**Summary of Findings:**

I've included the basic features of PagerDuty, OpsGenie, and VictorOps along with my overview and review of their feature-sets and what each of these services can bring to the existing Python stack that the dev team is working with. I understand that the existence of Python, integration with the dev team's Azure server, and the quality of the service's mobile app are important so I included expanded sections on those features.

I've attached to each service a link to test code I have written that performs basic CRUD operations that involve creating, acknowledging, and resolving incidents that come up frequently from the services. When Python libraries could be used for these purposes, Python code was included. REST API code that is initiated through **curl** requests is also included for the services where it is important.

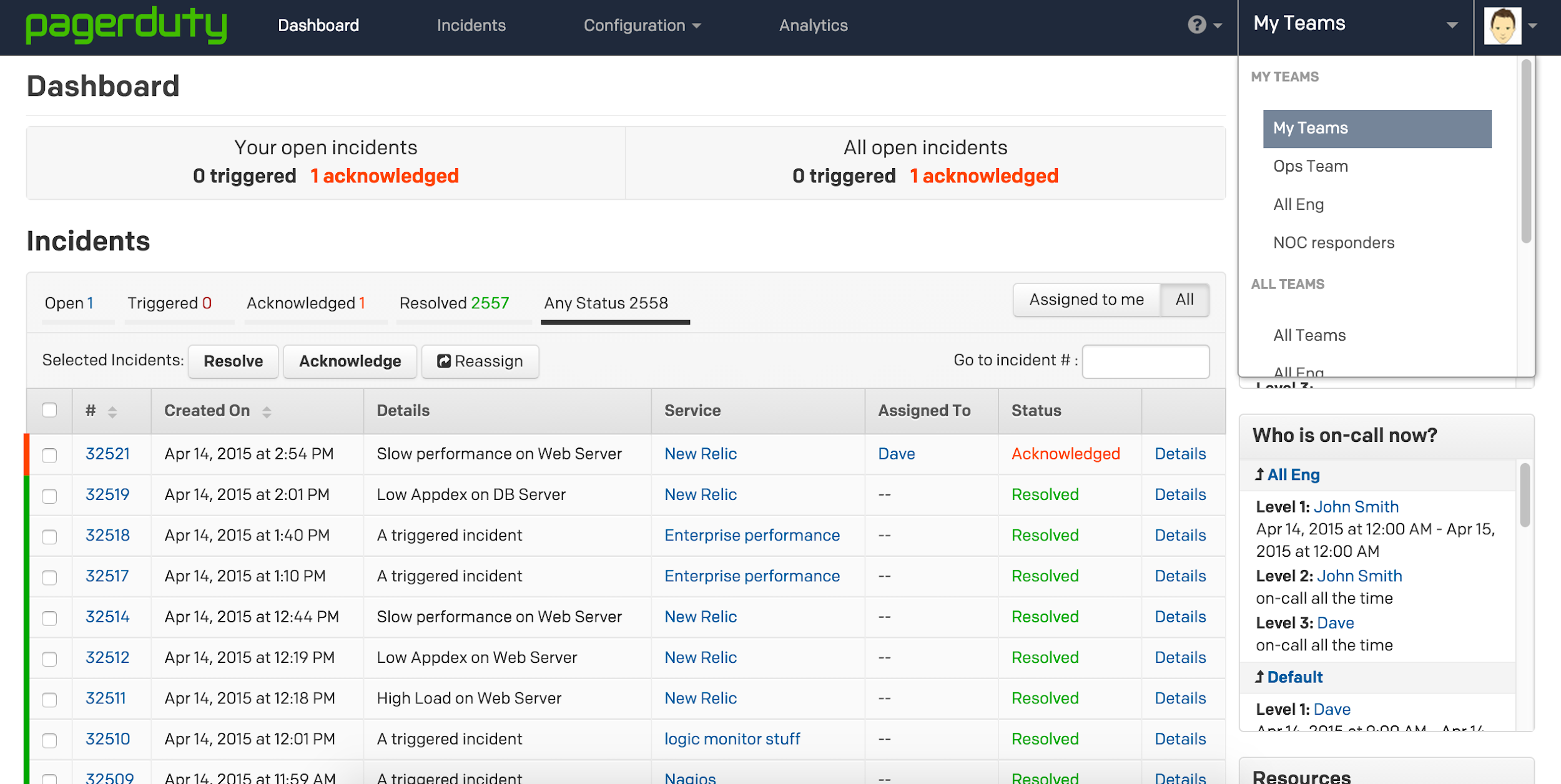
I'd like to conclude by giving a strong recommendation for the adoption of Pagerduty by the team for the reasons listed in the document below.

[**Link to final summary comparing PagerDuty and VictorOps**](https://docs.google.com/document/d/16zwa1mjftVsnhaz8IIwoyVGI25jXKt9akm1hsiDVgl8/edit?usp=sharing)

[**PagerDuty**](https://www.pagerduty.com/) **(Recommended) -** [**Python Test Code**](https://drive.google.com/open?id=1X9wcao-2p0SrMs5nLxqa4hnvxqryAdEz)**,** [**REST Test Code**](https://drive.google.com/open?id=1RHraiRukjuZqSJkfX9wwDjBjlCH6rfnz)

Basic Features

* Event Grouping & Enrichment
* Reliable & Rich Alerting
* Scheduling & Automated Escalations
* Mobile Incident Management
* Real-Time Collaboration
* System & User Reporting
* Always-On, Guaranteed Delivery
* Enterprise-Grade Security & Controls
* Service Grouping
* **API:** Community-supported, open-sourced Python API for use with the PagerDuty API
  + Note that the Python API ([pypd](https://github.com/PagerDuty/pagerduty-api-python-client/)) is not officially created by PagerDuty but is nearly complete (can Create, Read, Delete but can’t Update all elements yet)
  + Standard REST API included for use with all entities of a PagerDuty account. Platform is highly extensible to existing stacks
    - Can ingest events from monitoring tools and and route them through PagerDuty to be created as incidents (if integration is not already included)
* **Web App Features**: Web app also included to monitor incidents and view on-call individuals at any moment. Most detailed web app of all three options.
  + Reduces need to make personal calls to address incidents. PagerDuty knows who is on call and escalates the incident to them automatically



