please try again later."

Retry-After: 120 (optional)

Clear guidance helps clients recover smoothly.

4 BOUNDED RETRIES WITH JITTER

Clients should retry a limited number of times with exponential backoff and jitter to avoid adding more load.



- ☒ Use a limited number of retries.
- ☒ Increase the delay between retries.
- ☒ Add jitter to spread out retry attempts.
- ☒ Stop retrying after the limit.

5 MONITOR REJECTION RATE AND LATENCY

Track rejection rate (429s) and request latency to understand how your limits are working in practice.



- ☒ Measure 429 rate (rejections).
- ☒ Monitor request latency.
- ☒ Use the data to tune budgets, backoff, and user experience.
- ☒ Look for patterns and improve over time.

6 KEY TAKEAWAY



KEY TAKEAWAY:
Fairness needs a scope.

- ☒ Set budgets by client or tenant, not just globally.
- ☒ Use shared counters when instances enforce one limit.
- ☒ Return clear 429 guidance, optionally Retry-After.
- ☒ Use bounded retries with jitter.
- ☒ Monitor rejection rate and latency.
- ☒ Keep technical claims scoped as written.

Right scope, clear signals,
and thoughtful clients make
APIs fair for everyone.

Respect
Users
Build
Together

PLAN WITH A SCOPE

MEASURE REAL IMPACT

KEEP THINGS FAIR

BUILD A BRIGHTER TOMORROW

Better
APIs
Brighter
Tomorrow



WORKED EXAMPLE: RETRY STORM

TURN PRINCIPLES INTO A REAL SCENARIO.

Examples
Turn
Ideas
Into
Better
Systems
♥

Let's walk
through a real
example step
by step!
♥

SAME GOAL, REAL IMPACT

- ✓ USE FAIR BUDGETS
- ✓ APPLY SAFE BACKOFF
- ✓ MEASURE THE RESULTS
- ✓ IMPROVE WITH DATA

Fair
Design
Happier
Users
♥

REAL EXAMPLES

PRACTICAL DESIGN

MEASURE AND LEARN

A BRIGHTER TOMORROW

1 PROBLEM: 100 CLIENTS RETRY TOGETHER AFTER HTTP 429

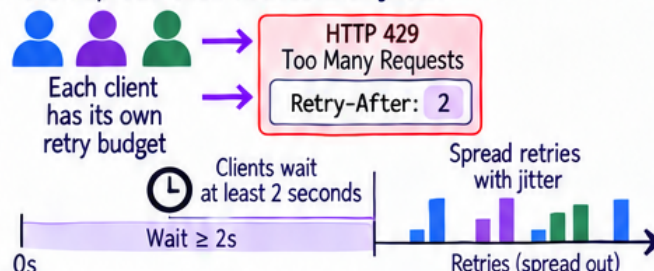
A burst of 100 clients all retry at the same time after receiving HTTP 429 (Too Many Requests). This can overload the service and keep everyone in a retry storm.



2 ACTION: APPLY PER-CLIENT BUDGETS AND SEND RETRY-AFTER: 2

Set per-client budgets to limit retries. When the service returns HTTP 429, include:

Retry-After: 2. Clients wait at least 2 seconds, then spread their retries with jitter.



3 CHECK: MEASURE 429 RATE AND LATENCY UNDER THE SAME LOAD

Run the same load (100 clients) and measure the 429 rate and request latency. Both should fall compared to the initial burst.



4 IMPROVE: TUNE BUDGETS FROM OBSERVED TRAFFIC

Use the measured results to tune per-client budgets. Adjust based on real traffic patterns to further reduce 429s and keep latency healthy.



Real examples
turn good ideas
into better systems.
♥

Fair
Systems
Stronger
Together
♥

MEASURE IN PRACTICE

TUNE WITH DATA

KEEP USERS FIRST

BUILD A BRIGHTER TOMORROW

Better
APIs
Brighter
Tomorrow
♥





RECAP: FAIR, BOUNDED, OBSERVABLE.

