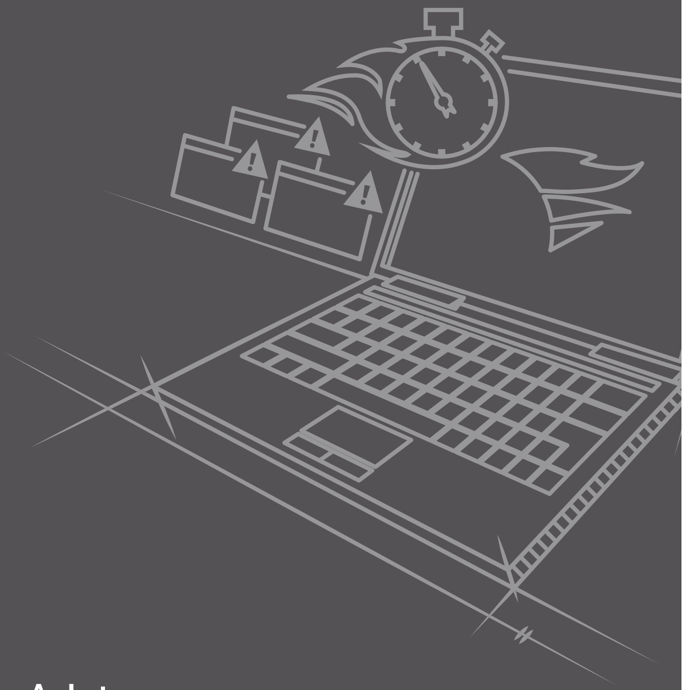
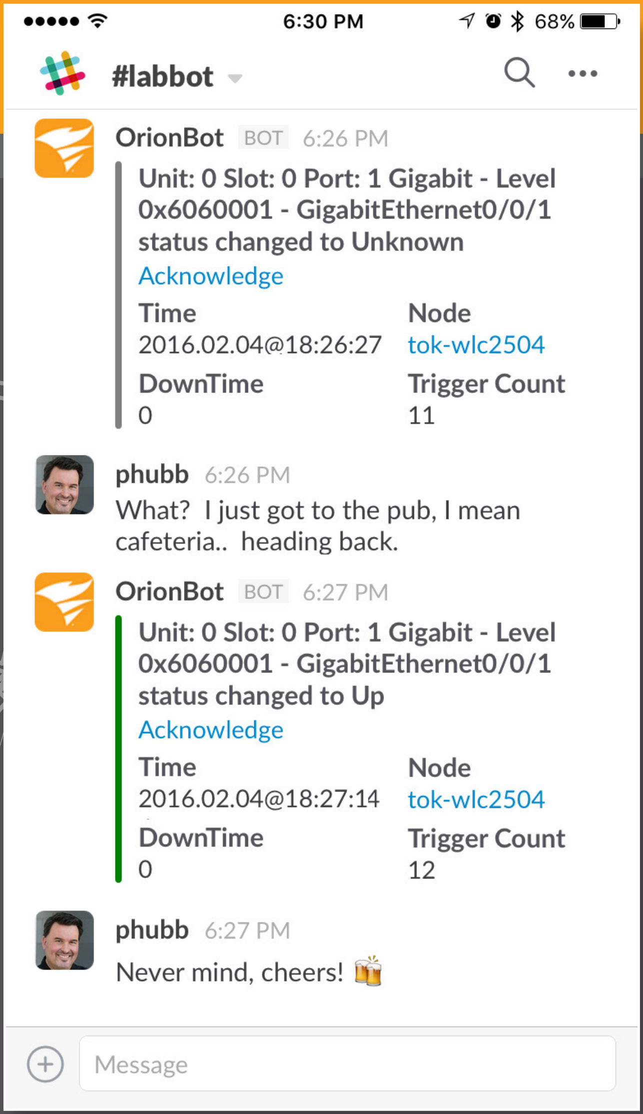
solarwinds

THE INCOMPLETE GUIDE TO INTEGRATING SOLARWINDS® ORION® INTO SLACK®

Get SolarWinds alerts into Slack in about 5 minutes (give or take)





6:30 PM

#labbot

OrionBot BOT 6:26 PM

Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Unknown

Acknowledge

Time	Node
2016.02.04@18:26:27	tok-wlc2504
DownTime	**Trigger Count**
0	11

phubb 6:26 PM

What? I just got to the pub, I mean cafeteria.. heading back.

OrionBot BOT 6:27 PM

Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Up

Acknowledge

Time	Node
2016.02.04@18:27:14	tok-wlc2504
DownTime	**Trigger Count**
0	12

phubb 6:27 PM

Never mind, cheers! 🍻

Message

by Leon Adato
& Patrick Hubbard

WELCOME

WHAT IS THIS GUIDE ABOUT?

This book will help you integrate two of the most amazing tools ever conceived of by the human mind: SolarWinds and Slack (we may be exaggerating just a bit, but you'll have to forgive our enthusiasm). Follow along with us to the end, and what you'll have is a system where alerts from SolarWinds will auto-magically appear in one (or more) channels in Slack, allowing teams to view in real-time, search through past events, and interact with those alerts to find out more information or even acknowledge them back in the SolarWinds system.