* **Smart Notifications and Scheduling:** Focuses on aggregating incidents from integrated services and dispatching services to appropriate team members related to the alarm
  + Uses machine learning to suppress irrelevant events and alerts to not notify any users
  + Add on-call schedules for all users and create a custom escalation policy for notifications
  + Detailed alert tracking and escalation process to create alerts only for relevant parties



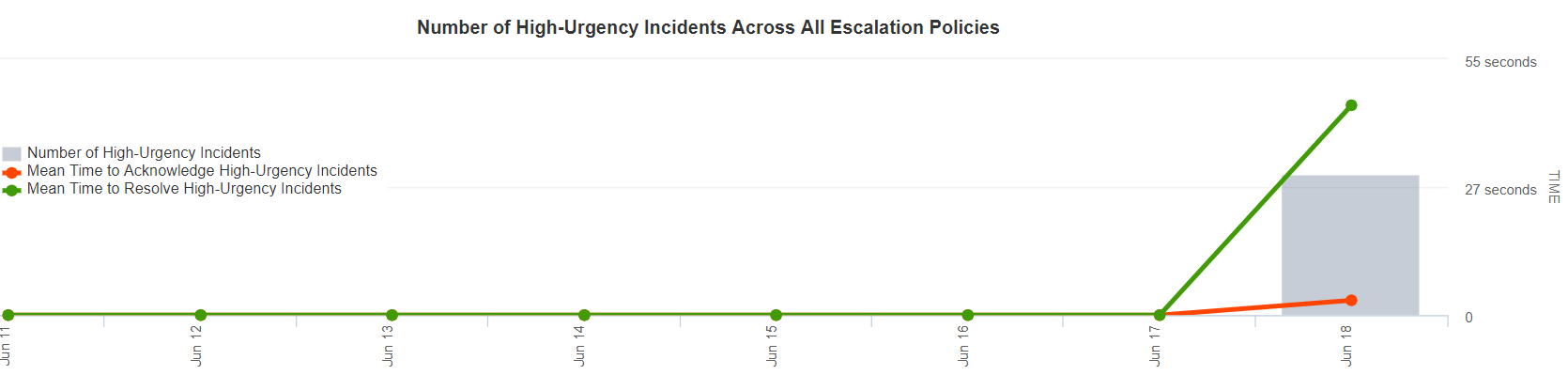
* **Heartbeat:** No native Heartbeat monitoring unless Dead Man’s Snitch (DMS) service is used: If a server or cron job does not make a request once every **x** number of minutes or hours, appropriate users are notified and an incident is created
* **Customer Service:** Strong community support with open-source client API and active community forum
  + 24/7/365 support through email, text messaging, and phone
* **Integrations:** Native Slack and Microsoft Azure integration (with 200+ other native integrations) along with ability to create custom integrations through the REST API

|  |
| --- |
| from os.path import dirname, join from pprint import pprint import pypd from pypd.errors import BadRequest  # set the from\_email to an appropriate email for the api key used pypd.api\_key = 'WyRLzYzEqE2JVmYayKQs' from\_email = 'justin.tran@aidoc.com'  # in this case assuming that there is already a service created service = pypd.Service.find\_one()  # assuming an escalation policy exists as well escalation\_policy = pypd.EscalationPolicy.find\_one()  # set some incident data with a incident\_key we can use to find it later if # we want to avoid duplicating until we act on any open incidents data = {  'type': 'incident',  'title': 'incident\_demo\_incident2',  'service': {  'id': service['id'],  'type': 'service\_reference',  },  'incident\_key': 'incident\_demo\_key',  'body': {  'type': 'incident\_body',  'details': 'testing creating an incident',  },  'escalation\_policy': {  'id': escalation\_policy['id'],  'type': 'escalation\_policy\_reference',  } }  # if the incident is already open it will error with BadRequest try:  incident = pypd.Incident.create(  data=data,  add\_headers={'from': from\_email, },  ) except BadRequest:  incident = pypd.Incident.find(incident\_key='incident\_demo\_key')[-1]  # mergeable incident data\_mergable = data.copy() mergable\_key = 'incident\_demo\_key\_mergable'  data\_mergable['incident\_key'] = mergable\_key  try:  to\_merge = pypd.Incident.create(  data=data\_mergable,  add\_headers={'from': from\_email, }  ) except BadRequest:  to\_merge = pypd.Incident.find(incident\_key=mergable\_key)[-1]  # ack it, snooze it, resolve it? pprint(incident) pprint(incident.json) incident.acknowledge(from\_email) incident.snooze(from\_email, duration=3600) incident.create\_note(from\_email, 'This is a note!') incident.merge(from\_email, [to\_merge, ])  # before we trigger an event get all currently triggered incidents triggered\_incidents = pypd.Incident.find(statuses=['triggered', ])  # let's see if events and alerts work! pypd.EventV2.create(data={  'routing\_key': '64e61052453e4ec0a3b42e93ac235375',  'event\_action': 'trigger',  'payload': {  'summary': 'this is an error event!',  'severity': 'error',  'source': 'pypd bot',  } })  # wait for our events to trigger incidents incidents = pypd.Incident.find(statuses=['triggered', ])  # resolve and finish up for i in incidents:  i.resolve(from\_email=from\_email, resolution='resolved automagically!') |

