**Summary**

I began by manually integrating the VictorOps platform with Azure alerts and was able to successfully integrate it this time using Logic Apps and a custom JSON payload that VictorOps knew how to parse. This serves to prove that even if VictorOps doesn’t support a specific platform, we could easily use their generic REST integration to work with any platform that can send HTTP POST requests.

Following this I reviewed the pricing details of PagerDuty and VictorOps. If both services are going to be subscribed to at their $49 rate, we essentially receive their respective full suite of features. If we are aiming for their $29 rate, the most important feature we would lack in VictorOps is the lack of runbook usage.

Whereas with PagerDuty, the only vital feature we would lose is access to “health reports” showing historical incident data. Again, the PagerDuty UI is arguably more cluttered to view compared to VictorOps’ UI. Because we would only be losing some runbook features with VictorOps, I’d **recommend VictorOps** at the $29 rate if we would like to save money.

I reviewed the **Logz.io** integration with both services and the integrations work essentially the same, displaying the vital data/message triggered by the ELK stack alert found in Logz.io. It worked smoothly in AWS (with Windows) and GCP (with Ubuntu) and I was able to ship logs regarding the VM from the ELK stack to Logz.io and of course to both PagerDuty and VictorOps.

If we are depending on Logz.io more than the native AWS and GCP alerts, either AWS or GCP are solid choices in that aspect.

Finally, I researched the integrations of both platforms with native AWS and GCP alerts in preparation for our move from Azure. The first vital piece of information was that GCP does not integrate with PagerDuty at the moment due to the use of an old API that has not been updated. All other integrations work adequately with the platforms.

I **personally preferred the AWS monitoring service** (Cloudwatch) compared to GCP’s service (Stackdriver). One major reason was the capability of creating alerts that could take actions in addition to notifying the user on their incident management platform of choice. These actions included restarting a VM instance of even stopping it altogether.

In addition, alerts could be triggered AND resolved automatically on a platform by AWS once the metric that triggered the alert returned to a normal state.

**A combination of VictorOps with AWS would be highly efficient and easy to set up.**

**Manually Integrating VictorOps with Azure**

Note: **This works for both metric alerts and log alerts. You simply have to connect either alert to the action group with the logic app’s webhook (which will send the message to VictorOps).**

In a support case I opened with Azure, the support engineer told me that there are plans on the horizon that with deprecation/merging of OMS, custom JSON could be added to all Azure Monitor alert types/rules.

For the time being, my fix is to use a Logic App to mold the payload into one VictorOps will accept the request. Here is a **summary** of what we’re about to do:

Consider any alert created by the Azure Monitor. Oftentimes we would simply link this alert to an action group with a webhook that links directly to an incident management platform. For our workaround, we will instead route the webhook URL to a logic app which will handle the alert payload (because we also cannot construct a custom JSON with the current version of Azure monitor).

In our logic app, whenever an HTTP request is received at our logic app’s URI, a HTTP 200 response is triggered and sent back to the original requester. In this case, the requester is the alert rule’s action group firing an HTTP request. In the logic app, we create an HTTP POST request and send it to VictorOps’ generic REST API. This POST request contains a JSON body that is written with the [Logic Apps Workflow Definition Language](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-workflow-definition-language) and contains the necessary JSON fields [required by the VictorOps REST endpoint](https://help.victorops.com/knowledge-base/victorops-restendpoint-integration/). More JSON fields can be added as necessary to make the incident descriptions more readable.

### **Logic App**

Create a [*Logic App*](https://docs.microsoft.com/en-us/rest/api/logic/). The Logic App will serve as the central structure for the integration with VictorOps. Follow these steps:

1. **Create a new Logic App** by clicking the *Create Resource* button in the top left corner of the Azure Portal. You can equivalently follow the first couple steps of [this](https://docs.microsoft.com/en-us/azure/logic-apps/quickstart-create-first-logic-app-workflow) documentation.
   1. **Name the application**.
   2. **Select** an existing **Resource Group**.
   3. **Create the logic app.**
2. From the dashboard, select the Logic App you have just created.
3. From the Logic App blade, select Logic App Designer
   1. For the trigger condition, select "**When an HTTP Request is received**"
   2. Click **New Step, s**elect **Add an Action**
   3. Select **Request Response**
      1. Leave it as responding with a 200 status code
   4. Click **New Step, s**elect **Add an Action**
   5. Select **HTTP - HTTP**
      1. Method: **POST**
      2. URL: This can be found in your VictorOps account under *Settings* > *Alert Behavior* > *Integrations* > *Generic REST*
      3. Headers: **Content-Type** | **application/json**

4. Body:

|  |
| --- |
| {  "data": "@triggerBody()",  "entity\_display\_name": "@{triggerBody()?['context']?['name']}",  "entity\_id": "@{triggerBody()?['context']?['id']}",  "message\_type": "@{if(equals(triggerBody()?['data']?['status'],'Activated'),'recovery','critical')}",  "state\_message": "@{triggerBody()?['context']?['description']}",  "monitoring\_tool":"Azure" } |

Note that this JSON Body will require some tweaking in the future to get the data we absolutely want in the incident. Once again, view the Logic Apps Workflow Definition Language article for more information. Most of the incident information sent to VictorOps is found in the **data** field.

5. In the top left of the blade, click **Save**

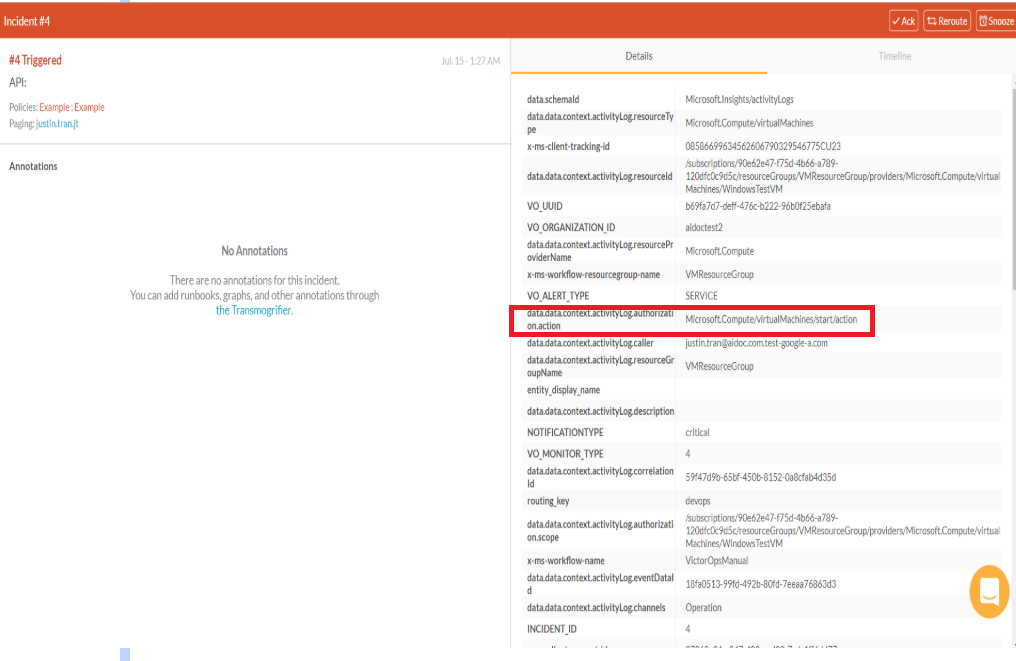
1. Back in the Logic App Designer and under the "*When an HTTP Request is Received*", the url has now been generated. **Copy this url** to the clipboard.

