Platform Known Error Database

FM-KED-000 Getting Started

The **Finmars Known Error Database** is a curated registry of recurring and well-understood operational issues observed in Finmars environments, together with documented recovery procedures and practical guidance.

Its purpose is not to list every possible failure, but to capture **patterns that are already known**, reproducible, and solvable through established algorithms. Each entry describes the symptoms, scope of impact, severity, and an expected recovery path based on real operational experience.

This document serves three audiences at once

- operators who need fast orientation during incidents
- customers who want transparency and predictability
- support teams who require a shared operational memory

Every known error in this database is classified by **severity** and **recovery class**, allowing readers to understand both the business impact and the typical effort required to restore normal operation. Where possible, step-by-step recovery instructions are provided to reduce response time and avoid unnecessary investigation.

Issues that are **not present** in this database are considered unknown by definition. Such cases may require discovery, diagnostics, or architectural analysis and are handled outside the scope of predefined recovery commitments.

The Finmars Known Error Database is a living document. It evolves with the platform, operational experience, and lessons learned from real incidents. Its goal is clarity, not completeness. Reliability, not illusion.

Each known issue carries **two orthogonal markers**:

1. **Severity** – impact on the business or system
2. **Recovery Class** – expected effort and time to restore service

---

Severity Levels (Impact-based)

S1 — Critical

- System unusable or data integrity at risk
- No viable workaround

- Immediate attention required

S2 — High

- Core functionality degraded
- Workarounds exist but are painful
- Business impact is noticeable

S3 — Medium

- Partial feature loss
- Clear workaround available
- Limited operational impact

S4 — Low

- Cosmetic or non-blocking behavior
- No impact on business operations

Severity answers the question:
“How bad is this for the user?”

Recovery Class A — Quick Fix

- Expected resolution: **under 1 hour**
- Fully documented procedure
- Typical for routine maintenance issues

Recovery Class B — Standard Recovery

- Expected resolution: **up to 4 hours**
- Known issue with multiple steps or validation
- Usually fits within monthly support window

Recovery Class C — Extended Recovery

- Expected resolution: **up to 1 business day**
- High complexity or cross-component dependency
- Not fully covered by standard support

Recovery Class D — Exploratory

- Resolution time **unknown**

- Partial or missing diagnostic data
- Discovery and investigation required

Recovery class answers the question:
“How long does it usually take when things go normally?”

How it looks in registry

Example

- **Issue ID:** FM-KED-###
- **Title:** Background jobs stuck in pending state
- **Severity:** S2 — High
- **Recovery Class:** B — Standard Recovery
- **Estimated Recovery Time:** up to 4 hours
- **Covered by Monthly Support:** Yes
- **Resolution Algorithm:** documented

For an unknown issue:

- **Severity:** S1 — Critical
- **Recovery Class:** D — Exploratory
- **Covered by Monthly Support:** No
- **Notes:** subject to separate agreement

FM-KED-001 — VM Disk Space Exhaustion

Severity: S2 — High
Recovery Class: A — Quick Fix
Covered by Monthly Support: Yes

Description

Disk space on a virtual machine reaches critical levels, leading to degraded system behavior, application instability, or failed background operations.

This issue is operational, recurrent, and typically caused by uncontrolled growth of logs, containers, temporary files, or Kubernetes artifacts.

Typical Symptoms

- Services failing to write logs or temporary files
- Background jobs failing without explicit errors
- Kubernetes pods entering `Evicted` or `Terminating` state
- System warnings related to low disk space

Diagnostic Checklist

Identify Top Disk Consumers

```
sudo du -ahx / | sort -rh | head -n 20
```

Recovery Procedure

Follow the steps below **as needed**, not necessarily all of them.