KEY STEPS FOR RESILIENT API DESIGN

RECAP CHECKLIST

FAIR, BOUNDED, OBSERVABLE.

1 SIZE CAPACITY AND REFILL FROM REAL SERVICE LIMITS.



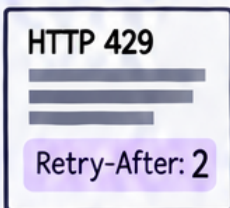
Size capacity and refill from real service limits.

2 SCOPE BUDGETS FAIRLY BY CLIENT OR TENANT.



Scope budgets fairly by client or tenant.

3 GIVE CLEAR 429 RETRY GUIDANCE.



Give clear 429 retry guidance.

4 BOUND ATTEMPTS, ADD JITTER, MEASURE 429 RATE AND LATENCY.



Bound attempts, add jitter, measure 429 rate and latency.



FINAL TIP



A limit is a production policy, not just a number.

Fair APIs
Brighter Tomorrow

PEOPLE FIRST
REAL IMPACT
CONTINUOUS LEARNING
A BRIGHTER TOMORROW



API KEY ROTATION

Overlap credentials and audit revocation.

Rotate with confidence. Keep your users running smoothly. ♡

PLAN
TRANSITION
VERIFY
REVOKE

- ✓ CONTINUITY
- ✓ OBSERVABILITY
- ✓ CLEAR PROCESS
- ✓ SAFER OUTCOMES

PRACTICAL SECURITY

REAL-WORLD OPERATIONS

BETTER QUESTIONS

BRIGHTER TOMORROW

OBJECTIVE

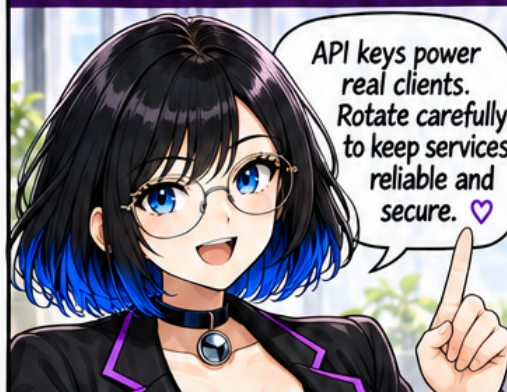


Objective:

Replace a key without breaking clients, then retire the old one safely.

Continuity today.
A safer tomorrow.

WHY IT MATTERS



API keys power real clients. Rotate carefully to keep services reliable and secure. ♡

HAPPY USERS

RELIABLE SERVICES

LOWER RISK

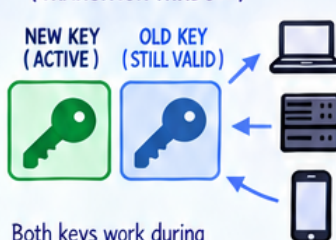
CLEAN AUDIT TRAILS

A SAFE ROTATION WINDOW

Create the new key first.



BOTH KEYS VALID (TRANSITION WINDOW)



Revoke the old key after verified migration.



Plan. Verify. Revoke.
That's how you rotate like a pro. ♡

Same Elly.
Better systems.
A brighter tomorrow.



ELLYPUNK

ellypunk.com/training/api-key-rotation-overlap-credentials-and-audit-revocation

© 2026 Ellypunk / NeetOps

KNOW EVERY KEY OWNER

Visibility turns risk into control.

Inventory
Clarity
Security
Confidence
Brighter
Tomorrow
♡

Before you rotate, know where every key lives and who owns it. ♡

SAME ACTION.
SAME CONFIDENCE.