### **Alerts**

Follow these steps:

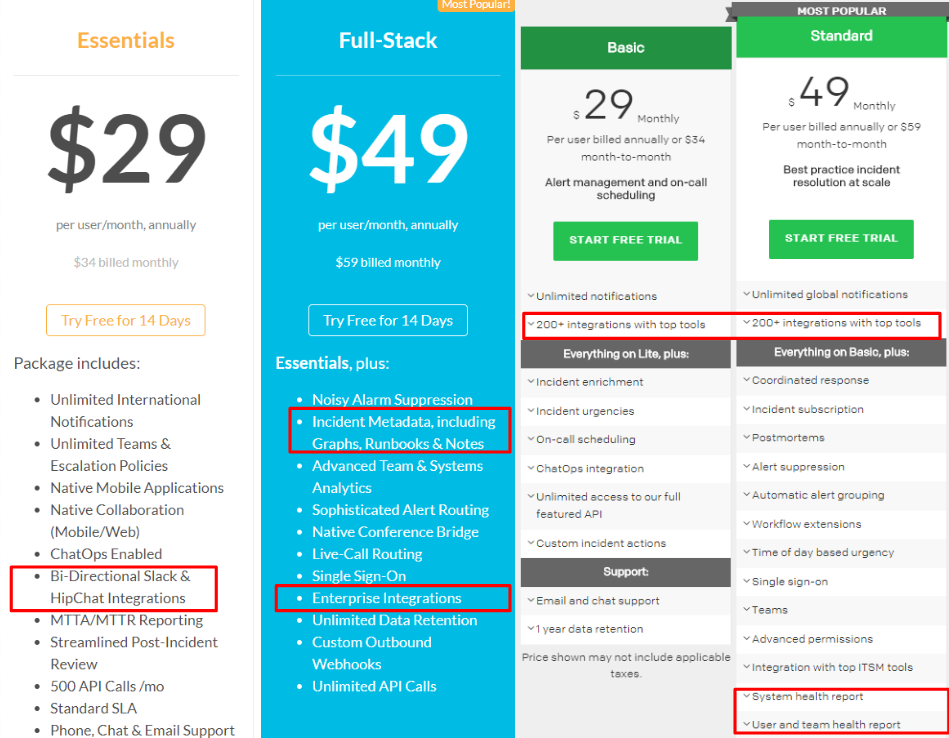
1. From the left menu pane, select **Monitoring** >> **Alerts** >> **New Alert Rule**
2. Define the alert however you would prefer to monitor things. For testing purposes, I find it easiest to monitor all administrative operations a condition (see Step 3)
3. *Whenever the "****Administrative Activity Log All Administrative operations****" has "****any****" level, with "****any****" status and event is initiated by* "**<admin\_email\_addr>**"
4. Define the alert details with any name and description
5. For the last step, select a **New Action Group**, this action group will fire a webhook towards
   1. For the action, select webhook
   2. For the url of the webhook, **paste the url copied earlier** from the Logic App
6. Save

### **Results**

The following incident is produced by VictorOps when the VM on Azure is started:

Because we are able to manually integrate VictorOps with Azure (for the time being without the need for a built-in integration), we wouldn’t be needing an email integration like the one created with PagerDuty.

Again, we would need to edit the JSON Body sent from the logic app to make it look nicer in VictorOps. Keep in mind that this example incident is not indicative of what an actual alert from our VMs would look like as this is an error with different alert fields from our own.**Pricing Differences**

* VictorOps does not allow us to include dynamic notes (Graphs, runbooks, notes) on incidents with the Essentials package
  + Despite the bullet points that say “Slack and Hipchat integrations”, the Essentials tier includes all integrations except for ServiceNow and StatusPage
* Meanwhile, PagerDuty does not allow us to view overall report data on the Basic package (number of incidents, speed at which they were acknowledged, etc.)

We would have to make a tradeoff and decide which of these features is more necessary if we are planning on paying for a $29 subscription. The $49 subscriptions for both services provide the full suite of their features and are fairly equal (except for the differences we discussed in our last meeting)

**Logz.io Integration with AWS/GCP and PagerDuty/VictorOps**

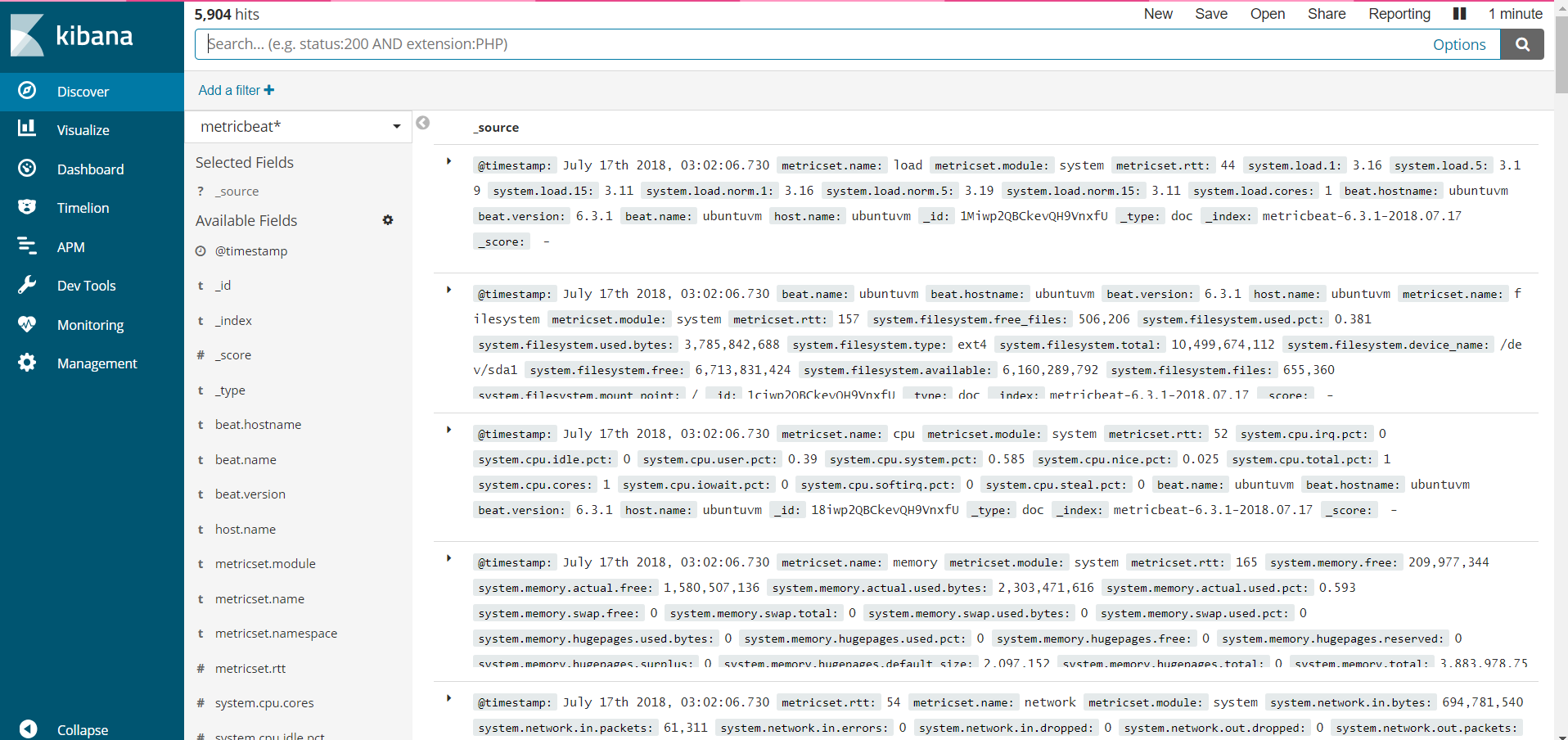
With AWS and Google, we first set up a VM instance (I personally used an Ubuntu instance in GCP but the process for setting up ELK on Windows which I did in AWS is also listed below) and install the ELK stack onto it. **Remember to open up the required ports on whichever platform you use!** We then navigate to [http://VM\_IP\_HERE:5601](http://my_ip_here:5601) to view the data in Kibana and create an index pattern*.* Finally, we ship the logs to Logz.io for display using FileBeat for automatic shipment (or a variety of other methods such as *cURL or rsyslog* for manual shipment) on our VM. For full details, refer to the following articles:

1. GCP: <https://logz.io/blog/elk-stack-google-cloud/>
2. Amazon: <https://logz.io/blog/install-elk-stack-amazon-aws/>
3. Windows on AWS or GCP: <https://logz.io/blog/elastic-stack-windows/>
4. How to ship logs once ELK is up and running on any instance: <https://app.logz.io/#/dashboard/data-sources/Filebeat>
5. An alternative way using GCP to collect logs and send them to Logz.io without installing an ELK stack: <https://logz.io/blog/google-pub-sub/>

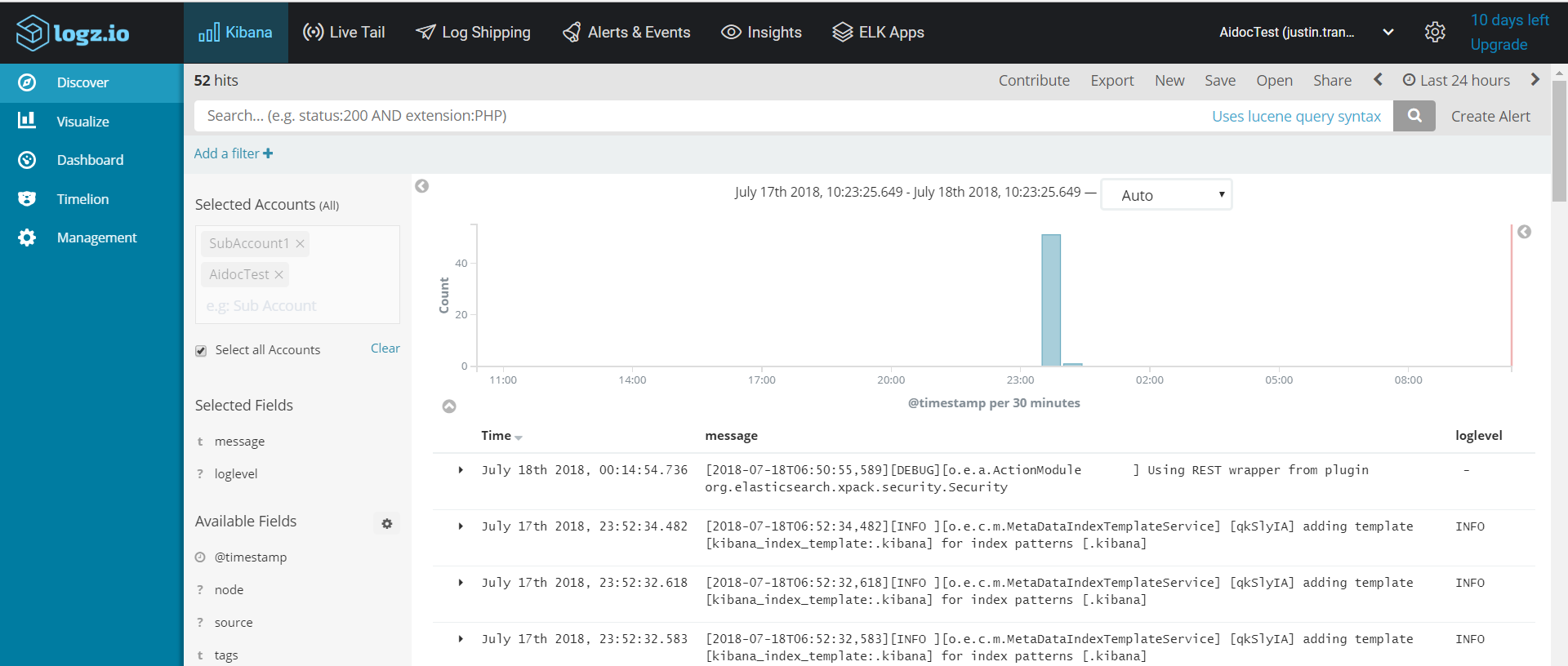
***Note on shipping logs:***

* Filebeat is the preferred method for automatic shipment of logs and requires a single setup process that you can create and forget about (much easier in Windows)
* Rsyslog requires opening up port 5000 and is manual
* cURL is manual as well
* All manual log shipping methods can be set up as cron jobs or Powershell scripts that automatically occur, rendering them as valid automatic shipment methods as well
* The log files are all found in the /var/log/elasticsearch directory on Linux and C:\ProgramData\Elastic\Elasticsearch\logs on Windows.

Another option is to use the Logz.io SaaS options in the AWS and Google Marketplaces respectively. They do have a subscription cost but you won’t have to go through the trouble of setting up the ELK stack. You will still have to configure a service to ship logs from your VM to Logz.io.



Example of logs being shown in a local Kibana server. These logs are from my GCP Ubuntu instance. The logs from the AWS Windows instance in Kibana look similar. These logs are then sent to Logz.io and can be seen as graphs.