---

1. Clean Package Manager Artifacts

```
sudo apt-get autoremove
sudo du -sh /var/cache/apt
sudo apt-get autoclean
sudo apt-get clean
```

2. Clean System Journals

```
sudo journalctl --vacuum-time=3d
```

3. Truncate Docker Logs

```
sudo truncate -s 0 /var/lib/docker/containers/**/*-json.log
```

4. Prune Docker Resources

```
sudo docker system prune
```

5. Remove Obsolete Kubernetes ReplicaSets

```
kubectl get rs -A -o wide | tail -n +2 | \
awk '{if ($3 + $4 + $5 == 0) print "kubectl delete rs -n "$1, $2 }' | sh
```

6. Clear Evicted Kubernetes Pods

```
kubectl get pods | grep Evicted | awk '{print $1}' | xargs kubectl delete pod
```

With explicit kubeconfig:

```
kubectl --kubeconfig bank.yaml get pods | grep Evicted | \
awk '{print $1}' | xargs kubectl --kubeconfig bank.yaml delete pod
```

7. Force Remove Stuck Terminating Pods

```
for p in $(kubectl --kubeconfig bank.yaml get pods | grep Terminating | awk '{print $1}');
do
  kubectl --kubeconfig bank.yaml delete pod $p --grace-period=0 --force
done
```

Optional Diagnostics

Inspect Memory Usage (for runaway processes)

```
ps -eo size,pid,user,command --sort -size | \
awk '{ hr=$1/1024 ; printf("%13.2f Mb ",hr) } \
{ for ( x=4 ; x<=NF ; x++ ) { printf("%s ",$x) } print "" }'
```

Preventive Notes

- Disk usage monitoring is strongly recommended
- Log rotation must be verified after updates
- Kubernetes cleanup should be part of routine maintenance

FM-KED-002 — PostgreSQL Table and Index Bloat (Missing or Ineffective VACUUM)

Severity: S2 — High
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes

Description

PostgreSQL database performance degrades over time due to table and index bloat caused by insufficient or ineffective `VACUUM` operations.

This issue manifests gradually and is commonly observed on systems with high write activity, long-running transactions, or misconfigured autovacuum settings.

Typical Symptoms

- Slow queries without obvious query plan changes
- Increased disk usage on database volumes
- Tables or indexes significantly larger than expected
- Elevated I/O usage
- Application timeouts under normal load

Diagnostic Checklist

Identify Database Size and Largest Tables

```
SELECT
  relname AS table_name,
  pg_size_pretty(pg_total_relation_size(relid)) AS total_size
FROM pg_catalog.pg_statio_user_tables
ORDER BY pg_total_relation_size(relid) DESC
```

```
LIMIT 10;
```

Check Autovacuum Activity

```
SELECT
  relname,
  last_vacuum,
  last_autovacuum,
  n_dead_tup
FROM pg_stat_user_tables
ORDER BY n_dead_tup DESC;
```

Recovery Procedure

Follow steps **carefully**. Some operations are I/O intensive.

---

1. Run Manual VACUUM (Non-blocking)

```
VACUUM (VERBOSE, ANALYZE);
```

Recommended for moderate bloat and active systems.

---

2. Vacuum Specific Tables

```
VACUUM (VERBOSE, ANALYZE) table_name;
```

Use when bloat is localized.

---

3. Reclaim Disk Space (Blocking)

```
VACUUM FULL table_name;
```

⚠ Locks the table for the duration of the operation
⚠ Use during maintenance windows only

---

4. Reindex Bloated Indexes

```
REINDEX TABLE table_name;
```

Or concurrently, when supported:

```
REINDEX INDEX CONCURRENTLY index_name;
```

Preventive Notes

- Ensure autovacuum is enabled and properly tuned
- Monitor `n_dead_tup` growth over time
- Avoid long-running transactions
- Schedule periodic maintenance for write-heavy tables

Operational Notes

- Disk space reclaimed by `VACUUM` is reusable by PostgreSQL, not always returned to the OS
- `VACUUM FULL` physically rewrites tables and should be used sparingly

---

FM-KED-003 — Network Connectivity Loss on Virtual Machine

Severity: S1 — Critical
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes (diagnostics only)

Description

A virtual machine becomes partially or fully unreachable due to loss of network connectivity. This may affect administrative access, application availability, or external integrations.

The root cause may lie in operating system configuration, firewall rules, cloud security settings, or infrastructure provider issues.

