Este trabalho é um projeto da disciplina de Engenharia de Software II da Universidade Federal de Minas Gerais, Departamento de Ciência da Computação. Feito pelos alunos Cláudio Manuel, Tauany Santos e Wallace Augusto, em Maio de 2017.
O objetivo do projeto é fazer uma documentação de um sistema open source, hospedado no GitHub e que seja popular (com mais de 1000 estrelas).
Foi escolhido o sistema StackStorm, uma plataforma que realiza a integração entre serviços de padrões operacionais comum.

1. Descrição do Sistema

1.1. Características e objetivo

O StackStorm é uma plataforma de automação de código aberto que realiza a integração entre serviços e ferramentas de padrões operacionais comuns. O foco principal da plataforma é tomar ações em resposta a eventos que ocorrem. Além disso, tem por objetivo proporcionar aos seus usuários uma forma organizada de conectar todos os aplicativos, serviços e fluxos de trabalho que ele utiliza de uma forma mais simples, tornando mais facilmente esse ambiente automatizado.
A plataforma é composta por Sensors, Triggers, Actions, Rules, Workflows, Packs e Audit Trail, todos estes componentes em conjunto realizam a conexão dos serviços e fluxos de trabalho.

- Os Sensors são plugins que fazem a integração da entrada e saída de eventos.
- Os Triggers são representações de eventos externos. É possível definir um novo disparador, para isso é necessários criar um novo plugin de sensor.
- As Actions são plugins na linguagem Python ou qualquer outro script. Existem ações genéricas, por exemplo “ssh”, ou ações personalizadas. Essas ações podem ser chamadas diretamente pelo usuário, através de uma API, por exemplo. Além disso, podem ser usadas e chamadas como regra e fluxos de trabalho (tópicos que estão explicados abaixo).
- As Rules desencadeiam ações. Elas aplicam critérios de correspond
  ência e fazem um mapeamento para definir como será a entrada de ações.
- Os Workflows definem a ordem e as condições de transações de dados. Em geral, são necessárias mais de uma ação para uma automação, quando ela possui mais de uma etapa. Os fluxos de trabalho podem ser chamados manualmente ou por regras.
- Os Packs são unidades de implantação de conteúdo utilizadas para simplificar o gerenciamento e compartilhamento de conteúdos.
- A Audit Trail de execução de ações pode ser manual ou automática, o usuário pode definir e a partir disso, é gravada e armazenada a ação, especificando o contexto e o resultado da execução.

O conteúdo que é armazenado na plataforma é todo em código. Além disso, os Packs, que são as unidades de implantação de conteúdo podem ser criados e compartilhados no GitHub ou enviados para o repositório da comunidade StackStorm.
A licença utilizada pelo projeto é a Licença Apache que pode ser lida no link abaixo:
Http://www.apache.org/licenses/LICENSE-2.0

1.2 Linguagem e Ferramenta

A linguagem principal utilizada na plataforma é o Python que é a linguagem utilizada para a criação dos plugins de Sensors e também para os plugins de Actions. A linguagem Python é uma linguagem de programação de alto nível, de script, imperativa e orientada a objetos. Além de Python, outras linguagens são utilizadas nas contribuições de Packs, como por exemplo a linguagem Shell e o Javascript.
Shell é uma linguagem interpretada, de script que é usada em diversos sistemas operacionais com diversos dialetos. Assim como o Python, Javascript é uma linguagem de script, orientada a objetos e é utilizada para conectar os objetos do ambiente.
Uma ferramenta que é utilizada na plataforma é a “CloudSlang” que faz o gerenciamento de aplicativos implantados. O objetivo do uso desta ferramenta é automatizar rapidamente as operações de TI diárias e definir um workflow estruturado e fácil de entender.
A “CloudSlang”, é um DSL que se baseia em uma linguagem YAML, que é utilizada para realizar consultas complexas no banco de dados. Os conteúdos da CloudSlang são divididos em dois tipos principais que são as operações e os fluxos. Estas operações podem ser escritas em duas linguagens diferentes, que são Java e Python.
Um outro mecanismo utilizado é o Jinja2 que é um modelo completo para a linguagem Python. Ele é utilizado para gerenciar os workflows da plataforma.

