Valkey

**Valkey – Introduction**

**What is Valkey?**

**Valkey** is an open-source, high-performance, in-memory data store, forked from Redis under the Linux Foundation in 2024. It retains **full compatibility with Redis commands and data structures** while aiming for enhanced community governance and transparent development.

Valkey is designed for **speed, scalability, and low latency** operations, making it suitable for caching, real-time analytics, message brokering, and more.

🔍 **Key Use Cases of Valkey**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Description** | **Example Scenarios** |
| **Caching** | Store frequently accessed data in-memory to reduce database or API load. | User session caching, API response caching, frequent DB query results. |
| **Real-time Analytics** | Fast reads/writes support analytics pipelines requiring sub-millisecond latency. | Real-time dashboards, metrics aggregation, leaderboard ranking. |
| **Pub/Sub Messaging** | Built-in publish/subscribe messaging pattern for lightweight messaging between services. | Notifications, event broadcasting, microservice communication. |
| **Rate Limiting** | Atomic increment and TTL commands make Valkey ideal for implementing rate limits. | API gateway rate limiting, login attempt throttling. |
| **Queueing System** | Lists and streams can be used to build queues for task distribution. | Background job queues, task scheduling pipelines. |
| **Geospatial Indexing** | Native support for storing and querying geospatial data. | Store locations, proximity-based services, delivery apps. |
| **Data Structure Server** | Supports various data types beyond simple key-value storage. | Hashes for object storage, sets for membership checks, sorted sets for scoring systems. |

📦 **What Valkey Includes – Features Table**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Data Structures** | Strings, Hashes, Lists, Sets, Sorted Sets, Bitmaps, HyperLogLogs, Streams, Geospatial indexes. |
| **Persistence Options** | RDB snapshots, AOF (Append Only File) logging for durability and recovery. |
| **Replication** | Supports master-replica replication for scalability and availability. |
| **High Availability** | Sentinel for monitoring, failover, and automatic reconfiguration. |
| **Cluster Mode** | Sharding across multiple nodes to scale horizontally. |
| **Pub/Sub** | Lightweight publish/subscribe messaging system. |
| **Lua Scripting** | Run Lua scripts atomically on the server for complex operations. |
| **Modules Support** | Extend Valkey functionalities via modules (same as Redis modules). |
| **High Performance** | Handles millions of requests per second with low latency. |
| **ACL & Security** | Role-based access control, TLS encryption, client authentication. |

ℹ️ **Additional Information**

* **Governance:** Managed under the **Valkey project in Linux Foundation**, encouraging open development and diverse community contribution.
* **Compatibility:** Fully compatible with Redis clients, commands, and modules. Migrating from Redis to Valkey is straightforward.
* **License:** BSD-3-Clause, ensuring permissive open-source usage in enterprise and open projects.
* **Release cadence:** Regular releases focused on stability, performance improvements, and community-requested features.
* **Support ecosystem:** Growing contribution from cloud vendors, database companies, and open-source contributors to maintain and enhance features.
* **Official website:** [https://valkey.io](https://valkey.io/)

**. Valkey Architecture Diagram**

**1.1. Standalone Architecture**

diff

CopyEdit

+-----------+

| Client |

+-----------+

|

v

+-----------+

| Valkey |

| Standalone|

+-----------+

|

v

[ Persistence (AOF, RDB) ]

➡️ **Description**: Single Valkey instance serving read/write requests. Persistence can be enabled via **AOF or RDB snapshots** for data durability.

**1.2. Master-Replica Architecture**

pgsql

CopyEdit

+-----------+

| Client |

+-----------+

|

v

+-----------+

| Master |

| Valkey |

+-----------+

|

+------+------+

| |

v v

+-----------+ +-----------+

| Replica 1 | | Replica 2 |

+-----------+ +-----------+

➡️ **Description**: Master handles writes, replicas replicate data for **high availability and read scalability**.