Python API example

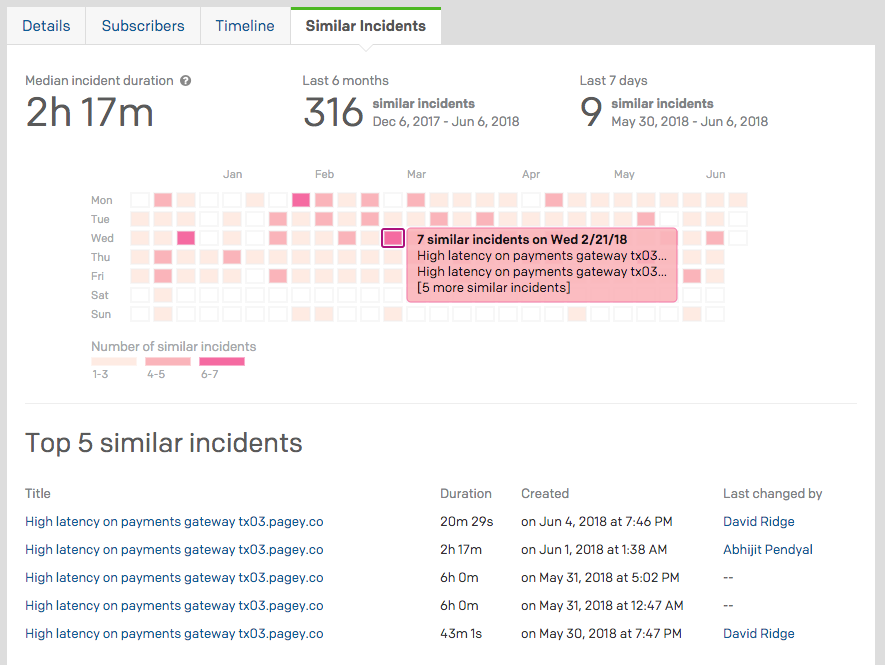
|  |
| --- |
| # Trigger an incident curl -X POST  --header 'Content-Type: application/json'  --header 'Accept: application/vnd.pagerduty+json;version=2'  --header 'From: justin.tran@aidoc.com'  --header 'Authorization: Token token=WyRLzYzEqE2JVmYayKQs' -d  '{  "incident": {  "type": "incident",  "title": "Overflow",  "service": {  "id": "PCMH9RC",  "type": "service\_reference"  }  } }' https://api.pagerduty.com/incidents  # Acknowledge an incident curl -X PUT  --header 'Content-Type: application/json'  --header 'Accept: application/vnd.pagerduty+json;version=2'  --header 'From: justin.tran@aidoc.com'  --header 'Authorization: Token token=WyRLzYzEqE2JVmYayKQs' -d  '{"incidents":   [{  "id": "PIRLLWV",  "type": "incident",  "status": "acknowledged",  }] }' https://api.pagerduty.com/incidents |

REST API example





* **Critical Feature:** Strong community support and all the basic functionality needed to be notified by incidents. Detailed analytics and product is overall extremely easy to use with a number of features available
  + Built-in machine learning algorithms automatically group together similar incidents to improve performance and also group related alerts into single incidents



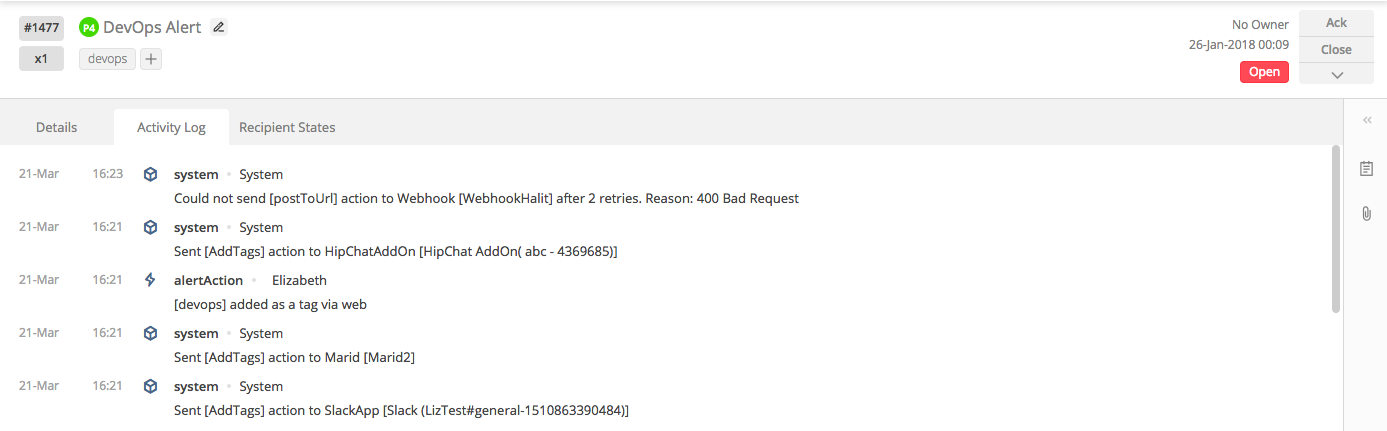
* **Cost:** Standard Pricing: $49 USD per user per month
  + Unlimited API requests
  + Unlimited data retention

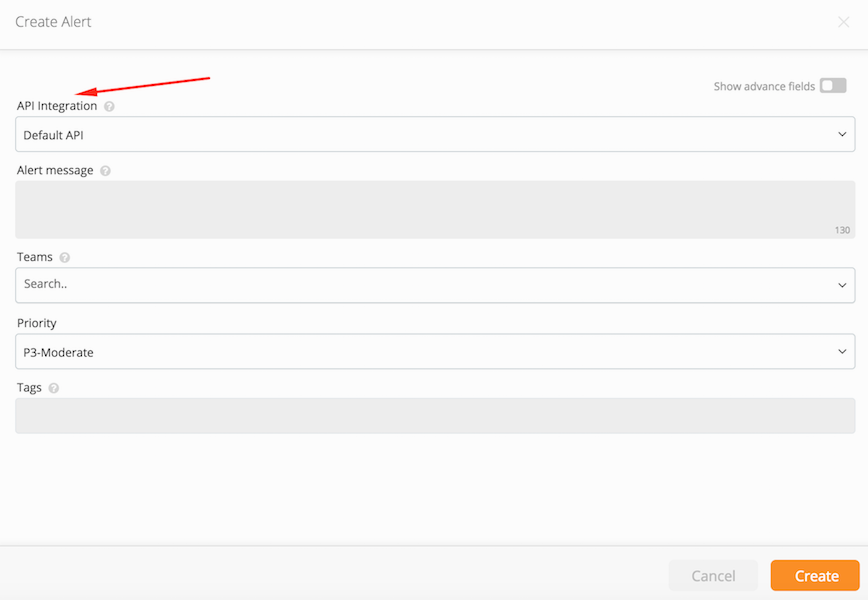
References:

1. <https://www.resolver.com/blog/how-to-choose-an-incident-management-system/>
2. <https://stackshare.io/stackups/opsgenie-vs-victorops-vs-pagerduty>
3. <https://github.com/PagerDuty/pagerduty-api-python-client> (Open-source client API)
4. <https://v2.developer.pagerduty.com/docs/getting-started>
5. <https://github.com/PagerDuty/API_Python_Examples> (Basic examples of REST API usage in Python using the **requests** module)
6. <https://v2.developer.pagerduty.com/v2/page/api-reference#!/API_Reference/get_api_reference>
7. <https://community.pagerduty.com/t/how-to-use-dead-mans-snitch-to-enable-heartbeat-monitoring/1429>
8. <https://support.pagerduty.com/v1/docs/event-intelligence?utm_source=web&utm_campaign=whats_new_feature_links&utm_medium=link>

[**OpsGenie**](https://www.opsgenie.com/) **-** [**Python Test Code**](https://drive.google.com/open?id=1cz3DqU3NkLLf1ACVZ-TmyAiS7d7pfIt0)

Basic Features

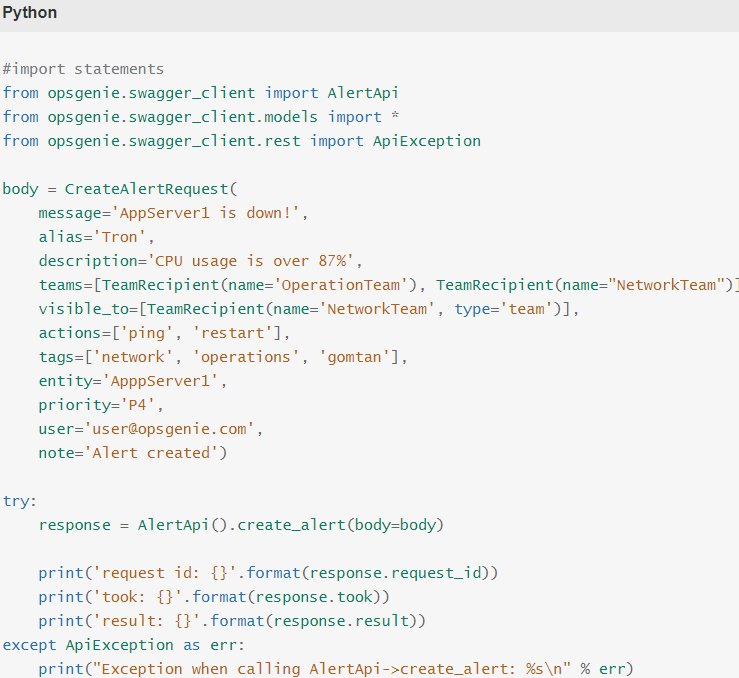
* On-Call Schedules
* Rich Notification
* Escalations
* Heartbeat Monitoring
* Detailed tracking
* Quick setup
* **API:** Contains a limited Python API for OpsGenie services making it easier for existing Python code to integrate with OpsGenie (Python API only covers 1 of 26 categories found in the full REST API)
  + Note: Python API only contains a small portion of the official REST API
  + Can also be used as a web application or through a command line utility
* **Smart Notifications and Scheduling:** On-Call schedule definitions with escalation policies
  + Routes critical alerts only to members that are on-call and are relevant to the alert. Then escalates the alert through all appropriate individuals in the process in any defined order
  + Detailed tracking of alerts (when, who has seen the alert, severity of alert). Alert queries can be very finely tuned (may be a learning curve involved)
  + Full control over notification rules sent to on-call individuals through apps, text messages, voice calls, or email. Responses can also be made from the app to address the incident
* **Critical Feature:** Heartbeat monitoring and Python API. Python API is unofficial but what little that exists is detailed.
* **Web app Features:** All the basics. Analytics pages are not working at the moment (404).



Web app alerts interface



Detailed cURL request creating an alert



Python API creating an alert

* **Integrations:** Native Slack and Azure integration (and 170+ more integrations) plus integrations created through Python or REST API
* **Customer Service:** 24/7 email, chat, phone support
* **Cost:** $39 USD per user per month
  + Unlimited API calls
  + Unlimited data retention

