Sincronização de arquivos com o Filesync

Lucas Bulegon Mello
lucasbmello96@gmail.com

[1]Eixo Tecnologia – Faculdade SENAC Porto Alegre (FSPOA)
Rua Coronel Genuíno 130 – Cidade Baixa – Porto Alegre – RS – Brasil

Abstract. *The system provides a solution for synchronizing digital files between workstations or servers within a network. This synchronization will be performed automatically through synchronization tasks previously configured by the tool operator.*

Resumo. *O sistema oferece uma solução para sincronia de arquivos digitais entre estações de trabalho ou servidores dentro de uma rede. Esta sincronia será realizada automaticamente através de tarefas de sincronia previamente configuradas pelo operador da ferramenta.*

1. Introdução

Em redes de grande porte, aonde existe um fluxo massivo de dados em trânsito, pode ser um desafio manter quaisquer informações sincronizadas entre os diversos membros desta rede. Grandes distâncias geográficas, diferentes configurações entre os equipamentos e diferentes características entre os links de comunicação podem gerar grandes atrasos ou falhas no envio destes dados.

Com o crescimento da internet na última década e com a transformação digital que diversas empresas vem passando, notou-se um grande aumento na quantidade de dados gerados e trocados entre redes. Juntamente deste aumento de demanda surge um problema, como manter estes dados sempre seguros, disponíveis e sincronizados dentro de uma rede massiva? E como manter essa necessidade crescente por tŕafego de dados?

A maioria das empresas baseadas na estrutura de matriz/filial gera um grande volume de dados e de tráfego de redes entre todas suas localidades. Seja este tráfego gerado pelos funcionários acessando os sistemas internos da empresa, realizando envio de e-mails e mensagens de texto, imagens ou documentos corporativos em geral.

Para que os arquivos gerados sempre estejam íntegros e disponíveis é imprescindível que existam sistemas confiáveis de gerência de arquivos, sincronias de dados e redundâncias de informações, para que nenhum dado seja perdido, em nenhuma circunstância. Levando em conta estes dois requisitos, desenvolveu-se o File-Sync.

Essa ferramenta permite a troca de dados entre computadores de uma rede, utilizando-se de tarefas de sincronia configuradas pelo operador de forma simplificada. Tarefas estas, facilitam a gerência sobre os dados sincronizados, a periodicidade de suas sincronias e se existiram alterações de conteúdo nos arquivos.

2. Fundamentação Teórica

2.1. Redes de Computadores

Neste item serão descritos os conceitos básicos de redes de computadores, sua origem com a conceituação de tecnologia, suas distinções de espaço, principais características e protocolos presentes.

> Tecnologia, num sentido amplo, pode ser definida como o conjunto de ferramentas, maquinários e técnicas desenvolvido pelo homem - desde a descoberta do fogo - como uma maneira de modificar o ambiente em seu favor. Mais recentemente na história humana, desenvolvimentos tecnológicos na área das telecomunicações e tecnologia da informação modificaram não só como as pessoas se comunicam, mas também como elas se relacionam com a própria tecnologia. [Picon et al. 2015]

> As redes de computadores têm inúmeros usos, tanto por empresas quanto por indivíduos, tanto em casa quanto em trânsito. Para as empresas, as redes de computadores pessoais que utilizam servidores compartilhados frequentemente oferecem acesso a informações corporativas, normalmente usando o modelo cliente-servidor [...]
> Grosso modo, as redes podem ser divididas em LANs, MANs, WANs e internets. As LANs abrangem um prédio e operam em altas velocidades. As MANs em geral abrangem uma cidade [...]. As WANs abrangem um país ou um continente. [Tanenbaum and Wetherall 2011]

Conforme destacado no texto de Tanenbaum e Wetherall, as redes estão presentes em quase todos os aspectos de nossa sociedade moderna. Essas redes são organizadas por padrões, ou protocolos de comunicação, conforme descrito a seguir.

> O software de rede consiste em protocolos ou regras pelas quais os processos se comunicam. A maioria das redes aceita as hierarquias de protocolos, com cada camada fornecendo serviços às camadas situadas acima dela e isolando-as dos detalhes dos protocolos usados nas camadas inferiores. [Tanenbaum and Wetherall 2011]

> O TCP (*Transmission Control Protocol* — Protocolo de Controle de Transmissão) e o IP (*Internet Protocol* — Protocolo da Internet) são dois dos mais importantes da Internet. O protocolo IP especifica o formato dos pacotes que são enviados e recebidos entre roteadores e sistemas finais. [Kurose and Ross 2014]

> As redes fornecem vários serviços a seus usuários. Esses serviços podem variar da entrega de pacotes por melhores esforços por serviços não orientados a conexões até a entrega garantida por serviços orientados a conexões. [Tanenbaum and Wetherall 2011]

2.2. Congestionamento

Em redes com muito tráfego de dados, podem ocorrer casos de congestionamentos, quando um determinado recurso ou equipamento não consegue atender a demanda com sua capacidade atual, gerando enfileiramentos e atrasos nas entregas de conteúdo.

