

Redis: Guia de Apoio ao Estudante

Material de Apoio Teórico e Prático



Índice

1. [Introdução Aprofundada](#)
 2. [Arquitetura Detalhada](#)
 3. [Estruturas de Dados Avançadas](#)
 4. [Comandos Fundamentais](#)
 5. [Performance e Otimização](#)
 6. [Persistência e Durabilidade](#)
 7. [Replicação e Alta Disponibilidade](#)
 8. [Clustering e Escalabilidade](#)
 9. [Pub/Sub e Messaging](#)
 10. [Segurança](#)
 11. [Monitoramento e Debugging](#)
 12. [Casos de Uso Avançados](#)
 13. [Integração com Linguagens](#)
 14. [Deployment e DevOps](#)
 15. [Troubleshooting](#)
 16. [Recursos Adicionais](#)
-

1. Introdução Aprofundada

1.1 História e Evolução

Redis foi criado em 2009 por **Salvatore Sanfilippo** (antirez) na Itália, inicialmente para resolver problemas de performance em uma aplicação web. O nome significa "**RE**mote **D**ictionary **S**erver".

Timeline importante:

- **2009:** Primeira versão pública
- **2010:** Suporte a estruturas de dados avançadas
- **2012:** Lua scripting
- **2013:** Redis Sentinel (alta disponibilidade)
- **2015:** Redis Cluster (clustering nativo)

- **2018:** Redis Streams
- **2020:** Redis 6 com ACLs e threading parcial

1.2 Filosofia e Design

Redis segue alguns princípios fundamentais:

Simplicidade: Interface simples mas poderosa

Simplicidade na sintaxe

SET chave valor

GET chave

Performance: Otimizado para velocidade máxima

- Dados em memória RAM
- Estruturas de dados eficientes em C
- Single-threaded event loop

Versatilidade: Múltiplas estruturas de dados nativas

- Não apenas key-value simples
- Estruturas complexas com operações atômicas

1.3 Quando Usar Redis

✓ **Redis é ideal para:**

- Cache de alta performance
- Sessões de usuário
- Contadores e métricas
- Filas de mensagens
- Pub/Sub em tempo real
- Leaderboards e rankings
- Rate limiting
- Dados temporários

✗ **Redis NÃO é ideal para:**

- Dados relacionais complexos
 - Queries SQL complexas
 - Dados que excedem a RAM disponível
 - Transações ACID complexas
 - Relatórios analíticos pesados
-

2. Arquitetura Detalhada

2.1 Arquitetura Interna

Arquitetura detalhada: [clique aqui](#)

Fluxo de Operações e Dados: [clique aqui](#)

2.2 Single-Threaded Model

Redis usa um **modelo single-threaded** para operações de dados:

Vantagens:

- Não há overhead de context switching
- Não precisa de locks/mutexes
- Operações são naturalmente atômicas
- Mais simples de debugar

Threading Parcial (Redis 6+):

Main Thread: [Command Processing] [Data Operations]

I/O Threads: [Network Read/Write]

BG Threads: [Persistence] [Expiration] [Eviction]

2.3 Protocolo RESP

Redis usa o **RESP (Redis Serialization Protocol)**:

Comando: `SET nome "João"`

Cliente → Servidor:

```
*3\r\n
$3\r\n
SET\r\n
$4\r\n
nome\r\n
$4\r\n
João\r\n
```

Servidor → Cliente:

```
+OK\r\n
```

Tipos de dados RESP:

- + String simples
 - - Erro
 - : Integer
 - \$ Bulk string
 - * Array
-

3. Estruturas de Dados Avançadas

3.1 Strings - Além do Básico

Strings no Redis podem armazenar:

- Texto (UTF-8)
- Números (integers, floats)
- Dados binários (imagens, serialized objects)

Operações avançadas:

```
# Operações bitwise
SETBIT usuario:1:permissions 0 1    # Permissão 0 = true
GETBIT usuario:1:permissions 0

# Operações em ranges
SETRANGE mensagem 6 "mundo"
GETRANGE mensagem 0 10

# Append
APPEND log ":INFO - Usuario conectado"

# Múltiplas operações
MSET user:1:nome "João" user:1:idade 25 user:1:ativo true
MGET user:1:nome user:1:idade
```

Casos de uso especiais:

- **Bitmaps:** Tracking de usuários ativos, features flags
- **Contadores distribuídos:** INCR/DECR com garantia atômica
- **Cache de objetos:** Serialização JSON/Protocol Buffers

3.2 Lists - Estruturas Versáteis

Lists são **listas duplamente ligadas**:

```
# Operações em ambas as extremidades
LPUSH tarefas "tarefa1"      # Adiciona à esquerda
RPUSH tarefas "tarefa2"      # Adiciona à direita
LPOP tarefas                 # Remove da esquerda
RPOP tarefas                 # Remove da direita

# Operações de bloco (blocking)
BLPOP tarefas 10             # Aguarda até 10s por elemento
BRPOP tarefas 5

# Operações por índice
LINDEX tarefas 0             # Primeiro elemento
LSET tarefas 0 "nova_tarefa"

# Trimming
LTRIM tarefas 0 99           # Mantém apenas 100 elementos
```

Padrões importantes:

- **Queue (FIFO):** LPUSH + RPOP
- **Stack (LIFO):** LPUSH + LPOP
- **Producer/Consumer:** LPUSH + BRPOP
- **Circular Buffer:** LPUSH + LTRIM

3.3 Sets - Operações de Conjunto

```
# Operações básicas
SADD linguagens "Python" "JavaScript" "Go"
SISMEMBER linguagens "Python"      # 1 (true)
SCARD linguagens                    # 3 (cardinalidade)

# Operações entre conjuntos
SADD frontend "JavaScript" "TypeScript" "HTML" "CSS"
SADD backend "Python" "Go" "Redis" "PostgreSQL"

SINTER frontend backend             # Interseção: vazio
SUNION frontend backend            # União: todos elementos
SDIFF backend frontend             # Diferença: backend - frontend

# Operações destrutivas
SPOP linguagens                    # Remove elemento aleatório
```

```
SRANDMEMBER linguagens 2          # 2 elementos aleatórios (sem  
remove)
```

Casos de uso:

- **Tags de artigos:** Relacionamentos many-to-many
- **Seguidores/Seguindo:** Relacionamentos sociais
- **Whitelist/Blacklist:** Controle de acesso
- **Deduplicação:** Elementos únicos

3.4 Hashes - Objetos Estruturados

```
# Operações básicas  
HSET produto:123 nome "Notebook Dell" preco 2500.00 categoria  
"informatica"  
HGET produto:123 nome  
HGETALL produto:123  
  
# Operações múltiplas  
HMSET usuario:456 nome "Maria" email "maria@email.com" idade 28  
ativo true  
HMGET usuario:456 nome email  
  
# Operações numéricas  
HINCRBY produto:123 estoque -1          # Decrementa estoque  
HINCRBYFLOAT produto:123 preco -100.0 # Desconto  
  
# Campos condicionais  
HSETNX usuario:456 criado_em "2024-01-15" # Set apenas se não  
existir  
  
# Iteração  
HSCAN usuario:456 0 MATCH "endereco*" # Busca campos por padrão
```

Vantagens dos Hashes:

- **Eficiência de memória:** Mais eficiente que múltiplas keys
- **Operações atômicas:** Múltiplos campos em uma operação
- **Estrutura natural:** Representa objetos facilmente

