Diretrizes de Gerência de Configuração

1. Introdução

Este documento registra as diretrizes de Gerência de Configuração do projeto do Sistema Sucuri Analytics.

2. Modelo de Ramificação

O modelo de ramificação a ser adotado para os repositórios de código será o *Git Flow,* devido ao nível de maturidade da equipe e diversidade de *branches* que oferece.

3. Gerenciamento de versões

3.1. Branches

O padrão de nomenclatura de *branches* no modelo será: `<tag da branch/nome-da-branch>`

O nome da *branch* deve seguir o *kebab-case*. As *tags* padrões de *branch* para o projeto são:

feature	Ramificação para desenvolvimento de uma funcionalidade nova do sistema.
bugfix	Ramificação para resolução de bugs do sistema.
release	Ramificação que contém as versões estáveis do sistema.
develop	Ramificação que contém as versões em desenvolvimento do sistema.
main	Ramificação que contém as versões estáveis e implantáveis do sistema.

3.2. Commits

Estrutura do *commit*: `<*tag* primária>: <Mensagem do *commit*>`

As *issues* podem ser referenciadas dentro da mensagem de *commit*, de modo que se possa até fechá-las com a própria mensagem.

Exemplo: *fix*: Padronização de *unicodes close* #29 e Apagamento dos *logs close* #30

Palavras chaves (fecham, editam *issues*): *close, closes, closed, fixes, fixed, resolve, resolves, resolved.*

build	Alterações que afetam o sistema de construção ou dependências externas. Exemplos de escopos: gulp, broccoli, npm;	*perf*	Uma alteração de código que melhora o desempenho;
ci	Altera a configuração de documentos e scripts de integração contínua (CI). Exemplos de escopos: Travis, Circle, BrowserStack, SauceLabs;	*refactor*	Tipo utilizado em quaisquer mudanças que sejam executadas no código, porém não alteram a funcionalidade final da tarefa impactada;
docs	Referem-se a inclusão ou alteração somente de arquivos de documentação;	*style*	Alterações referentes a formatações na apresentação do código que não afetam o significado do código, como por exemplo: espaço em branco, formatação, ponto e vírgula ausente, etc.;
feat	Tratam adições de novas funcionalidades ou de quaisquer outras novas implantações ao código;	*test*	Adicionando testes ausentes ou corrigindo testes existentes nos processos de testes automatizados (TDD);
fix	Essencialmente define o tratamento de correções de bugs;	*env*	Utilizado na descrição de modificações ou adições em arquivos de configuração em processos e métodos de integração contínua (CI), como parâmetros em arquivos de configuração de contêineres.
chore	Alterações que não modificam arquivos de código, como por exemplo, mudanças no *gitignore*.		

3.3. Pull requests

O código será mergeado se, e somente se, aprovado por ao menos dois integrantes, preferencialmente que não estejam envolvidos diretamente com o desenvolvimento e produção.

4. Gerenciamento de mudanças

4.1. Issues

O título da *issue* deve ser sucinto e explicativo. Criar uma *issue* para cada questão/problema, de forma a modularizar adequadamente o trabalho, com uma descrição clara e precisa. Caso seja uma issue de *bug*, é obrigatório uma demonstração do erro na descrição da *issue*, seja por foto ou vídeo de execução.

É imprescindível a manutenção do *status* da *issue* atualizado.

nome da *tag*	descrição da *tag*	cor	nome da *tag*	descrição da *tag*	cor
bug	Algo não está funcionando.	vermelha	*help wanted*	Atenção extra é requerida.	verde
documentation	Melhorias e/ou adições na documentação.	azul	*invalid*	Não parece correto.	amarelo
duplicate	Essa *issue* ou *pull request* já existe.	cinza	*question*	É necessário informação adicional/complementar.	rosa
enhancement	Nova funcionalidade ou pedido de funcionalidade.	verde claro	*wontfix*	Essa *issue* não será trabalhada no momento.	marrom

5. Plataforma de integração contínua e entrega contínua (CI/CD)

5.1. GitHub Actions

A ferramenta *GitHub Actions* será utilizada para a Integração e Entrega Contínua por ser completamente integrada ao *GitHub*, eliminando a necessidade de criar e configurar uma infraestrutura externa com outras ferramentas de CI e CD.

A ferramenta de automação também possui uma comunidade muito ativa que oferece milhares de ações pré-construídas, além de empresas como a *Microsoft* e o próprio *GitHub* que contribuem com seus repositórios. As *actions* são *open-source* e funcionam em qualquer plataforma, linguagem e nuvem. Além disso, também rodam no *Linux*, *macOS*, *Windows* e contêineres.

Fluxos de trabalho

São processos automatizados configuráveis que executam um ou mais trabalhos. São definidos por um arquivo YAML que deve estar no repositório no diretório `.github/workflows` e que serão executados quando acionados por um evento, manualmente ou de acordo com um cronograma previamente definido.

