BROCADE

AUSNOG

BGP Auto-remediation

Darin Sikanic

Why Lazy Engineers Are The Best Engineers

“Laziness makes you write labor-saving programs to reduce overall energy expenditure”

– Larry Wall (original author of Perl)

How to solve so many manual problems

Theoretically

More Engineers

- More fingers on CLI
- Expensive – high human to device ratio

Programmers

- Proprietary code
- Expensive skill set
- Expensive to maintain (bug fixes, extensions)
- Keep them away from buses

Tools Based Approach

- Commodity code
- Community driven (more eyes on code)
- Extensible by operators

Operations Auto Remediation

DevOps for Networks

- Operations Workflow Generation
  - Over time many NOC engineers develop fault finding procedures that are repetitive in nature.
  - These procedures can be decomposed and represented as a series of steps in a workflow

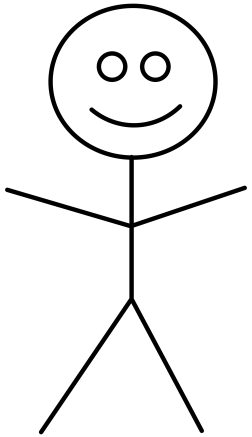
Todays Use Case

BGP Policy Remediation

- BGP policies are pretty much set and forget
  - We write BGP peering policies to follow very absolute conditions
    - Peer Fails then use next preferred route
  - BGP NLRI is evaluated against a very strict (and for good reasons) selection process
    - Highest Local Pref
    - Shortest AS path
    - Lowest MED
    - EGP over IGP
  - A workflow engine gives us a mechanism to automate some intelligence into how we handle failures

Automating Dave the Operations guy..

Dave works for BigCo Content Service Provider



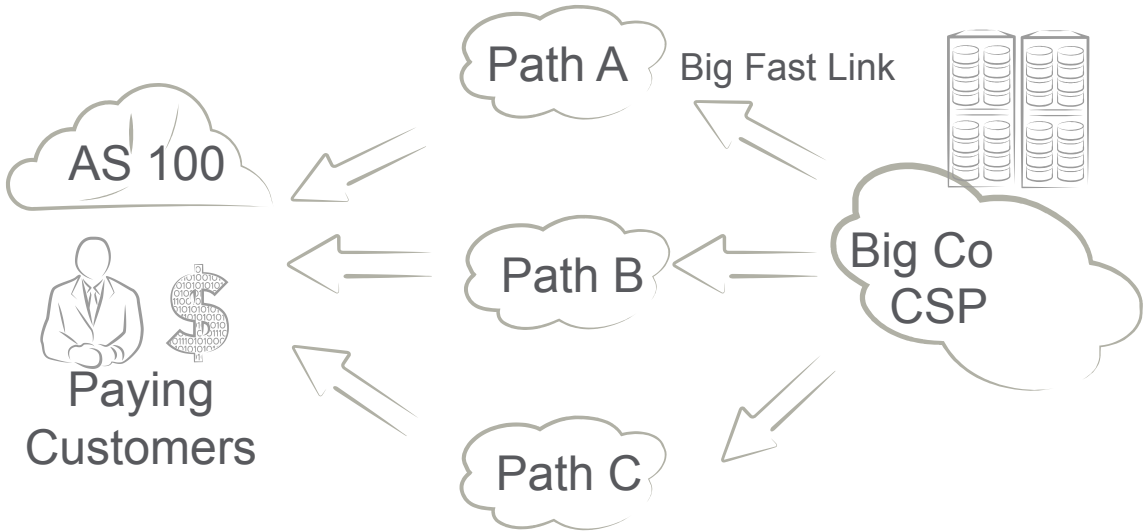
Dave

- He is very important
  - Manages BGP peering
  - Is on call when something goes wrong
- Typically when something does go wrong, he has to login to a router and issue some commands
  - This takes a non deterministic amount of time
    - prone to errors, and Dave being awake and perhaps sober..
- Could do with an automated way to fix common scenarios.