2. Equipe de Desenvolvimento

O projeto StackStorm é uma plataforma de automatização que possui uma versão open-source com suporte da comunidade. Seu desenvolvimento e construção se fez através de uma comunidade no Slack. O Slack funciona como uma rede social, através dele, qualquer pessoa pode criar uma sala de bate-papo, com múltiplos canais, dentro de uma empresa, com assuntos distintos e para grupos diferentes. Dentro de cada um desses canais, a ferramenta permite troca de mensagens de texto, documentos e compartilhamento de mídias, como fotos e vídeos.
Tanto o team principal quanto os diversos usuários e desenvolvedores utilizam essa ferramenta para desenvolver a plataforma StackStorm, que, no caso, já são mais de 200 pessoas envolvidas. Abaixo, a sua equipe de fundação:

Dmitri Zimine
Chief Stormer & Co-Fundador
Ajudou a liderar a primeira onda de automação de operações enquanto servia como arquiteto-líder e chefe de engenharia na Opalis.

Winson Chan
Stormer
Focado no desenvolvimento de produtos. Altamente qualificado na concepção e desenvolvimento de autoatendimento, soluções automatizadas e gerenciamento de pilhas VM na nuvem híbrida da empresa.

Patrick Hooboom
Stormer
Stormer focado em DevOps. Tem como objetivos alavancar as ferramentas de gerenciamento de configuração e criar scripts personalizados para melhorar os processos em toda a organização.

Lakshmi Kannan
Stormer
Responsável por construir uma plataforma previsível, de melhor performance e utilizável. Anteriormente, trabalhou na Amazon e Rackspace construindo sistemas altamente distribuídos e de baixa latência.

Manas Kelshikar
Stormer
Desenvolvedor da plataforma. Trabalhos anteriores na VMware como membro da equipe de clientes vSphere, onde contribuiu para a arquitetura e implementação do produto de nova geração.

Tomaz Muraus
Stormer
Auxilia na construção da plataforma. Trabalhou anteriormente na Cloudkick, Rackspace e DivyCloud como desenvolvedor e operando sistemas altamente distribuídos.

3. Evolução do sistema

Nesta seção, serão descritas as principais releases da plataforma StackStorm, retirados dos log de releases do projeto no git e do blog de desenvolvedores. Releases de teste, não estáveis, como beta e alpha, não serão descritas.

- **v0.8.0 - Disponibilizada em 3 de Março de 2015**
  Primeiro release do StackStorm, com 143 *commits*, desenvolvido por 2 meses. StackStorm 0.8 possui uma interface web para usuário que facilita o uso da plataforma. Esta interface possui 3 visões: a página Actions lista todas as ações registradas na instalação da plataforma; A página History exibe todas as ações que estão sendo executadas em real-time e seus status, além de todas as ações anteriormente executadas; A página Rules indica todas as regras registradas que podem ser executadas pela interface.
- **v0.9.1 - Disponibilizada em 13 de Maio de 2015**
  Após várias correções de *bugs* e com 7159 *commits*, esta versão foi apresentada com 2 melhorias: criação da opção de ignorar checagem de Certificados SSL e retorno de HTTP BAD REQUEST quando requerido token TTL.
- **v0.11 - Disponibilizada em 5 de Junho de 2015**
  Com 6689 *commits*, esta versão apresentou melhorias na configuração CLI da plataforma, como a criação de comandos *get, list* e *re-emit*. Algumas outras melhorias no funcionamento da API.
- **v0.13 - Disponibilizada em 24 de Agosto de 2015**
  Após 5542 *commits*, esta versão revelou a construção de uma autenticação no backend da plataforma, além de aperfeiçoamentos na estrutura da

plataforma. Também foram executadas melhorias na configuração CLI, com a criação de novas *features.*