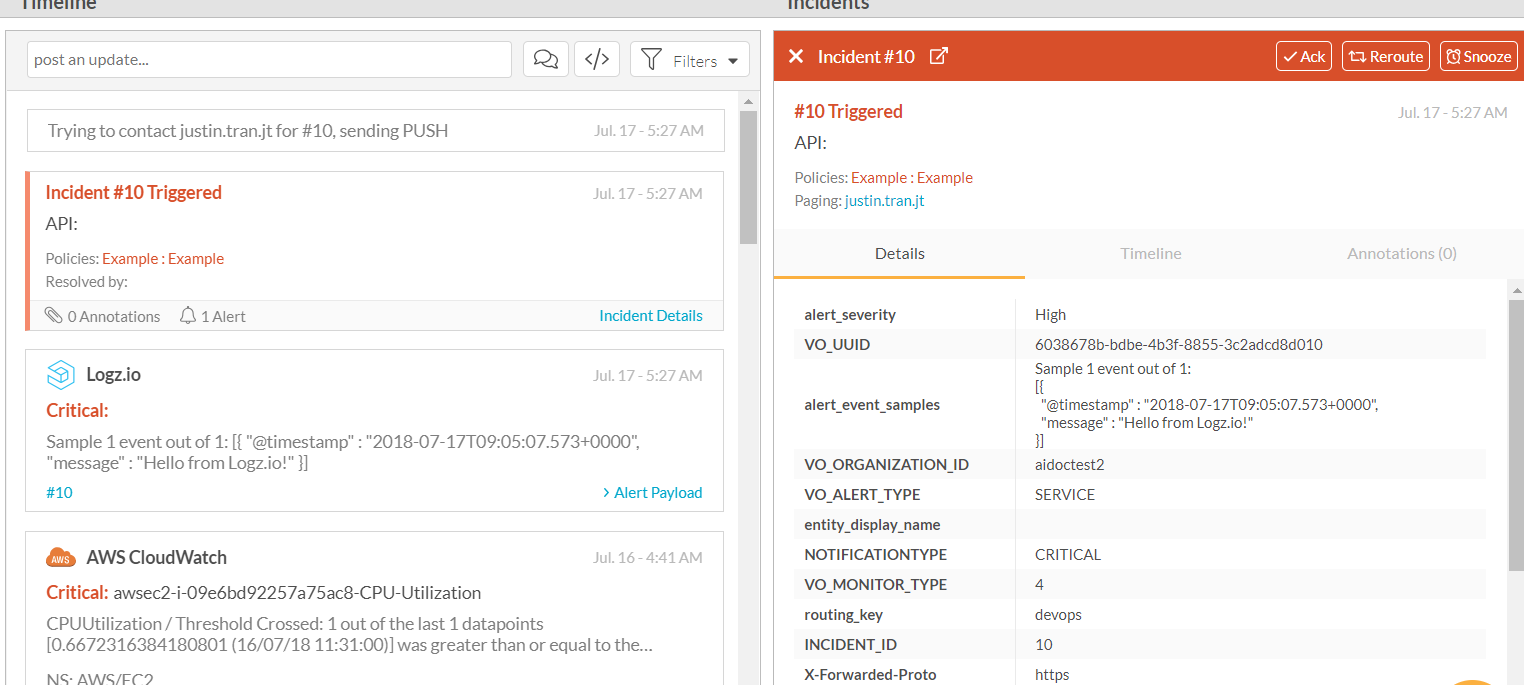
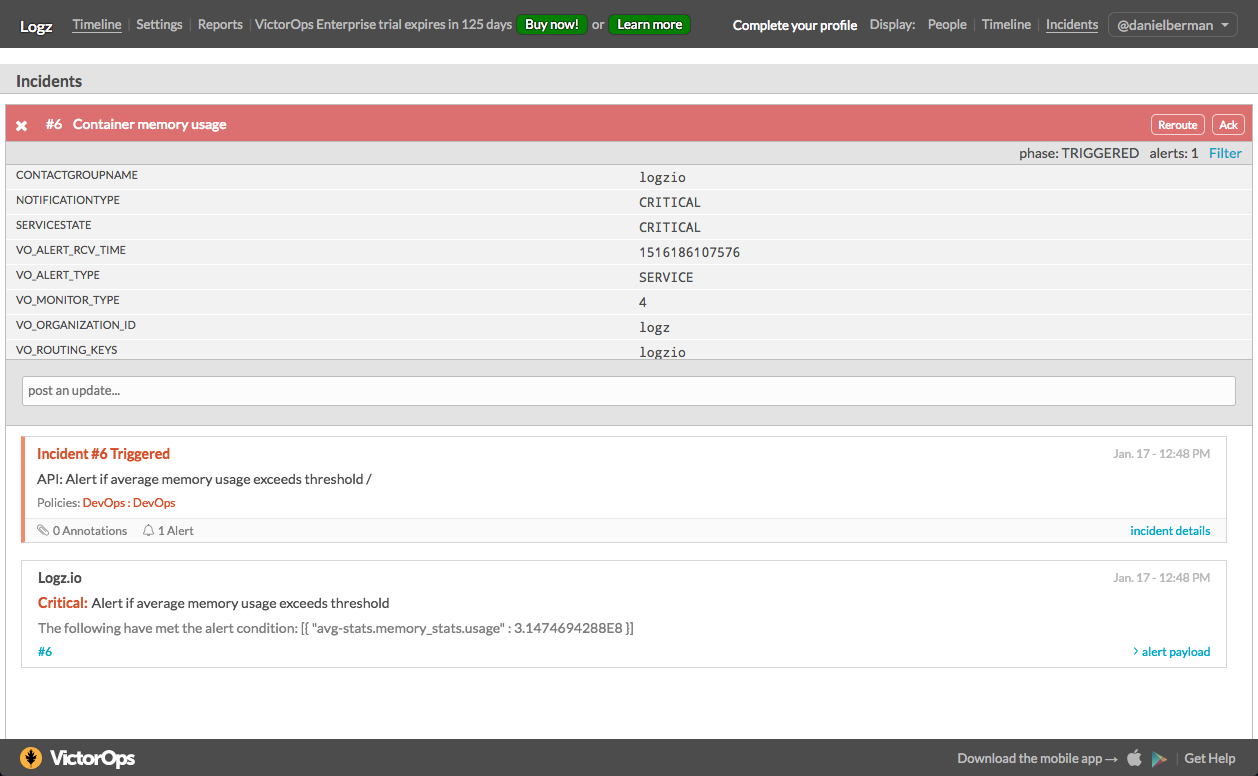


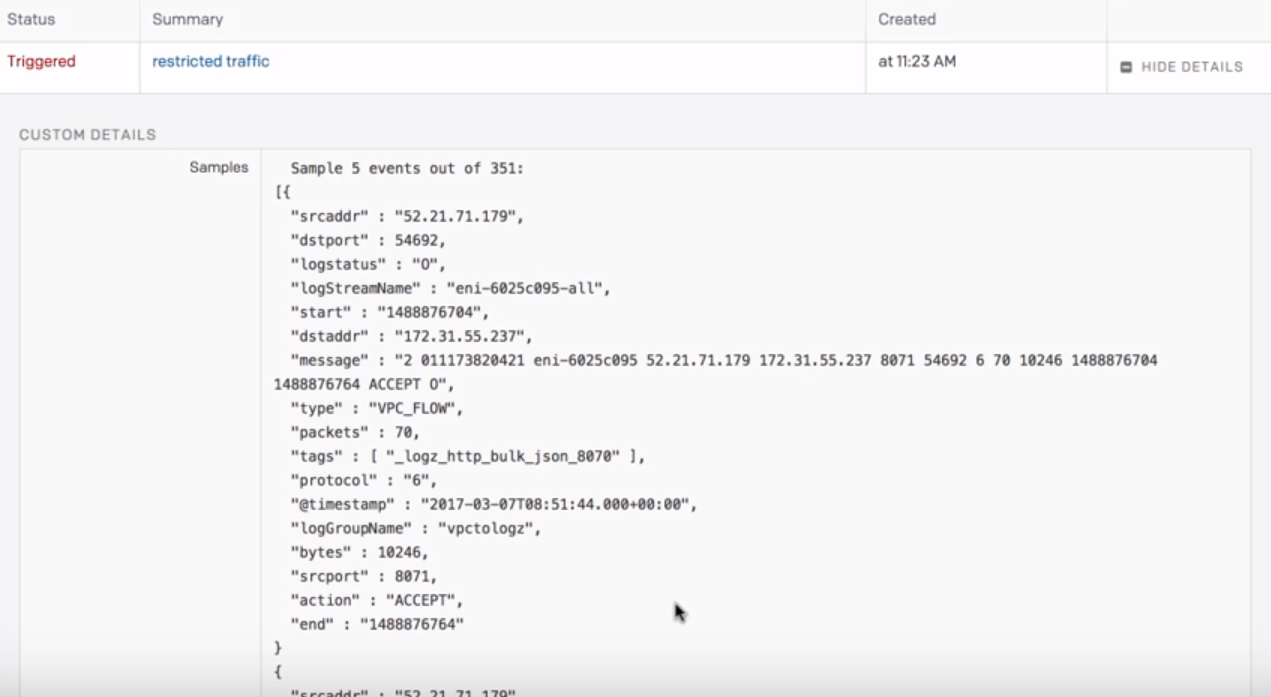
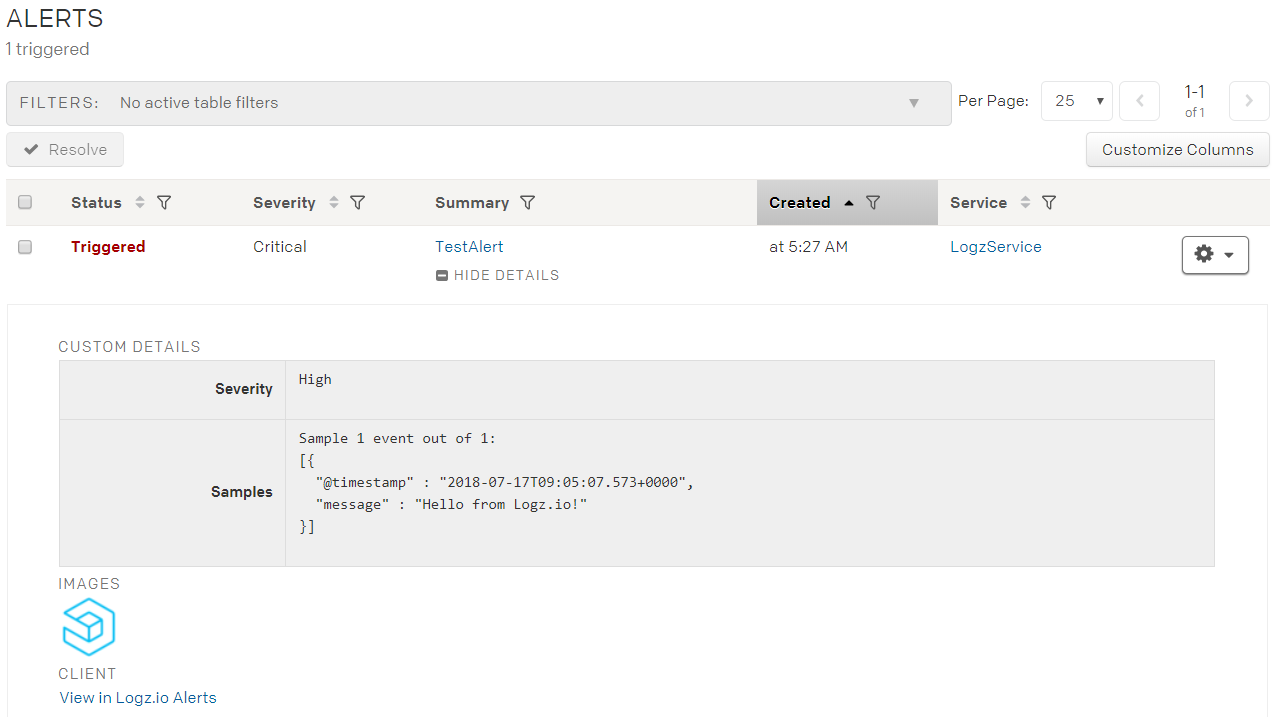
An example of logs successfully shipped from the ELK stack to Logz.io