Content Service Provider

A very Basic Use Case

- Big Co wants to ensure high throughput to their customers.
- Dave is on call to re-configure peering policy to preference the least utilized link towards AS 100 if the currently preferred link goes down.
- Dave decides to write a workflow to auto-remediate this common scenario

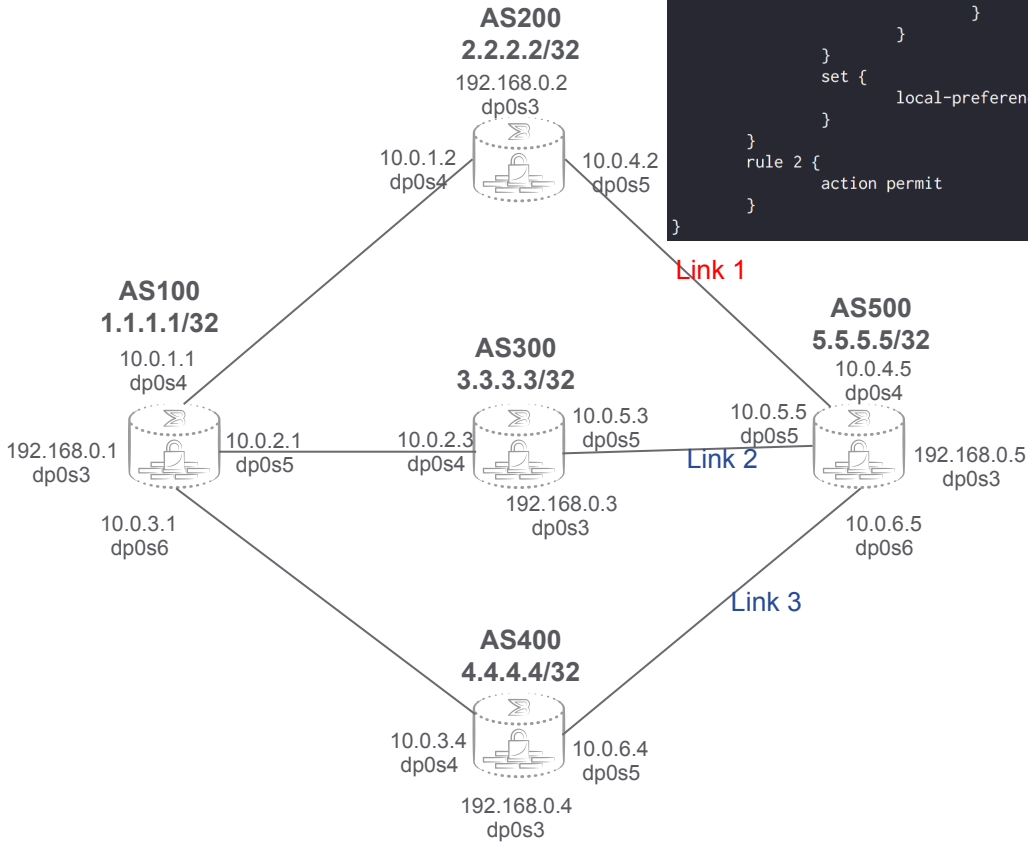


Content Service Provider

Typical Routing Problem.

- Typical Peering Scenario
  - BGP Policies are used to influence path selection
  - BGP Policy is **static** in nature and can only use standard attributes for path selection
- An automated workflow can select paths based on real-time information

