The new Check-API

CHECKMK CONFERENCE #6 – MUNICH, APRIL 29, 2020

Moritz Kiemer
Developer
tribe29
Agenda

1. RECAP – WHY BUILD A NEW CHECK-API
2. PRINCIPLES OF THE NEW CHECK-API
3. WHAT THAT MEANS FOR YOU
From few to 1900+ plugins, API hasn’t changed

<table>
<thead>
<tr>
<th>Past</th>
<th>Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 plugins + Check-API v1</td>
<td>1900+ plugins + 3rd party checks + Check-API v1 + some more functions</td>
</tr>
</tbody>
</table>

thumb_up thumb_down
Overarching goal: A new way to write plugins

**Goal**

- Check plugins are modules
- Plugins import a versioned, well-defined API
What we promised a year ago...

From Checkmk 1.7:

✔ API and plugins are being transformed into Python modules
✔ Most existing plugins are being auto-migrated
✔ Complete switch-over to Python 3
Agenda

1. RECAP – WHY BUILD A NEW CHECK-API
2. PRINCIPLES OF THE NEW CHECK-API
3. WHAT THAT MEANS FOR YOU
Principles of the new API

1. Establish conventions & consistency
2. Things doing the same thing should look the same
3. Make implicit explicit
4. Reduce unnecessary options
5. Ensure „fail early“ principle
Principles of the new API

1. Establish conventions & consistency
   - Establish consistent naming
   - Make naming more „telling“
     - `check_function` → `check_function`
     - `default_levels_variable` → `check_default_parameters`
     - `group` → `check_ruleset_name`
Principles of the new API

2. Things doing the same thing should look the same

- Various ways to achieve the same result - historically grown
  - Example: Rule sets for discovery functions don’t work like those of check functions
    - we changed that now!
- Not one consistent best practice
- Confusing if you look at the code
Principles of the new API

3 Make implicit explicit

- Many things were implicitly assumed, without any explicit notion of whether it works as intended or not
  - Cluster Compatibility
  - Parse function (used to be optional $\rightarrow$ not anymore)
Principles of the new API

4 Reduce unnecessary options

- Limitations in API „force“ you to do things in a certain way
  - OID_BIN → gone
  - OID_END_BIN → gone
  - OID_END_OCTET_STRING → gone
  - OID_STRING → gone
  - OID_END → OIDEnd()
  - BINARY(.) → OIDBytes(.)
  - CACHED(.) → OIDCached(.)

- Don’t need two different things doing exactly the same
  - Check function either as function or generator → now always a generator
Principles of the new API

Ensure “fail early” principle

- Working with new Check API should make errors visible early on (not just later at runtime)
- Putting checks in python modules makes testing easier and running them independently possible
- Calling a register function instead of filling a dictionary allows for extensive validation of arguments
Agenda

1. RECAP – WHY BUILD A NEW CHECK-API
2. PRINCIPLES OF THE NEW CHECK-API
3. WHAT THAT MEANS FOR YOU
Checks just got a lot easier

- Python modules mean you can run checks separately
- Python modules make it possible to leverage IDEs
- You can actually test your code!
- You know whether it worked: If config is generated without errors, your plugin (probably) is correct
- Many improvements in the details (e.g. cluster compatibility)
The good news...

- Auto-migration will do most of the work for you
- Auto-migration fails for only 63 of 1920 plugins (>96% success)
- Cluster aware plugins will not be auto-migrated, though
How to prepare for the new API

- **Werk #10601** lists anticipated reasons why auto-migration fails → Avoid those 😊
  - Most important example: Complex SNMP scan functions
- If you don’t expect auto-migration to work, consider the following:
  - Implement parse function (using only Python built-ins)
  - Make discover & check function a generator
  - Put creation of host labels into separate generator function (expecting parsed data as argument)
  - In cluster case expect this input: parsed = {"node1": data1, "node2": data2, ... }
Any questions?
Thank you