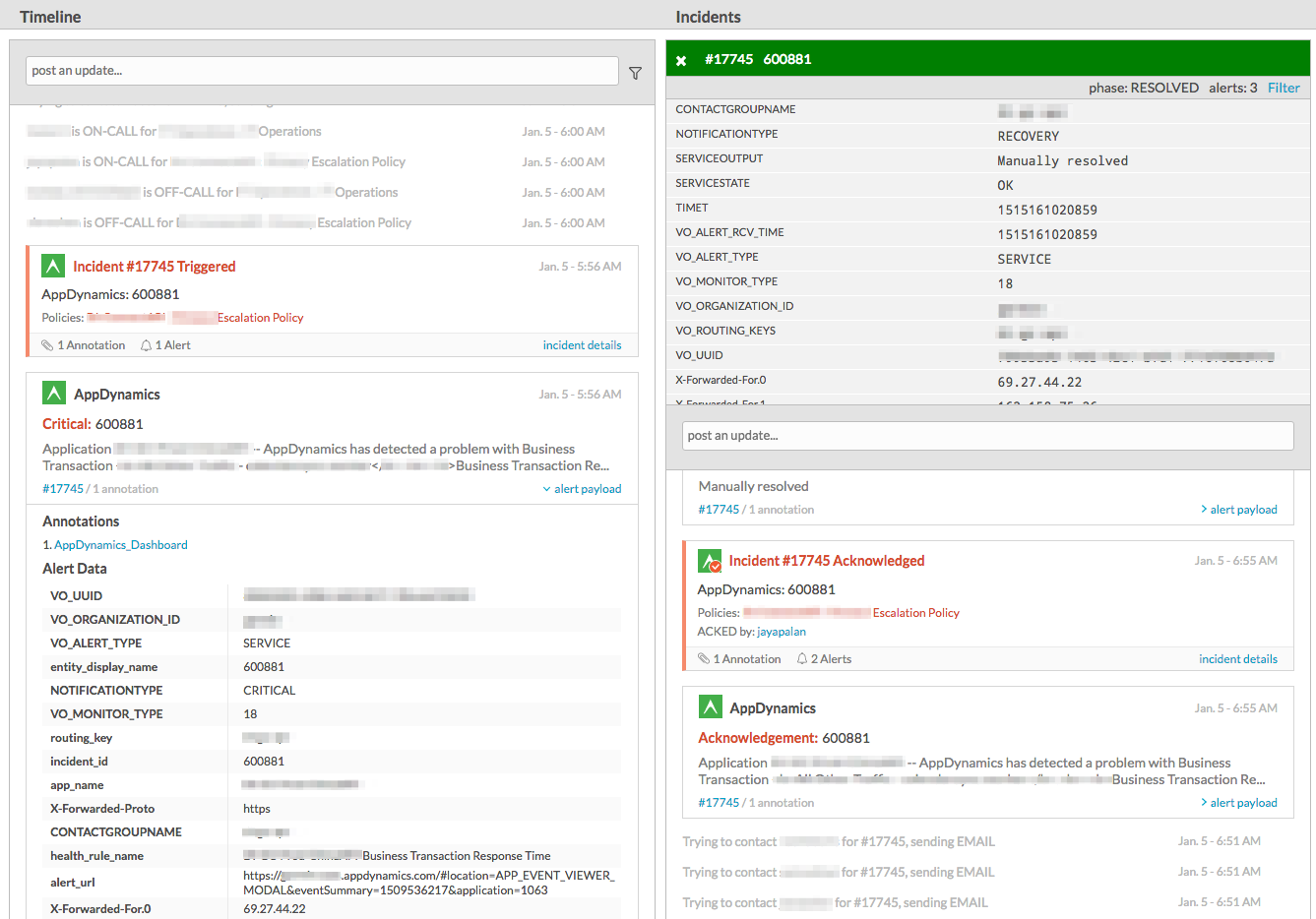
References:

1. <https://docs.opsgenie.com/docs/opsgenie-python-api>
2. <https://docs.opsgenie.com/docs/notification-preferences>
3. <https://www.opsgenie.com/features>
4. <https://www.g2crowd.com/products/opsgenie/reviews>
5. <https://www.opsgenie.com/pricing#pricing-details>

[**VictorOps**](https://victorops.com/) **-** [**REST Test Code Here**](https://drive.google.com/open?id=1UyddQYxZS_2PWE0w9QIrvgQQo-mwXi8v)

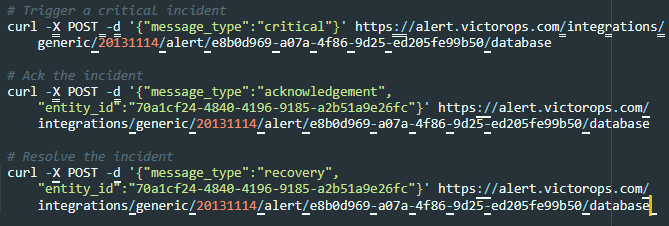
**Basic Features**

* On-Call Schedules
* Noise Suppression
* Live Call Routing
* Reporting
* API and Webhooks
* ChatOps
* Mobile
* Runbooks and Graphs
* Delivery Insights
* **API:** Contains a standard REST API but no Python API with VictorOps features. Also contains a web app
  + No command line utilities or Python API (less versatility)
  + All requests must be made through REST API (most feature-rich), the web app (limited in features), or the mobile applications
* **Web app Features:** On-Call schedule definitions with escalation policies
  + Routes critical alerts only to members that are on-call and are relevant to the alert. Then escalates the alert through all appropriate individuals in the process
  + **Critical Feature:** Incident Automation handling that kicks off escalations and smart suggestions for alert rules



Web app alerts interface

* Detailed web app containing many fields of details for incidents



Example cURL request for creating, acknowledging and resolving an incident

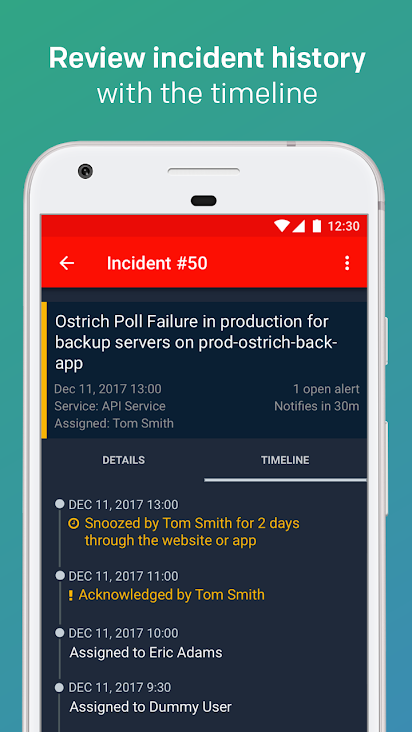
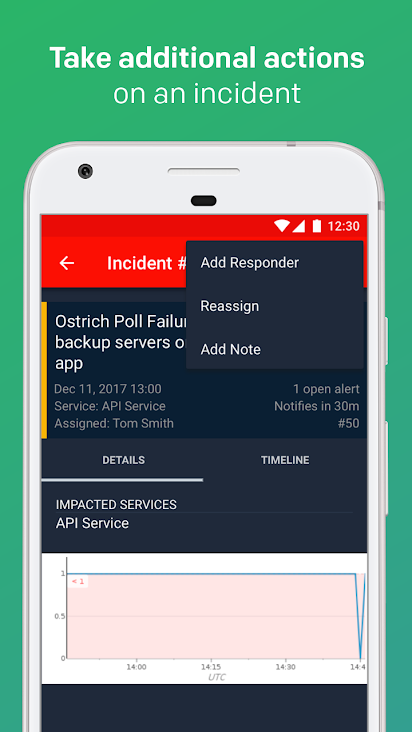
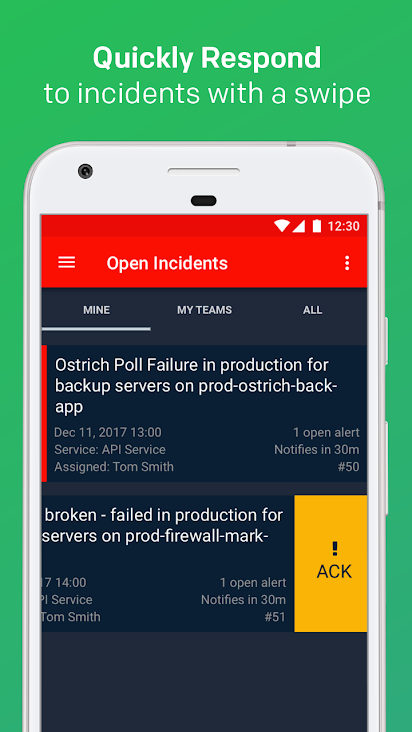
* **Heartbeat:** Heartbeat monitoring only when integrated with the Nagios system and network monitoring software
* **Customer Service:** Email, chat, phone support only supported from 5PM IST to 2AM IST on American weekdays
* **Integrations:** Over 100+ integrations as well (plus Slack and Azure Operations Management Suite)
* **Cost:** Standard Pricing: $49 USD per user per month
  + Unlimited API requests
  + Unlimited data retention