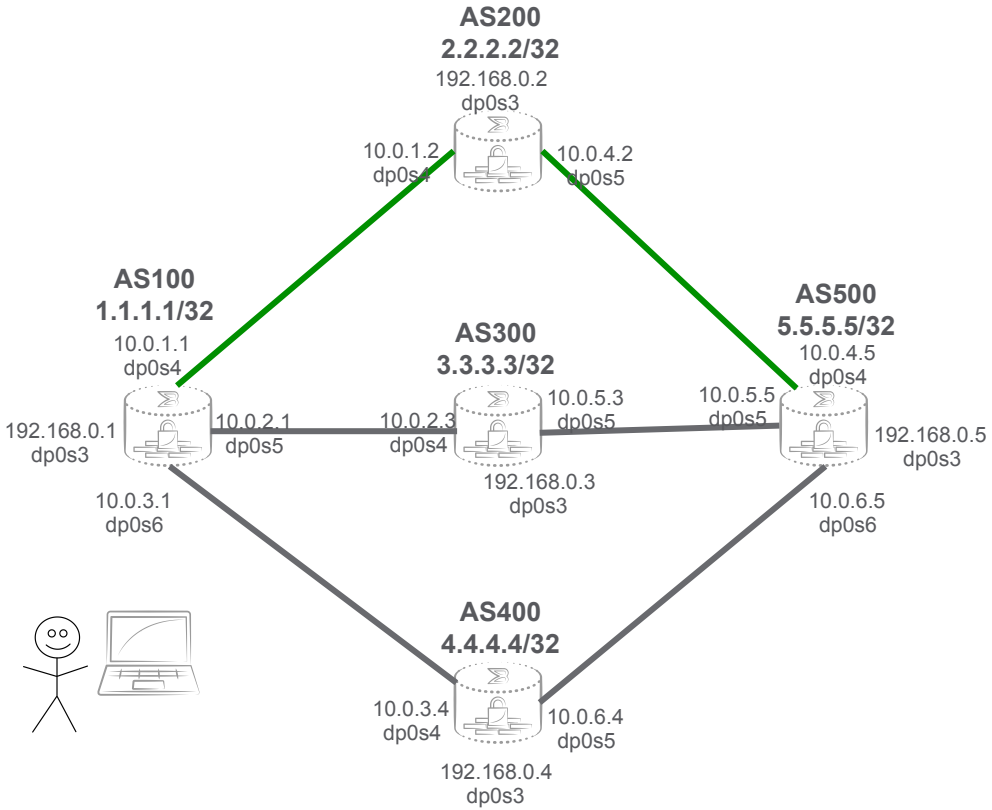
```
route-map peer-10.0.4.2 {
        rule 1 {
                action permit
                match {
                        ip {
                                address {
                                        prefix-list route1
                                }
                        }
                }
                set {
                        local-preference 300
                }
        }
        rule 2 {
                action permit
        }
}
```



The “Old” Way

Workflow Takes a long time!

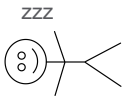
- Peer Goes down
  - NOC gets an alarm
  - BGP automatically reroutes
    - Pre configured routing policy
- Customer complains of traffic Loss/ Degradation
  - NOC engineer investigates
  - NOC engineer adjusts routing policy
- Traffic to redirected to preferred path
- **Time to Resolve = Hours**