3.5 Sorted Sets - Rankings e Ordenação

```
# Adicionando com scores
```

```

ZADD ranking_jogadores 1500 "jogador1" 1750 "jogador2" 1200
"jogador3"

# Consultas por ranking
ZRANGE ranking_jogadores 0 -1 WITHSCORES      # Ordem crescente
ZREVRANGE ranking_jogadores 0 -1 WITHSCORES    # Ordem decrescente

# Consultas por score
ZRANGEBYSCORE ranking_jogadores 1400 1600
ZCOUNT ranking_jogadores 1500 2000           # Conta elementos no
range

# Operações em scores
ZINCRBY ranking_jogadores 50 "jogador1"       # Adiciona 50 pontos
ZSCORE ranking_jogadores "jogador1"          # Retorna score atual

# Ranking de elemento
ZRANK ranking_jogadores "jogador2"            # Posição (0-based)
ZREVRANK ranking_jogadores "jogador2"        # Posição reversa

```

Casos de uso avançados:

- **Leaderboards com timestamps:** Score = timestamp
- **Auto-complete:** Score = frequência de uso
- **Rate limiting:** Score = timestamp, remove por range
- **Prioridade de tarefas:** Score = prioridade

3.6 Streams - Event Sourcing

Redis Streams (v5.0+) para logs de eventos temporais:

```

# Adicionando eventos
XADD eventos * usuario "joao" acao "login" timestamp 1642284000
XADD eventos * usuario "maria" acao "logout" timestamp 1642284060

# Lendo eventos
XREAD STREAMS eventos 0                      # Todos desde início
XREAD COUNT 10 STREAMS eventos $             # Próximos 10

# Consumer Groups
XGROUP CREATE eventos grupo1 0                # Cria grupo de
consumidores
XREADGROUP GROUP grupo1 consumidor1 STREAMS eventos >

```

```
# Informações do stream
XINFO STREAM eventos
XLEN eventos                                     # Número de entradas
```

Casos de uso:

- **Event sourcing:** Log de eventos de domínio
- **Activity feeds:** Timeline de atividades
- **IoT data:** Séries temporais de sensores
- **Audit logs:** Rastreamento de ações

4. Comandos Fundamentais

4.1 Comandos de Keys

```
# Informações sobre keys
TYPE chave                                     # Tipo da estrutura
TTL chave                                     # Time to live (-1 = sem expiração)
PTTL chave                                    # TTL em milissegundos
EXISTS chave1 chave2                         # Verifica existência (retorna count)

# Expiração
EXPIRE chave 3600                             # Expira em 1 hora
EXPIREAT chave 1642284000                     # Expira em timestamp
PERSIST chave                                # Remove expiração

# Renomeação e movimentação
RENAME chave nova_chave                      # Renomeia (sobrescreve)
RENAMENX chave nova_chave                    # Renomeia apenas se destino não
existe
MOVE chave 1                                 # Move para database 1

# Busca e iteração
KEYS pattern                                 # ⚠ Evitar em produção!
SCAN cursor [MATCH pattern] [COUNT count]  # Iteração segura
```

4.2 Comandos de Database

```
# Databases
```



```
SELECT 0                # Seleciona database 0 (0-15 por
padrão)                  # Limpa database atual
FLUSHDB                  # Limpa todas databases
FLUSHALL

# Informações
DBSIZE                   # Número de keys na database atual
INFO                     # Informações completas do servidor
INFO memory              # Informações de memória
INFO stats               # Estatísticas de uso
```

4.3 Comandos de Transação

```
# Transação simples
MULTI                    # Inicia transação
SET conta:1 1000
SET conta:2 500
EXEC                     # Executa todas

# Com verificação condicional
WATCH conta:1            # Observa mudanças
MULTI
DECRBY conta:1 100
INCRBY conta:2 100
EXEC                     # Executa apenas se conta:1 não mudou

# Cancelar transação
DISCARD                  # Cancela comandos em fila
```

5. Performance e Otimização

5.1 Benchmarking

```
# Benchmark básico
redis-benchmark -h localhost -p 6379 -n 100000 -c 50

# Teste específico
redis-benchmark -t set,get -n 100000 -q

# Com pipeline
```

```
redis-benchmark -h localhost -p 6379 -n 100000 -P 10

# Resultados típicos (hardware moderno):
# SET: 80,000-120,000 ops/sec
# GET: 100,000-150,000 ops/sec
# Com pipeline: 500,000+ ops/sec
```

5.2 Pipeline para Performance

Sem Pipeline:

```
import redis
r = redis.Redis()

# Lento: 1000 round-trips
for i in range(1000):
    r.set(f"key:{i}", f"value:{i}")
```

Com Pipeline:

```
# Rápido: 1 round-trip
pipe = r.pipeline()
for i in range(1000):
    pipe.set(f"key:{i}", f"value:{i}")
pipe.execute()
```

5.3 Otimização de Memória

Configurações importantes:

```
# redis.conf
hash-max-ziplist-entries 512      # Otimização para hashes pequenos
hash-max-ziplist-value 64
list-max-ziplist-size -2         # Otimização para listas
set-max-intset-entries 512       # Sets de integers pequenos
```

Estratégias de otimização:

```
# Use hashes para objetos relacionados
```

```
# Em vez de:
SET user:1:name "João"
SET user:1:email "joao@email.com"
SET user:1:age 25

# Use:
HMSET user:1 name "João" email "joao@email.com" age 25

# Monitore uso de memória
MEMORY USAGE user:1
DEBUG OBJECT user:1
```

5.4 Políticas de Eviction

```
# redis.conf
maxmemory 256mb
maxmemory-policy allkeys-lru

# Políticas disponíveis:
# noeviction          - Retorna erro quando memória cheia
# allkeys-lru         - Remove keys menos recentemente usadas
# volatile-lru        - Remove keys com TTL menos usadas
# allkeys-random      - Remove keys aleatórias
# volatile-random     - Remove keys com TTL aleatórias
# volatile-ttl        - Remove keys com menor TTL
# allkeys-lfu         - Remove keys menos frequentemente usadas (Redis 4.0+)
# volatile-lfu        - Remove keys com TTL menos frequentes
```

6. Persistência e Durabilidade

6.1 RDB (Redis Database Backup)

Configuração:






```
# redis.conf
save 900 1      # Snapshot se pelo menos 1 key mudou em 15min
save 300 10     # Snapshot se pelo menos 10 keys mudaram em 5min
save 60 10000   # Snapshot se pelo menos 10000 keys mudaram em 1min
```

```
dbfilename dump.rdb
dir /var/lib/redis/
```

Comandos manuais:

```
SAVE          # Snapshot síncrono (bloqueia servidor)
BGSAVE        # Snapshot assíncrono (em background)
LASTSAVE      # Timestamp do último snapshot
```

Características do RDB:

-  Arquivo compacto
-  Recuperação rápida
-  Ideal para backups
-  Pode perder dados entre snapshots
-  Fork pode ser custoso

6.2 AOF (Append Only File)

Configuração:

```
# redis.conf
appendonly yes
appendfilename "appendonly.aof"

# Política de sync
appendfsync everysec    # Recomendado
# appendfsync always    # Mais seguro, mais lento
# appendfsync no        # Menos seguro, mais rápido
```






Reescrita automática:

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

Comandos manuais:

```
BGREWRITEAOF    # Reescreve AOF em background
```

Características do AOF:

-  Maior durabilidade
-  Log human-readable
-  Recuperação automática de corrupção
-  Arquivo maior
-  Recuperação mais lenta