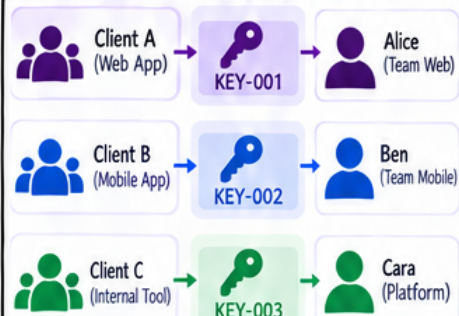
- ✓ FIND OWNERS
- ✓ TRACK USAGE
- ✓ LOCATE DEPLOYMENTS
- ✓ ENABLE ROTATION

Better Questions
Better Systems
A Brighter Tomorrow
♡

INVENTORY
ACCOUNTABILITY
REAL-WORLD SECURITY
BRIGHTER TOMORROW

1 Map each key to a client and owner.

Build a complete inventory. Every key must be linked to the client or service that uses it and a clear owner.



Clear ownership.
Fewer surprises.

2 Track a non-secret key ID and last-use signal.

Record a safe identifier and when the key was last used. This helps you understand what is still active and what might already be safe to remove.



Track signals, not secrets. ♡

Key ID	Client / Service	Owner	Last Use
KEY-001	Web App	Alice	23 Sep 2026
KEY-002	Mobile App	Ben	21 Sep 2026
KEY-003	Internal Tool	Cara	15 Sep 2026
KEY-004	Analytics	Dan	03 Sep 2026
KEY-005	Reports	Erin	Never

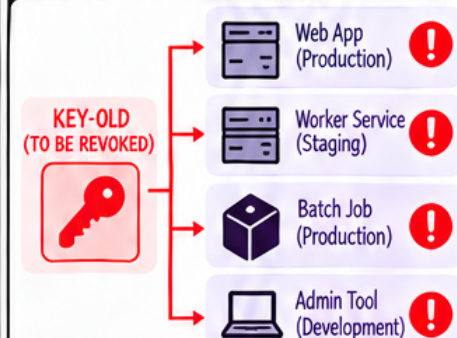
Usage signals show you what still matters.

3 Find every deployment that still uses the old key.

Search your environments to locate all deployments that still reference the old key. Update or retire them before revocation.



No stragglers. Rotate with confidence. ♡



Find it everywhere.
Update or remove it.



Never log the key value itself. ♡

Same action.
Same record.
A brighter tomorrow.

PEOPLE
PROCESS
VISIBILITY
SAFER SYSTEMS
BRIGHTER TOMORROW



CONTROL THE OVERLAP

Two valid keys. A safer transition.

Control the overlap to keep services reliable and secure. ♡

PLAN
SECURE
MONITOR
REVOKE

- ✓ SAME RESTRICTIONS
- ✓ USE A SECRET MANAGER
- ✓ BOUNDED TRANSITION
- ✓ MONITOR KEY USAGE
- ✓ REVOKE OLD KEY

PRACTICAL SECURITY
REAL-WORLD OPERATIONS
BETTER QUESTIONS
BRIGHTER TOMORROW

1 • SAME RESTRICTIONS

Give the new key the same needed restrictions.

NEW KEY
(ACTIVE)

OLD KEY
(STILL VALID)



- ✓ Same permissions
- ✓ Same scope
- ✓ Same resource access
- ✓ Same usage policies

Consistency reduces risk.

2 • USE A SECRET MANAGER

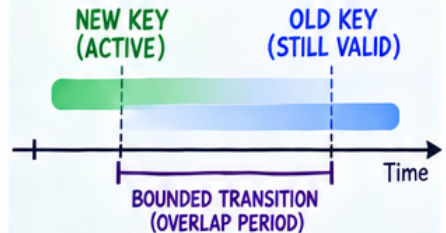
Store it in a secret manager, not source code.



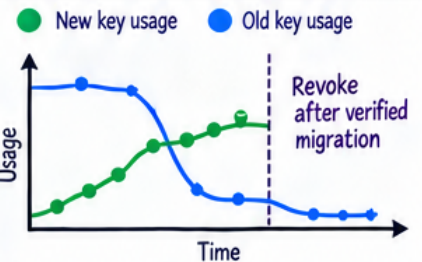
Keep credentials out of code.
Keep systems safer.