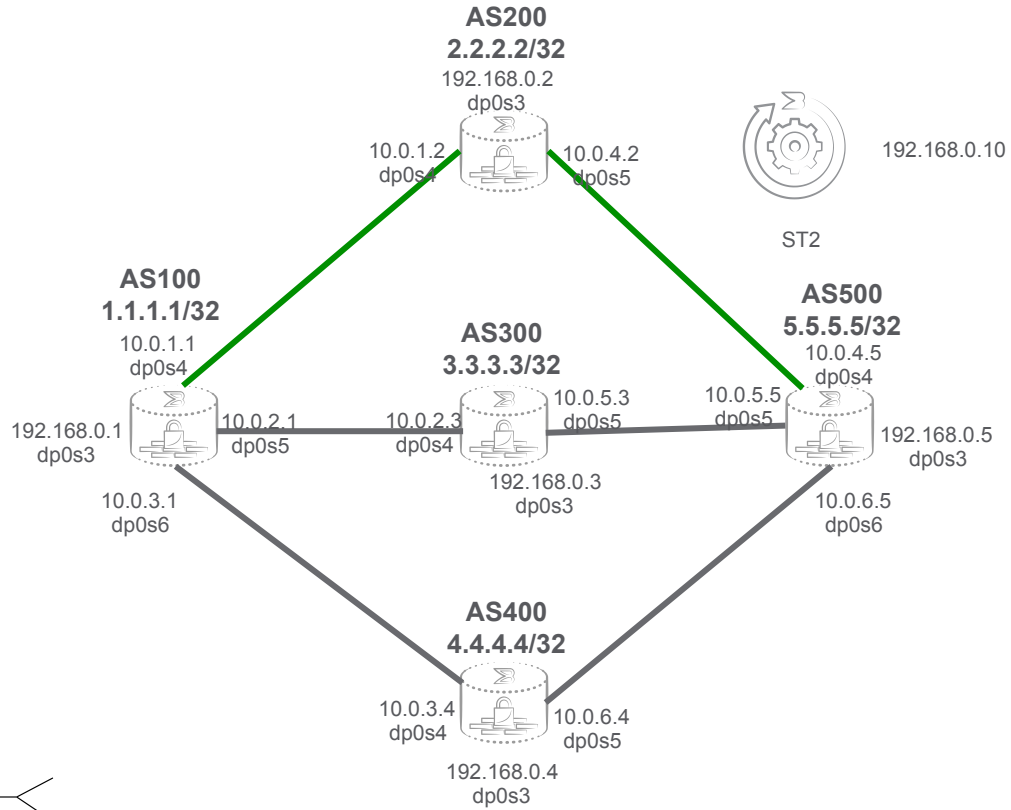


The “Lazy” Way

React in seconds

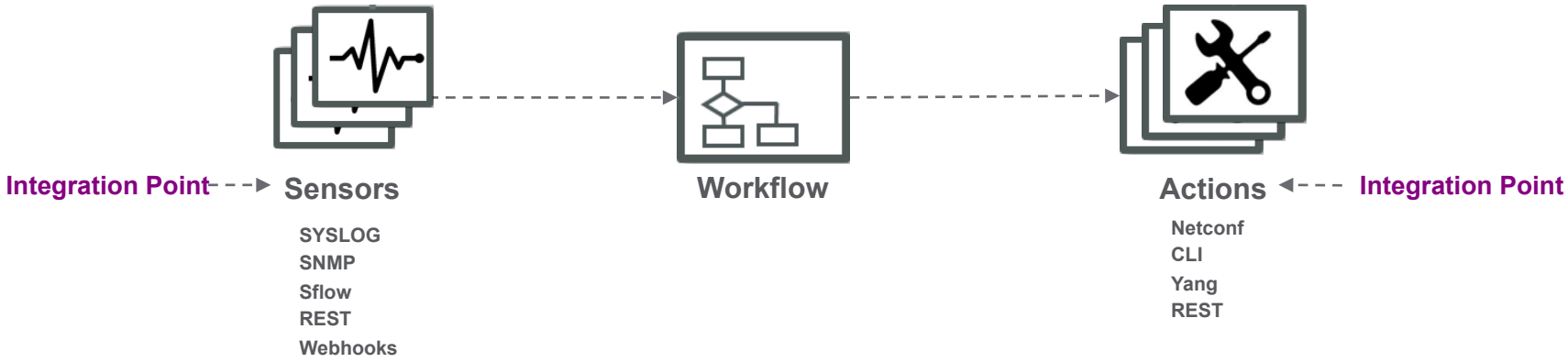
- BGP Peer goes down
  - Syslog alarm triggers workflow
- Workflow is run
  - Log into router
  - Check the route resolution
    - Looks for interface with least utilization
      - Can include other checks (ping/trace/ looking glass path tests, complex scripting)
  - Workflow engine adjusts routing policy
- Traffic redirected/Customer happy
- **Time to Resolve = Seconds!**
- **And Dave’s still in bed!**





Anatomy of a Workflow

Workflow: a set of steps to achieve a specific outcome. Workflow defines order, transitions, conditions, and data flow using a simple declarative language like Mistral.