> *Congestion occurs when resource demands exceed the capacity. As users come and go, so do the packets they send; Internet performance is therefore largely governed by these inevitable natural fluctuations.* [Welzl 2005]

2.3. Arquivos de Dados

> Todas as aplicações de computadores precisam armazenar e recuperar informações. Enquanto um processo está sendo executado, ele pode armazenar uma quantidade limitada de informações dentro do seu próprio espaço de endereçamento. [Tanenbaum and Bos 2015]

Arquivos são gerados a todo o momento durante o uso de computadores. Ao mesmo tempo que eles são fundamentais ao nosso uso cotidiano, podem se tornar facilmente um problema quando seu volume é muito grande ou quando está distribuído em diversos locais diferentes.

> Um arquivo é um mecanismo de abstração. Ele fornece uma maneira para armazenar informações sobre o disco e lê-las depois. Isso deve ser feito de tal modo que isole o usuário dos detalhes de como e onde as informações estão armazenadas, e como os discos realmente funcionam. [Tanenbaum and Bos 2015]

2.4. Sincronia de Dados e Algoritmos de Sincronia

Tratando de sincronia de dados, não é possível deixar de mencionar os algoritmos que são executados para identificar alterações de arquivos. Um dos mais utilizados é o rsync no Linux. Seu algoritmo troca informações criptográficas de checkagem (checksum) entre o cliente e servidor, para determinar se os arquivos necessitam de sincronia.

> *The algorithm identifes parts of the source file which are identical to some part of the destination file, and only sends those parts which cannot be matched in this way.* [Tridgell and Mackerras 1996]

2.5. CLI ou Shell

Segundo [Tanenbaum and Bos 2015], usuários Linux tem a opção e a preferência pelo uso de interfaces de linha de comando (**C**ommand **L**ine **I**nterface em Inglês) para operação de seus sistemas, ou simplesmente Shell. A interface de linha de comando mais famosa e amplamente utilizada é o BASH (**B**ourne **A**gain **SH**ell), inspirada no clássico Shell Unix original, ou Shell Bourne.

> Quando o usuário digita uma linha de comando, o shell extrai a primeira palavra dele, onde palavra aqui significa uma série de caracteres delimitados por um espaço ou guia (tab). Ele então presume que essa palavra é o nome de um programa a ser executado, pesquisa por esse programa e se o encontra, executa-o. O shell então suspende a si mesmo até o programa terminar, momento em que ele tenta ler o próximo comando. [Tanenbaum and Bos 2015]

Também seguindo [Tanenbaum and Bos 2015], existem diversas categorias de software que são executados via linha de comando, são eles:

1. Comandos para manipulação de arquivos e diretórios.
2. Filtros.
3. Ferramentas de desenvolvimento de programas, como editores e compiladores.
4. Processamento de texto.
5. Administração de sistema.
6. Miscelâneas.

3. Tecnologias Utilizadas

3.1. Arquivos JSON

No campo de consistência de dados, utilizaram-se arquivos no padrão JSON, que é um formato de estruturação de dados baseado em texto que utiliza padrões de chave e valor para organização dos dados, podendo até ser considerado um banco de dados não relacional.

> *JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data.* [GoogleInc 2014]

A leitura destes dados contidos nos arquivos JSON pela linguagem Python acontece de forma nativa, aonde a estrutura de dados do arquivo é convertida para a tipagem da linguagem, seguindo um padrão definido, facilitando o acesso a estes dados.

3.2. Python

Será descrito um breve histórico da linguagem, bem como os principais módulos utilizados no projeto.

3.2.1. O que é Python?

Python é uma linguagem de programação de alto nível, de tipagem é dinâmica forte, interpretada e não compilada, o que permite execuções como *script*, chamado na documentação de módulo, ou via um conjunto de módulos, chamado de pacote. Permite uso de paradigma funcional, modular, orientado a objetos e imperativo procedural. [Python 2020]

Foi lançada em 1991 por Guido van Rossum, que trabalhou até 2019 na Dropbox e ainda se mantém ativo na comunidade Python, autodeclarando-se ”Benevolent Dictator for Life”ou algo como ditador benevolente vitalício. A linguagem tem uma comunidade ativa gigantesca e vem passando por atualizações constantes, tendo sua versão 2.7 sendo descontinuada e sua versão 3.8 como estável em uso corrente.

O Python é uma linguagem multiplataforma, o que possibilita o desenvolvimento de códigos compatíveis com a maioria do sistemas operacionais, como Linux, Unix, Windows e MacOS, contendo apenas alguns requisitos mínimos.