3 • BOUNDED TRANSITION

Keep both keys valid only for a bounded transition.



MONITOR KEY USAGE



Watch which key IDs are used before you revoke. ♡

Same Elly.
Better systems.
A brighter tomorrow. ♡



WORKED EXAMPLE: ONE CLIENT LAGS

Find it. Fix it. Verify it. Revoke it safely.

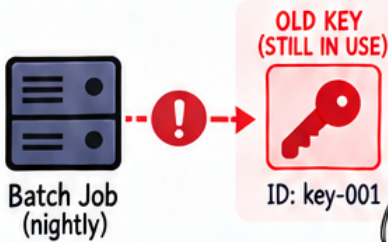
SAME
PEOPLE
SAFER
SYSTEMS
A BRIGHTER
TOMORROW

Real systems
have stragglers.
Let's handle one
together. ♡

OBSERVE
FIX
VERIFY
REVOKE
CONTINUOUSLY IMPROVE

1. PROBLEM

One batch job still uses the old key.



Most clients have switched,
but this batch job is still using
the old key.

2. ACTION

Update its secret, then rerun the job.



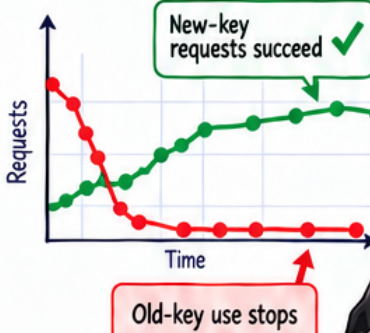
Update the batch job
to use the new key,
then rerun it. ♡

3. CHECK

New-key requests succeed; old-key use stops.

KEY USAGE (MONITORING)

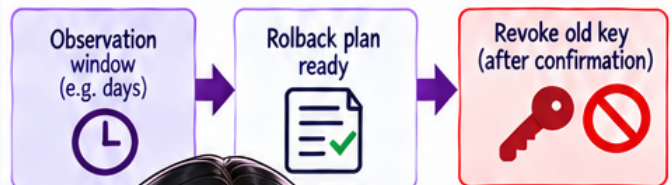
● New key (key-002) ● Old key (key-001)



Monitor to confirm
the new key is working
and the old key is no
longer being used. ♡

4. IMPROVE

Revoke only after an observation window
and rollback plan.



A quiet key
may still belong
to a rare job. ♡



ROTATION CLOSEOUT

Finish strong. Prove it. Keep it safe.

AUDIT TIMELINE (EXAMPLE)



Document each step with evidence.

ROTATION CLOSEOUT CHECKLIST

1 Confirm every client moved to the new key.



- ☒ All clients updated
- ☒ No clients using the old key
- ☒ Monitor usage during transition

2 Observe through the rare-job cycle.



- ☒ Monitor for the full rare-job cycle
- ☒ Verify expected jobs run with the new key
- ☒ Check for any stragglers

Some services run infrequently. Give it time and verify.

3 Disable the old key; test new-key access.



- ☒ Old key disabled
- ☒ Test new-key access works
- ☒ Confirm no errors in normal workflows

Disabling stops further use. Testing confirms everything still works.

4 Delete the old key after the rollback window.



- ☒ Rollback window completed
- ☒ Old key deleted
- ☒ Final audit record captured

Delete only after the rollback window to allow for safe recovery.

A rotation ends with evidence of safe revocation.

Secure
Clients
Stable Systems
Happier People

PEOPLE
PROCESS
TECHNOLOGY
TRUST



RECOVERY DRILLS: PROVE A DATABASE BACKUP CAN RESTORE



OBJECTIVE



OBJECTIVE:
Prove the recovery point
and recovery time with
a real test restore.

KEY MESSAGE



A backup is only as good
as your ability to restore it.
Validate recovery with a
test restore in an isolated
environment.