Typical Symptoms

- SSH access unavailable or unstable
- Applications unreachable from external networks
- Inability to access external services or the public internet
- Timeouts in inter-service communication

Diagnostic Checklist

Proceed in order. Each step narrows the responsibility boundary.

---

1. Verify SSH Connectivity

```
ssh user@vm_ip
```

If SSH is unreachable:

- Verify correct IP and credentials

- Check whether the VM responds to ICMP (ping), if allowed

---

2. Verify Internet Access from the VM

```
ping -c 3 8.8.8.8
curl https://example.com
```

Distinguish between:

- No outbound connectivity
- DNS resolution issues

---

3. Check OS-Level Firewall Rules

```
sudo iptables -L -n
sudo ufw status
```

Verify that required inbound and outbound traffic is allowed.

---

4. Check Cloud Security Groups and Network Rules

- Review inbound and outbound rules in the cloud provider console
- Confirm correct ports, protocols, and source ranges
- Verify network routing and subnet configuration

---

5. Escalation to Infrastructure Provider

If all checks above are inconclusive:

- Collect timestamps, VM identifiers, and observed symptoms
- Open a support ticket with the cloud provider
- Attach diagnostic evidence and test results

This step marks the transition beyond Finmars operational control.

Preventive Notes

- Restrict firewall changes to controlled processes
- Audit security group changes regularly
- Maintain documented network topology and access rules

Responsibility Boundary

Finmars SCSA provides best-effort diagnostics and configuration verification.
Network outages caused by infrastructure providers, underlying hardware, or provider-managed networks are outside Finmars SCSA responsibility.

FM-KED-004 — Backup Failure Due to Excessive Backup Size

Severity: S1 — Critical
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes

Description

Automatic daily backups fail because the generated database dump exceeds available disk capacity on the backup worker or application node.

This condition usually indicates uncontrolled growth of historical data and must be addressed immediately to restore backup continuity.

Typical Symptoms

- Daily backup job not completed or failing repeatedly
- Backup files missing or incomplete
- Disk space exhaustion on backup or worker node
- Alerts indicating failed or skipped backup runs

Root Cause

- Excessive volume of historical records retained in the database
- Insufficient disk capacity on the worker node where
  - the Authorizer VM is deployed, or
  - the Finmars Community Edition is installed

Diagnostic Checklist

Verify Backup Job Status

- Confirm backup job execution logs
- Identify failure reason and timestamps

Check Disk Space on Backup Node

```
df -h
```

Estimate Database Dump Size

```
pg_dump --format=custom --file=/tmp/test.dump db_name
ls -lh /tmp/test.dump
```

Recovery Options

Choose **one** or a combination, depending on business requirements.

Option 1: Remove Obsolete Historical Records

- Identify historical tables with excessive row counts
- Confirm data retention requirements
- Delete or archive obsolete records
- Re-run backup after cleanup

⚠ Data deletion is irreversible and must be explicitly approved.

Option 2: Increase Disk Capacity on Backup Worker Node

- Extend disk size of the worker node
- Ensure sufficient free space for full backup generation
- Re-run the backup job and verify completion

This option preserves all historical data.

Preventive Notes

- Define and enforce data retention policies
- Monitor backup file sizes over time
- Monitor free disk space on backup and worker nodes
- Periodically validate backup completion, not only existence

---

Responsibility Boundary

Finmars SCSA provides diagnostics, recommendations, and operational guidance.
Infrastructure changes such as disk resizing may depend on customer approval or cloud provider action.

FM-KED-005 — SSL/TLS Certificate Expiration

Severity: S1 — Critical
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes

Description

SSL/TLS certificates used by Finmars services expire, causing secure connections to fail and rendering applications inaccessible over HTTPS.

This issue is time-based and entirely recoverable through certificate renewal.

Typical Symptoms

- Browsers displaying certificate expiration warnings
- HTTPS connections rejected by clients or integrations
- API calls failing due to TLS handshake errors
- Monitoring alerts related to certificate validity

Diagnostic Checklist

Verify Certificate Expiration

```
openssl s_client -connect domain:443 -servername domain | openssl x509 -noout -dates
```

Identify Certificate Termination Point

- Nginx reverse proxy
- Kubernetes Ingress controller

Recovery Procedure

Follow the procedure relevant to the deployment model.