Um repositório pode ter fluxos de trabalho diferentes para:

- Testar *pull requests.*
- Implantar o aplicativo toda vez que uma nova versão for criada.
- Adicionar uma etiqueta toda vez que alguém abre uma nova *issue*.

Arquivo de fluxo de trabalho

nome(opcional)	**Nome do fluxo de trabalho a ser exibido na aba "Ações" do repositório.**
run-name	Nome das execuções do fluxo de trabalho.
on	Especifica o evento/gatilho para o fluxo de trabalho.
jobs	Agrupa todos os trabalhos executados no fluxo de trabalho.
runs on	Determina o executor para o trabalho.
steps	Agrupa todas as tarefas que são executadas no trabalho.
uses	Especifica qual ação será executada durante a etapa.
run	Instrui o trabalho a executar um comando no executor (pode ser mais de um).

Para incluir ações em um fluxo de trabalho basta adicioná-las como novas etapas dentro de *jobs*. Atentar-se às entradas que as ações podem exigir, que por sua vez devem ser definidas no fluxo de trabalho.

Para adicionar uma ação de um repositório diferente é necessário incluir a referência `{owner}/{repo}@{ref}` em *uses*, especificando também a versão da ação (pode ser uma tag, um SHA ou *branches*).

6. Convenções de Código

Na presente seção do documento, definem-se convenções de código - baseadas nos PEP 8, PEP 20, PEP 257, PEP 484, PEP 526 e PEP 3131 - a serem seguidas durante o desenvolvimento do projeto.

6.1. Disposição de Código

Para cada nível de recuo, devem-se utilizar 4 espaços (e não tabulações). É necessário realizar o alinhamento de elementos agrupados com as linhas de continuação, por meio da junção de linha implícita do Python entre parênteses, colchetes e colchetes com recuo deslocado.

Quanto ao uso de recuo deslocado, deve-se atentar para a não disposição de argumentos na primeira linha e para o uso do recuo adicional para fazer a distinção de linha de continuação.

Para condicionais *if* com uma grande quantidade de instruções, deve-se adicionar indentação adicional nas linhas de continuação e realizar a quebra antes do operador binário.

A chave/colchete/parêntese de fechamento em construções de várias linhas deve ser alinhada sob o primeiro caractere sem espaço em branco da última linha da lista. O comprimento de uma linha deve-se ater a no máximo 79 caracteres. Caso a linha seja de docstring ou comentário, está limitada a 72 caracteres. Deve-se utilizar a continuação de linhas implícitas de Python (feita entre parênteses, colchetes e chaves) para quebrar linhas longas.

6.2. Linhas em Branco

Funções de nível superior e definições de classe devem ser envolvidas com duas linhas em branco. Definições de métodos dentro de uma classe, por sua vez, devem estar cercadas por uma única linha em branco.

Para separar grupos de funções relacionadas, devem-se utilizar duas linhas em branco, dispensáveis caso sejam one-liners relacionados.

Para indicar seções lógicas, utilize uma linha em branco.

6.3. Codificação do Arquivo de Origem

O código na distribuição principal do Python deve estar em UTF-8, sem declaração de codificação.Faça uso de caracteres não ASCII apenas para denotar nomes humanos e lugares. Caracteres Unicode ruidosos como zalgo e marcas de ordem de byte são vetados.

Todos os identificadores na biblioteca padrão do Python devem usar identificadores somente ASCII e estar em inglês sempre que possível.

6.4. Importações

As importações devem ser feitas no início do arquivo, após comentários/docstrings do módulo e antes de de constantes e globais do módulo. Primeiro são feitas as importações de bibliotecas padrão, seguidas das de terceiros relacionadas e, por fim, as importações específicas de bibliotecas locais. Cada grupo de importação deve ser separado por uma linha em branco.

Cada importação por *import* deve ser feita em linha separada. Considerando o uso de *from* | *import*, pode-se importar mais de um componente na mesma linha. Prefere-se o uso de importações absolutas devido a sua legibilidade e melhor comportamento. As importações relativas explícitas devem ser empregadas se, e somente se o pacote for extremamente complexo e a importação absoluta seja desnecessariamente detalhada.

Deve-se evitar importações curinga () e usar apenas importação absoluta para bibliotecas padrão.Importações futuras devem aparecer antes de qualquer código (inclusive nomes *Dunder*), exceto *docstrings*.

6.5. Nomes *Dunder*

Dunders de nível de módulo (ou seja, nomes com dois sublinhados à esquerda e dois à direita), como __all__, __author__, __version__, etc., devem ser colocados após a docstring do módulo, mas antes de qualquer instrução de importação, exceto importações.

6.6. Aspas

Deve-se utilizar apenas aspas duplas em *strings*, salvo a exceção de métodos ou classes de bibliotecas externas e dependências que exigem aspas simples.