6.3 Estratégia Híbrida

Configuração recomendada para produção:

```
# Ambos habilitados
save 900 1
appendonly yes
appendfsync everysec

# RDB para backup
# AOF para durabilidade
```

Processo de recuperação:

1. Redis procura por AOF primeiro
 2. Se AOF não existe, usa RDB
 3. Se ambos corrompidos, inicia vazio
-

7. Replicação e Alta Disponibilidade

7.1 Master-Slave Replication

Configuração do Slave:

```
# redis.conf do slave
replicaof 192.168.1.100 6379      # IP e porta do master
masterauth senha_do_master      # Se master tem senha

# Configurações adicionais
replica-read-only yes           # Slave apenas leitura
replica-priority 100            # Prioridade para promoção
```

Comandos de replicação:

```
# No master
INFO replication                # Status da replicação

# No slave
SLAVEOF 192.168.1.100 6379      # Inicia replicação
SLAVEOF NO ONE                  # Para replicação (torna master)
```

7.2 Redis Sentinel

Configuração do Sentinel:

```
# sentinel.conf
port 26379
sentinel monitor mymaster 192.168.1.100 6379 2
sentinel auth-pass mymaster senha
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 60000
sentinel parallel-syncs mymaster 1
```

Características do Sentinel:

- Monitora masters e slaves
- Detecta falhas automaticamente
- Executa failover automático
- Notifica aplicações sobre mudanças
- Mínimo 3 instâncias para quorum

Comandos Sentinel:

```
# Conectar ao sentinel (porta 26379)
SENTINEL masters                # Lista masters monitorados
SENTINEL slaves mymaster        # Lista slaves do master
SENTINEL get-master-addr-by-name mymaster
SENTINEL failover mymaster      # Força failover manual
```

7.3 Alta Disponibilidade com Aplicações

Exemplo em Python:

```
import redis.sentinel
```

```
# Configuração dos sentinels
sentinels = [('192.168.1.10', 26379),
              ('192.168.1.11', 26379),
              ('192.168.1.12', 26379)]

sentinel = redis.sentinel.Sentinel(sentinels)

# Descobre master atual
master = sentinel.master_for('mymaster', socket_timeout=0.1)
slave = sentinel.slave_for('mymaster', socket_timeout=0.1)

# Escreve no master, lê do slave
master.set('chave', 'valor')
valor = slave.get('chave')
```

8. Clustering e Escalabilidade

8.1 Redis Cluster

Características:

- Sharding automático de dados
- 16,384 hash slots
- Múltiplos masters
- Replicação automática
- Sem proxy necessário

Configuração de Cluster:

```
# redis.conf para cada nó
cluster-enabled yes
cluster-config-file nodes-6379.conf
cluster-node-timeout 15000
cluster-require-full-coverage yes
```

Criando cluster:

```
# Inicia 6 instâncias Redis (3 masters, 3 slaves)
redis-server --port 7000 --cluster-enabled yes --cluster-config-file
nodes.conf
```

```
redis-server --port 7001 --cluster-enabled yes --cluster-config-file
nodes.conf
# ... mais instâncias

# Cria cluster
redis-cli --cluster create \
  127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \
  127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \
  --cluster-replicas 1
```

8.2 Hash Slots e Sharding

Como funciona o sharding:

$\text{CRC16}(\text{key}) \% 16384 = \text{slot}$

Exemplos:

$\text{CRC16}(\text{"user:1"}) \% 16384 = 9842 \rightarrow \text{Node A}$

$\text{CRC16}(\text{"user:2"}) \% 16384 = 5461 \rightarrow \text{Node B}$

$\text{CRC16}(\text{"product:1"}) \% 16384 = 1234 \rightarrow \text{Node C}$

Hash tags para agrupamento:

```
# Keys com mesmo hash tag ficam no mesmo slot
SET {user:1}:profile "..."
SET {user:1}:settings "..."
SET {user:1}:posts "..."
# Todas ficam no mesmo nó
```

8.3 Comandos de Cluster

```
# Informações do cluster
CLUSTER INFO
CLUSTER NODES
```

```
# Gerenciamento de slots
CLUSTER KEYSLOT chave           # Descobre slot da chave
CLUSTER COUNTKEYSINSLOT 9842    # Conta keys no slot
CLUSTER GETKEYSINSLOT 9842 10   # Lista keys no slot
```



```
# Failover manual
CLUSTER FAILOVER                                # No slave que deve virar master
```

8.4 Limitações do Cluster

Comandos não suportados:

- Multi-key operations entre diferentes slots
- Transações multi-key
- Algumas operações de database (SELECT, FLUSHALL)

Workarounds:

```
# Em vez de:
MGET user:1 user:2 user:3                        # Pode falhar se users em slots
diferentes

# Use:
GET {users}:1
GET {users}:2
GET {users}:3                                    # Garantido no mesmo slot
```

9. Pub/Sub e Messaging

9.1 Pub/Sub Básico

Publisher:

```
PUBLISH canal_noticias "Nova versão do Redis lançada!"
PUBLISH sistema:alertas "CPU usage > 90%"
```

Subscriber:

```
SUBSCRIBE canal_noticias sistema:alertas
# Aguarda mensagens...

# Pattern subscription
PSUBSCRIBE sistema:*                        # Todos canais que começam com
"sistema:"
```

9.2 Implementação com Linguagens

Python - Subscriber:

```
import redis

r = redis.Redis()
pubsub = r.pubsub()
pubsub.subscribe('noticias', 'alertas')

for message in pubsub.listen():
    if message['type'] == 'message':
        print(f"Canal: {message['channel']}")
        print(f"Mensagem: {message['data']}")
```

Python - Publisher:

```
import redis
import time

r = redis.Redis()

while True:
    r.publish('alertas', f'Heartbeat: {time.time()}')
    time.sleep(30)
```

9.3 Padrões Avançados

Fan-out messaging:

- # Um publisher, múltiplos subscribers
- # Cada subscriber recebe TODAS as mensagens

Work queue com Lists:

```
# Producer
r.lpush('trabalhos', json.dumps({'task': 'processar_imagem', 'id': 123}))
```

```
# Consumer
while True:
    job = r.brpop('trabalhos', timeout=1)
    if job:
        process_job(json.loads(job[1]))
```

Priority queue com Sorted Sets:

```
# Adiciona trabalho com prioridade
r.zadd('trabalhos_prioritarios', {'job1': 1, 'job2': 5, 'job3': 3})

# Processa por prioridade
job = r.zpopmin('trabalhos_prioritarios')
```

10. Segurança

10.1 Autenticação

Configuração básica:

```
# redis.conf
requirepass minhaSenhaSegura123

# Multiple passwords (Redis 6+)
user default on +@all ~* &* >senhaAntiga
user admin on +@all ~* &* >senhaAdmin
```

ACL (Access Control Lists) - Redis 6+:

```
# Criar usuário com permissões limitadas
ACL SETUSER app_user on +@read ~app:* >senhaApp

# Usuário só pode:
# - Fazer login (on)
# - Executar comandos de leitura (+@read)
# - Acessar keys que começam com "app:" (~app:*)
# - Usar senha "senhaApp" (>senhaApp)
```

```
# Listar usuários
ACL LIST
ACL WHOAMI
```

10.2 Rede e Firewall

Configurações de rede:

```
# redis.conf
bind 127.0.0.1 192.168.1.10      # IPs permitidos
port 6379                       # Porta padrão
protected-mode yes              # Proteção adicional

# Desabilitar comandos perigosos
rename-command FLUSHDB ""
rename-command FLUSHALL ""
rename-command KEYS ""
rename-command CONFIG "CONFIG_b9f3794abc"
```