Option 1: Renew Certificate in Nginx Proxy

- Generate or obtain renewed certificate
- Replace certificate and private key in Nginx configuration
- Reload Nginx configuration

```
sudo nginx -t
sudo systemctl reload nginx
```

Option 2: Renew Certificate in Kubernetes Ingress

- Renew certificate via the configured certificate manager
- Update or recreate TLS secret used by the Ingress
- Verify Ingress reload and certificate propagation

Preventive Notes

- Track certificate expiration dates
- Use automated renewal where possible
- Monitor certificate validity proactively

Responsibility Boundary

Finmars SCSA provides best-effort renewal guidance and validation.
Certificate issuance authority availability and DNS control remain customer responsibilities.

FM-KED-006 — Kubernetes Cluster Certificate Expiration

Severity: S1 — Critical
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes

Description

Internal Kubernetes certificates expire, leading to partial or complete cluster malfunction. This may affect control plane communication, node registration, API access, or workload scheduling.

This issue typically appears in long-running clusters where certificate rotation was not automated or monitored.

Typical Symptoms

- `kubectl` commands failing with TLS or x509 errors
- Nodes switching to `NotReady` state
- Control plane components restarting or failing
- Ingress, networking, or admission controllers malfunctioning

---

Diagnostic Checklist

Verify Certificate Expiration

On control plane node:

```
sudo kubeadm certs check-expiration
```

If `kubeadm` is not available, inspect certificates directly:

```
openssl x509 -in /etc/kubernetes/pki/apiserver.crt -noout -dates
```

Recovery Procedure

⚠ Perform these steps on the **control plane node**
⚠ Requires administrative access

---

1. Renew Kubernetes Certificates

```
sudo kubeadm certs renew all
```

This renews all cluster certificates managed by kubeadm.

---

2. Restart Control Plane Components

```
sudo systemctl restart kubelet
```

Kubernetes will automatically recreate static pods for:

- kube-apiserver
- kube-controller-manager
- kube-scheduler

---

3. Refresh Local kubeconfig Files

```
sudo cp /etc/kubernetes/admin.conf ~/.kube/config
sudo chown $(id -u):$(id -g) ~/.kube/config
```

Repeat for any other kubeconfig files in use.

---

4. Verify Cluster Health

```
kubectl get nodes
kubectl get pods -A
```

Ensure all nodes return to `Ready` state and system pods stabilize.

Preventive Notes

- Monitor certificate expiration dates regularly
- Schedule certificate renewal before expiration
- Prefer automated rotation where supported

- Avoid running clusters indefinitely without maintenance

Responsibility Boundary

Finmars SCSA provides best-effort operational guidance.
Clusters not managed via kubeadm or heavily customized may require additional investigation beyond standard support scope.

FM-KED-007 — 502 Bad Gateway Error (Application Unreachable)

Severity: S1 — Critical
Recovery Class: B — Standard Recovery
Covered by Monthly Support: Yes (known causes only)

Description

Nginx returns a **502 Bad Gateway** error because the Django application backend becomes unreachable.

In the majority of observed cases, this is caused by **Out Of Memory (OOM)** conditions on the virtual machine hosting the application worker, leading to termination of Gunicorn or equivalent application processes.

Typical Symptoms

- HTTP 502 responses from Nginx
- Application intermittently unavailable
- Gunicorn workers restarting or disappearing
- Kernel logs indicating OOM events

Primary Root Cause

- Application requests producing excessive memory usage
- Large datasets loaded into memory
- Insufficient RAM on the worker virtual machine

When memory limits are exceeded, the operating system terminates the application process, leaving Nginx without a valid upstream.

Diagnostic Checklist

Confirm OOM Condition

```
dmesg | grep -i oom
journalctl -k | grep -i kill
```

Check Available Memory

```
free -h
```

Verify Application Server Status

```
systemctl status gunicorn
```

Recovery Options

Apply one or more of the following, depending on constraints.