Both PagerDuty and VictorOps contain integrations with Logz.io and they work seamlessly with Logz.io. When an alert in Logz.io is triggered by a metric in the ELK stack, a corresponding incident is triggered in PagerDuty and VictorOps. The logs work perfectly and are similar in visuals to other alerts sent to the interfaces such as Azure alerts.

1. PagerDuty Integration guide: <https://www.pagerduty.com/docs/guides/logz-io-integration-guide/>
2. VictorOps Integration guide (uses REST integration): <https://help.victorops.com/knowledge-base/victorops-logz-io-integration/>

Below are the examples of the test alerts triggered in VictorOps and PagerDuty:

Above and below are 2 separate alerts triggered by Logz.io and seen in VictorOps

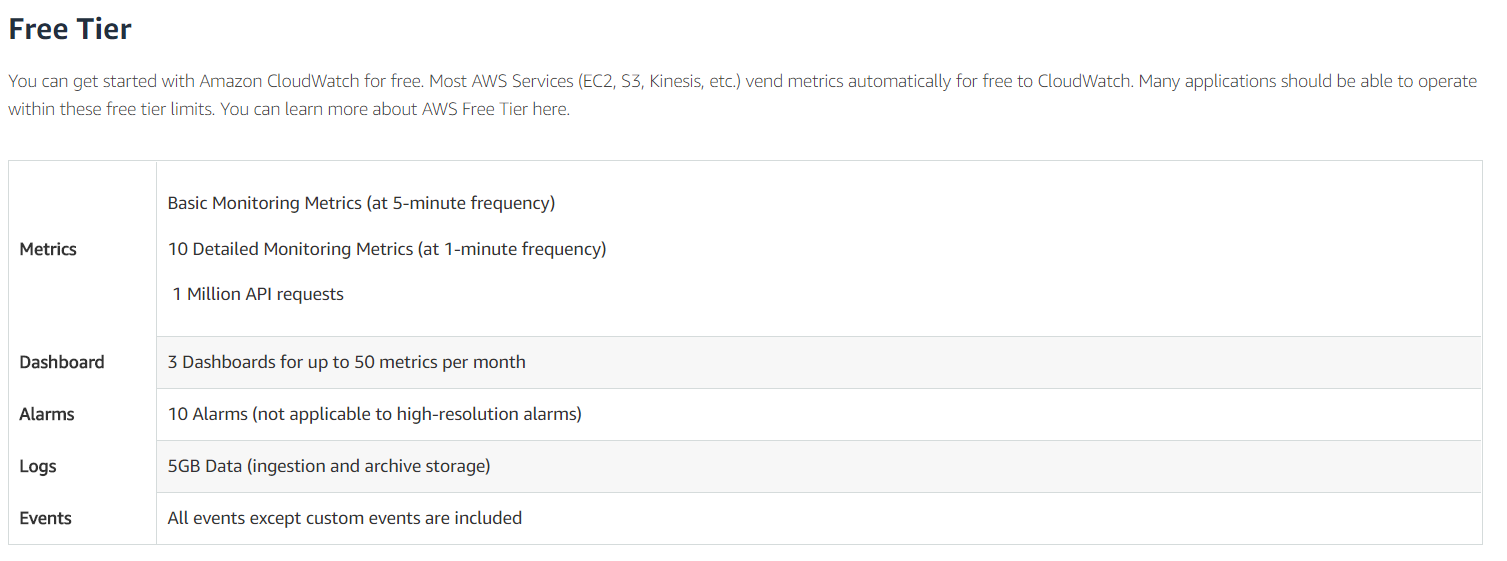


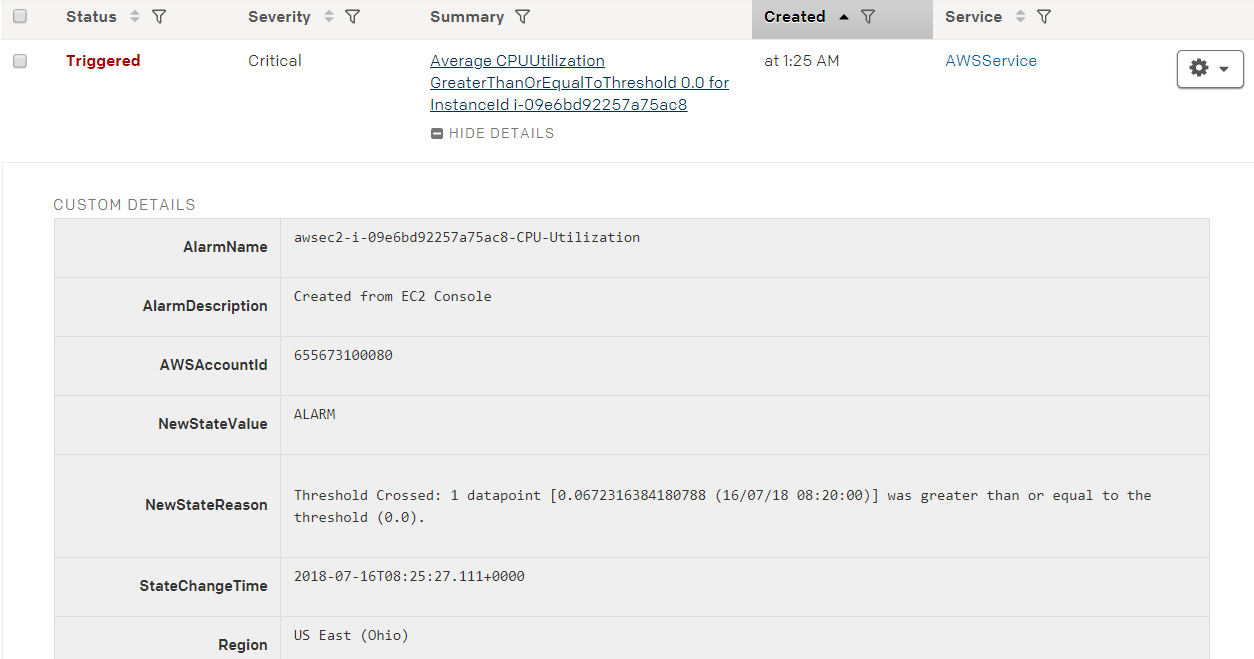
Example of 2 Logz.io alerts sent to PagerDuty