**1.3. Sentinel-based High Availability**

pgsql

CopyEdit

+-----------+

| Client |

+-----------+

|

v

+-----------+

| Sentinel |

+-----------+

/ | \

v v v

+-----------+ +-----------+ +-----------+

| Master | | Replica 1 | | Replica 2 |

| Valkey | +-----------+ +-----------+

+-----------+

➡️ **Description**: **Sentinel monitors Valkey nodes** for failover, reconfiguration, and service discovery.

**1.4. Cluster Mode Architecture**

pgsql

CopyEdit

+-----------+

| Client |

+-----------+

|

v

+-----------------+

| Valkey Cluster |

+-----------------+

/ | \

v v v

+-------+ +-------+ +-------+

| Node1 | | Node2 | | Node3 |

+-------+ +-------+ +-------+

➡️ **Description**: **Sharding data across multiple nodes** for horizontal scalability and fault tolerance.

⚙️ **2. Deployment Guides**

**2.1. Standalone Deployment**

**Install from package manager**

bash

CopyEdit

# For Debian/Ubuntu

sudo apt update

sudo apt install valkey-server

# Start service

sudo systemctl start valkey-server

sudo systemctl enable valkey-server

**2.2. Docker Deployment**

**Using Docker Hub image**

bash

CopyEdit

docker run -d --name valkey -p 6379:6379 valkey/valkey

➡️ **Data persistence example**:

bash

CopyEdit

docker run -d --name valkey \

-p 6379:6379 \

-v /mydata:/data \

valkey/valkey valkey-server --save 60 1000 --appendonly yes

**2.3. Cluster Deployment (manual basic)**

1. **Start multiple Valkey instances with cluster enabled**:

bash

CopyEdit

valkey-server --port 7000 --cluster-enabled yes --cluster-config-file nodes-7000.conf --cluster-node-timeout 5000 --appendonly yes

1. Repeat for ports 7001, 7002, etc.
2. **Create cluster**:

bash

CopyEdit

valkey-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \

--cluster-replicas 1

➡️ For production use, deploy across **multiple nodes** with replicas for HA.

**2.4. Kubernetes Deployment**

**Using Helm (recommended)**

bash

CopyEdit

helm repo add valkey https://charts.bitnami.com/bitnami

helm repo update

helm install my-valkey valkey/valkey

➡️ Configure values.yaml for:

* Persistence settings
* Cluster mode
* Password authentication

Alternatively, deploy via Kubernetes manifests defining StatefulSets for persistence.

🔍 **3. Valkey vs Redis Comparison Table**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Valkey** | **Redis** |
| **Origin** | Forked from Redis under Linux Foundation (2024) | Originally developed by Salvatore Sanfilippo (antirez), now Redis Inc. |
| **Governance** | Community-driven under **Linux Foundation** | Redis Inc. governed with Open Core licensing for enterprise features |
| **License** | BSD-3-Clause (same as Redis OSS) | Redis OSS: BSD-3-Clause, Redis Enterprise features proprietary |
| **Compatibility** | Fully compatible with Redis commands and clients | Native |
| **Performance** | Same as Redis OSS (forked from v7.2) with future community improvements | Highly optimised with continuous development |
| **Modules** | Supports Redis modules compatibility | Native modules support, commercial modules in Redis Enterprise |
| **Features roadmap** | Community decides new features collaboratively | Redis Inc. sets roadmap for Redis OSS and Enterprise |
| **Deployment options** | Standalone, Cluster, Sentinel, Kubernetes | Standalone, Cluster, Sentinel, Kubernetes |
| **Commercial support** | Community and third-party | Official Redis Inc. support with Enterprise features |
| **Use Cases** | Same as Redis: caching, queues, pub/sub, geospatial, real-time analytics | Same |

ℹ️ **Additional Notes**

* **Why Valkey was created:** Due to licensing and governance concerns, aiming to keep Redis technology fully open under a **neutral foundation**.
* **Migration:** Compatible with existing Redis setups. Migration involves pointing clients to Valkey instances with minimal or no code change.