Test today.
Recover
tomorrow.

BACKUP FILE



- ✓ File exists
- ✓ Looks valid
- ✓ Stored safely

A backup file alone
is not proof of a
usable recovery.

TEST RESTORE



- ✓ Restore completes
- ✓ Data is usable
- ✓ Recovery time known
- ✓ Proves readiness

A successful test
restore proves
your backups work.



SET THE RECOVERY TARGET

Define how much data loss is acceptable and how long you can be down.

Clear targets help you test the right things.

RESILIENT DATA
REAL RECOVERY
STRONGER TOMORROW

SAME SKILLS.
BRIGHTER TOMORROW.

RECOVERY POINT OBJECTIVE (RPO)

Maximum acceptable data loss, measured as time.



Example target:
30 min



RECOVERY TIME OBJECTIVE (RTO)

Maximum acceptable outage duration.



Example target:
60 min



A restore drill measures the actual gap and outage against these targets.

These are example targets, not universal guarantees.



DESIGN A SAFE RESTORE DRILL



**PRODUCTION
(DO NOT MODIFY)**



**ISOLATED
TEST TARGET**



1

Inventory the backup chain and required keys.



2

Restore to an isolated test target.



3

Check data and application access.



4

Record the recovered point and elapsed time.



**A file check is not
a substitute for a restore.**



WORKED EXAMPLE: THE RESTORE DRILL

Follow the steps, prove the outcome, then keep getting better. That's how backups build confidence. ♡

A restore drill turns a backup into a proven recovery path.

PRACTICE
RECOVER
BE READY ♡

BACKUP

ISOLATED
RESTORE ENVIRONMENT

- ✓ Practice
- ✓ Validate
- ✓ Learn
- ✓ Improve

1 PROBLEM

A backup exists, but recovery is unproven.

?

BACKUP

- ✓ Backup file exists
- ✓ Looks valid
- ✓ Stored safely

2 ACTION

Restore it to an isolated target.

Restore to an isolated system so you can test safely, without affecting production.

ISOLATED
RESTORE ENVIRONMENT

BACKUP

3 CHECK

Data and app checks pass.
Data gap: 20 min. Outage: 45 min.

It works!

- ✓ Data checks pass
- ✓ Application works
- ✓ Data gap: 20 min.
- ✓ Outage: 45 min.

4 IMPROVE

Record missing access or slow steps for the next drill.

Turn what you learn into a better, faster, more reliable recovery.

IMPROVEMENTS FOR NEXT DRILL

- ✓ Missing access to request
- ✓ Steps that were slow
- ✓ Anything that caused delay



Example targets:
RPO 30 min; RTO 60 min.
Both met.

Drills turn hope into evidence.
Keep practicing. ♡

RECOVER FASTER

VALIDATE REGULARLY

REDUCE RISK

BE A MORE RELIABLE YOU

Same backups.
A brighter tomorrow.
♡



RECAP: PROVE THE PATH



1  **Know the RPO and RTO targets.**

2  **Restore on an isolated target.**

3  **Test data and application access.**

4  **Record gaps, duration, and repairs.**



M401 • MASTERY

25 September 2026

PAGE 1

DATABASE SCHEMA ROLLOUTS

Expand, Migrate, Contract

Evolve your database without breaking your users. ♡

Same Data.
More Possibilities. ♡

SCHEMA EVOLUTION

RELIABLE RELEASES

DATA FOR PEOPLE

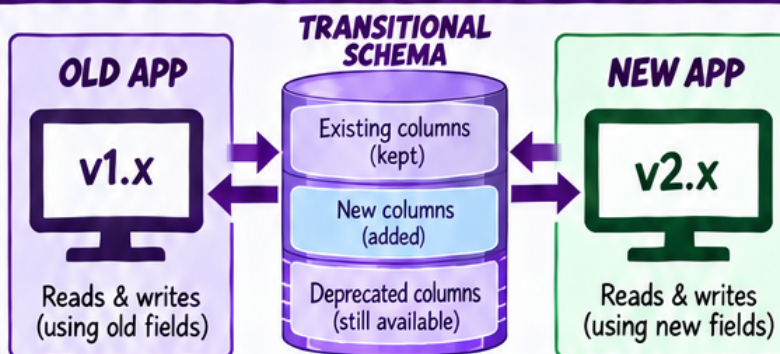
A BRIGHTER TOMORROW



OBJECTIVE

Change a live schema while old and new application versions overlap.