- **v1.1.0 - Disponibilizada em 27 de Outubro de 2015**
  Adicionado YAQL v1.0 para suporte do Mistral, já que as versões anteriores foram descontinuadas. Updates para os comandos da CLI. Nova autenticação no backend da plataforma para servidores LDAP. Melhorias na API, tais como chaves que não expiram como tokens de autenticação. Adicionada opção de verificação de certificado SSL para requisição HTTPS. Correção de *bugs* na execução do Paramiko SSH. Total de 4364 *commits*.
- **v1.2.0 - Disponibilizada em 8 de Dezembro de 2015**
  Com 3866 *commits,* esta versão apresenta correção de *bugs* na execução do Paramiko SSH, tais como conexão com parâmetros ao seu alvo. Melhorias na API da plataforma, como por exemplo na criação de validação de parâmetros durante a criação de novas regras. Nova *feature* para o campo de notificações. Classes mock adicionadas aos *scripts* para facilitar testes.
- **v1.3.0 - Disponibilizada em 22 de Janeiro de 2016**
  Posteriormente, 3390 *commits* foram feitos a esta versão, que trouxe a introdução de novos parâmetros para a instalação da plataforma. Além de novas *features,* como novas ações em Trace e em uma lista de regras. Suporte para objetos presentes no banco de dados. Adição de *flag*s ao *script.* Melhorias na utilização das classes mock.
- **v1.4.0 - Disponibilizada em 18 de Abril de 2016**
  Depois de 2717 *commits,* várias melhorias foram apresentadas, como melhorias no suporte da execução SSH da plataforma, através de ações de autenticação de senha, suporte a porta padrão (22), entre outros. Aperfeiçoamentos no ChatOps e na API. Acesso ao banco de dados através da utilização de Python. Novas *features,* como operadores *regex* e *iregex.* Adição de novas *flags* no script.

- **v2.1.0 - Disponibilizada em 5 de Dezembro de 2016**
  Poucos *commits* realizados, apenas 1016, porém novas *features* foram apresentadas nessa nova versão, tais como a adição de novos comandos CLI e a adaptação da mesma com JSON. Melhorias no desempenho da API no tratamento de exceções. Melhorias nos packs, para consistência.
- **v2.2.0 - Disponibilizada em 25 de Fevereiro de 2017**
  Correção de vários *bugs* na API. Validações nas opções de configuração de criação de regras no sistema. Correções apropriadas nas políticas de cancelamento de ações.
- **v2.2.1 - Disponibilizada em 3 de Abril de 2017**

Versão atual, com apenas 576 *commits,* que apresenta melhorias em seu *script*, para correção de erros padrões. Correção de vários *bugs,* como configurações na API ou suporte de bibliotecas para testes.

4. Arquitetura e Componentes do Sistema

O StackStorm é um serviço que possui uma arquitetura modular. Ele compreende componentes de serviços acoplados que se comunicam por meio de barramentos de mensagens e escala horizontalmente para enviar automaticamente em escala. Para isso, o StackStorm utiliza REST API, que é um meio de providenciar a troca e o uso de informações entre os sistemas computacionais na Internet. Ele também utiliza CLI client, que é uma interface para linha de comando, para administradores e usuários operarem localmente ou remotamente o sistema. Além disso, o StackStorm utiliza a biblioteca Python client para a conveniência do desenvolvedor. Ainda não possui uma interface Web mas em breve estará disponível.
O diagrama a seguir mostra a arquitetura do sistema, incluindo seus componentes:

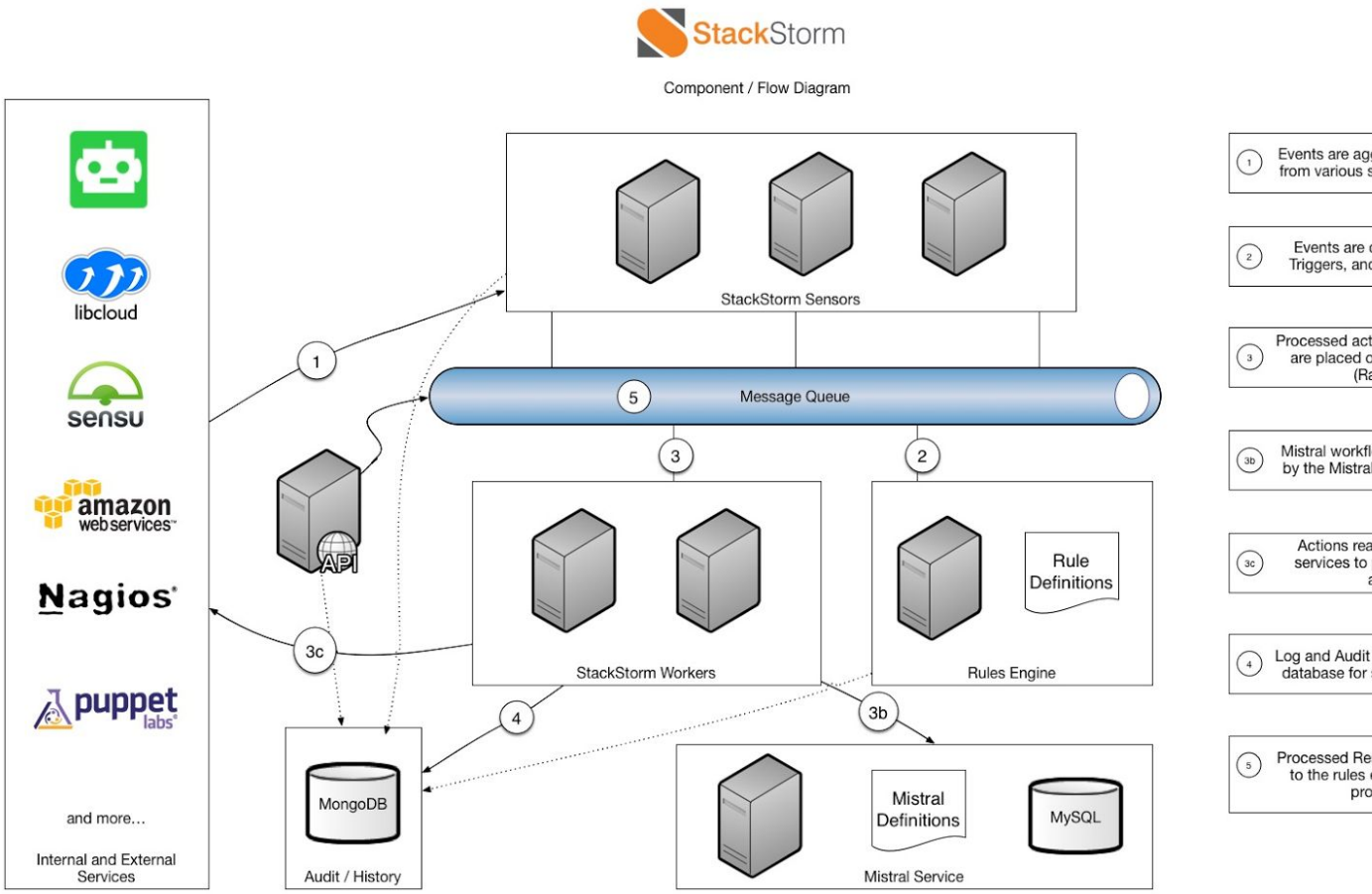


Diagrama arquitetural do StackStorm

1. Os eventos são agregados, via Push/Pull, de vários serviços por meio de Sensors;

2. Events são comparados com Triggers e geram Actions;
3. As ações processadas dos workflows são colocadas na lista de mensagem (RabbitMQ);