3.2.2. Principais Módulos e Pacotes Utilizados

O projeto faz uso de diversas ferramentas nativas ao Linux, por isso utilizam-se basicamente módulos de acesso aos recursos do sistema operacional.

3.2.3. Pacote Subprocess

Para as execuções de comandos no sistema operacional, utilizou-se o módulo nativo *subprocess*. Este é um módulo Python usado para criação de novos processos [Astrand 2003] dentro do sistema operacional, tendo possibilidade de interação com as entradas, saídas e erros do Linux através de PIPES (interligação lógica entre as entradas e saídas dos processos).

O *subprocess* realiza as execuções da ferramenta rsync no Linux, com isso, gerencia todas as tarefas que envolvem envio de dados na rede.

3.2.4. Pacote JSON

Nas funções de serialização e desserialização dos arquivos JSON, utilizou-se o módulo nativo *json*, que já trás consigo todos os recursos para gerência de arquivos neste formato. [Foundation 2020b]

3.2.5. Pacote Crontab

Para administração das tarefas agendadas no cron do Linux, utilizou-se o módulo de terceiros, *python-crontab*.

> *Crontab module for reading and writing crontab files and accessing the system cron automatically and simply using a direct API.* [Owens 2020]

3.2.6. Pacote SYS

> *This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.* [Foundation 2020c]

Utilizou-se o pacote *sys* no acesso aos argumento repassados durante a execução da ferramenta via linha de comando, realizando consulta direta ao *argv* do Linux (lista ordenada com os parâmetros executados via shell) para coleta dos parâmetros.

Utilizando uma lógica nas entradas, com os parâmetros repassados via CLI considerando o padrão de dois traços antes de seu nome, para identificação da intenção da execução dentro do código, sendo possível controlar as diferentes sessões da ferramenta.

Por exemplo, quando se faz necessário alterar alguma configuração de domínio, utiliza-se o padrão *'–domain'*, que aciona o módulo de CLI responsável por todas as configurações de domínio na ferramenta.

3.2.7. Pacote Fnmatch

Para busca de padrões em *strings* de texto, utilizou-se *fnmatch*, que é um módulo Python que não acompanha a instalação padrão da linguagem, sendo necessário instalação à parte. [Foundation 2020a]

A fim de localizar o padrão desejado dentro do texto, o módulo utiliza uma linguagem específica, próxima à expressões regulares. Aplicou-se este módulo para identificar o padrão de dois traços nos parâmetros, utilizado largamente no módulo de CLI.

3.2.8. Framework Flask

Para construção da API utilizou-se o *framework Flask*. [Pallets 2010]

3.2.9. Ambientes Virtuais Python

A fim de padronizar as execuções de código Python dentro do projeto, utilizaram-se ambientes virtuais Python.

> *The venv module provides support for creating lightweight "virtual environments" with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (which matches the version of the binary that was used to create this environment) and can have its own independent set of installed Python packages in its site directories.* [Meyer 2011]

Com a utilização de ambientes virtuais é possível manipular as dependências mais facilmente, permitindo que cada projeto em execução/desenvolvimento possua seu ambiente Python totalmente independente, podendo utilizar versões da linguagem e dependências diferentes em cada um dos ambientes virtuais criados.

Para facilitar a configuração do ambiente, utiliza-se um recurso nativo ao manipulador de pacotes PIP, que faz a leitura das dependências necessárias e suas versões adequadas de um arquivo de configuração padrão. Contido na raiz do projeto e nomeado como `requirements.txt`.

3.3. Git

Segundo [Conservancy 2020b], o sistema foi criado a partir da necessidade de versionamento e controle do código desenvolvido para o *kernel* Linux em 2005. Tem foco na distribuição, velocidade, não linearidade (vários "galhos"de desenvolvimento simultâneos e eficácia com projetos grandes.

> Controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde. [Conservancy 2020a]

3.3.1. GitHub

A ferramenta de Git utilizada como controle de versão do projeto foi o GitHub. Plataforma de uso gratuito, comprada pela Microsoft no ano de 2018.

4. Funcionamento da Ferramenta

Os módulos desenvolvidos utilizam-se largamente de recursos presentes nos sistemas Linux, como SSH, crontab e rsync. Para isso, são realizadas leituras nos arquivos de dados e posteriormente as operações nesses contextos dentro do sistemas operacional.

Durante a instalação da ferramenta, os arquivos de configuração tem seu conteúdo zerado, permanecendo apenas com sua estrutura de dados. Para execução da ferramenta, são necessárias alguns passo:

1. Criação ou ingresso em domínio file-sync.
2. Adição dos hosts (manual ou automaticamente durante o ingresso em domínio).
3. Criação das tarefas de sincronia (manual ou automaticamente durante o ingresso em domínio).
4. Inicio das sincronias de dados, respeitando os tempos definidos nas tarefas.