Option 1: Reduce Request Scope

- Apply stricter filters to API requests
- Limit requested date ranges
- Reduce number of portfolios, instruments, or entities per request
- Avoid bulk data retrieval in a single call

This reduces memory pressure at the application level.

Option 2: Increase RAM on Worker Virtual Machine

- Increase memory allocation on the worker VM
- Restart application services after resizing
- Verify stability under previous load

This addresses the issue at the infrastructure level.

Escalation and Unknown Issues

If the issue persists after:

- request scope reduction, and
- sufficient memory allocation

then the incident is classified as an **unknown issue**.

Such cases require investigation, profiling, or architectural analysis and are **not covered** by the standard monthly support allocation.

Preventive Notes

- Avoid unbounded API queries
- Monitor memory usage trends
- Define safe defaults and limits at API level
- Prefer asynchronous processing for heavy workloads

Responsibility Boundary

Finmars SCSA provides best-effort diagnostics and guidance for known memory-related causes. Application design decisions and infrastructure capacity planning beyond documented scenarios require separate analysis.

FM-KED-008 — HTTP 500 Internal Server Error (Application Error)

Severity: S2 — High
Recovery Class: D — Exploratory
Covered by Monthly Support: No (fix requires product change)

Description

The application returns an **HTTP 500 Internal Server Error**, indicating an unhandled exception or logic failure inside the Finmars application.

In most cases, this represents a **software defect** rather than an infrastructure or configuration issue.

Typical Symptoms

- API requests returning HTTP 500
- Application stack traces in logs
- Errors reproducible with the same input
- No corresponding infrastructure or resource alerts

Primary Meaning

A 500 error usually means:

- the request reached the application correctly
- the application failed while processing it

This class of error cannot be reliably resolved through operational actions alone.

Required Action

Register an Issue in Finmars Core Repository

All confirmed 500 errors should be reported via GitHub:

https://github.com/finmars-platform/finmars-core/issues

When creating an issue, include:

- request details (endpoint, parameters)
- error messages or stack traces
- timestamps
- environment details

Resolution Path

- Issue is analyzed by Finmars maintainers
- Fix is implemented in the product codebase
- Resolution is delivered via a **new Finmars release**

Customers must upgrade to the fixed version to resolve the issue.

Urgency Handling

- **Most urgent bugs** are addressed in **best-effort time**
- Priority depends on severity, reproducibility, and impact
- No guaranteed fix timelines are implied

Estimated Recovery Time

Not predictable
Depends on investigation complexity and release cycle

Preventive Notes

- Keep Finmars versions up to date
- Monitor application logs for early signals
- Avoid unsupported customizations where possible

Responsibility Boundary

Finmars SCSA provides guidance, triage assistance, and escalation support.
Bug fixes require product changes and fall outside standard operational support.

FM-KED-009 — Missing Data in Reports

Severity: S3 — Medium
Recovery Class: A — Quick Fix
Covered by Monthly Support: Yes

Description

Reports return incomplete results or missing values because required **Prices** and/or **FX Rates** are not available for the requested reporting date.

This is not an application error. Finmars can only calculate reports based on data that exists in the system.

Typical Symptoms

- Empty or partially populated report fields
- Missing valuations or calculated figures
- Reports returning results for some instruments but not others
- No application or infrastructure errors present

Primary Meaning

The reporting engine executed successfully, but the **input data set is incomplete**.

Most commonly:

- prices are missing for one or more instruments
- FX rates are missing for one or more currencies
- data is not available for the exact requested date

Diagnostic Checklist

Identify Report Date

- Confirm the exact date used in the report

Verify Price Availability

- Check that instrument prices exist for the report date

Verify FX Rate Availability

- Check that FX rates exist for all required currency pairs

Recovery Procedure

Import Missing Data into Finmars

- Import required **Prices** for the requested date
- Import required **FX Rates** for the requested date
- Re-run the report after data import

Once data is present, the report will calculate correctly.

Preventive Notes

- Ensure price and FX data feeds are complete
- Monitor data import success regularly
- Align report dates with available market data

Responsibility Boundary

Finmars SCSA provides diagnostics and guidance.
Data availability and correctness depend on customer data imports or connected providers.