Rules

Rule are made up of:

- **Triggers**
  - Specifies which incoming events (produced by sensors) should be inspected for potential match against this rule
- **Criteria**
  - The condition(s) needed to be met for the action defined in the rule to be executed.
- **Actions**
  - The action/workflow to be executed on a successful match on a trigger and an optional set of criteria. The parameters to an action are also specified here.

```
---
    name: "rule_name"                       # required
    pack: "examples"                        # optional
    description: "Rule description."        # optional
    enabled: true                           # required

    trigger:                                # required
        type: "trigger_type_ref"

    criteria:                               # optional
        trigger.payload_parameter_name1:
            type: "regex"
            pattern : "^value$"
        trigger.payload_parameter_name2:
            type: "iequals"
            pattern : "watchevent"

    action:                                 # required
        ref: "action_ref"
        parameters:                         # optional
            foo: "bar"
            baz: "{{trigger.payload_parameter_1}}"
```

Sensors

Sensors are:

- Pieces of Python code that follow the StackStorm sensor interface.
- They are adapters that allow for integration with external systems.
- Sensors dispatch triggers which are in-turn used by rules to perform a series of actions or workflows.

```python
import eventlet

from st2reactor.sensor.base import Sensor


class HelloSensor(Sensor):
    def __init__(self, sensor_service, config):
        super(HelloSensor, self).__init__(sensor_service=sensor_service, config=config)
        self._logger = self.sensor_service.get_logger(name=self.__class__.__name__)
        self._stop = False

    def setup(self):
        pass

    def run(self):
        while not self._stop:
            self._logger.debug('HelloSensor dispatching trigger...')
            count = self.sensor_service.get_value('hello-st2.count') or 0
            payload = {'greeting': 'Yo, StackStorm!', 'count': int(count) + 1}
            self.sensor_service.dispatch(trigger='hello-st2.event1', payload=payload)
            self.sensor_service.set_value('hello-st2.count', payload['count'])
            eventlet.sleep(60)

    def cleanup(self):
        self._stop = True

    # Methods required for programmable sensors.
    def add_trigger(self, trigger):
        pass

    def update_trigger(self, trigger):
        pass

    def remove_trigger(self, trigger):
        pass
```

Actions and Workflows

Actions are:

- Pieces of code (any language) that can perform arbitrary tasks
- They are accompanied by a metadata file that describes the action and what parameters it requires.

Workflows are:

- Used when multiple actions are needed to complete a task
- Strings actions together and orchestrates their execution by calling the right action at the right time with the right input, keeping state and passing data.
- Can use workflow engine such as Mistral or built-in one called ActionChain

```
version: '2.0'

examples.mistral-basic:
    description: A basic workflow that runs an arbitrary linux command.
    type: direct
    input:
        - cmd
    output:
        stdout: <% $.stdout %>
    tasks:
        task1:
            action: core.local cmd=<% $.cmd %>
            publish:
                stdout: <% task(task1).result.stdout %>
                stderr: <% task(task1).result.stderr %>
```

Metadata file (`my_echo_action.yaml`):

```
---
name: "echo_action"
runner_type: "python-script"
description: "Print message to standard output."
enabled: true
entry_point: "my_echo_action.py"
parameters:
    message:
        type: "string"
        description: "Message to print."
        required: true
        position: 0
```

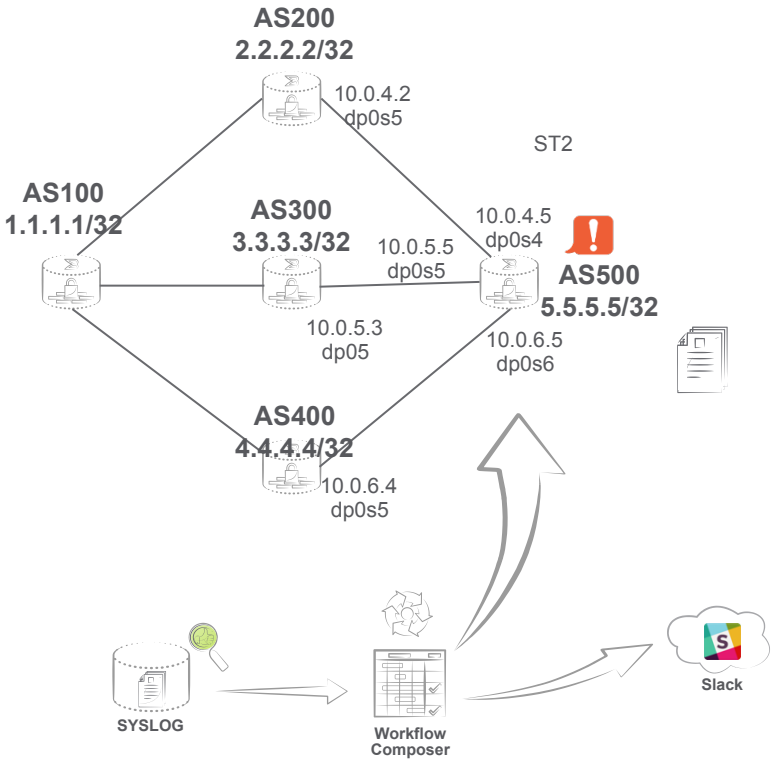
Action script file (`my_echo_action.py`):

```
import sys

from st2actions.runners.pythonrunner import Action

class MyEchoAction(Action):
    def run(self, message):
        print(message)

        if message == 'working'
            return (True, message)
        return (False, message)
```

