Módulo Store

O módulo **store** fornece um sistema de armazenamento persistente de dados que utiliza namespacing automático. Cada valor armazenado é automaticamente associado ao identificador de execução (EXEC_IDENT) da métrica ou script, criando um namespace isolado para cada um. Isso significa que, por padrão, diferentes métricas não compartilham dados, mas múltiplas execuções da mesma métrica podem acessar e modificar os valores que ela armazenou anteriormente. O módulo é útil para armazenar estados, configurações ou históricos que precisam persistir entre reinícios do agente.

- Armazenamento chave-valor persistente em disco
- Namespacing automático baseado no identificador da métrica/script
- Timestamp automático para cada entrada
- Thread-safe (seguro para uso concorrente)
- Persistência garantida entre reinícios do agente

Diferença entre Store, Registry e Cache

Store vs Registry:

- **Store**: Persiste em disco, sobrevive a reinícios do agente mas é global para todas as métricas/scripts
- **Registry**: Armazenamento apenas em memória, perdido no reinício

Store vs Cache:

- **Store**: Persistente, sem políticas de expiração automática
- **Cache**: Otimizado para acesso rápido, com políticas de expiração

Namespacing:

- **Funções** `put`/`get`/`delete`: Usam namespacing automático (chave + ident)
- **Funções** `*_global`: Ignoram namespacing (chave pura)

Funções Disponíveis

1. `store.put(chave, valor)`

Armazena um valor no store com namespacing automático.

Parâmetros:

- **chave** (string): Identificador para o valor (será prefixado com o ident)
- **valor** (qualquer tipo Lua): Valor a ser armazenado (string, número, booleano, tabela, nil)

Retorno:

- **nil**: A função não retorna valor

Comportamento:

1. Prefixa a chave com o identificador de execução atual (se disponível)
2. Armazena o valor associado à chave prefixada
3. Registra automaticamente o timestamp atual (UTC)
4. Sobrescreve qualquer valor existente com a mesma chave
5. Persiste automaticamente em disco

Exemplo de Uso:

```
-- Armazenar dados específicos do script atual
store.put("ultima_execucao", os.time())
store.put("contador_falhas", 0)
store.put("config", {
    timeout = 30,
    intervalo = 60,
    alertas = true
})

-- Dados serão armazenados com prefixo do ident
-- Se EXEC_IDENT = "monitor-cpu", a chave se torna "monitor-cpu:ultima_execucao"
```

2. `store.get(chave)`

Recupera um valor do store com namespacing automático.

Parâmetros:

- **chave** (string): Chave do valor a ser recuperado (será prefixada com o ident)

Retorno:

- **tuple**: `(valor, timestamp)` onde:
  - **valor** (qualquer tipo ou nil): Valor armazenado, ou `nil` se a chave não existir
  - **timestamp** (número ou nil): Timestamp Unix (segundos) da última atualização, ou `nil` se a chave não existir

Comportamento:

1. Prefixa a chave com o identificador de execução atual
2. Retorna o valor e timestamp se a chave existir
3. Retorna `(nil, nil)` se a chave não existir
4. Preserva o tipo original do valor armazenado
5. O timestamp é em segundos desde a epoch Unix (UTC)

Exemplo de Uso:

```
-- Recuperar dados do script atual
local ultima_exec, timestamp = store.get("ultima_execucao")
if ultima_exec then
    local diferenca = os.time() - timestamp
    print("Última execução há", diferenca, "segundos")
end

-- Recuperar configuração
local config, ts = store.get("configuracao")
if config then
    print("Configuração carregada (atualizada em", os.date("%H:%M:%S", ts), ")")
    for chave, valor in pairs(config) do
        print("  ", chave, "=", valor)
    end
end

-- Verificar existência com valor padrão
local limite, _ = store.get("limite_temperatura")
limite = limite or 70  -- Valor padrão se não configurado
```

3. `store.delete(chave)`

Remove uma entrada do store com namespacing automático.

