SPARKL®

Automate Confidence Scoring

How SPARKL® integrates with Ansible for a Powerful End-to-End Service Management Solution

Solution Brief

Current State-of-the-Art

In enterprise computing, solutions such as Ansible represent the state-of-the-art for configuration of infrastructure artefacts. Principally, these artefacts are servers - but Ansible is being considered for network switch configuration as well.

Configurations are captured as desired state descriptions, which the management tool will seek to enforce. Typically, Ansible is used to configure applications running on servers which together serve to effect the delivery of services. It comes with a broad range of configuration modules for this task.

Ansible can also be used to deploy and tear-down artefacts, representing a mature configuration management solution for service delivery automation.

Why Ansible?

The use of Ansible is particularly powerful through apt technology integrations, and its presence in the market will be promoted through making these integrations possibilities explicit. Ansible is an excellent tool for configuration management and there are opportunities for extending its impact in some key areas:

- By integrating with an autonomics tool for implementation of closed-loop management, in an enterprise computing context.
- Integration with distributed ledger technology so that immutable records of actions are maintained.
- By moving beyond an enterprise computing context, to new pastures such as the Internet of Things.

Ansible is perfect for pushing out changes to machines en masse, but it relies on technology integrations for orchestrating changes - deciding how and when changes should occur. Ansible is also good for describing the configuration of individual components, but will need to rely on a tool such as SPARKL to determine when to initiate changes, and support with resolving ordering dependencies between updates to components.

For example, when rolling out a new release of an application and/or service, or carrying out a security update, certain questions cropping up during the process can be resolved by a tool like SPARKL when working with Ansible - e.g. which application components should be updated, to what version, and in what order.

Autonomic Opportunities

Autonomics focuses on continuously moving a system towards its desired state. An autonomics solution will continuously execute a closed-loop of monitoring the state of the environment, analysing it (e.g. with respect to its divergence from the desired system state), planning a course of mitigation actions, and executing the actions.

This is the MAPE cycle of autonomic computing, shown in **Figure 1**. With regard to the MAPE cycle, Ansible principally provides support for the execution stage alone.

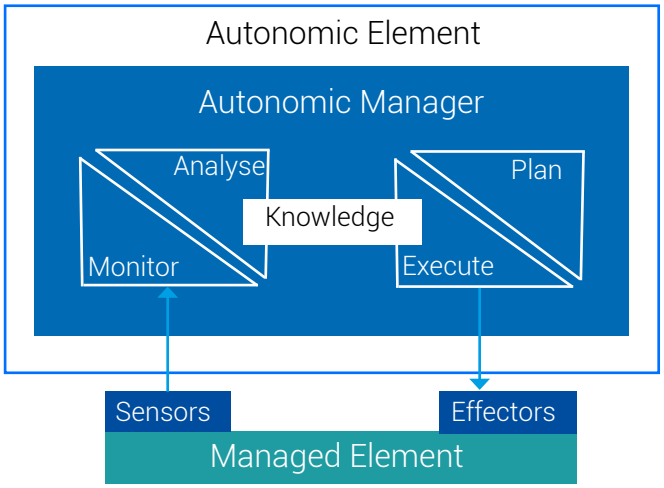

Figure 1: the Monitor, Analyse, Plan, Execute (MAPE) cycle of an Autonomic Element

The SPARKL orchestration tool completes this cycle - it decides how, when and what actions should be done based on events observed in its environment, and defer to Ansible as one of its principal technology integrations to do the enforcement. SPARKL's two main business contexts are enterprise computing and the Internet of Things, and it uses Ansible for both.

For enterprise computing, the management task in service delivery has become increasingly burdensome.

Rather than manual intervention with procedures such as runbooks, the value of SPARKL lies in its effective automation of maintaining desired system state by performing analytics over observed event streams and planning actions based on those observations.

We propose SPARKL coupled with Ansible as a powerful **Configuration Management solution for Enterprise Computing**.

Example - Confidence Scoring

Consider the following example where SPARKL is performing confidence scoring of security threats based on host and network fingerprints, such as duplicate system processes running on a host, or traffic on a network.

We describe orchestrations in SPARKL as mixes, and these can be written in XML.

A mix may consist of different types of operations - a notification is called by logic external to the mix, such as a SPARKL service adapter for an incoming event stream, running in a different mix. Other types of operations are used by SPARKL to plan a mitigation in response.

For example, event streams from host and network sensors may be initially processed by a Complex Event Processing (CEP) application such as Apache Storm, and a consolidated view may be forwarded to SPARKL, as shown in **Figure 2**.

Here, events are routed to an Event Stream Adapter Mix, where logic in the mix may forward events to a number of processing mixes. One such mix may define a notification, `RaiseSystemProcessDuplicate`, which handles events concerning the anomaly of duplicate system processes on a host which can be indicative of malware.