**Direct AWS and GCP Integration with PagerDuty/VictorOps (without using Logz.io)**

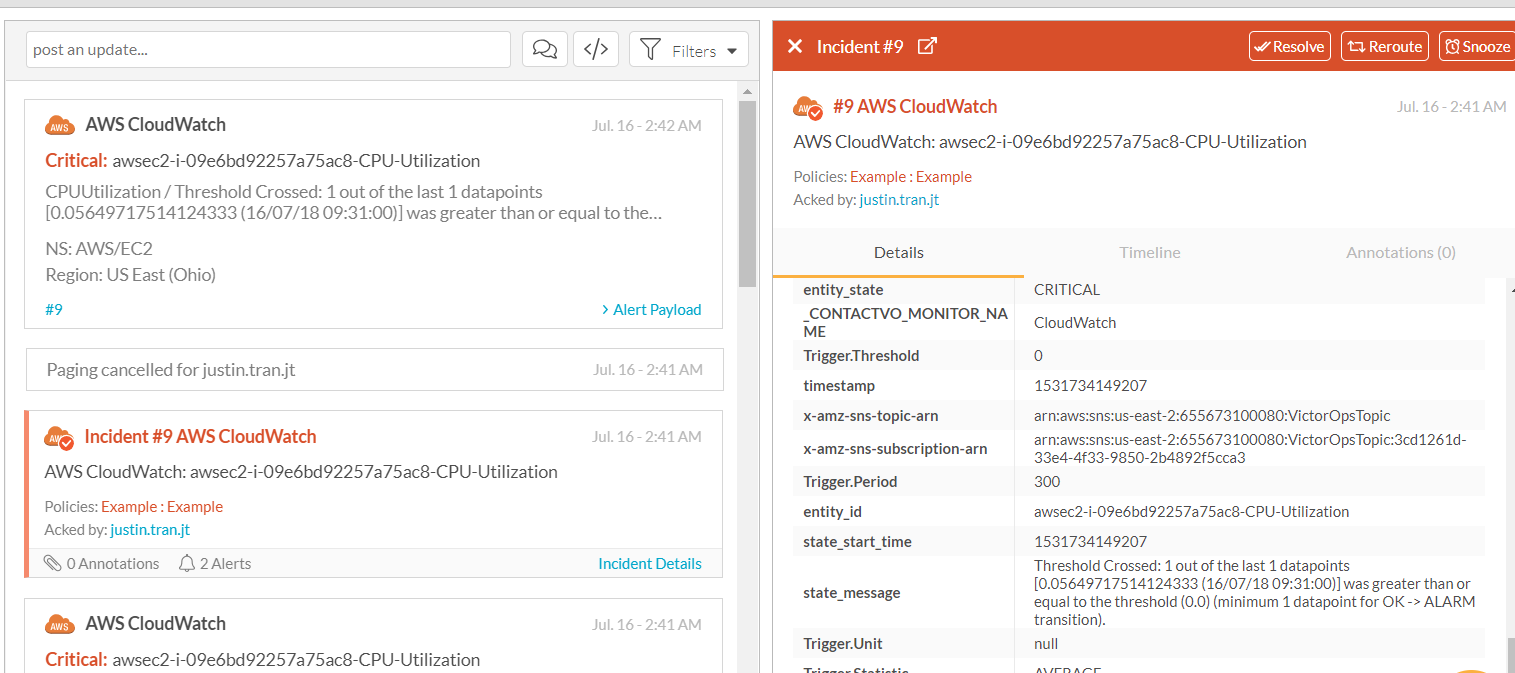
[AWS](https://aws.amazon.com/cloudwatch/pricing/) ([PagerDuty](https://www.pagerduty.com/docs/guides/aws-cloudwatch-integration-guide/), [VictorOps](https://help.victorops.com/knowledge-base/victorops-cloudwatch-integration/)):

AWS Cloudwatch is the service that provides metrics and logs/alarms for AWS platform. It only provides metrics and alerts for AWS services whereas GCP can provide metrics and alerts for a variety of cloud platforms. The integrations work by using Cloudwatch and AWS SNS (simple notification service).

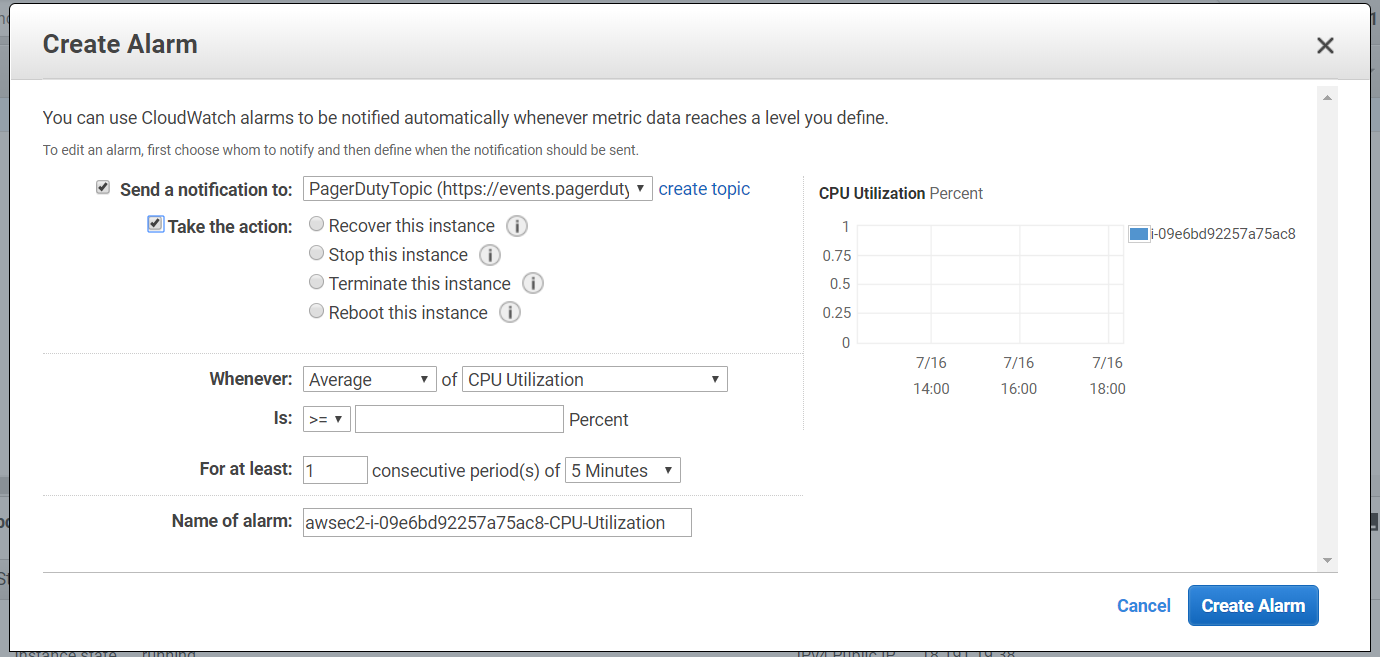
To test PagerDuty and VictorOps, I created a test alert in AWS that monitored the metric of a VM in AWS EC2 using Cloudwatch and SNS. These services all worked together incredibly easy and creating alerts was a very simple process that is more customizable than both Azure and GCP. For more on the capabilities of Alerts, scroll down. For now, the next two images show what test alerts in PagerDuty and VictorOps look like. They are fairly standard but detailed.