**Customer Benefits and Results with Valkey**

🎯 **1. Key Customer Benefits**

|  |  |  |
| --- | --- | --- |
| **Benefit** | **Description** | **Impact/Result** |
| **High Performance** | In-memory data store with sub-millisecond latency. | Faster application response times, improved user experience, supports high concurrency. |
| **Open Governance & Community-driven** | Under Linux Foundation, ensuring transparent development and vendor neutrality. | Reduced risk of vendor lock-in, long-term product stability and trust. |
| **Cost Efficiency** | Open-source with BSD-3 license, no licensing costs for core features. | Significant savings vs. proprietary caching/messaging solutions. |
| **Flexible Deployment Options** | Supports standalone, replication, sentinel, cluster, Docker, and Kubernetes deployments. | Fits various customer architectures, on-premise or cloud-native. |
| **Scalable Architecture** | Cluster mode allows horizontal scaling across nodes. | Handles growing workloads without performance degradation. |
| **Rich Data Structures** | Beyond key-value: lists, sets, sorted sets, hashes, streams, geospatial data. | Enables building complex applications like leaderboards, chat apps, real-time analytics with minimal development effort. |
| **Compatibility with Redis** | Fully compatible with existing Redis commands and clients. | Easy migration from Redis with minimal or no code changes. |
| **High Availability** | Supports replication and Sentinel-based failover for resilient services. | Minimizes downtime, ensuring business continuity. |
| **Secure** | TLS encryption, ACL-based authentication. | Meets enterprise security standards for sensitive applications. |
| **Rapid Development Enablement** | Simple API design and wide language client support. | Faster time-to-market for features leveraging Valkey as a backend. |

📈 **2. Typical Customer Results**

✔️ **Performance Improvements**

* Achieve **~1 million requests per second per node** with sub-millisecond latency, depending on infrastructure.
* Reduce database read load by **60-90%** when used as a caching layer.

✔️ **Cost Reduction**

* Eliminates licensing costs associated with commercial caching or database acceleration solutions.
* Reduces cloud database read IOPS costs when integrated as a cache fronting RDBMS.

✔️ **Scalability Gains**

* Seamless horizontal scaling in cluster mode allows businesses to handle **traffic spikes without redesigning applications**.
* Supports auto-sharding, distributing data across nodes efficiently.

✔️ **Increased Reliability**

* Sentinel and cluster failover configurations reduce **RTO (Recovery Time Objective) to seconds**, ensuring high availability SLAs.

✔️ **Developer Productivity**

* Native support for rich data structures simplifies development of:
  + Leaderboards (using sorted sets)
  + Real-time chat systems (using pub/sub or streams)
  + Rate limiting (using atomic operations)
  + Session management (using key expiry)
* Developers can implement **complex use cases without additional backend systems or libraries**.

✔️ **Future-proofed Open Source Strategy**

* Confidence in an open governance model avoiding abrupt license changes, ensuring product sustainability and compliance with open-source strategy policies.

💡 **3. Example Customer Scenarios**

|  |  |  |
| --- | --- | --- |
| **Industry** | **Use Case** | **Result** |
| **E-commerce** | Product page caching and session management | Reduced page load times by ~80%, improved conversion rates. |
| **Gaming** | Real-time leaderboard and matchmaking system | Enabled low-latency updates for millions of players concurrently. |
| **Financial Services** | Risk calculations and rate limiting APIs | Achieved sub-millisecond access times for critical risk models with minimal infrastructure. |
| **Telecom** | Pub/Sub-based notification systems | Delivered real-time messaging to millions of subscribers reliably. |

🔑 **Summary**

✔️ Valkey delivers **high performance, scalability, cost efficiency, and flexibility** for customers needing real-time data processing, caching, and messaging systems with the confidence of an **open, community-governed solution**.