10.3 TLS/SSL

Configuração TLS:

```
# redis.conf
port 0                          # Desabilita porta
não-criptografada
tls-port 6380                   # Porta TLS
tls-cert-file redis.crt
tls-key-file redis.key
tls-ca-cert-file ca.crt
```

Cliente com TLS:

```
import redis

r = redis.Redis(
    host='redis.exemplo.com',
    port=6380,
    ssl=True,
    ssl_cert_reqs='required',
```

```
ssl_ca_certs='ca.crt'  
)
```

10.4 Monitoramento de Segurança

```
# Log de comandos perigosos  
CONFIG SET slowlog-log-slower-than 0  
CONFIG SET slowlog-max-len 128  
  
# Monitor conexões  
CLIENT LIST  
CLIENT KILL 192.168.1.100:54321  
  
# Auditoria  
ACL LOG # Log de violações ACL
```

11. Monitoramento e Debugging

11.1 Comandos de Monitoramento

Informações básicas:

INFO	# Informações completas
INFO memory	# Uso de memória
INFO stats	# Estatísticas
INFO replication	# Status replicação
INFO clients	# Conexões ativas

Monitoramento em tempo real:

MONITOR	# ⚠ Impacta performance
CLIENT LIST	# Lista conexões
SLOWLOG GET 10	# Últimos 10 comandos lentos

11.2 Métricas Importantes

Memória:

```
INFO memory
# used_memory: Memória usada pelo Redis
# used_memory_rss: Memória física usada
# used_memory_peak: Pico de memória
# maxmemory: Limite configurado
```

Performance:

```
INFO stats
# total_connections_received: Total de conexões
# total_commands_processed: Total de comandos
# instantaneous_ops_per_sec: Operações por segundo
# keyspace_hits: Cache hits
# keyspace_misses: Cache misses
```

Conexões:

```
INFO clients
# connected_clients: Clientes conectados
# client_recent_max_input_buffer: Maior buffer de entrada
# client_recent_max_output_buffer: Maior buffer de saída
```

11.3 Ferramentas de Monitoramento

RedisInsight:

- GUI oficial da Redis Labs
- Visualização de dados em tempo real
- Análise de performance
- Browser de keys
- Profiler de comandos

Prometheus + Grafana:

```
# docker-compose.yml
version: '3'
services:
  redis-exporter:
    image: oliver006/redis_exporter
    environment:
      REDIS_ADDR: "redis://redis:6379"
```

```
ports:
  - "9121:9121"
```

Comandos para scripts de monitoramento:

```
#!/bin/bash
# monitor-redis.sh

# Verifica se Redis está rodando
redis-cli ping > /dev/null 2>&1
if [ $? -ne 0 ]; then
    echo "ALERT: Redis não está respondendo"
    exit 1
fi

# Verifica uso de memória
MEMORY_USAGE=$(redis-cli INFO memory | grep used_memory: | cut -d:
-f2 | tr -d '\r')
MAX_MEMORY=$(redis-cli CONFIG GET maxmemory | tail -1)

if [ "$MAX_MEMORY" != "0" ]; then
    PERCENTAGE=$((MEMORY_USAGE * 100 / MAX_MEMORY))
    if [ $PERCENTAGE -gt 90 ]; then
        echo "ALERT: Uso de memória em ${PERCENTAGE}%"
    fi
fi

# Verifica replicação
REPL_STATUS=$(redis-cli INFO replication | grep master_link_status |
cut -d: -f2 | tr -d '\r')
if [ "$REPL_STATUS" != "up" ] && [ "$REPL_STATUS" != "" ]; then
    echo "ALERT: Replicação com problemas: $REPL_STATUS"
fi
```

11.4 Debugging e Troubleshooting

Análise de comandos lentos:

```
# Configurar slowlog
CONFIG SET slowlog-log-slower-than 10000 # 10ms
CONFIG SET slowlog-max-len 100
```

```
# Analisar comandos lentos
SLOWLOG GET 10
# Resultado mostra: ID, timestamp, duration, command, client_addr
```

Profiling de aplicação:

```
import redis
import time

class RedisProfiler:
    def __init__(self, redis_client):
        self.redis = redis_client

    def execute_command(self, *args):
        start_time = time.time()
        result = self.redis.execute_command(*args)
        duration = (time.time() - start_time) * 1000

        if duration > 10: # Log se > 10ms
            print(f"SLOW: {args} took {duration:.2f}ms")

        return result

# Uso
r = redis.Redis()
profiled_redis = RedisProfiler(r)
```

Análise de memória por key:

```
# Comando para analisar uso de memória
MEMORY USAGE chave
```

```
# Script Lua para analisar múltiplas keys
redis-cli --eval memory_analysis.lua
```

memory_analysis.lua:

```
local keys = redis.call('KEYS', ARGV[1] or '*')
local result = {}
```



```

for i=1,#keys do
    local memory = redis.call('MEMORY', 'USAGE', keys[i])
    table.insert(result, {keys[i], memory})
end

return result

```

12. Casos de Uso Avançados

12.1 Sistema de Cache Inteligente

Cache com invalidação automática:

```

import redis
import json
import hashlib
from functools import wraps

class SmartCache:
    def __init__(self, redis_client, default_ttl=300):
        self.redis = redis_client
        self.default_ttl = default_ttl

    def cache_key(self, func_name, args, kwargs):
        # Gera chave única baseada na função e argumentos
        key_data = f"{func_name}:{args}:{sorted(kwargs.items())}"
        return f"cache:{hashlib.md5(key_data.encode()).hexdigest()}"

    def cached(self, ttl=None):
        def decorator(func):
            @wraps(func)
            def wrapper(*args, **kwargs):
                cache_key = self.cache_key(func.__name__, args,
                kwargs)

                # Tenta buscar no cache
                cached_result = self.redis.get(cache_key)
                if cached_result:
                    return json.loads(cached_result)

```

```

        # Executa função e armazena resultado
        result = func(*args, **kwargs)
        self.redis.setex(
            cache_key,
            ttl or self.default_ttl,
            json.dumps(result)
        )
        return result
    return wrapper
return decorator

# Uso
cache = SmartCache(redis.Redis())

@cache.cached(ttl=600)
def buscar_usuario(user_id):
    # Simulação de consulta cara ao banco
    return database.query(f"SELECT * FROM users WHERE id = {user_id}")

```

Cache com tags para invalidação:

```

class TaggedCache:
    def set_with_tags(self, key, value, tags, ttl=300):
        # Armazena valor
        self.redis.setex(key, ttl, json.dumps(value))

        # Associa key às tags
        for tag in tags:
            self.redis.sadd(f"tag:{tag}", key)
            self.redis.expire(f"tag:{tag}", ttl + 60) # Tag vive um
pouco mais

    def invalidate_by_tag(self, tag):
        # Busca todas as keys da tag
        keys = self.redis.smembers(f"tag:{tag}")
        if keys:
            # Remove todas as keys
            self.redis.delete(*keys)
            # Remove a tag
            self.redis.delete(f"tag:{tag}")

# Uso

```

```

tagged_cache = TaggedCache()

# Cache com tags
tagged_cache.set_with_tags(
    "produto:123",
    {"nome": "Notebook", "preco": 2500},
    tags=["produtos", "categoria:informatica", "marca:dell"]
)

# Invalida todos produtos da marca Dell
tagged_cache.invalidate_by_tag("marca:dell")