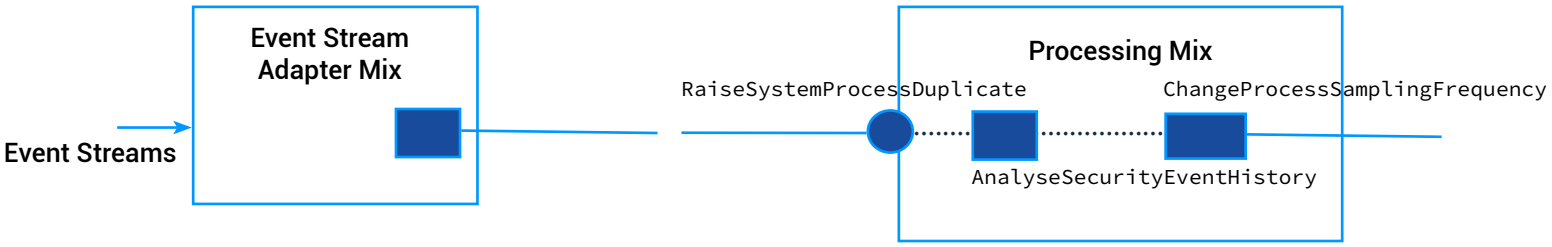

Figure 2: Handling duplicate process anomalies, using SPARKL and Ansible

When this notification is called within an event, SPARKL will perform planning (shown by the dotted lines) and ultimately may call `AnalyseSecurityEventHistory`.

This operation will determine what course of action should be taken, and through some confidence scoring, may decide that all hosts in the web server group should report their process dumps at 10x the frequency. This change would then be imposed via Ansible en masse by connecting to each of the machines and changing their configurations.

This is carried out by the `ChangeProcessSamplyingFrequency` operation, shown in the XML excerpt for the mix in Figure 3. The Ansible configuration to be enforced is given in the SPARKL service `AnsibleEngine`, named in the operation. The end-to-end orchestration described here is simply handled through the integration of SPARKL with Ansible.

```
...
<service name="SecurityEventCorrelation"
  provision="script">
  <prop name="script.src"
    type="eclipse-clp"><![CDATA[

% Get alarm from SPARKL
field(alarm, Alarm),

% Event correlation logic, pulls data from mongodb event database
calculate_frequency(Frequency),
% Write result
resultname("Ok"),
resultfield(frequency, Frequency).
  ]]></prop>
</service>

<service name="AnsibleEngine"
  provision="script">
    <prop name="script.src"
      type="ansible"><![CDATA[

        hosts: webservers
        sudo: True
        vars:
            setfreq: /usr/bin/setfreq
        tasks:
        - name: change process sampling frequencing
          shell: "{{ setfreq }} {{ frequency }}"
     ]]></prop>

<notify name="RaiseSystemProcessDuplicate"
  service="Sequencer"
  clients="NotifySecurityAlarms"
  fields="alarm">
  <prop name="txm_analytics.source.type"
    value="securityalarms"/>
</notify>

<request name="AnalyseSecurityEventHistory"
  service="SecurityEventCorrelation"
  fields="alarm">
  <reply name="ChangeFrequency"
    fields="frequency"/>
  <reply name="DoNothing"
    fields="EVENT_PROCESSED"/>
  <reply name="Error"
    fields="reason"/>
</request>

<consume name="ChangeProcessSamplingFrequency"
  service="AnsibleEngine"
  fields="frequency"/>
...
```

Figure 3: An excerpt of a processing mix for handling duplicate system process anomalies

The inventory, including hosts which are in the `webservers` group as named in the Ansible script fragment, is managed dynamically by SPARKL. SPARKL field data passed into the `ChangeProcessSamplyingFrequency` operation is made available as Ansible vars - note how the frequency field is used in the command string for the shell task.

To enable these features, SPARKL implements its own native support for Ansible in the guise of the `AnsibleEngine` service type.

SPARKL determines what should be done in response to incoming event streams, and Ansible is used to put the determined measures into effect across multiple machines in parallel, possibly at a massive scale.

The Internet of Things

A principal business context for SPARKL is the Internet of Things (IoT) - be that for home automation, smart cities, industrial applications and so on. SPARKL has adopted the use of Ansible in its IoT solution as its principal mechanism for enforcing state of machines, typically compute containers.

SPARKL supports a range of technology integrations for analytics. One such integration is shown in **Figure 4**.