💰 **Valkey Subscription Costs**

**1. Core Valkey Costs**

✅ **Valkey is completely free and open source.**

* **License:** BSD-3-Clause
* **Cost:** $0 for software usage
* **Included:** All core features, clustering, replication, persistence, and compatibility with Redis commands.

**2. Why No Subscription Costs?**

Valkey is governed under the **Linux Foundation** as an open community project, ensuring:

* **No commercial licensing fees** for software usage.
* **No feature lock behind subscriptions** – unlike proprietary forks.
* **Freedom to deploy** on any infrastructure (on-premises, private cloud, public cloud) without vendor restrictions.

**3. Potential Associated Costs**

While Valkey itself has no direct subscription cost, customers should consider:

|  |  |  |
| --- | --- | --- |
| **Cost Type** | **Description** | **Notes** |
| **Infrastructure** | Compute, memory, storage for running Valkey instances. | Depends on your deployment (VMs, bare metal, containers, Kubernetes nodes). |
| **Managed Services** | If using a third-party Valkey/Redis-compatible managed service provider. | Costs vary by provider, instance size, and SLAs. |
| **Support Contracts** | Optional contracts with cloud vendors or third-party support firms. | E.g. proactive monitoring, production support, consultancy. |
| **Operational Overhead** | Maintenance, upgrades, monitoring, backups, and scaling. | Often managed internally by DevOps/SRE teams or automated via Kubernetes/Helm deployments. |

**4. Comparison with Redis Subscription Costs**

|  |  |  |
| --- | --- | --- |
| **Product** | **Subscription Cost** | **Notes** |
| **Valkey** | $0 | Fully open source under BSD-3, community-supported, no feature gating. |
| **Redis OSS** | $0 | Free for open source Redis features only. |
| **Redis Enterprise** | Varies by subscription tier | Costs depend on RAM usage, cloud provider, features like Active-Active geo-replication, proprietary modules, 24x7 support. Starts from **$300-500+/month** for small production deployments, scaling to **thousands USD/month** for high availability multi-region clusters. |

**5. Optional Commercial Support Options for Valkey**

Currently, Valkey does not have an official commercial support entity like Redis Inc. However:

* Organizations may engage **third-party database support companies** for enterprise SLAs.
* Some cloud vendors may offer Valkey-compatible managed services with monthly costs covering hosting and support.

📝 **Summary**

✔️ **Valkey itself has no subscription cost**, enabling organisations to deploy an enterprise-grade in-memory data store without licensing fees.  
✔️ **Total cost of ownership (TCO)** depends only on **infrastructure, operational support, and any optional third-party services**, making Valkey a highly cost-efficient alternative to commercial caching and data store solutions.

📘 **Valkey User Guide**

**1. Introduction**

**Valkey** is an open-source, in-memory key-value data store, forked from Redis under the Linux Foundation. It is used for:

* Caching
* Real-time analytics
* Message brokering (Pub/Sub)
* Queues and rate limiting

**2. Installation**

**2.1. Installing on Linux (Debian/Ubuntu)**

bash

CopyEdit

sudo apt update

sudo apt install valkey-server

**2.2. Starting Valkey**

bash

CopyEdit

sudo systemctl start valkey-server

sudo systemctl enable valkey-server

➡️ Confirm it is running:

bash

CopyEdit

valkey-cli ping

# Output: PONG

**2.3. Docker Deployment**

bash

CopyEdit

docker run -d --name valkey -p 6379:6379 valkey/valkey

**2.4. Kubernetes Deployment via Helm**

bash

CopyEdit

helm repo add valkey https://charts.bitnami.com/bitnami

helm repo update

helm install my-valkey valkey/valkey

➡️ Modify values.yaml for persistence, authentication, and cluster settings as needed.