6.7. Espaço em branco em expressões e instruções

Não faça uso de espaço em branco nas seguintes situações:

1. Imediatamente dentro de parênteses, colchetes ou chaves.
2. Entre uma vírgula à direita e um parêntese de fechamento seguinte.
3. Imediatamente antes de uma vírgula, ponto e vírgula ou dois pontos.
4. No entanto, considerando um fatiamento, os dois pontos atuam como um operador binário e devem ter valores iguais em ambos os lados (tratando-o como o operador com a menor prioridade). Em uma fatia estendida, ambos os dois-pontos devem ter o mesmo espaçamento aplicado. Exceção: quando um parâmetro de fatia é omitido, o espaço é omitido.
5. Imediatamente antes do parêntese aberto que inicia a lista de argumentos de uma chamada de função.
6. Imediatamente antes do parêntese aberto que inicia uma indexação ou divisão.
7. Mais de um espaço ao redor de um operador de atribuição (ou outro) para alinhá-lo com outro.
8. Espaços em branco à direita em qualquer lugar.
9. Espaços ao redor do = quando usado para indicar um argumento de palavra-chave ou quando usado para indicar um valor padrão para um parâmetro de função não anotado

Sempre coloque um espaço em cada lado de operadores binários (atribuição, atribuição aumentada, comparações, Booleanos). Se operadores com prioridades diferentes forem usados, adicione um espaço em branco em ao redor de operadores com prioridades distintas -mantenha sempre a mesma quantidade em ambos os lados do operador.

As anotações de função devem usar as regras normais para dois-pontos e sempre ter espaços ao redor da -> seta (se existir). Ao combinar uma anotação de argumento com um valor padrão, no entanto, use um espaço ao redor do sinal de igualdade. Não faça uso de declarações compostas (várias instruções na mesma linha), salvo uso para um corpo pequeno de if/for/while.Declarações compostas (várias instruções na mesma linha) geralmente são desencorajadas.

6.8. Nomenclatura

Os nomes devem ser representativos, ou seja, devem ser capazes de transmitir o que representam. Evite nomes genéricos - como "temp", "data", "valor" - por dificultarem a compreensão. Utilize apenas caracteres alfanuméricos e sublinhados (_). Não faça uso de caracteres especiais.
A convenção de nomenclatura *snake_case* será utilizada para a nomenclatura de Variáveis,Módulos, Funções e Constantes na separação de palavras em nomes compostos. Variáveis, Módulos,Funções devem ter nomes em letras minúsculas. Constantes são nomeadas com letras maiúsculas (convenção SCREAMING_SNAKE_CASE ou MACRO_CASE). Classes utilizam a convenção PascalCase (também conhecida como CapWords ou UpperCamelCase). Recomenda-se evitar o uso de afixos desnecessários para o nome das classes. Pacotes são escritos em letras minúsculas e não devem ter nomes muitos generalizados, no intuito de evitar conflitos entre pacotes.

6.9. Comentários

Comentários devem ser adicionados para facilitar a compreensão do código. Devem ser simples e organizados de modo a manter a legibilidade do código. Apenas comentar onde é preciso uma explicação.
Comentários devem existir para explanar trechos complexos do código ou para fornecer informações adicionais se necessárias. Devem se manter atualizados, acompanhando as mudanças e versionamento do código. Comentários em bloco devem seguir o mesmo nível de recuo do código ou conjunto de código a qual se refere. Cada linha do comentário de bloco começa com um # e também um único espaço (a menos que o texto dentro do comentário possua recuo). Parágrafos dentro do bloco devem ser separados por uma linha com um único #. Comentários embutidos devem ser utilizados apenas se necessário e se contiverem uma informação de fato relevante. São fortemente desencorajados por, na grande maioria dos casos, reiterar o óbvio e apenas causar distrações. São escritos na mesma linha de instrução, separados por dois espaços de recuo e iniciados por # e um espaço em branco.

6.10. Docstrings

Classes, métodos, funções e módulos públicos devem ser iniciados por *docstrings*. Métodos não públicos não precisam de docstrings desde que estejam acompanhados de um comentário que descreva sua função. Se é multilinha, seu delimitador `"""` deve estar sozinho na linha final. Caso contrário, o delimitador deve se manter na mesma e única linha. Sempre utilize triplas aspas duplas como delimitador.

7. Histórico de versões do documento

A presente seção apresenta o histórico de versões desse documento.

Versão	Publicação	Autor(es)	Ações realizadas
1.0	26/04/2023	Lucas Henrique Alves Borth, Júlia Alves Corazza, Murilo Matias e Pedro Rodrigues Arantes	- Versão inicial das diretrizes de Gerência de Configuração do projeto.

1.1	17/05/2023	Lucas Henrique Alves Borth, Júlia Alves Corazza, Murilo Matias e Pedro Rodrigues Arantes	- Delimitação das Diretrizes de Código. - Nome da Aplicação