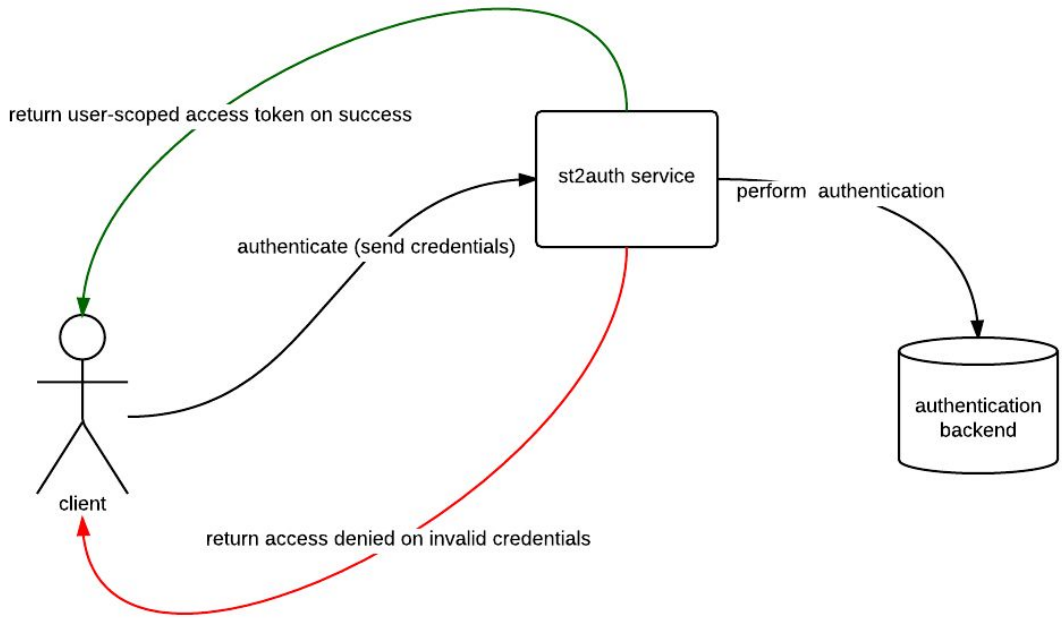
3b. Mistral workflows são processados pelo Mistra Service (Opicional);

3c. Actions alcançam vários serviços para executarem ações do workflow;

4. O Log e o Audit History são armazenados no banco de dados (MongoDB);
5. Processed Results são enviados de volta para o mecanismo de regras para processamento adicional.

Os seguintes componentes foram discutidos na primeira parte desse manual: Sensors, Triggers, Actions, Rules, Workflows, Packs e Audit trail.

StackStorm implementa o controle Role Based Access (RBAC) que permite administradores e operadores do sistema a restringirem o acesso do usuário e limitar as operações que podem ser realizadas. Além disso, o banco de dados operador pode ter acesso apenas ao banco de dados relacionado às ações que serão efetuadas.
Quanto à autenticação do sistema, é incluso um serviço de autenticação responsável pela autenticação do usuário e gerar o token de tempo limite de acesso. Quando o modo de autenticação está ativo (modo default), esse token de acesso é utilizado para autenticar junto ao StackStorm REST APIs. A imagem a seguir mostra o funcionamento da autenticação do sistema:



Serviço de autenticação do StackStorm

4.1. Padrões de código

A maioria dos repositórios utilizam arquivos de configuração compartilhados do Flake8 e PyLint. O StackStorm segue alguns padrões que devem ser utilizados durante o desenvolvimento do código. São eles:

- PEP8 Python Style Guide;
- 4 espaços para o tab;
- No máximo 100 caracteres em uma linha;
- Arquivos editados não devem conter espaço em branco à esquerda;
- Todos os arquivos fonte devem conter o cabeçalho da licença do Apache 2.0;
- Verificar se as modificações não quebram nenhuma regra. Isso pode ser feito rodando o script “make flake8”.

4.2. Diretriz geral do código

- Logging
  É importante por aumentar a visibilidade e fazer o projeto mais fácil de ser entendido e dar suporte. Toda a declaração de log deve conter o maior número possível de informações adicionais possível. Essas informações devem ser incluídas no dicionário que é passado via argumento para o método logger. O formato default do log que é utilizado inclui esse contexto adicional como parte da mensagem que ajudará o usuário a encontrar a informação relevante.
  Para obter uma referência à instância logger, deve-se utilizar a função “st2common.log.getLogger”. Deve-se usar essa função ao invés da função do módulo logging do stdlib pois é declarado um nível de log personalizado e faz mais algumas coisas que está disponível apenas nos loggers que são obtidos através da versão do getLogger. Na maioria das vezes, isso deve ser feito no início do módulo, após os imports e esse logger deve ser utilizado no módulo.

```
from st2common import log as logging

LOG = logging.getLogger(__name__)
LOG.debug('....')
```

  O dicionário deve conter valores que são relevantes ao log da mensagem em questão, como por exemplo, criado banco de dados do objeto, usuário que fez a ação.