```

12.2 Rate Limiting Avançado

Sliding Window Rate Limiter:

```

import time
import redis

class SlidingWindowRateLimiter:
    def __init__(self, redis_client):
        self.redis = redis_client

    def is_allowed(self, key, limit, window_seconds):
        now = time.time()
        pipeline = self.redis.pipeline()

        # Remove entradas antigas
        pipeline.zremrangebyscore(key, 0, now - window_seconds)

        # Conta requisições na janela atual
        pipeline.zcard(key)

        # Adiciona requisição atual
        pipeline.zadd(key, {str(now): now})

        # Define expiração
        pipeline.expire(key, window_seconds)

        results = pipeline.execute()
        current_requests = results[1]

        return current_requests < limit

```

```
# Uso
rate_limiter = SlidingWindowRateLimiter(redis.Redis())

def api_endpoint(request):
    user_id = request.user.id

    if not rate_limiter.is_allowed(f"rate_limit:user:{user_id}",
100, 3600):
        return {"error": "Rate limit exceeded"}, 429

    return process_request(request)
```

Rate Limiting com múltiplas janelas:

```
class MultiWindowRateLimiter:
    def __init__(self, redis_client):
        self.redis = redis_client

    def check_limits(self, key, limits):
        """
        limits = [
            (10, 60),    # 10 req/min
            (100, 3600), # 100 req/hour
            (1000, 86400) # 1000 req/day
        ]
        """
        now = time.time()

        for limit, window in limits:
            window_key = f"{key}:{window}"

            if not self.is_allowed(window_key, limit, window):
                return False, f"Exceeded {limit} requests per
{window} seconds"

        return True, None
```

12.3 Distributed Locking

Lock distribuído com Redis:

```

import uuid
import time

class DistributedLock:
    def __init__(self, redis_client, key, timeout=10):
        self.redis = redis_client
        self.key = f"lock:{key}"
        self.timeout = timeout
        self.identifier = str(uuid.uuid4())

    def acquire(self):
        end = time.time() + self.timeout

        while time.time() < end:
            # Tenta adquirir lock
            if self.redis.set(self.key, self.identifier, nx=True,
ex=self.timeout):
                return True
            time.sleep(0.001) # 1ms

        return False

    def release(self):
        # Script Lua para release atômico
        release_script = """
        if redis.call("GET", KEYS[1]) == ARGV[1] then
            return redis.call("DEL", KEYS[1])
        else
            return 0
        end
        """
        return self.redis.eval(release_script, 1, self.key,
self.identifier)

    def __enter__(self):
        if self.acquire():
            return self
        raise Exception("Could not acquire lock")

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.release()

# Uso
with DistributedLock(redis.Redis(), "process_payments"):

```

```
# Código que deve rodar em apenas uma instância
process_pending_payments()
```

12.4 Session Store

Sistema de sessões escalável:

```
import json
import time
from datetime import datetime, timedelta

class RedisSessionStore:
    def __init__(self, redis_client, session_timeout=1800): # 30
        min

        self.redis = redis_client
        self.timeout = session_timeout

    def create_session(self, user_id, data=None):
        session_id = str(uuid.uuid4())
        session_data = {
            "user_id": user_id,
            "created_at": time.time(),
            "last_accessed": time.time(),
            "data": data or {}
        }

        # Armazena sessão
        self.redis.setex(
            f"session:{session_id}",
            self.timeout,
            json.dumps(session_data)
        )

        # Indexa por usuário
        self.redis.sadd(f"user_sessions:{user_id}", session_id)
        self.redis.expire(f"user_sessions:{user_id}", self.timeout)

        return session_id

    def get_session(self, session_id):
        session_data = self.redis.get(f"session:{session_id}")
        if session_data:
            data = json.loads(session_data)
```

```

        # Atualiza último acesso
        data["last_accessed"] = time.time()
        self.redis.setex(
            f"session:{session_id}",
            self.timeout,
            json.dumps(data)
        )

        return data
    return None

def update_session(self, session_id, updates):
    session_data = self.get_session(session_id)
    if session_data:
        session_data["data"].update(updates)
        self.redis.setex(
            f"session:{session_id}",
            self.timeout,
            json.dumps(session_data)
        )

def destroy_session(self, session_id):
    session_data = self.get_session(session_id)
    if session_data:
        user_id = session_data["user_id"]

        # Remove sessão
        self.redis.delete(f"session:{session_id}")

        # Remove do índice do usuário
        self.redis.srem(f"user_sessions:{user_id}", session_id)

def get_user_sessions(self, user_id):
    session_ids =
self.redis.smembers(f"user_sessions:{user_id}")
    sessions = []

    for session_id in session_ids:
        session_data = self.get_session(session_id.decode())
        if session_data:
            sessions.append({
                "session_id": session_id.decode(),
                **session_data
            })

```

```
    })

    return sessions
```

12.5 Real-time Analytics

Sistema de métricas em tempo real:

```
import time
from datetime import datetime

class RealTimeAnalytics:
    def __init__(self, redis_client):
        self.redis = redis_client

    def track_event(self, event_type, user_id=None,
properties=None):
        timestamp = int(time.time())

        # Contador geral
        self.redis.incr(f"events:{event_type}:total")

        # Contador por minuto
        minute_key = f"events:{event_type}:minute:{timestamp // 60}"
        self.redis.incr(minute_key)
        self.redis.expire(minute_key, 3600) # Expira em 1 hora

        # Contador por hora
        hour_key = f"events:{event_type}:hour:{timestamp // 3600}"
        self.redis.incr(hour_key)
        self.redis.expire(hour_key, 86400 * 7) # Expira em 7 dias

        # Usuários únicos (HyperLogLog)
        if user_id:
            self.redis.pfadd(f"unique_users:{event_type}:daily:{timestamp // 86400}", user_id)

        # Propriedades do evento
        if properties:
            for key, value in properties.items():
                self.redis.hincrby(f"events:{event_type}:properties:{key}", value,
```


1)

```
def get_metrics(self, event_type, timeframe="hour"):
    if timeframe == "minute":
        return self._get_minute_metrics(event_type)
    elif timeframe == "hour":
        return self._get_hour_metrics(event_type)
    elif timeframe == "day":
        return self._get_daily_metrics(event_type)

def _get_hour_metrics(self, event_type):
    now = int(time.time())
    current_hour = now // 3600

    metrics = {}
    for i in range(24): # Últimas 24 horas
        hour = current_hour - i
        key = f"events:{event_type}:hour:{hour}"
        count = self.redis.get(key) or 0
        metrics[hour] = int(count)

    return metrics

def get_unique_users(self, event_type, days_back=7):
    unique_counts = {}
    now = int(time.time())
    current_day = now // 86400

    for i in range(days_back):
        day = current_day - i
        key = f"unique_users:{event_type}:daily:{day}"
        count = self.redis.pfcount(key)
        unique_counts[day] = count

    return unique_counts

# Uso
analytics = RealTimeAnalytics(redis.Redis())

# Tracking
analytics.track_event("page_view", user_id="user123",
properties={"page": "/home"})
analytics.track_event("purchase", user_id="user456",
properties={"amount": 99.99})
```

```
# Métricas
hourly_views = analytics.get_metrics("page_view", "hour")
unique_users = analytics.get_unique_users("page_view")
```