References:

1. <https://portal.victorops.com/public/api-docs.html>
2. <https://community.spiceworks.com/topic/766081-pagerduty-opsgenie-experiences>
3. <https://victorops.com/pricing>
4. <https://victorops.com/incident-automation>

**Comparing Mobile Apps**

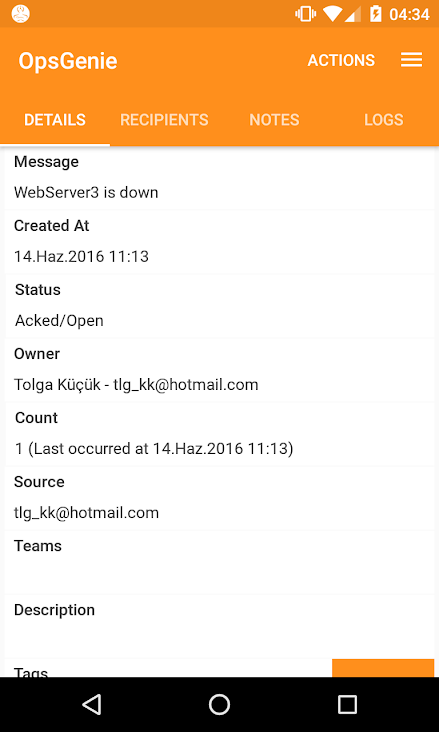
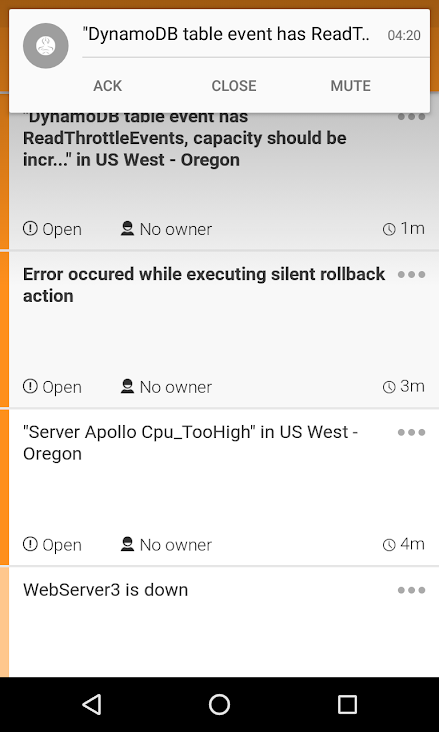
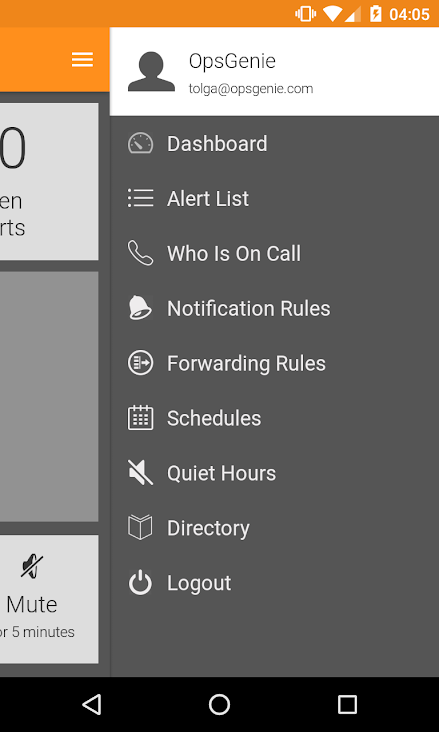
**PagerDuty**



**Features:**

* View extensive incident details
* Acknowledge and close incidents
* View an exact timeline of when the incident was triggered, the details of the alert that triggered it, and post notes to the timeline of the incident for others to see
* **“Run a play”** and automatically notify other specific users of the incident
* Attach annotations and notes to document the process
* View and configure the escalation policy to send notifications via app, SMS, or email based on urgency of the incident
* View all other colleagues that are on-call and view a daily schedule of the on-call list