COMPATIBLE TRANSITIONAL SCHEMA



RELEASE TIMELINE

1

EXPAND



Add new columns (tolerate both versions).

2

MIGRATE



Deploy new app version and let both versions run.

3

VERIFY



Confirm data is consistent and complete.

4

CONTRACT



Remove old columns after all traffic is on the new version.

5

COMPLETE



Simpler schema. Cleaner future.

Evolve deliberately.
Keep your systems running.
Ship with confidence. ♡

Evolve
Build
Simplify
Together ♡

SCHEMA CHANGES

LIVE SYSTEMS

SMOOTHER TRANSITIONS

BRIGHTER TOMORROWS

Better Data
Brighter People ♡



ELLYPUNK

| ellypunk.com/training/database-schema-rollouts-expand-migrate-contract | © 2026 Ellypunk / NeetOps

THREE SAFE PHASES

Expand, Migrate, Contract

Big changes are safer when you do them in three phases.

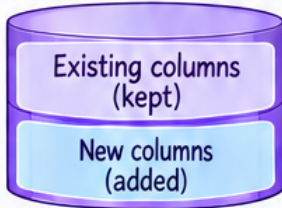
SCHEMA CHANGES
LIVE SYSTEMS
SMOOTHER TRANSITIONS
BRIGHTER TOMORROWS

Better Data
Brighter People

1

EXPAND

Add the new shape without removing the old.



- ✓ Existing app keeps working.
- ✓ New columns are available.
- ✓ No behaviour changes for current users.

2

MIGRATE

Dual-write, backfill, and move readers after validation.



- ✓ Write to both old and new.
- ✓ Backfill existing data.
- ✓ Move readers to the new fields after validation.
- ✓ Run with both versions in parallel.

3

CONTRACT

Remove old fields only after every old consumer has retired.



- ✓ Confirm all old consumers have retired.
- ✓ Remove old columns.
- ✓ Leave a simpler, cleaner schema.
- ✓ Lower long-term maintenance costs.



APPLICATION TIMELINE (HIGH LEVEL)



Old App

Reads & writes (uses old fields)



New App

Reads & writes (uses new fields)

1 EXPAND

Using old fields

2 MIGRATE

Reading & writing new fields

3 CONTRACT

Retires after migration

Only new fields (kept)

Rollback is easiest before Contract.

SCHEMA EVOLUTION
RELIABLE RELEASES
DATA FOR PEOPLE
A BRIGHTER TOMORROW

Evolve
Build
Simplify
Together



ELLYPUNK

| ellypunk.com/training/database-schema-rollouts-expand-migrate-contract

| © 2026 Ellypunk / NeetOps

GUARD THE OVERLAP

Expand, Migrate, Contract

Keep both versions working correctly.
Protect your users during the overlap.

Same Data.
More Possibilities.

PLAN CAREFULLY

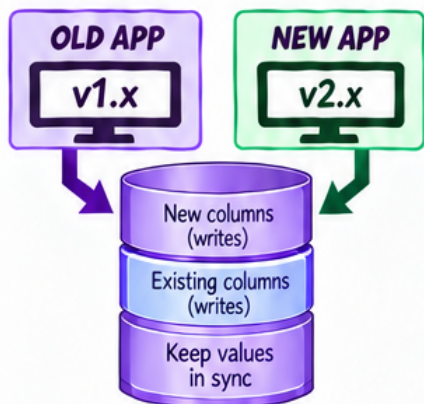
VERIFY CONTINUOUSLY

PROTECT USERS

A BRIGHTER TOMORROW

1 DUAL WRITES

Keep old and new values consistent while both app versions run.