13. Integração com Linguagens

13.1 Python - Avançado

Pool de conexões otimizado:

```
import redis
from redis.connection import ConnectionPool

# Pool de conexões
pool = ConnectionPool(
    host='localhost',
    port=6379,
    db=0,
    max_connections=20,
    retry_on_timeout=True,
    socket_keepalive=True,
    socket_keepalive_options={}
)

redis_client = redis.Redis(connection_pool=pool)

# Classe para operações de alto nível
class RedisManager:
    def __init__(self, redis_client):
        self.redis = redis_client

    def atomic_increment(self, key, amount=1):
        """Incremento atômico com retry"""
        max_retries = 3
        for attempt in range(max_retries):
            try:
                with self.redis.pipeline() as pipe:
                    pipe.watch(key)
                    current_value = pipe.get(key) or 0
                    pipe.multi()
```

```

        pipe.set(key, int(current_value) + amount)
        pipe.execute()
        return int(current_value) + amount
    except redis.WatchError:
        if attempt == max_retries - 1:
            raise
        continue

def get_or_set(self, key, func, ttl=300):
    """Get com fallback para função"""
    value = self.redis.get(key)
    if value is None:
        value = func()
        self.redis.setex(key, ttl, json.dumps(value))
    else:
        value = json.loads(value)
    return value

```

Async Redis com asyncio:

```

import asyncio
import aioredis
import json

class AsyncRedisManager:
    def __init__(self):
        self.redis = None

    async def connect(self):
        self.redis = await aioredis.from_url(
            "redis://localhost:6379",
            max_connections=20
        )

    async def cached_async(self, key, coro, ttl=300):
        # Tenta buscar no cache
        cached = await self.redis.get(key)
        if cached:
            return json.loads(cached)

        # Executa corrotina
        result = await coro
        await self.redis.setex(key, ttl, json.dumps(result))

```

```

        return result

    async def pubsub_handler(self, channel):
        pubsub = self.redis.pubsub()
        await pubsub.subscribe(channel)

        async for message in pubsub.listen():
            if message['type'] == 'message':
                yield message['data']

# Uso
async def main():
    manager = AsyncRedisManager()
    await manager.connect()

    # Cache assíncrono
    result = await manager.cached_async(
        "api_data",
        fetch_from_api(), # Corrotina
        ttl=600
    )

    # Pub/Sub assíncrono
    async for message in manager.pubsub_handler("updates"):
        await process_message(message)

asyncio.run(main())

```

13.2 Node.js

Cliente Redis otimizado:

```

const redis = require('redis');
const { promisify } = require('util');

class RedisManager {
    constructor() {
        this.client = redis.createClient({
            host: 'localhost',
            port: 6379,
            retry_strategy: (options) => {
                if (options.error && options.error.code ===
'ECONNREFUSED') {

```

```

        return new Error('Redis server refused
connection');
    }
    if (options.total_retry_time > 1000 * 60 * 60) {
        return new Error('Retry time exhausted');
    }
    return Math.min(options.attempt * 100, 3000);
}
});

// Promisify métodos
this.get = promisify(this.client.get).bind(this.client);
this.set = promisify(this.client.set).bind(this.client);
this.del = promisify(this.client.del).bind(this.client);
this.pipeline = () => this.client.pipeline();
}

async cachedFunction(key, func, ttl = 300) {
    try {
        const cached = await this.get(key);
        if (cached) {
            return JSON.parse(cached);
        }

        const result = await func();
        await this.client.setex(key, ttl,
JSON.stringify(result));
        return result;
    } catch (error) {
        console.error('Redis cache error:', error);
        return await func(); // Fallback
    }
}

async publishWithRetry(channel, message, maxRetries = 3) {
    for (let attempt = 1; attempt <= maxRetries; attempt++) {
        try {
            return await this.client.publish(channel,
JSON.stringify(message));
        } catch (error) {
            if (attempt === maxRetries) throw error;
            await new Promise(resolve => setTimeout(resolve,
1000 * attempt));
        }
    }
}

```

```

    }
  }
}

// Uso com Express.js
const express = require('express');
const app = express();
const redisManager = new RedisManager();

app.get('/user/:id', async (req, res) => {
  try {
    const user = await redisManager.cachedFunction(
      `user:${req.params.id}`,
      () => database.findUser(req.params.id),
      600 // 10 minutos
    );

    res.json(user);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

13.3 Java - Spring Boot

Configuração Redis com Spring:

```

@Configuration
@EnableCaching
public class RedisConfig {

    @Bean
    public LettuceConnectionFactory redisConnectionFactory() {
        return new LettuceConnectionFactory(
            new RedisStandaloneConfiguration("localhost", 6379));
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate() {
        RedisTemplate<String, Object> template = new
RedisTemplate<>();
        template.setConnectionFactory(redisConnectionFactory());
        template.setKeySerializer(new StringRedisSerializer());
    }
}

```

```

        template.setValueSerializer(new
GenericJackson2JsonRedisSerializer());
        return template;
    }

    @Bean
    public CacheManager cacheManager() {
        RedisCacheManager.Builder builder = RedisCacheManager
            .RedisCacheManagerBuilder
            .fromConnectionFactory(redisConnectionFactory())
            .cacheDefaults(cacheConfiguration());

        return builder.build();
    }

    private RedisCacheConfiguration cacheConfiguration() {
        return RedisCacheConfiguration.defaultCacheConfig()
            .entryTtl(Duration.ofMinutes(10))

        .serializeKeysWith(RedisSerializationContext.SerializationPair
            .fromSerializer(new StringRedisSerializer()))

        .serializeValuesWith(RedisSerializationContext.SerializationPair
            .fromSerializer(new
GenericJackson2JsonRedisSerializer()));
    }
}

@Service
public class UserService {

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    @Cacheable(value = "users", key = "#userId")
    public User findUser(Long userId) {
        // Busca no banco de dados
        return userRepository.findById(userId);
    }

    @CacheEvict(value = "users", key = "#user.id")
    public User updateUser(User user) {
        return userRepository.save(user);
    }
}

```

```
public void trackUserActivity(Long userId, String activity) {  
    String key = "user_activity:" + userId;  
    redisTemplate.opsForList().leftPush(key, activity);  
    redisTemplate.expire(key, Duration.ofDays(30));  
}  
}
```

14. Deployment e DevOps

14.1 Docker

Dockerfile otimizado:

```
FROM redis:7-alpine  
  
# Cria usuário não-root  
RUN addgroup -g 1001 redis && \  
    adduser -D -u 1001 -G redis redis  
  
# Copia configuração customizada  
COPY redis.conf /usr/local/etc/redis/redis.conf  
  
# Ajusta permissões  
RUN chown redis:redis /usr/local/etc/redis/redis.conf && \  
    mkdir -p /data && \  
    chown redis:redis /data  
  
# Muda para usuário redis  
USER redis  
  
# Expõe porta  
EXPOSE 6379  
  
# Comando customizado  
CMD ["redis-server", "/usr/local/etc/redis/redis.conf"]
```

docker-compose.yml para desenvolvimento:

```
version: '3.8'
```



```

services:
  redis:
    image: redis:7-alpine
    container_name: redis-dev
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
      - ./redis.conf:/usr/local/etc/redis/redis.conf
    command: redis-server /usr/local/etc/redis/redis.conf
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 30s
      timeout: 10s
      retries: 3
    networks:
      - app-network

  redis-insight:
    image: redislabs/redisinsight:latest
    container_name: redis-insight
    ports:
      - "8001:8001"
    volumes:
      - redis_insight_data:/db
    networks:
      - app-network

volumes:
  redis_data:
  redis_insight_data:

networks:
  app-network:
    driver: bridge

```

14.2 Kubernetes

Redis StatefulSet:

```

apiVersion: apps/v1
kind: StatefulSet

```

```
metadata:
  name: redis
spec:
  serviceName: redis
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:7-alpine
          ports:
            - containerPort: 6379
          resources:
            requests:
              memory: "256Mi"
              cpu: "250m"
            limits:
              memory: "512Mi"
              cpu: "500m"
          volumeMounts:
            - name: redis-data
              mountPath: /data
            - name: redis-config
              mountPath: /usr/local/etc/redis
      livenessProbe:
        exec:
          command:
            - redis-cli
            - ping
        initialDelaySeconds: 30
        timeoutSeconds: 5
      readinessProbe:
        exec:
          command:
            - redis-cli
            - ping
        initialDelaySeconds: 5
        timeoutSeconds: 1
```

```

    volumes:
      - name: redis-config
        configMap:
          name: redis-config
    volumeClaimTemplates:
      - metadata:
          name: redis-data
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 1Gi

```

Redis Cluster no Kubernetes:

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis-cluster
spec:
  serviceName: redis-cluster
  replicas: 6
  selector:
    matchLabels:
      app: redis-cluster
  template:
    metadata:
      labels:
        app: redis-cluster
    spec:
      containers:
        - name: redis
          image: redis:7-alpine
          ports:
            - containerPort: 6379
            - containerPort: 16379
          command:
            - redis-server
          args:
            - /etc/redis/redis.conf
            - --cluster-enabled
            - "yes"
            - --cluster-config-file

```

```

- /data/nodes.conf
- --cluster-node-timeout
- "5000"
volumeMounts:
- name: redis-data
  mountPath: /data
- name: redis-config
  mountPath: /etc/redis
volumeClaimTemplates:
- metadata:
    name: redis-data
  spec:
    accessModes: ["ReadWriteOnce"]
    resources:
      requests:
        storage: 1Gi

```

14.3 Terraform

Redis na AWS ElastiCache:

```

# terraform/redis.tf
resource "aws_elasticache_parameter_group" "redis" {
  family = "redis7"
  name    = "redis-params"

  parameter {
    name = "maxmemory-policy"
    value = "allkeys-lru"
  }

  parameter {
    name = "timeout"
    value = "300"
  }
}

resource "aws_cloudwatch_log_group" "redis" {
  name          = "/aws/elasticache/redis"
  retention_in_days = 7
}

output "redis_endpoint" {

```

```
    value =  
aws_elasticache_replication_group.redis.configuration_endpoint_address  
}
```

14.4 Ansible

Playbook para instalação Redis:

```
# ansible/redis.yml  
---  
- hosts: redis_servers  
  become: yes  
  vars:  
    redis_version: "7.0.8"  
    redis_port: 6379  
    redis_bind: "127.0.0.1"  
    redis_maxmemory: "256mb"  
    redis_maxmemory_policy: "allkeys-lru"  
  
  tasks:  
    - name: Install dependencies  
      package:  
        name:  
          - wget  
          - gcc  
          - make  
          - tcl  
        state: present  
  
    - name: Create redis user  
      user:  
        name: redis  
        system: yes  
        shell: /bin/false  
        home: /var/lib/redis  
  
    - name: Download Redis  
      get_url:  
        url: "http://download.redis.io/releases/redis-{{  
redis_version }}.tar.gz"  
        dest: "/tmp/redis-{{ redis_version }}.tar.gz"
```

```
- name: Extract Redis
  unarchive:
    src: "/tmp/redis-{{ redis_version }}.tar.gz"
    dest: /tmp
    remote_src: yes

- name: Compile Redis
  make:
    chdir: "/tmp/redis-{{ redis_version }}"
    jobs: "{{ ansible_processor_vcpus }}"

- name: Install Redis binaries
  make:
    chdir: "/tmp/redis-{{ redis_version }}"
    target: install

- name: Create Redis directories
  file:
    path: "{{ item }}"
    state: directory
    owner: redis
    group: redis
    mode: '0755'
  loop:
    - /etc/redis
    - /var/lib/redis
    - /var/log/redis

- name: Generate Redis configuration
  template:
    src: redis.conf.j2
    dest: /etc/redis/redis.conf
    owner: redis
    group: redis
    mode: '0640'
  notify: restart redis

- name: Create Redis systemd service
  template:
    src: redis.service.j2
    dest: /etc/systemd/system/redis.service
  notify:
    - reload systemd
    - restart redis
```

```
- name: Start and enable Redis
  systemd:
    name: redis
    state: started
    enabled: yes
    daemon_reload: yes
```

handlers:

```
- name: reload systemd
  systemd:
    daemon_reload: yes

- name: restart redis
  systemd:
    name: redis
    state: restarted
```

Template de configuração (redis.conf.j2):

```
# Redis Configuration Template
bind {{ redis_bind }}
port {{ redis_port }}
protected-mode yes

# Memory
maxmemory {{ redis_maxmemory }}
maxmemory-policy {{ redis_maxmemory_policy }}

# Persistence
save 900 1
save 300 10
save 60 10000

dbfilename dump.rdb
dir /var/lib/redis/

# AOF
appendonly yes
appendfilename "appendonly.aof"
appendfsync everysec

# Logging
loglevel notice
logfile /var/log/redis/redis-server.log
```

```
# Security
{% if redis_password is defined %}
requirepass {{ redis_password }}
{% endif %}

# Limits
timeout 300
tcp-keepalive 300

# Slow log
slowlog-log-slower-than 10000
slowlog-max-len 128
```

15. Troubleshooting

15.1 Problemas Comuns

15.1.1 Redis não inicia:

```
# Verificar logs
sudo journalctl -u redis -f

# Verificar configuração
redis-server --test-config /etc/redis/redis.conf

# Verificar permissões
ls -la /var/lib/redis/
ls -la /var/log/redis/

# Verificar portas em uso
netstat -tulpn | grep 6379
```

15.1.2 Conectividade:

```
# Teste local
redis-cli ping

# Teste remoto
redis-cli -h redis.exemplo.com -p 6379 ping
```



```
# Verificar firewall
sudo ufw status
sudo iptables -L

# Teste de rede
telnet redis.exemplo.com 6379
```

15.1.3 Performance lenta:

```
# Verificar comandos lentos
SLOWLOG GET 10

# Verificar uso de memória
INFO memory

# Verificar fragmentação
INFO memory | grep mem_fragmentation_ratio

# Verificar keys expiradas
INFO stats | grep expired_keys
```

15.2 Debugging Avançado

Script de diagnóstico:

```
#!/bin/bash
# redis-diagnosis.sh

echo "=== Redis Diagnosis ==="
echo "Date: $(date)"
echo

echo "=== Basic Info ==="
redis-cli INFO server | grep -E
"(redis_version|process_id|uptime_in_seconds)"
echo

echo "=== Memory Usage ==="
redis-cli INFO memory | grep -E
"(used_memory_human|used_memory_peak_human|maxmemory_human|mem_fragm"
```

```

entation_ratio)"
echo

echo "=== Client Connections ==="
redis-cli INFO clients
echo

echo "=== Slow Queries ==="
redis-cli SLOWLOG GET 5
echo

echo "=== Key Distribution ==="
for db in {0..15}; do
    keys=$(redis-cli -n $db DBSIZE)
    if [ "$keys" != "0" ]; then
        echo "Database $db: $keys keys"
    fi
done
echo

echo "=== Top Keys by Memory ==="
redis-cli --bigkeys
echo

echo "=== Replication Status ==="
redis-cli INFO replication
echo

echo "=== Configuration Issues ==="
redis-cli CONFIG GET "*memory*"
redis-cli CONFIG GET timeout
redis-cli CONFIG GET maxclients