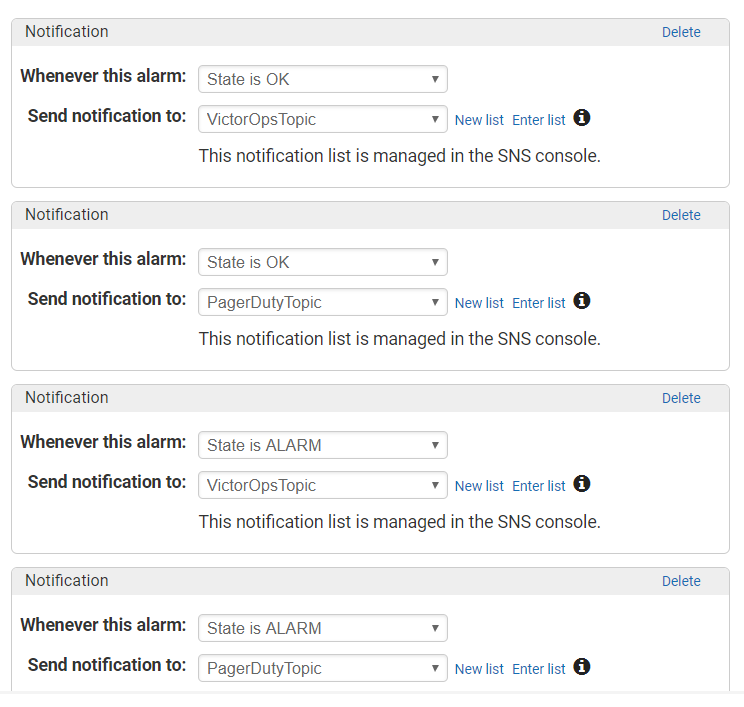
PagerDuty



VictorOps

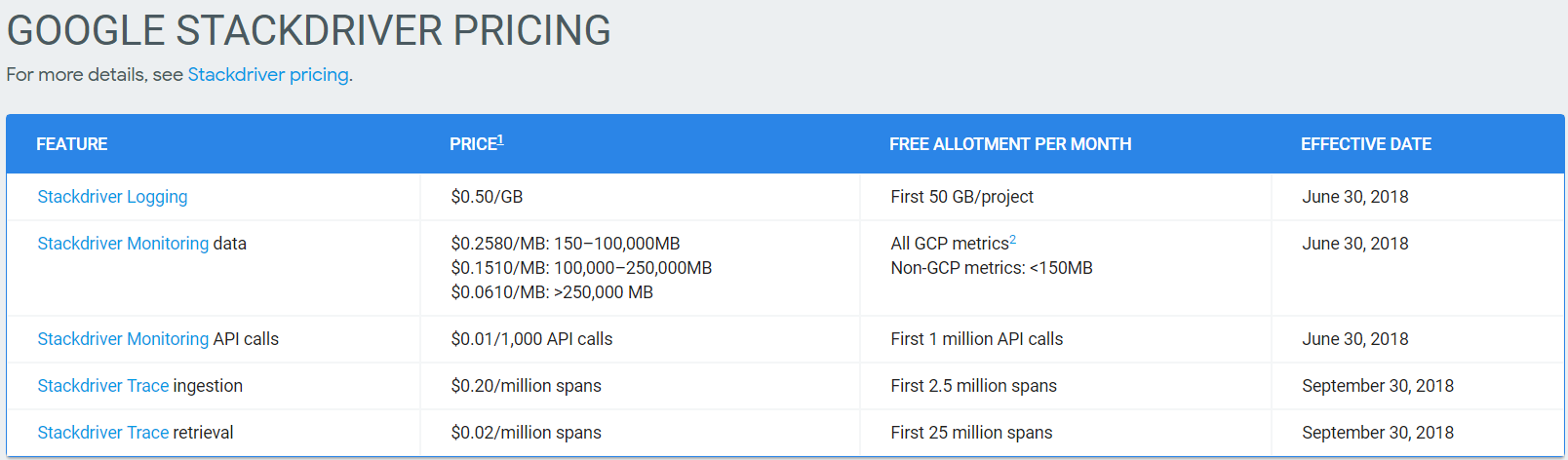


One significant feature of AWS that is easy to use is the ability to take automatic action when a specific alert is triggered. For instance if CPU usage is at 100% for a length of time, the VM instance can be rebooted or stopped automatically without a user having to handle it manually upon seeing an alert (but the user would still receive the alert notification).