Um mesmo nó pode ser utilizado como cliente e servidor da ferramenta, possibilitando que se forme a hierarquia para envio dos dados entre os nós. Para que este funcionamento de cliente/servidor ocorra é necessário que a ferramenta esteja disponível e corretamente configurada em cada nó, apontando cada sucessor para envio dos dados, formando o desenho de árvore para envio.

A ferramenta está disponível no GitHub em:
`https://github.com/lbmello/file-sync`

5. Validação

Para realizar a correta validação referente aos envios de dados, criou-se um módulo de *logs*, que captura os status sobre as sincronizações, dados sobre domínios, alterações nos arquivos dentro dos compartilhamentos, além de capturar os erros da ferramenta.

Este módulo de *logs* oferece a possibilidade de armazenamento e coleta local ou remota dos dados gerados pelas operações da ferramenta.

6. Ambiente de Testes

Inicialmente foram realizados testes com ambiente virtualizado *Vagrant*, que provê uma interface automatizada e baseada em código para criação dos ambientes. Porém estas máquinas virtuais apresentaram poucas possibilidades de personalização e falta de conexão de rede entre as máquinas virtuais e os sistemas de emulação e redes como o GNS3.

Além disso, a configuração inicial do ambiente apresentou muita demora, mostrando-se ineficiente quando manipularam-se diversas máquinas virtuais.

Optou-se então pela utilização de máquinas virtuais simples, criadas manualmente no virtualizador *opensource* VirtualBox.

6.1. Endereçamento

Utilizou-se um endereçamento na faixa 192.168.15.0/24. Mesmo existindo um servidor DHCP configurado nesta rede, os endereços de IP são definidos estaticamente em cada máquina virtual, seguindo a ordem de endereçamento após o IP 192.168.15.100.

Todas as máquinas virtuais utilizam placas de rede em modo privado (nomeado no VirtualBox como placa em modo NAT), o que possibilita executar este ambiente em qualquer infraestrutura para testes.

Além das máquinas virtuais se comunicarem entre si, também existe comunicação com a Internet através do sistema operacional hospedeiro, sem necessidade de roteamento manual.

Não existe limitação de banda ou restrições de acesso aplicados neste ambiente.

6.2. Recursos de Hardware

Cada máquina foi configurada com 512MB de memória RAM, um núcleo de processamento sem limite de execução por núcleo no *host* hospedeiro e 8GB de espaço em disco, sem nenhuma partição. Configurou-se também um padrão de nome para a estação Linux e para a máquina virtual em si dentro do VirtualBox.

Em função das máquinas virtuais utilizadas para testes serem inicialmente iguais, optou-se pela utilização dos discos no formato de clone *linkado*.

Em função das máquinas virtuais utilizadas para testes serem inicialmente iguais, optou-se em função de economia de espaço em disco no servidor hospedeiro, que todas as máquinas virtuais utilizam o sistema de disco em modo clone *linkado*. Isto faz com que os dados sejam armazenados apenas no disco da primeira máquina configurada e as demais máquinas fazem a leitura destes dados em comum no disco e gravam em seus discos próprios, apenas o conteúdo diferencial.

6.3. Configurações do Ambiente

Para que os arquivos do projeto estejam disponíveis em todos os nós da rede, sem necessidade de atualizações do pacote a cada modificação no código, criaram-se pastas compartilhadas entre o sistema hospedeiro e as máquinas virtuais.

No sistema hospedeiro foi configurado um compartilhamento SAMBA, apontando a raiz do diretório do projeto (semelhante à estrutura do código no GitHub) como raiz do mapeamento de rede. Concedeu-se permissão de escrita e gravação neste compartilhamento e a partir da máquina virtual são executados os *scripts* de instalação e configuração da ferramenta.

Os módulos de instalação do projeto tratam as dependências, tanto no Linux quanto no Python, e por sua vez, os módulos de configuração do projeto certificam-se que o código estará disponível em seu diretório padrão `/etc/file-sync/` com os arquivos de configuração apenas contendo suas estruturas de dados.

7. Arquitetura da Solução

O projeto foi desenvolvido utilizando a linguagem de programação Python, mais especificamente em sua versão 3.8, executada em sistema Linux.

O projeto subdivide-se nos módulos de dados, entrada, saída, interno, api, cli e sincronia.

7.1. Módulo de Dados

O módulo de dados gerencia todas as interações de leitura e escrita nos arquivos de configuração do projeto.

Seu funcionamento inicial durante a execução consiste em realizar a leitura dos arquivos JSON e por meio de funções que retornem o conteúdo bruto dos dados contidos nos arquivos de configuração do projeto. Estes dados são consumidos pelo módulo de entrada.

Possui também um submódulo chamado `write`, que é responsável pela gravação dos dados nos arquivos de configuração do projeto.

A seguir, serão descritos os arquivos de dados que organizam a ferramenta.