Se estiver passando uma instância de uma classe personalizada como valor, deve ser implementado o método “to_dict” nessa classe. Esse método é responsável por retornar uma representação do dicionário desse objeto que pode ser serializável como JSON. É importante lembrar que esse método já está implementado para todos os banco de dados do StackStorm (ActionDB, RunnerTypeDB, etc).

```
action_db = ...
user_db = ...
remote_addr = ...

extra = {'action_db': action_db, 'user_db': user_db, 'remote_addr': remote_addr}
LOG.debug('New action has been created. ActionDB.id=%s' % (action_db.id),
          extra=extra)
```

Todos os objetos datetime que estão sendo utilizados no código devem estar na timezone correta e representadas em UTC. O mesmo vale para datas armazenadas do banco de dados, se não puder utilizar timestamp, as datas armazenadas devem ser representadas em UTC.

- Instanciando classes modelo
  Quando instanciadas as classes modelo, como ActionDB, RuleDB, SensorTypeDB, deve-se passar todos os valores como argumentos ao construtor ao invés de atribuir variáveis à instância da classe. Passar todos os campos como argumentos ao construtor significa que a funcionalidade do construtor é preservada. Pode também deixar mais claro e óbvio para os desenvolvedores quando os valores estão disponíveis e permite uma análise estática básica no código.
  Como deve ser:

```
action_db = ActionDB(pack='mypack', name='myaction', enabled=True)
```

Como não deve ser:

```
action_db = ActionDB()
action_db.pack = 'mypack'
action_db.name = 'myaction'
action_db.enabled = True
```

4.3. Estrutura do código

- StackStorm/st2 Repo
  O StackStorm é feito por serviços individuais viz. st2auth, st2api, st2rulesengine, st2sensorcontainer, st2actionrunner, st2notifier.
  StackStorm também possui um CLI e uma biblioteca client. Os códigos para todos esses serviços e componentes são apresentados em diferentes pastas no diretório raiz.
- st2common
  Código comum que contém o banco de dados e os modelos dos dados API, funções de utilidades e código de serviço comum que todo o StackStorm necessita. Cada pacote de serviço (deb/rpm) tem uma dependência no pacote st2common sendo disponível. Então todo código comum que deve ser compartilhado entre componentes deve ser adicionado neste diretório.
- st2api
  Esse diretório contém o código para os controladores API do StackStorm. As APIs e os controladores são versionados. Controladores experimentais são adicionados ao diretório exp dentro de st2api e, quando maduros, são movidos para a pasta dos controladores versionados.
- st2auth
  Contém o código para o ponto final do st2auth. Já que esse ponto final precisa ser implementado separadamente do st2api, ele está disponível como um aplicativo separado.
- st2actions
  Contém o código para as ações que ocorrem no StackStorm. Uma nova ação para o StackStorm deve ser adicionada em st2actions/st2actions/runners. Esse diretório também contém o código para o nó trabalhador que verifica rabbitmq para execuções de entrada.
  Notifier e results tracker também fazem parte desse código base. Notifier é o componente que envia disparadores de notificações e de ações até o final execução da ação. Results tracker é uma lista assíncrona avançada para certos tipos de corredores, como mistral, onde a execução do workflow é parada remotamente e é necessário acionar o mistral API para coletar os resultados.
- st2client
  Contém o código tanto para a biblioteca StackStorm client quanto para StackStorm em apenas um diretório. Isso será eventualmente separado.
- st2debug

O código apresentado neste diretório vem da ferramenta st2-submit-debug-info. Essa ferramenta fornece um jeito de compartilhar logs e outras informações com o engenheiro de solução de problemas do StackStorm.

- st2reactor
  Contém o código para o sensor e para o mecanismo de regras.
- st2tests
  Código compartilhado que contém utilitários de testes que ajudam a testar individualmente e a integração para todos os componentes do StackStorm.

5. Referências

https://github.com/StackStorm/st2
https://stackstorm.com/