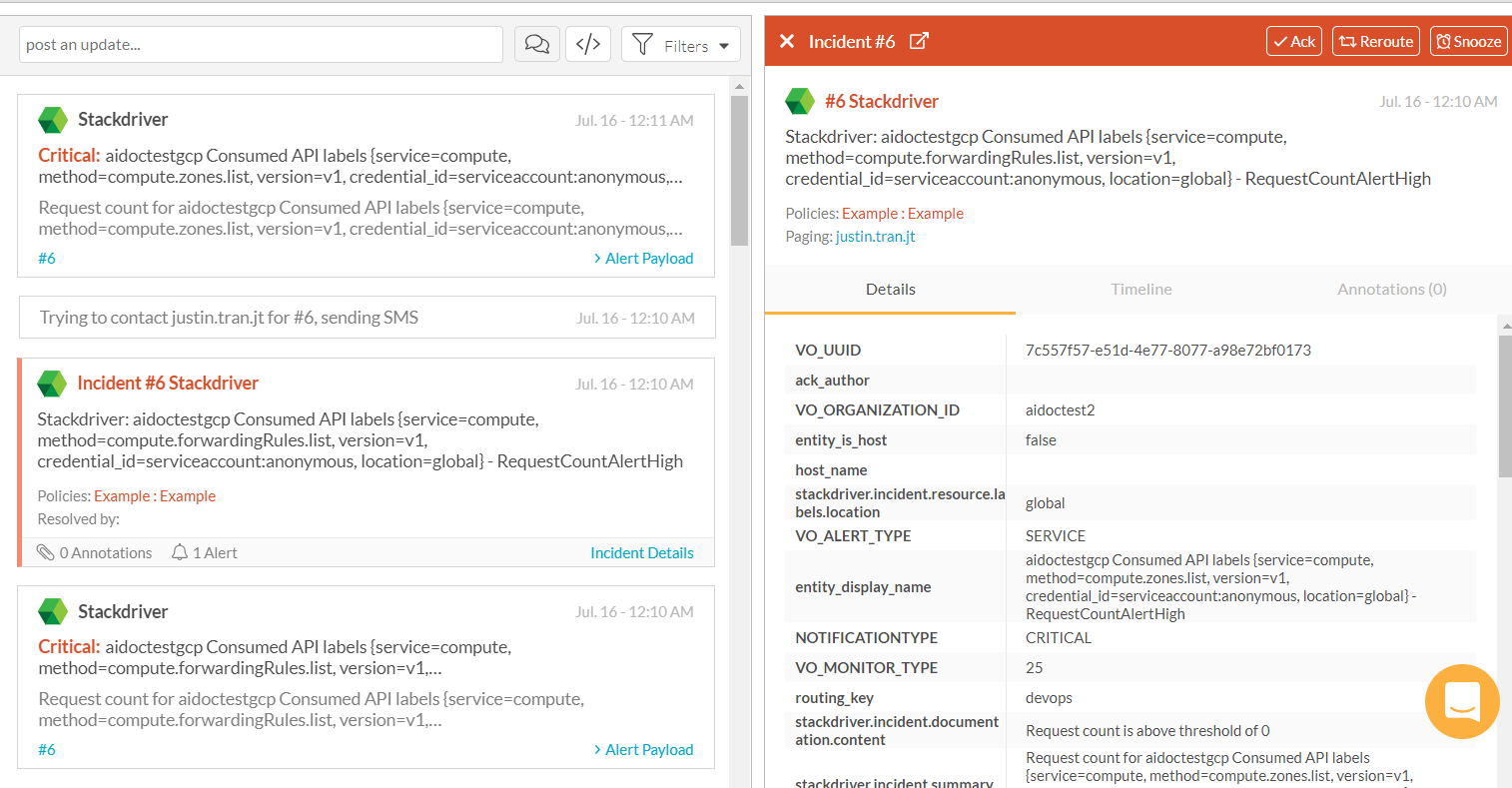


Another interesting feature is the ability to not only trigger alarms when a threshold is met, but to also automatically resolve incidents in integrated incident management platforms when the state of the metric returns to normal or has been resolved automatically by the system.

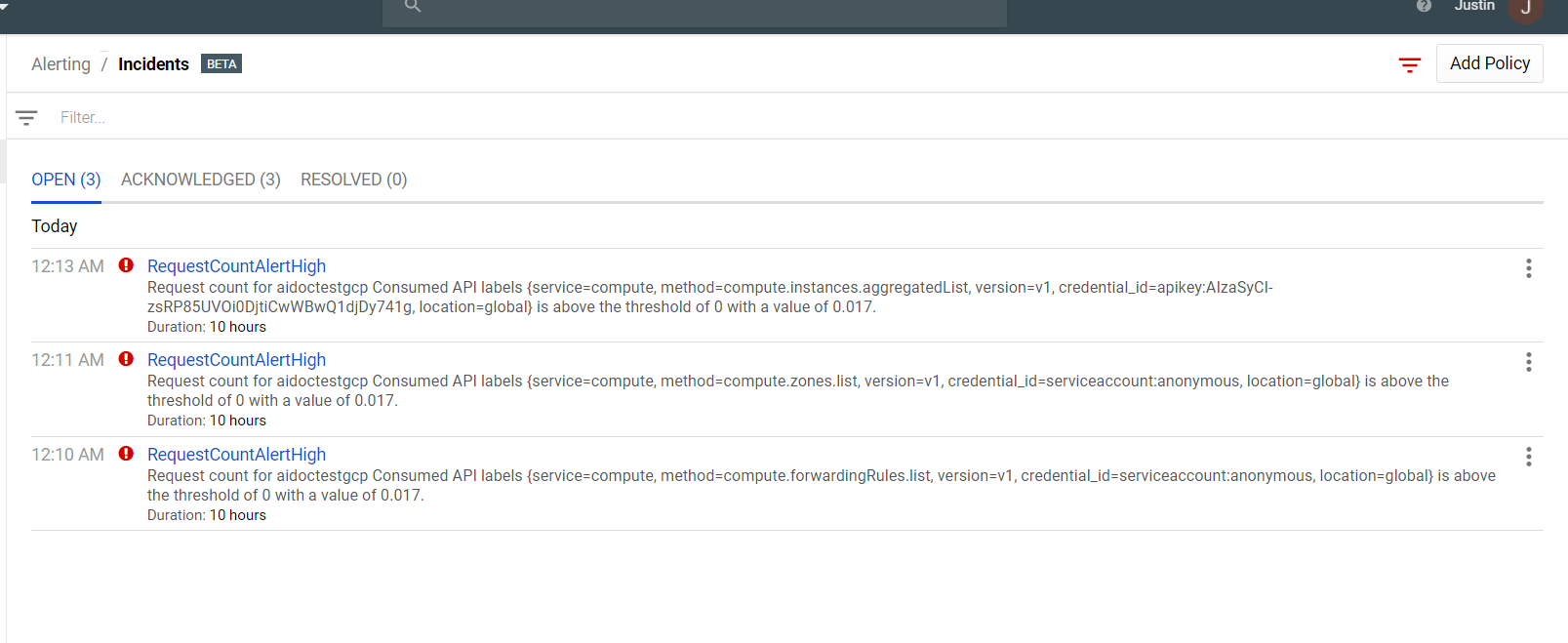
[GCP](https://cloud.google.com/stackdriver/) ([~~PagerDuty~~](https://www.pagerduty.com/docs/guides/stackdriver-integration-guide/), [VictorOps](https://help.victorops.com/knowledge-base/victorops-stackdriver-integration/)):

PagerDuty and VictorOps contain integrations with GCP through a program called Google Stackdriver which can actually monitor and log data from practically any cloud computing service (including AWS and GCP).

**NOTE: The current version of Stackdriver integration with PagerDuty is deprecated and does not work with the current PagerDuty API. PagerDuty is planning on updating the integration but has not pushed an update yet.**

VictorOps provides an automatic integration with Stackdriver when it’s connected to GCP. As an example, I created an alert condition that triggered in Stackdriver when log requests were made. This triggered successfully and sent immediately to VictorOps where the incident can be clearly seen below.

Also note that Stackdriver also has its own incident management features (though they are somewhat rudimentary). You can see an example of the triggered alerts showing up as incidents in Stackdriver itself. One of the rudimentary parts of Stackdriver’s incident management is the inability to mass acknowledge or mass resolve alerts when they appear. There is also no on-call system supporting multiple users that can be scheduled.



Presentation Process:

1. Re-enable VictorOps and PagerDuty phone notifications
2. Turn on Logz.io alerts
3. Start GCP VM and sudo service \* start (Elasticsearch, Kibana, metricbeat)
4. Navigate to <http://35.185.82.245:5601>
5. Show all metrics being logged in Kibana and explain how they are shipped to Logz.io where they are then triggered and show up as incidents in PagerDuty/VictorOps
6. Log into PagerDuty and VictorOps on Mobile and Web
7. Log into Logz.io
   1. In VM, execute *curl -T /var/log/elasticsearch/gc.log.0.current* [*http://listener.logz.io:8021/file\_upload/qVsLZjOcBGEAkiKLjMTeqxvZrVDkKEPB/elasticsearch*](http://listener.logz.io:8021/file_upload/qVsLZjOcBGEAkiKLjMTeqxvZrVDkKEPB/elasticsearch)or sudo curl -T /var/log/metricbeat/metricbeat<http://listener.logz.io:8021/file_upload/qVsLZjOcBGEAkiKLjMTeqxvZrVDkKEPB/metricbeat>
   2. Show what the Logz.io interface looks like for filtering data
   3. This should trigger a test alert in both PagerDuty and VictorOps
      1. We can create a test alert that triggers when # events >= 0
      2. We can also create a test alert from metrics that show up
   4. Show how the interface looks in both PagerDuty and VictorOps. Note the differences when viewing the raw JSON in PagerDuty
      1. Also note how VictorOps groups similar incidents! PagerDuty doesn’t
8. Show the alerts that come from Email integration. Send the email
   1. PagerDuty doesn’t parse the Email! Only VictorOps does.
9. Show mobile app capabilities and what they display
10. Show annotations made by transmogrifier in VictorOps
    1. Explain that you can input the fields found in the incident payload into the transmogrifier so that the links change dynamically with each incident
    2. Works with graph pictures as long as they are just image files