```

Monitoramento de comandos suspeitos:

```

import redis
import time
from collections import defaultdict

class RedisMonitor:
    def __init__(self, redis_client):
        self.redis = redis_client
        self.command_stats = defaultdict(list)

```

```

def monitor_commands(self, duration=60):
    """Monitorea comandos por X segundos"""
    monitor = self.redis.monitor()
    start_time = time.time()

    for command in monitor.listen():
        if time.time() - start_time > duration:
            break

        if command['command']:
            cmd_parts = command['command'].split()
            if cmd_parts:
                cmd_name = cmd_parts[0].upper()
                self.command_stats[cmd_name].append({
                    'timestamp': command['time'],
                    'client': command['client_addr'],
                    'command': command['command']
                })

    return self.analyze_commands()

def analyze_commands(self):
    analysis = {}

    for cmd, occurrences in self.command_stats.items():
        analysis[cmd] = {
            'count': len(occurrences),
            'clients': len(set(occ['client'] for occ in
occurrences)),
            'avg_per_second': len(occurrences) / 60
        }

        # Detectar comandos sospeitos
        if cmd in ['KEYS', 'FLUSHALL', 'FLUSHDB']:
            analysis[cmd]['warning'] = 'Potentially dangerous
command'

        elif analysis[cmd]['avg_per_second'] > 100:
            analysis[cmd]['warning'] = 'High frequency command'

    return analysis

# Uso
monitor = RedisMonitor(redis.Redis())

```

```
stats = monitor.monitor_commands(duration=60)

for cmd, data in stats.items():
    print(f"{cmd}: {data['count']} times,
{data['avg_per_second']:.1f}/sec")
    if 'warning' in data:
        print(f"  WARNING: {data['warning']}")
```

15.3 Recuperação de Desastres

Backup automatizado:

```
#!/bin/bash
# redis-backup.sh

BACKUP_DIR="/backups/redis"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=7

# Criar diretório de backup
mkdir -p $BACKUP_DIR

# Backup RDB
redis-cli BGSAVE
sleep 5 # Aguarda conclusão

# Copia arquivo RDB
cp /var/lib/redis/dump.rdb $BACKUP_DIR/dump_$DATE.rdb

# Backup AOF (se habilitado)
if [ -f /var/lib/redis/appendonly.aof ]; then
    cp /var/lib/redis/appendonly.aof
    $BACKUP_DIR/appendonly_$DATE.aof
fi

# Comprime backups
gzip $BACKUP_DIR/*_$DATE.*

# Remove backups antigos
find $BACKUP_DIR -name "*.gz" -mtime +$RETENTION_DAYS -delete

echo "Backup completed: $DATE"
```

```
Restore de backup:
#!/bin/bash
# redis-restore.sh

BACKUP_FILE=$1
REDIS_DATA_DIR="/var/lib/redis"

if [ -z "$BACKUP_FILE" ]; then
    echo "Usage: $0 <backup_file>"
    exit 1
fi

# Para Redis
sudo systemctl stop redis

# Faz backup do estado atual
sudo cp $REDIS_DATA_DIR/dump.rdb $REDIS_DATA_DIR/dump.rdb.backup

# Restaura backup
if [[ $BACKUP_FILE == *.gz ]]; then
    gunzip -c $BACKUP_FILE > $REDIS_DATA_DIR/dump.rdb
else
    cp $BACKUP_FILE $REDIS_DATA_DIR/dump.rdb
fi

# Ajusta permissões
sudo chown redis:redis $REDIS_DATA_DIR/dump.rdb

# Inicia Redis
sudo systemctl start redis

# Verifica se está funcionando
redis-cli ping
```

16. Recursos Adicionais

16.1 Bibliografia e Documentação

Documentação Oficial:

- Redis Documentation: <https://redis.io/documentation>
- Redis Commands Reference: <https://redis.io/commands>

- Redis Configuration: <https://redis.io/topics/config>

Livros Recomendados:

- "Redis in Action" by Josiah Carlson
- "Redis Essentials" by Maxwell Dayvson da Silva
- "Mastering Redis" by Jeremy Nelson

Cursos Online:

- Redis University: <https://university.redis.com/>
- Introduction to Redis Data Structures: <https://redis.io/topics/data-types-intro>

16.2 Ferramentas Úteis

Interfaces Gráficas:

- **RedisInsight**: GUI oficial da Redis Labs
- **Redis Desktop Manager**: Cliente multiplataforma
- **Medis**: Cliente para macOS
- **Redis Commander**: Interface web

Monitoramento:

- **redis-stat**: Monitor de estatísticas
- **RedisLive**: Dashboard em tempo real
- **Prometheus Redis Exporter**: Métricas para Prometheus
- **Grafana Redis Plugin**: Dashboards customizados

Desenvolvimento:

- **Redis CLI**: Interface de linha de comando
- **redis-benchmark**: Ferramenta de benchmark
- **redis-migrate**: Migração entre instâncias
- **RediSearch**: Módulo de busca full-text

16.3 Comunidade

Forums e Comunidades:

- Redis Slack Community
- Stack Overflow - Redis Tag
- Reddit r/redis
- Redis Google Group

Eventos:

- RedisConf: Conferência anual

- Redis Meetups locais
- Webinars da Redis Labs

16.4 Checklist de Produção

Antes de ir para produção:

Configuração:

- ☐ `maxmemory` configurado adequadamente
- ☐ `maxmemory-policy` definida
- ☐ Persistência configurada (RDB + AOF)
- ☐ `timeout` e `tcp-keepalive` configurados
- ☐ Comandos perigosos renomeados/desabilitados

Segurança:

- ☐ `requirepass` configurado
- ☐ ACLs configuradas (Redis 6+)
- ☐ `bind` restrito aos IPs necessários
- ☐ `protected-mode` habilitado
- ☐ TLS configurado (se necessário)
- ☐ Firewall configurado

Monitoramento:

- ☐ Logs configurados e rotacionados
- ☐ Monitoramento de métricas implementado
- ☐ Alertas configurados
- ☐ Slowlog configurado
- ☐ Backup automatizado

Alta Disponibilidade:

- ☐ Replicação configurada
- ☐ Sentinel ou Cluster implementado
- ☐ Procedimentos de failover testados
- ☐ Aplicação preparada para reconexão

Performance:

- ☐ Benchmark realizado
- ☐ Pipeline implementado onde apropriado
- ☐ Pool de conexões configurado
- ☐ Keys design otimizado

16.5 Glossário

AOF (Append Only File): Arquivo de log que registra todas as operações de escrita.

Cluster: Modo de operação distribuído do Redis com sharding automático.

Eviction: Processo de remoção de keys quando a memória está cheia.

Hash Slot: Uma das 16.384 partições usadas no Redis Cluster.

HyperLogLog: Estrutura probabilística para contagem de elementos únicos.

LRU (Least Recently Used): Algoritmo que remove itens menos recentemente usados.

Pipeline: Técnica de envio de múltiplos comandos sem aguardar respostas.

Pub/Sub: Sistema de mensageria publisher/subscriber.

RDB: Formato de snapshot binário do Redis.

RESP: Redis Serialization Protocol - protocolo de comunicação.

Sentinel: Sistema de alta disponibilidade do Redis.

Sharding: Distribuição de dados entre múltiplos servidores.

TTL (Time To Live): Tempo de vida de uma key antes de expirar.