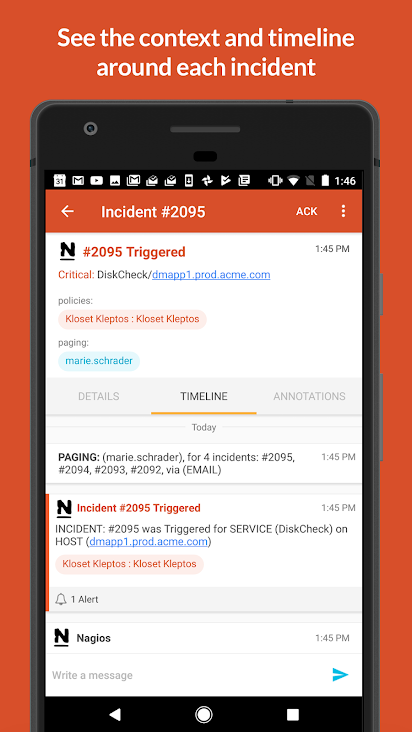
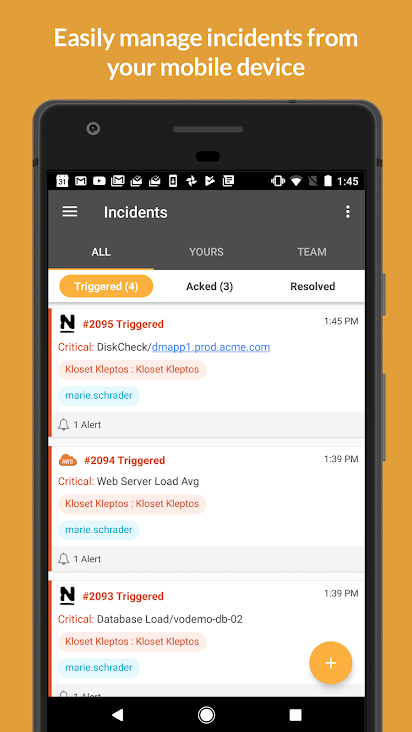
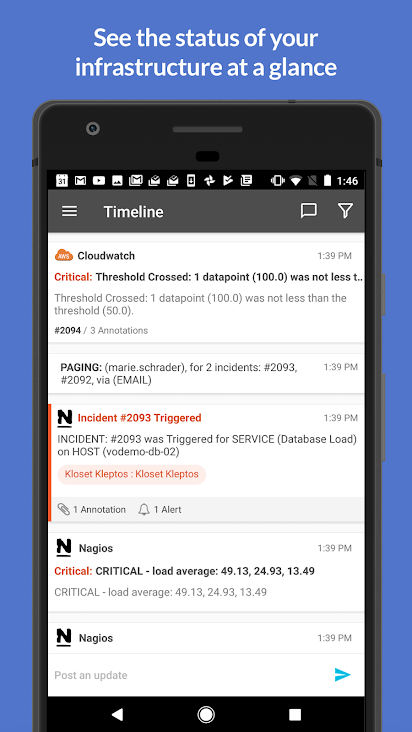
**OpsGenie (would NOT recommend based on the app)**



**Features:**

* View extensive incident details
* Confusing interface
* Acknowledge and close incidents
* Temporarily mute notifications for any amount of time and set quiet hours when you don’t want to receive notifications
* View an exact timeline of when the incident was triggered, the details of the alert that triggered it, and post an update to the timeline of the incident for others to see
* Attach annotations and notes to document the process
* View all incidents in the company, all incidents in the team, or all incidents that you are responsible for
* View all other colleagues that are on-call

**VictorOps**

****

**Features:**

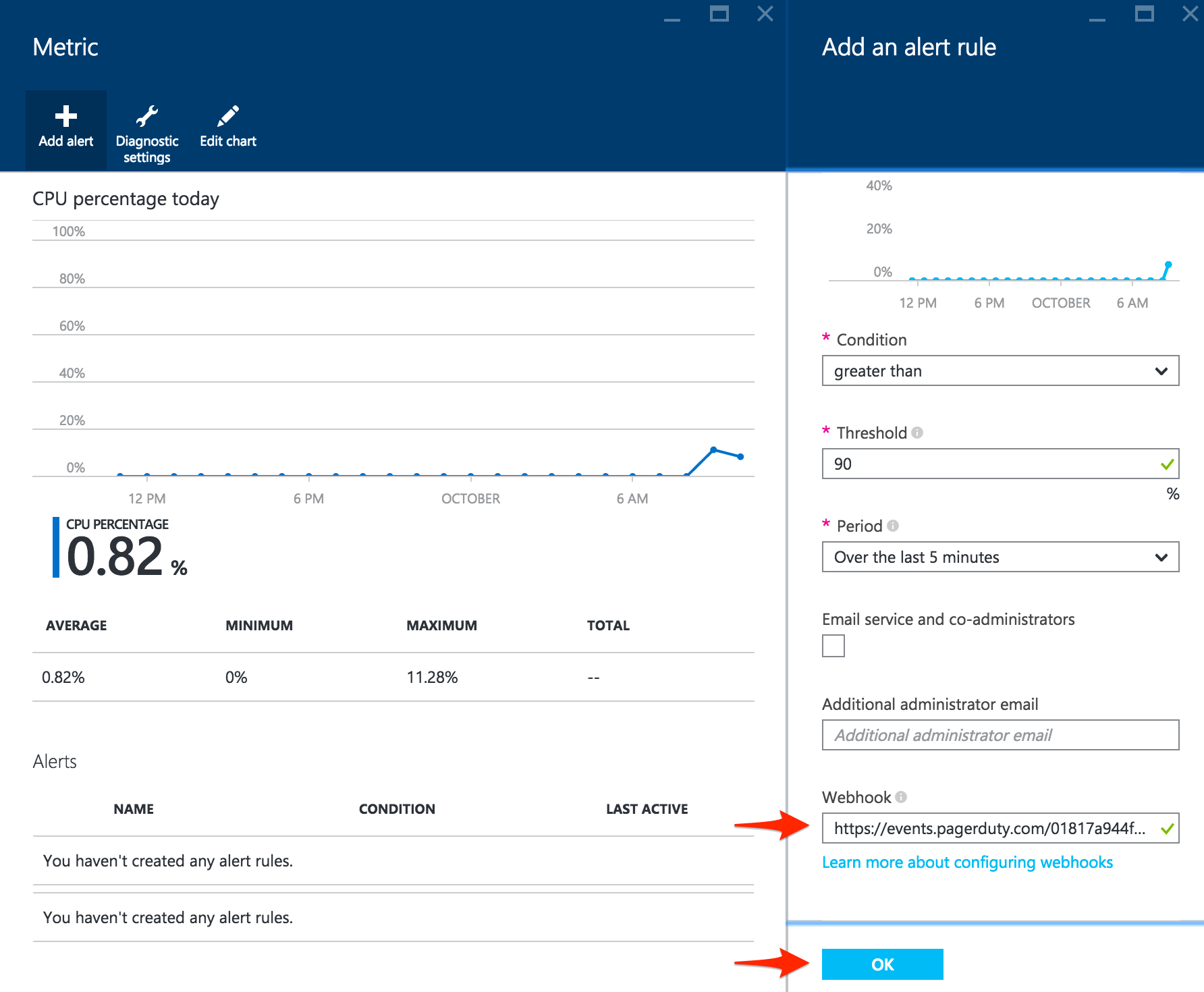
* View extensive incident details
* Acknowledge and close incidents
* View an exact timeline of when the incident was triggered, the details of the alert that triggered it, and post an update to the timeline of the incident for others to see
* Attach annotations and notes to document the process
* View all incidents in the company, all incidents in the team, or all incidents that you are responsible for
* View all other colleagues that are on-call