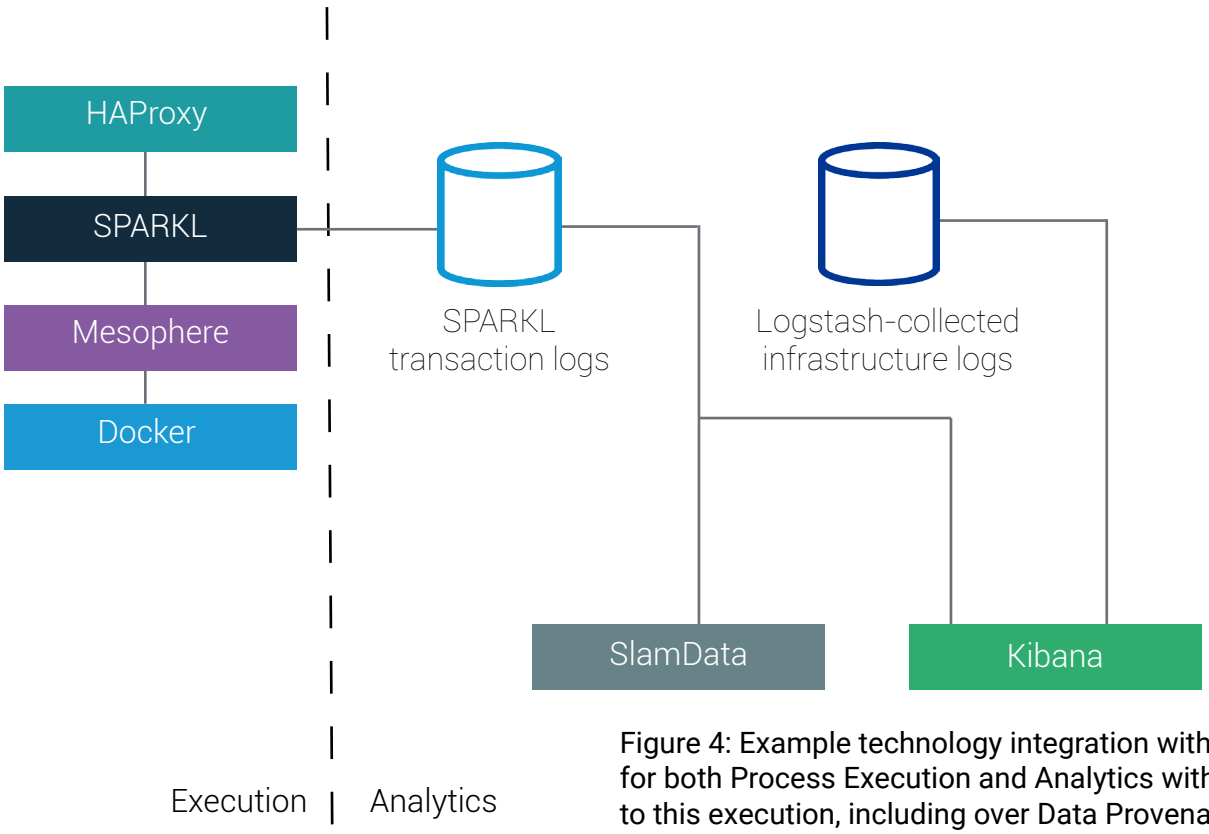

Figure 4: Example technology integration with SPARKL for both Process Execution and Analytics with respect to this execution, including over Data Provenance trails

In orchestrating service artefacts, SPARKL records everything that it does as **events**. These events can be pushed out to a range of tools for further analysis. An example integration is to use SPARKL with MongoDB and SlamData. SlamData offers powerful querying, visualisation and reporting capabilities- so the operations SPARKL carries out on system components via Ansible can be queried, analysed and reported on.

One feature of SlamData is its SQL-like query language. As an example, the following query would get the values of the frequency data fields (in times per hour) in **Figure 4**.

```
SELECT c.frequency, c.timestamp FROM `/sparkl/events` AS c WHERE c.opname =
“ChangeProcessSamplingFrequency”
+-----------+-------------------+
| frequency |         timestamp |
+===========+===================+
| 60        | 1441580462.841065 |
+-----------+-------------------+
| 600       | 1441580612.353648 |
+-----------+-------------------+
```

Figure 5: SPARKL Analytics

Blockchain Support

SPARKL integrates with a variety of blockchain solutions in order to demonstrate the integrity of event logs that it produces. For matters of compliance, say, it is crucial to have an accurate account of not only the business workflows executed (in part, by SPARKL), but also the operation workflows managing system components (via SPARKL and Ansible).

SPARKL has built-in support for pushing records of its events to typical blockchains. This means that, at any time, we are able to prove that the event logs that are being used for analytics and reporting (e.g. for compliance) have not been tampered with since being generated.