7.1.1. Config.JSON

Arquivo geral de configuração das tarefas de sincronia.

Devem ser respeitados padrões de nome para correta leitura dos dados.

Nos campos onde caminhos de sistema são passados deve-se respeitar letras maiúsculas e minúsculas, em função das características do sistema operacional Linux que processa estas tarefas.

Tarefa Global

Caso algum dos parâmetros das tarefas de sincronia não sejam informados, existe uma tarefa configurada com os parâmetros globais aplicáveis à todas as demais tarefas subsequentes, caso algum valor não tenha sido informado.

Exemplo de configuração da tarefa global.

```
1            "GLOBAL": {
2                "NAME" : "SHARE",
3                "DESCRIPTION" : "SHARE DEFAULT.",
4                "AUTHOR" : "LBMELLO",
5                "ENVIROMENT" : "LINUX",
6                "SYNC_LEVEL" : "0",
7                "NODE" : "MASTER",
8                "SOURCE" : "/tmp/share",
9                "USER" : "root",
10               "DESTINY" : "/tmp/share",
11               "TIME": "DEFAULT"
12           }
```

7.2. Parâmetros aplicáveis às Tarefas de Sincronia

Representa os valores a serem configurados nas tarefas de sincronia. Várias tarefas podem ser configuradas no mesmo servidor, apenas criando uma nova entrada na lista *SHARE*.

- *NANE* – Nome do compartilhamento. Informação utilizada para organização das tarefas dentro do sistema. Pode utilizar caracteres alfanuméricos, diferenciando letras maiúsculas de minúsculas.
- *DESCRIPTION* – Descrição do compartilhamento. Informação utilizada para facilitar a identificação do conteúdo das tarefas de sincronia.
- *AUTHOR* – Autor da Tarefa. Identificação do operador que configurou a tarefa dentro do sistema.
- *ENVIROMENT* – (AINDA SEM USO) Sistema operacional onde a ferramenta será executada. Até o presente momento, só existe compatibilidade com ambientes Linux.
- *SYNC LEVEL* – (AINDA SEM USO) Nível de sincronia do nó dentro da árvore do compartilhamento. Isso definirá após a implantação, se o nó está iniciando o cenário de ondas ou recebendo os dados de seus antecessores.
- *NODE* – (AINDA SEM USO) Identificação do tipo do nó (*master* ou *client*).
- *SOURCE* – Diretório de origem dos arquivos. Aonde os arquivos serão localizados localmente para envio aos demais nós da rede. Utilizado também o padrão *case sensitive* para os diretórios.
- *USER* – Usuário SSH utilizado para cópia. Existe necessidade da existência deste mesmo usuário no *host* de destino, necessita também realizar troca de chaves SSH entre os sistemas para não existir necessidade de incluir a senha durante as sincronias de arquivos.
- *IP* – Endereço de IP da máquina destino.
- *DESTINY* – Diretório de destino dos arquivos na estação remota. Caso o diretório não exista no destino, o mesmo será criado.
- *TIME* – Configuração de tempo para a tarefa. Diretamente conectado com as janelas de tempo criadas no arquivo `Time.JSON`. O valor declarado nesse parâmetro deve sempre existir no arquivo `Time.JSON`.

```
1          "SHARE": [
2              {
3                  "ID" : "001",
4                  "NAME" : "TST01",
5                  "DESCRIPTION" : "Teste_do_Projeto",
6                  "AUTHOR" : "LBMELLO",
7                  "ENVIROMENT" : "LINUX",
8                  "SYNC_LEVEL" : "0",
9                  "NODE" : "MASTER",
10                 "SOURCE" : "/tmp/sharemod",
11                 "USER" : "fs",
12                 "DESTINY" : "/tmp/rsync/",
13                 "TIME" : "24x7&1HS"
14             }
```

7.3. Time.JSON

Arquivo onde são setadas as configurações das janelas de tempo utilizadas nas tarefas de sincronia.

As configurações de tempo criadas neste arquivo são mencionadas no arquivo `Config.JSON`, no item *TIME* para agendamento da tarefa de sincronia.

Utiliza-se uma sequência lógica entre os valores na montagem do padrão de tempo. Em *OPERATION-TYPE* definimos como queremos executar esta tarefa, em *FREQUENCY* definimos um número que indicará a quantidade de execuções ou repetições, em *TIMEUNITY* definimos uma unidade temporal para esta execução. Finalmente em *SCHEDULE* definimos quais dias da semana a tarefa é executada.

- *OPERATION-TYPE* – Tipo de operação de tempo, indicando a frequência de execução da tarefa. Este parâmetro define como o sistema tratará a execução da tarefa.

  – *EVERY* – Caso indicado em *EVERY* a tarefa será executada na modalidade "a cada", o que significa que o valor do campo *FREQUENCY* será considerado juntamente com o campo *TIMEUNITY* para definir a janela de tempo da tarefa.