**3. Connecting to Valkey**

**Using CLI**

bash

CopyEdit

valkey-cli -h <host> -p <port>

➡️ For password-protected instances:

bash

CopyEdit

valkey-cli -h <host> -p <port> -a <password>

**Using Application Clients**

Valkey is compatible with **all Redis clients** in multiple languages. Example with Python:

python

CopyEdit

import redis

r = redis.StrictRedis(host='localhost', port=6379, db=0)

r.set('key', 'value')

print(r.get('key'))

**4. Basic Commands**

|  |  |
| --- | --- |
| **Command** | **Description** |
| SET key value | Set a key to hold the string value. |
| GET key | Get the value of a key. |
| DEL key | Delete a key. |
| EXPIRE key seconds | Set a timeout on a key. |
| INCR key | Increment integer value of a key by one. |
| LPUSH list value | Push value onto a list. |
| LRANGE list 0 -1 | Get all elements from a list. |
| SADD set member | Add member to a set. |
| SMEMBERS set | Get all members of a set. |
| ZADD zset score member | Add member with score to sorted set. |
| ZRANGE zset 0 -1 WITHSCORES | Retrieve sorted set members with scores. |
| PUBLISH channel message | Publish message to channel. |
| SUBSCRIBE channel | Subscribe to a channel. |

**5. Persistence**

Valkey supports:

✅ **RDB snapshots** – saves data at intervals.  
✅ **AOF (Append Only File)** – logs every write for durability.

**Enable AOF**

In valkey.conf:

nginx

CopyEdit

appendonly yes

Reload config:

bash

CopyEdit

valkey-cli CONFIG REWRITE

**6. High Availability**

**6.1. Replication (Master-Replica)**

In replica server config:

php-template

CopyEdit

replicaof <master-ip> <master-port>

➡️ The replica will automatically sync data from the master.

**6.2. Sentinel for Failover**

Valkey Sentinel monitors master and replica instances for **automatic failover**.

Start Sentinel with:

bash

CopyEdit

valkey-sentinel /etc/valkey/sentinel.conf

Define monitored master in sentinel.conf:

php-template

CopyEdit

sentinel monitor mymaster <master-ip> <master-port> 2

**7. Cluster Mode**

Valkey supports clustering for **sharding data across nodes**.

**7.1. Cluster Setup (Basic)**

1. Start multiple nodes with cluster mode enabled:

bash

CopyEdit

valkey-server --port 7000 --cluster-enabled yes --cluster-config-file nodes-7000.conf --appendonly yes

1. Repeat for ports 7001, 7002, etc.
2. Create cluster:

bash

CopyEdit

valkey-cli --cluster create \

127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \

--cluster-replicas 1

➡️ Production clusters should be spread across multiple servers with proper replication.

**8. Security**

✅ **Authentication** – set requirepass in valkey.conf  
✅ **ACL** – supports role-based access control  
✅ **TLS Encryption** – configure for secure data transmission

**9. Backup and Restore**

**Backup**

* **RDB snapshot:** Copy the dump file (default: dump.rdb) to backup storage.

**Restore**

* Place dump.rdb in the Valkey data directory and restart the server.

**10. Monitoring**

Use:

* **INFO** command for stats
* **MONITOR** for real-time command inspection
* Integration with **Prometheus exporters** for detailed metrics.

**11. Best Practices**

✔️ Use AOF with fsync for durability-critical data.  
✔️ Set reasonable TTL for cache keys to avoid memory exhaustion.  
✔️ Monitor memory usage and set maxmemory with eviction policy.  
✔️ For production, always deploy with replication or Sentinel/Cluster for HA.  
✔️ Secure with authentication and TLS in all environments.

**12. Additional Resources**

* [Official Valkey Website](https://valkey.io/)
* [Valkey GitHub Repository](https://github.com/valkey-io/valkey)
* Redis Command Reference (compatible)
* Linux Foundation Valkey Project

✨ **Summary**

Valkey offers a **fast, scalable, and open-source in-memory data store**, ideal for caching, real-time processing, and low-latency applications with **no vendor lock-in**.