SPARKL enables the use of any technology stack by a business to maintain event logs, as shown in **Figure 5**. Different businesses will have different needs regarding their technology choices, which are heavily determined by their reporting and analytics needs. SPARKL enables the reconciliation of arbitrary event logs with arbitrary distributed ledger solutions through its novel approach.

At any time, we can show the integrity of an event log, which may include records of actions carried out on service infrastructure via Ansible. **Figure 6** shows the output from running the SPARKL log-checking tool. This tool compares the contents of an event log that's being used for reporting purposes, versus what is kept '***on chain***'.

Note that in this case the event log is kept in MongoDB (to enable SlamData analytics), but the blockchain is kept in BigchainDB, a distributed ledger solution. SPARKL offers complete flexibility in the technology choices that can be made.

```
> python -vvv check.py
-Checking---
--- Events: type: mongodb, uri: mongodb://sparkl:sparkl@127.0.0.1:27017/sparkl, db: sparkl,
collection: events
--- Blockchain: type: bigchaindb, collection: sse1@127.0.0.1
-BLOCK_ON_CHAIN---N-12OW-1957-8
---CHECKING TRANSACTION FROM BLOCK---
N-12OW-1957-8 1C3131EC4CFAD9B0EC5538B58A5A4E54A7125D58901CC95667080CDF823DB979
N-W2L-1Z5-3H 7A130F988D458FC0CE4FF4E20A0CABCEDDB035F5F76357B6177234BDB51DF0B8
N-W2L-1Z5-4T 0CD07EF811175234D86D50E37811FAAB47B9D154E1F33A401C14CFA3C41BDD44
N-W2L-1Z5-51 649DE1276E4100B52FE15320B2487EBC77DA05B4E2500BBF03E6A1222BD6B1D3
N-W2L-1Z5-LO 07C8CD80A271288426201B62C7B5B028B14EC148E0C8BE5DF3A45DB86F5C6B75
N-W2L-1Z5-59 A06E9D4667033EA03091A419C2B429108094570l3066E22E0FFDA2C142FD6B9D
7027B89AA59DCF0C2161C04BF3A1408B6720FF9FF13672A1F41C5F6F7B836488
---BLOCK PASSED---
-BLOCK_ON_CHAIN---N-SQG-IJU-7
---CHECKING TRANSACTION FROM BLOCK---
N-SQG-IJU-7 1CD861C6A2235E947D283D8D073CDC4043DD9DEAC97C9F8C5B0654BE603A45D6
N-W2L-1D4G-17 28E83B7E1BA464F638A4A4C892B1D0CE25D3EDE4347736753127E3DA25EF2D41
N-W2L-1D4G-DE A4E77F41CCD72FB4757F7AD7ED7942727479E43362BCC97257BC54DEF8367DB8
N-W2L-1D4G-6J A415FACFF042D72CA73FF5ECF359C32501B96B5B4D8D332C858CFF1161636DA2
N-W2L-1D4G-44 6A97B7738D8E49E1F087B5DB63E797AD3BEAE3B8A0223771126170E8B628A62C
N-W2L-1D4G-7F 2F4AFEB19650C67F5AED61F42286422D84BB3D26C65EA3E830248670FB96250E
N-W2L-1D4G-8Z 5E8DAB2C50D902AB433AB0FB1A3D022F398902A7A1D4DF5C100C14FEFEABF629
N-W2L-1D4G-97 C4554B3B951DF1CF076271A4208A1E255D53E72FF9721757D81AA3E813DE2FBB
N-W2L-1D4G-L A1D03BC03CF6122AE61AF8530EA04ECBBC9E8267707E1D1F1F06EFA70DD1CF62
N-W2L-1D4G-KY 5917931A7814804FE76F17721CBECC638CAFB55BA515699BBF36B824476D8C6A
N-W2L-1D4G-BG F5C5F3E7E83DC0307FF29691869ECDA71E58A7ACCA0DF58C4D4E780146414EA9
N-W2L-1D4G-BW C074E6C647768085AFD3B1DCC9D2BFD3B3105C849689B6E366490B3C0DF77214
N-W2L-1D4G-CC 5A07102B33421D42540260C8618FD4756F80641BD51A7017CBB641082FBF6046
N-W2L-1D4G-CK 154EB9311555058E8AAD9EB3553A036635018347937DAD6058232310123B336D
701BF48D2D073835F78068142E624E3E6C6ADA32E08F4E2721D896AC5B5D7C0B
---CHECKING BLOCK HASH------BLOCK PASSED---
-SUCCESS: Valid chain.---
```

Figure 6: Showing the validity of an event log against the record of events '*on chain*'

Let’s talk

- Need more info? Drop us an e-mail at talk@sparkl.com
- See SPARKL tutorials and demos at sparkl.com/docs