  Ex.: Execução agendada a cada 10 minutos
  OPERATION-TYPE = EVERY
  FREQUENCY = 10
  TIMEUNITY = MINUTE

  – *JUST* – Caso indicado em *JUST* a tarefa será executada na modalidade "somente", que pode ser declarada para execuções pontuais, em determinados dias da semana ou horários específicos.
  Ex.: Execução agendada somente às quartas e sextas-feiras
  OPERATION-TYPE = JUST
  SCHEDULE = __W_F__

  – *ONCE* – Execução realizada somente uma vez, em uma data definida.

  – *SPECIAL* – Caso o operador special seja informado, o parâmetro *TIME_UNITY* poderá ser suprimido e os operadores especias podem ser usados. São eles: *HOURLY* (a cada hora), *DAILY* (diariamente), *WEEKLY* (semanalmente), *MONTHLY* (mensalmente) e *REBOOT* (a cada reiniciada do host).

- *FREQUENCY* – Frequência de execução. Declarado sempre com um numeral inteiro, pois isto definirá quantas vezes a tarefa será executada, dependendo dos demais parâmetros.

- *TIME_UNITY* – Unidade de tempo utilizada nas operações, pode ser *MINUTE, HOUR, DAY, WEEK e MONTH.*

- *SCHEDULE* – Dias da semana que a tarefa será executada, sempre deve ser declarado em maiúsculo, utilizando o padrão de nomenclatura em inglês.

```
        "LAB":{
            "NAME": "LAB",
            "SCHEDULE": "MTWTF__",
            "OPERATOR": "EVERY",
            "FREQUENCY": "5",
            "TIME_UNITY":  "MIN"
        }
```

Tempo Global

Assim como as tarefas de sincronia, existe uma entrada global de tempo cadastrada. Caso nenhum valor seja atribuído nas tarefas de tempo, os valores desta tarefa serão utilizados.

```
            "DEFAULT":{
                "NAME": "DEFAULT",
                "SCHEDULE": "MTWTFSS",
                "OPERATOR": "EVERY",
                "FREQUENCY": "3",
                "TIME_UNITY":  "HRS"
        }
```

7.4. Hosts.JSON

Arquivo onde os nós da rede devem ser declarados.

- *IP* – Endereço de IP IPV4 da estação/servidor sendo declarado.
- *DESCRIPTION* – Descrição da estação/servidor para facilitar a identificação daquele nó em inventário.
- *UID* – Identificação numérica única do nó.
- *SYNC-LEVEL* – Nível de Sincronia.
- *LEVEL-N* – Nome do nível de sincronia sendo declarado.
- *UID* – Grupo de UIDs pertencentes aquele nível de sincronia.

```
        "NODES": {
            "fs-01": {
                        "IP": "192.168.15.100",
                        "DESCRIPTION": "cliente fs-01",
                        "UID": "001",
                "EDGE": "002"
            }
```

7.5. Domain.JSON

Arquivo onde os dados de doínio são armazenados.

- *NAME* - Define-se o nome do domínio
- *DOMAIN_ID* - Define-se o identificador do domínio, listado com o comando `file-sync --domain list`

- *KNOWN_MEMBER* - IP de algum host conhecido, pertencente ao domínio declarado nos campos acima.

Exemplo do arquivo conf/Domain.JSON:

```
{
    "NAME": "lab_domain",
    "DOMAIN_ID": "169
       a01c91744946f7cb356577200b789",
    "KNOWN_MEMBER": "172.10.0.10"
}
```

7.6. Módulo de Entrada

Faz toda a entrada dos dados na ferramenta, todos provenientes dos arquivos de configuração do projeto, lidos pelo módulo de dados,

Essa sessão recebe o conteúdo dos arquivos do módulo de dados e cria os objetos que serão usados em todo o projeto. Estes objetos ficam disponíveis para todos os demais módulos através de variáveis globais declaradas durante sua execução.

7.7. Módulo de Saída

Divisão do projeto responsável por administrar as saídas de dados do projeto, como na geração de *logs*.

7.8. Módulo Interno

Módulo que administra a ferramenta em si, utilizando-se dos objetos lidos pela classe de dados e instanciados na classe de entrada. Dentre suas funções, este módulo cria os diretórios de configuração dentro de cada *share*, gerencia as tarefas agendadas no *crontab*, identifica quando existem alterações em arquivos, mantém uma relação atualizada do conteúdo de cada diretório e administra as relações SSH com os vizinhos.

7.8.1. Crontab

Classe criada no módulo `internal/cron.py`.

Através dos objetos de tempo da classe de entrada, interpreta os padrões de texto e converte estes dados para as entradas típicas do serviço *Crontab*, presente em sistemas Linux. Após interpretar os dados, cria as entradas de execução no arquivo *crontab* padrão para o usuário definido na tarefa de sincronia (arquitvo `Config.JSON`, na lista *[SHARE]* chave *[USER]*).