🔒 **Valkey Security Policies**

**1. Overview**

Valkey, as an in-memory data store, is highly performant but must be secured properly to:

✅ Prevent unauthorized data access  
✅ Protect sensitive cached or transient data  
✅ Maintain availability against attacks or misconfigurations

**2. Authentication Policy**

**2.1. Enable Password Authentication**

Set a strong password in valkey.conf:

php-template

CopyEdit

requirepass <strong\_password\_here>

➡️ Recommended:

* Use **complex passwords** (minimum 16 characters, with symbols and numbers).
* Rotate passwords **regularly** as per organisational policy (e.g., every 90 days).
* Store passwords securely in a secrets manager (Vault, AWS Secrets Manager, Azure Key Vault).

**2.2. Use ACLs for Fine-grained Access**

If multiple applications or users connect to Valkey:

* Configure **user roles with least privilege** using ACLs.
* Example ACL setup:

bash

CopyEdit

ACL SETUSER readonly on >password ~\* +get

ACL SETUSER readwrite on >password ~\* +get +set +del

**3. Network Security Policy**

**3.1. Bind to Specific Interfaces**

By default, Valkey binds to 127.0.0.1. For production:

* Bind only to trusted interfaces:

nginx

CopyEdit

bind 10.0.0.1

* Avoid bind 0.0.0.0 unless protected by strict network-level controls.

**3.2. Firewall Rules**

✅ Only allow inbound traffic to Valkey ports (default 6379) from:

* Application servers that require access
* Monitoring systems (if necessary)
* Management hosts (jump servers or bastions)

**4. Encryption Policy**

**4.1. Enable TLS**

Valkey supports TLS to encrypt data in transit.

* Generate server certificates and keys.
* Update valkey.conf:

swift

CopyEdit

tls-port 6379

port 0

tls-cert-file /etc/valkey/certs/server.crt

tls-key-file /etc/valkey/certs/server.key

tls-ca-cert-file /etc/valkey/certs/ca.crt

➡️ Force clients to connect via TLS for confidentiality and integrity.

**5. Data Protection Policy**

**5.1. Persistence File Security**

* RDB and AOF files can contain sensitive data.
* Ensure:

✅ File system permissions restrict access only to Valkey process owner.  
✅ Backup these files securely with encryption at rest (e.g., S3 bucket with SSE).  
✅ For extremely sensitive environments, consider disabling persistence if data can be rehydrated externally.

**6. Availability and DoS Protection**

Valkey is vulnerable to misuse that exhausts memory.

**6.1. Configure Max Memory and Policies**

In valkey.conf:

php-template

CopyEdit

maxmemory <value>

maxmemory-policy allkeys-lru

➡️ Prevents OOM errors that crash the instance.

**6.2. Monitor Connections**

* Use maxclients setting to limit simultaneous connections:

nginx

CopyEdit

maxclients 10000

* Helps mitigate connection flooding attacks.

**7. Logging and Monitoring Policy**

✅ Enable logging of access and errors.  
✅ Integrate logs with SIEM for security monitoring.  
✅ Monitor:

* Unexpected command patterns
* High connection rates
* Frequent authentication failures

**8. Backup Security Policy**

✔️ Store backups in **encrypted storage**.  
✔️ Implement **access control** on backup locations.  
✔️ Test restore procedures periodically to ensure data recoverability.

**9. Update and Patch Policy**

✔️ Regularly check for Valkey updates and security advisories.  
✔️ Apply patches in non-production first, then schedule production upgrades.  
✔️ Subscribe to Valkey GitHub and community channels for CVE alerts.