Demo Walkthrough

Troubleshooting & remediation

BGP Peers



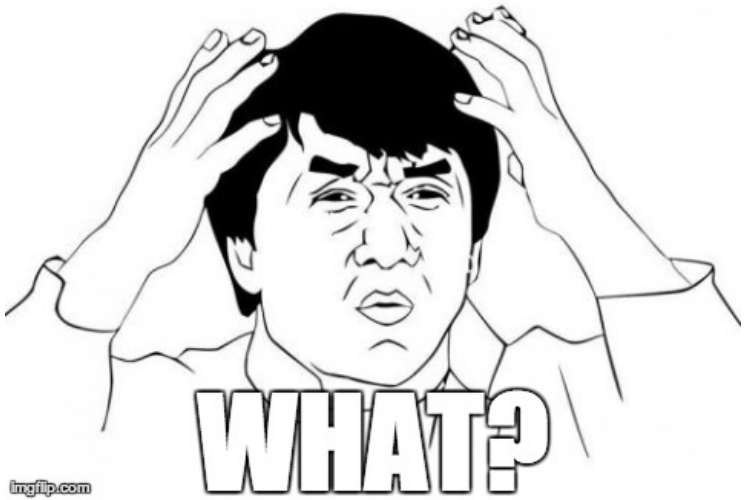
BGP Peer Failure

1. HOST - BGP Peer goes down - HOST
2. HOST - Send SYSLOG Message
3. SYSLOG - Plugin Matches Message REGEX
4. SYSLOG - Sensor Triggers BGP Autoremediation Workflow
5. ST2– Extract:
    - Routes for destination supplied in rule
    - Get the interfaces that are using those routes
    - Gather some statistics on those interfaces
    - Choose the least utilised interface
6. ST2– Using rest interface perform the following action:
    - Set local preference for preferred peer to be the highest
7. ST2– ALERT
    - Post messages to Slack for each step.

ChatOps

What is it?

- Conversation driven development

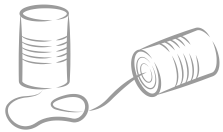


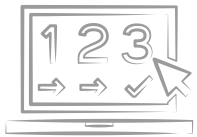
ChatOps

TLDR and why should I care

- A chat room (Slack Channel), where members can issue self-service tasks (show CPU usage for all hosts).
- Reduces feedback loop between team members and between disparate systems (OpenStack, Jira, your Tesla?)

The evolution of how Operations operates…





ChatOps

Demo

Further Reading

StackStorm Technologies

- http://stackstorm.com
  - Sensors, Actions, Rules, more..
  - Integration Packs
- https://stackstorm.com/blog/
  - Interesting use cases
- https://stackstorm.com/community/
  - Join StackStorm community Slack group for Q&A with StackStorm developers and users
- https://github.com/StackStorm/st2contrib
  - Find and contribute StackStorm integration packs

LEARN MORE ABOUT PYTHON PROGRAMMING

"LEARN PYTHON THE HARD WAY"

http://learnpythonthehardway.org/

CODE ACADEMY PYTHON COURSE

https://www.codecademy.com/learn/python

BROCADE

Thank You