Nesta classe, é interpretado um padrão específico para entradas de tempo, definido dentro do projeto no arquivo `Time.JSON`, no campo *SCHEDULE*.

Este padrão utiliza sete posições referentes a cada dia da semana, ordenadas em uma sequência de caracteres, representando cada um, a primeira letra do nome daquele dia em inglês, de segunda-feira à domingo. Com isso, definem-se os dias da execução da tarefa de sincronia.

Ex.: `MTWTFSS`

A representação acima, realiza a execução em todos os dias da semana.

Caso seja necessário não realizar a tarefa de sincronia em algum dos dias, basta alterar o padrão da letra referente ao dia, por um *underline* (_).

Ex.: `MT_TFSS`

O exemplo acima, não realiza a execução às quartas-feiras, pois o *underline* foi colocado na terceira posição da lista, referente à letra W (***W**ednesday* = quarta-feira).

7.8.2. SSH

Classe criada no módulo `internal/ssh.py`.

Gerencia as relações SSH entre os *hosts*. Cria e troca as chaves criptográficas entre os membros da rede.

7.8.3. Domínio

Classe criada no módulo `internal/domain.py`.

Organiza às relações de confiança entre os *hosts* que trocam arquivos e dados sobre os compartilhamentos. Os servidores precisam pertencer ao mesmo domínio para existir relação de confiança.

7.8.4. Gerência de Pastas

Classe criada no módulo `internal/folder.py`.

Sua função principal é criar a pasta oculta do projeto, dentro do diretório de cada *share*. Esta pasta armazena os dados de alteração dos arquivos daquele compartilhamento.

7.8.5. Alterações em Arquivos

Classe criada no módulo `internal/changes.py`.

Captura toda a árvore de diretórios e arquivos do compartilhamento, através da função `walk()` do pacote `os`, padrão ao Python.

Para cada arquivo e pasta dentro desta árvore, serão gerados *hashes* MD5, que acompanhadas da data de alteração do arquivo/pasta, compões o arquivo `CHANGES.JSON`, contido na pasta oculta do projeto, dentro do compartilhamento.

7.8.6. Localhost

Classe criada no módulo `internal/localhost.py`.

A classe é solicitada quando existe necessidade de cadastrar ou atualizar a informação na ferramenta referente ao próprio servidor que executa a ferramenta. Estes dados ficam contidos no arquivo `Hosts.JSON`, em uma chave especial inserida no início da lista de *NODES*.

É executada, na maioria das vezes, quando o registro de *localhost* não existe em `Hosts.JSON`. Quando os arquivos de configuração da ferramenta não contém dados.

7.8.7. Sincronia

Classe criada no módulo `internal/sync.py`.

Com os dados repassados das demais classes, realiza as sincronias através do *rsync*, utilizando *subprocess* do Python para execução no *host*.

7.9. API

Também é um pacote de entrada de dados, mas para evitar erros na linguagem de *circular import* (erro gerado pelo python quando se realiza a importação de um módulo dentro de outro módulo do mesmo pacote, gerando um *loop* na execução) com os módulos internos, optou-se pela separação deste pacote.

Gerencia as interações de cliente e servidor entre os *hosts*. Recebe os dados de alterações em arquivos dos compartilhamentos nos vizinhos, permite manipular *hosts*, compartilhamentos e entradas de tempo, além de todas realizar todas as validações de domínio entre os *hosts*.

7.10. CLI

É o pacote que organiza a integração da ferramenta com o operador via linha de comando. É divididos em alguns sub-módulos, que fazem a interação com as diferentes sessões possíveis do módulo CLI.

7.10.1. sync

Gerencia a sessão de sincronia da ferramenta.

Inicia as sincronias de arquivos para todos os *shares*, com o comando:

```
file-sync --sync all
```

Ou inicia somente a sincronia do *share* especificado:

```
file-sync --sync NOME-DO-SHARE
```

7.10.2. install

Faz a instalação das dependências (Python e ambiente virtual) e instala a ferramenta em si no *host*.

Instalação do Python3 mais dependências:

```
file-sync --install python3
```

Instalação da ferramenta:

```
file-sync --install
```

7.10.3. crontab

Gerencia o módulo cron da ferramenta.

Cria as entradas de execução no *Crontab* para todos os *shares*, com o comando:

```
file-sync --crontab all
```

Ou cria somente as entradas de execução no *Crontab* do *share* especificado:

```
file-sync --crontab NOME-DO-SHARE
```

7.10.4. config

Gerencia os dados contidos no arquivo Conf.JSON.

Adiciona uma nova configuração de *share*, solicitando os dados necessários via formulário na sequência de execução do seguinte comando:

```
file-sync --config add
```

7.10.5. domain

Administra as interações de domínio, criando um novo domínio e listando as informações do domínio já criado.