**10. Compliance Policy**

Ensure Valkey configurations and operations comply with organisational standards and applicable frameworks such as:

* **ISO 27001**
* **SOC 2**
* **PCI-DSS** (if caching cardholder data, validate compliance implications)
* **HIPAA** (for PHI data, enforce encryption and strict access controls)

**11. Incident Response Policy**

In case of suspected breach or vulnerability:

1. **Isolate** affected Valkey instances from networks if needed.
2. **Rotate passwords and revoke compromised credentials** immediately.
3. **Review logs** for unauthorized access patterns.
4. **Restore data** from last known safe backups if integrity is compromised.
5. **Patch vulnerabilities** before bringing back to production.

🔑 **Summary**

|  |  |
| --- | --- |
| **Policy Area** | **Key Action** |
| Authentication | Enable requirepass and configure ACLs. |
| Encryption | Enforce TLS for all connections. |
| Network | Bind to trusted interfaces; restrict via firewall. |
| Data | Protect persistence files and backups. |
| Availability | Set max memory, eviction policies, and maxclients. |
| Monitoring | Enable logging, integrate with SIEM, monitor usage. |
| Patching | Regular updates for security vulnerabilities. |
| Compliance | Validate against applicable standards. |

⭐ **Golden Rules – Definition and Application**

**1. What are Golden Rules?**

✔️ **Golden rules are fundamental, non-negotiable principles** that guide behaviours, decisions, and processes to ensure consistency, quality, and alignment with organisational or technical standards.

They are:

* **Simple to understand**
* **Widely applicable**
* **Critical for success, safety, or compliance**

**2. Why are Golden Rules Important?**

✅ Ensure **standardisation and reliability**  
✅ Reduce risk and human error  
✅ Enable quick decision-making under pressure  
✅ Build a culture of accountability and excellence

**3. Types of Golden Rules**

Here are examples across different contexts:

|  |  |
| --- | --- |
| **Context** | **Golden Rule Examples** |
| **Engineering** | - Always validate user inputs.  - Never deploy to production without peer review.  - Automate what you do more than twice. |
| **Security** | - Least privilege: give only necessary access.  - Assume breach: design for compromise.  - Encrypt data in transit and at rest. |
| **Operations / SRE** | - Never make changes without rollback plans.  - Monitor everything that can fail.  - Test in staging before production. |
| **Personal conduct / Leadership** | - Treat others as you want to be treated.  - Communicate clearly and honestly.  - Take ownership of mistakes and solutions. |

**4. Golden Rules for Technical Excellence (Examples)**

Here is a **set of golden rules** you can adapt for your platform and DevOps teams:

1. **Keep it Simple.**  
   Complexity is the enemy of reliability.
2. **Fail Fast and Loud.**  
   Detect failures early to avoid cascading issues.
3. **Automate Everything Possible.**  
   Reduce manual errors and increase efficiency.
4. **Version Control All Artifacts.**  
   Infrastructure as code, configurations, scripts, and documents.
5. **Test Before You Trust.**  
   Always validate in staging or sandbox environments.
6. **Secure by Default.**  
   Security is not optional or an afterthought.
7. **Design for Scale and Failure.**  
   Assume growth and unexpected outages in your architecture.
8. **Document as You Build.**  
   If it’s not documented, it does not exist for the team.
9. **Review, Learn, Improve.**  
   Blameless post-mortems to enhance systems and processes.
10. **Communicate Clearly.**  
    Over-communicate rather than under-communicate during incidents or changes.

**5. The Ultimate Golden Rule**

**“Treat others as you want to be treated.”**

This applies in all contexts – technical or personal – to build **trust, respect, and high-performing teams**.

💡 **Summary**

✔️ **Golden rules** are your **non-negotiable guiding principles**.  
✔️ They **simplify decisions, reduce risks, and strengthen culture**.  
✔️ Define them clearly within your team or organisation, communicate them regularly, and **embed them into onboarding, reviews, and day-to-day operations**.