- ✓ Write to both schemas
- ✓ Validate data as you write
- ✓ Handle failures safely
- ✓ Monitor for inconsistencies

2 BACKFILL

Work in bounded batches and verify every migrated row.



- ✓ Process data in bounded batches
- ✓ Verify each batch (row counts, samples, checksums)
- ✓ Track progress and handle errors
- ✓ Re-run safely if needed

3 OBSERVE

Track mismatches, errors, and old-reader traffic.



- ✓ Monitor mismatch counts
- ✓ Alert on errors
- ✓ Track old-reader traffic
- ✓ Confirm data is consistent before moving to contract

Do not contract while any old consumer remains.

OBSERVE REALITY
RESPECT YOUR USERS
EVOLVE DELIBERATELY
BRIGHTER TOMORROWS

Better Questions.
Better Software.
A Brighter Tomorrow.





RENAME WITHOUT A CLIFF

Expand, Migrate, Contract

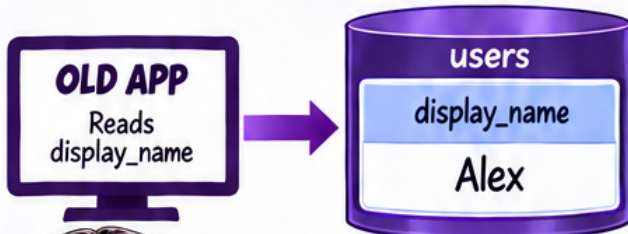
Better Data
Brighter Futures

SCHEMA CHANGES
LIVE SYSTEMS
SMOOTHER TRANSITIONS
BRIGHTER TOMORROWS



1 PROBLEM

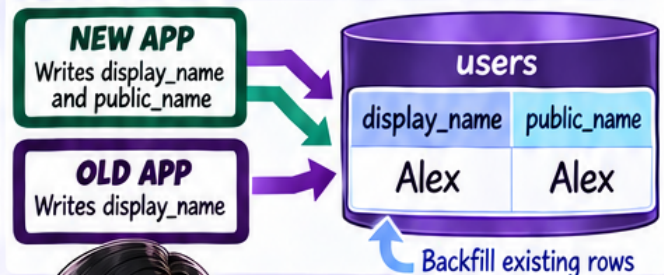
Old app reads display_name.



Right now we only have one column: display_name. The old app reads it.

2 ACTION

Add public_name. Dual-write and backfill.



We add public_name, dual-write in the new app, and backfill existing rows.

3 CHECK

Compare values. Test both readers.



Compare values to make sure display_name and public_name match. Test both readers.

4 IMPROVE

Move new reads. Retire old field only after old readers stop.



The new app reads public_name, while the old app still reads display_name. Retire the old field only after old readers stop.

**Evolve deliberately.
Keep your systems running.
Ship with confidence.**

Same Data.
More Possibilities.



CONTRACT ONLY WHEN READY

**Expand for compatibility.
Migrate with evidence.
Contract last.**

Finish strong.
Contract only
when you're
ready.



Same
Data.
More
Possibilities.

SCHEMA EVOLUTION
RELIABLE RELEASES
DATA FOR PEOPLE
A BRIGHTER TOMORROW



CONTRACT READINESS CHECKLIST

**CHECK BEFORE
YOU CONTRACT**

1



Old and new
values match.

2



Backfill is
complete.

3



Old-reader
traffic is zero.

4



Rollback and
backup plan
is reviewed.

**The last drop
is the least
reversible step.**



Evolve
Build
Simplify
Together

SCHEMA CHANGES
LIVE SYSTEMS
SMOOTHER TRANSITIONS
BRIGHTER TOMORROWS

Better
Data
Brighter
People