Cria domínio novo:

```
file-sync --domain create
```

Lista ID do domínio existente:

```
file-sync --domain list
```

8. Considerações Finais

Buscando facilitar a sincronia de arquivos entre estações de trabalho, foi desenvolvida a ferramenta file-sync, que oferece uma interface simplificada para o operador criar suas tarefas de sincronia.

Com o desenvolvimento atual da ferramenta, é possível criar as tarefas de sincronia de arquivos agendas em um local centralizado e padronizado, conforme a devida necessidade do usuário.

A fim de complementar o funcionamento da ferramenta, serão desenvolvidos os seguintes itens futuramente:

8.1. Envio de dados apenas com alterações entre compartilhamentos

Atualmente a ferramenta gera a árvore de diretórios e arquivos com suas respectivas *hashes* de alteração de arquivos. A ideia é que estes dados sejam trocados entre os nós para comparação do estado de edição dos arquivos. Caso existam diferenças entre a origem e o destino a sincronia será ativada.

8.2. Execução da API Desacoplada do Código

A implementação do módulo de API realiza a execução do *framework Flask* em modo de *debug*, dentro do próprio código. Este tipo de implantação é utilizada somente em ambiente de testes, pois impacta em diversos parâmetros e recursos do *framework* ficarem limitados. Em função dessa implementação, existe bloqueio de entrada via *Shell*, impossibilitando o uso do módulo CLI. A ideia em estudo para o futuro é implementar o módulo API desacoplado da execução do projeto, utilizando um servidor web padrão com *fastCgi* ou um *container Docker* configurado com o ambiente.

8.3. Módulo de Redes

Futuramente será desenvolvido um módulo que testa a qualidade da conexão com os demais *hosts* para identificar os melhores nós possíveis para troca de dados. Estes testes de rede irão considerar se o servidor remoto responde às conexões, se tem boa latência e *jitter* baixo e quantos saltos são realizados até o destino. A informação do melhor nó para troca estará disponível no campo *EDGE* do arquivo `Hosts.JSON`.

Referências

[Astrand 2003] Astrand, P. (2003). Pep 324 - python-subprocess. https://www.python.org/dev/peps/pep-0324.

[Conservancy 2020a] Conservancy, S. F. (2020a). Git. https://git-scm.com/book/pt-br/v2/.

[Conservancy 2020b] Conservancy, S. F. (2020b). Git - histórico. https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git.

[Foundation 2020a] Foundation, P. S. (2020a). fnmatch — unix filename pattern matching. https://docs.python.org/3/library/fnmatch.html.

[Foundation 2020b] Foundation, P. S. (2020b). json — json encoder and decoder. https://docs.python.org/3/library/json.html.

[Foundation 2020c] Foundation, P. S. (2020c). sys — system-specific parameters and functions. https://docs.python.org/3/library/sys.html.

[GoogleInc 2014] GoogleInc (2014). The javascript object notation (json) data interchange format. https://tools.ietf.org/html/rfc7159.

[Kurose and Ross 2014] Kurose, J. F. and Ross, K. W. (2014). *Redes de computadores e a internet uma abordagem top-down*. Pearsons, Avenida Santa Marina, 1193 - São Paulo.

[Meyer 2011] Meyer, C. (2011). Pep 405 – python virtual environments. https://www.python.org/dev/peps/pep-0405/.

[Owens 2020] Owens, M. (2020). python-crontab 2.5.1. https://pypi.org/project/python-crontab/.

[Pallets 2010] Pallets (2010). Flask - userguide. https://flask.palletsprojects.com/en/1.1.x/.

[Picon et al. 2015] Picon, F., Karam, R., Breda, V., Restano, A., Silveira, A., and Spritzer, D. (2015). Precisamos falar sobre tecnologia. http://rbp.celg.org.br/detalhe_artigo.asp?id=177.

[Python 2020] Python (2020). Python. https://docs.python.org/pt-br/3/faq/general.html#what-is-python.

[Tanenbaum and Bos 2015] Tanenbaum, A. S. and Bos, H. (2015). *Sistemas Operacionais Modernos*. Pearsons, Avenida Santa Marina, 1193 - São Paulo.

[Tanenbaum and Wetherall 2011] Tanenbaum, A. S. and Wetherall, D. J. (2011). *Redes de Computadores*. Pearsons, Avenida Santa Marina, 1193 - São Paulo.

[Tridgell and Mackerras 1996] Tridgell, A. and Mackerras, P. (1996). The rsync algorithm. https://openresearch-repository.anu.edu.au/bitstream/1885/40765/3/TR-CS-96-05.pdf.

[Welzl 2005] Welzl, M. (2005). *Network Congestion Control - Managing Internet Traffic*. John Wiley Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England.