Parâmetros:

- **chave** (string): Chave da entrada a ser removida (será prefixada com o ident)

Retorno:

- **nil**: A função não retorna valor

Comportamento:

1. Prefixa a chave com o identificador de execução atual
2. Remove completamente a entrada do store
3. Não faz nada se a chave não existir
4. Libera o espaço em memória e disco
5. Operação atômica e thread-safe

Exemplo de Uso:

```
-- Remover dados específicos do script
store.delete("cache_temporario")
store.delete("dados_processados")

-- Limpar todos os dados do script atual
local function limpar_dados_script()
    -- Nota: Não há listagem direta de chaves
    -- É necessário conhecer as chaves usadas
    local chaves_conhecidas = {
        "configuracao",
        "cache",
        "estado",
        "historico",
        "lock_backup"
    }

    for _, chave in ipairs(chaves_conhecidas) do
        store.delete(chave)
        log.debug("Removido:", chave)
```

```
    end
end

-- Remover após processamento
local function processar_e_limpar(chave)
    local dados, timestamp = store.get(chave)

    if dados then
        -- Processar dados
        local resultado = processar_dados(dados)

        -- Remover após processamento
        store.delete(chave)
        log.info("Dados processados e removidos:", chave)

        return resultado
    else
        log.warn("Chave não encontrada:", chave)
        return nil
    end
end
```

4. `store.put_global(chave, valor)`

Armazena um valor no store SEM namespacing (chave global).

Parâmetros:

- **chave** (string): Identificador global para o valor (não é prefixado)
- **valor** (qualquer tipo Lua): Valor a ser armazenado

Retorno:

- **nil**: A função não retorna valor

Comportamento:

1. Armazena o valor com a chave exata fornecida
2. Não aplica prefixo do identificador de execução
3. Compartilhado entre todos os scripts (global)
4. Persiste em disco

5. Sobrescreve qualquer valor existente com a mesma chave

Exemplo de Uso:

```
-- Armazenar dados globais compartilhados
store.put_global("versao_agente", "2.5.1")
store.put_global("ultima_atualizacao", os.time())
store.put_global("config_global", {
    timezone = "America/Sao_Paulo",
    log_level = "info",
    retencao_logs = 30  -- dias
})

-- Dados serão acessíveis por todos os scripts
-- Chave exata: "versao_agente" (sem prefixo)
```

5. `store.get_global(chave)`

Recupera um valor do store SEM namespacing (chave global).

Parâmetros:

- **chave** (string): Chave global do valor a ser recuperado (não é prefixada)

Retorno:

- **tuple**: `(valor, timestamp)` onde:
  - **valor** (qualquer tipo ou nil): Valor armazenado, ou `nil` se a chave não existir
  - **timestamp** (número ou nil): Timestamp Unix (segundos) da última atualização, ou `nil` se a chave não existir

Comportamento:

1. Busca o valor usando a chave exata fornecida (sem prefixo)
2. Retorna o valor e timestamp se a chave existir
3. Retorna `(nil, nil)` se a chave não existir
4. Preserva o tipo original do valor armazenado
5. O timestamp é em segundos desde a epoch Unix (UTC)
6. Acessa dados compartilhados por todos os scripts

Exemplo de Uso:

```
-- Recuperar dados globais compartilhados
```

```
local versao, ts_versao = store.get_global("versao_agente")
if versao then
    print("Versão do agente:", versao, "(atualizada em", os.date("%Y-%m-%d %H:%M:%S",
ts_versao), ")")
end

-- Recuperar configuração global
local config_global, ts_config = store.get_global("config_global")
if config_global then
    print("Configuração global carregada:")
    for chave, valor in pairs(config_global) do
        print("  ", chave, "=", valor)
    end
end

-- Verificar e usar valor padrão para configuração global
local timezone, _ = store.get_global("timezone")
timezone = timezone or "UTC"  -- Valor padrão se não configurado
print("Timezone configurado:", timezone)

-- Verificar existência de flag global
local manutencao, ts_manutencao = store.get_global("modo_manutencao")
if manutencao then
    log.warn("Sistema em modo de manutenção desde", os.date("%H:%M:%S", ts_manutencao))
    -- Pular execuções não críticas
    return
end
```

Revision #1
Created 3 February 2026 16:45:43 by Marc
Updated 3 February 2026 16:45:55 by Marc