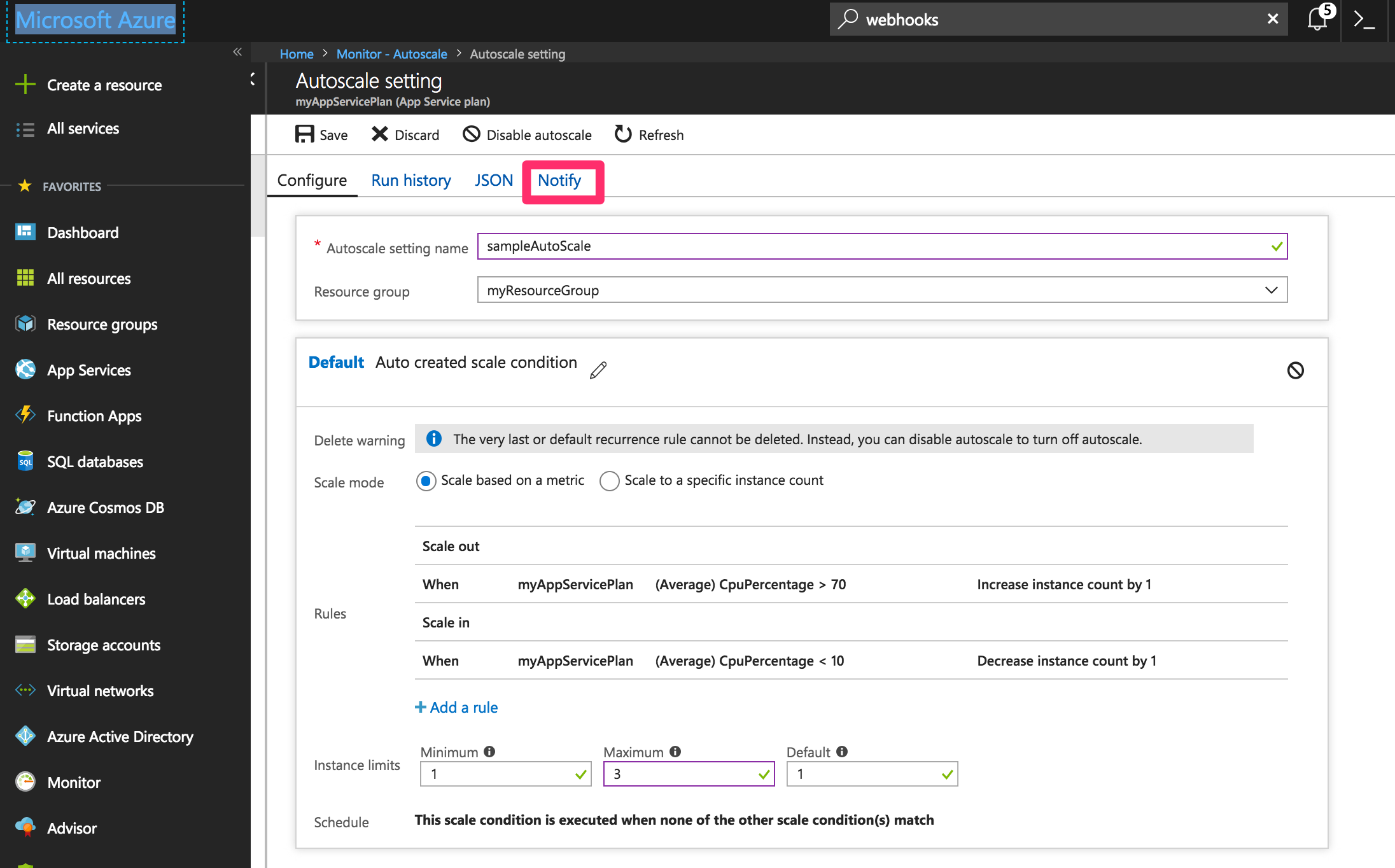
**Azure Integration**

* In general, it can be used to monitor CPU usage, disk I/O, etc. A threshold can be placed on every monitored statistic at which an alert is created and an incident is produced on the appropriate service via a webhook

**PagerDuty and OpsGenie**

* + Platform-level metrics for Azure services
  + Application Insights (for standard and custom metrics)
  + Autoscale Notifications
  + Audit Log Events

****



**VictorOps (not recommended)**

* + Azure OMS alerts that allow for monitored searches (limited)

References:

1. <https://www.pagerduty.com/docs/guides/azure-integration-guide/>
2. <https://docs.opsgenie.com/docs/microsoft-azure-integration>
3. <https://help.victorops.com/knowledge-base/microsoft-azure-oms-integration-guide-victorops/>