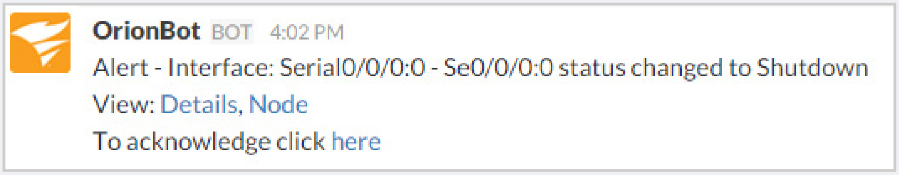

OrionBot BOT 4:02 PM
Alert - Interface: Serial0/0/0:0 - Se0/0/0:0 status changed to Shutdown
View: Details, Node
To acknowledge click here

TABLE OF CONTENTS

ALL ABOUT

ABOUT THE AUTHORS

Leon Adato

Leon Adato is a Head Geek™ and technical evangelist at SolarWinds, and is a Cisco® Certified Network Associate (CCNA®), MCSE and SolarWinds Certified Professional®. His experience spans financial, healthcare, food and beverage, and other industries.

Before he was a SolarWinds Head Geek, Adato was a SolarWinds user for over a decade. His expertise in IT began in 1989 and has led him through roles in classroom training, desktop support, server support, and software distribution.

Leon's essays can be found anywhere they will let him post, as well as on Twitter® as @LeonAdato, and on solarwinds.com.

Patrick Hubbard

Patrick Hubbard is a Head Geek and technical product marketing director at SolarWinds. With over 20 years of IT experience spanning network management, data center, storage networks, VoIP, virtualization, and more. Hubbard's career in technology began with in the technogenesis of skunkworks IT at American Airlines®. Since then, his career has included product management and strategy, technical evangelism, sales engineering and software development. Hubbard's focus is application and service delivery for startups and Fortune 500® companies alike that span high tech, transportation, financial services and telecom industries.

You can find Patrick's thoughts strewn across the internet like seeds on the Ethernet winds, on Twitter as @FerventGeek, or on solarwinds.com.

ABOUT SLACK

Slack is a web-based tool that facilitates team communication. More than a simple instant messenger client, Slack allows teams to share not only text messages, but also files, images, documents and more. Anything that is shared can be commented on, updated, indexed for searching, and synchronized to other web services.

Finally, Slack is extremely customizable. It supports hooks for external functions like stock market lookups, sales lead management, curation of cute cat videos, and even alerts from monitoring systems.

Which brings us to our next point:

ABOUT SOLARWINDS

SolarWinds is nothing more or less than the greatest monitoring, management, and automation solution in the world. How can we be so sure? 150,000 IT Pros have told us so—they've voted both with words and cold-hard cash. Started in 1998 by 18 employees in Tulsa, OK, with the idea that all the tools a network engineer needs should be in one place and accessed through a simple interface. The company has grown to thousands of employees in dozens of locations, providing dozens of amazing tools to IT professionals of all stripes and specializations

ABOUT YOU

If you were sitting at your desk thinking, ***"I love the alerts I'm getting in SolarWinds, but I wish I could get them in Slack, where my whole team could see them and respond collaboratively,"*** then you, my friend have some pretty strange thoughts and maybe you should get up from that desk every once in a while.

Also, this book is for you.

Also, you DEFINITELY can be our friend. Because we were thinking the same thing. Only, we decided to take a shot at it and see if we could make such a crazy idea happen.

What you should already know:

Before we dig into this, there are a few things that we're assuming you know about, or know how to do. We're not trying to be elitist or anything, it's just that if you aren't comfortable with the skills we list below, you're going to find a lot of the concepts – let alone tasks – to be frustrating.

That's not to say you're out of luck. We happen to know a community of folks who are more than willing to help guide you in your journey.

Head over to thwack.com and start reading forum posts, asking questions, and rubbing elbows with people from across the world who have been EXACTLY in your position (for some folks, it wasn't even that long ago!) and who would love to help shepherd you along.

So take heart, young Padawan! When the student is ready, the teacher will appear!

With all of that said, here's what you ought to be comfortable with:

- Generally moving around in Slack
- Generally moving around in SolarWinds
- Creating and modifying alerts in SolarWinds you don't have to be an expert, just generally familiar
- Working in the Windows® command prompt (i.e. the DOS-ish prompt) – Sorry if this one feels like a bit of a bummer, but this is one of those times when CLI (command line interface) is where the magic is going to happen.
- PowerShell™ – You don't have to be a PowerShell guru, but you ought to know what it is, how to get into the PowerShell command prompt (there's that CLI again!), and have a decent sense of the PowerShell programming syntax.

GETTING STARTED

WHAT YOU'LL NEED TO GET THIS DONE:

Now we just listed out skills or areas of knowledge, but there are a few specific THINGS (such as software, accounts, and access) you will need as well:

- A working SolarWinds installation – This can be any "standalone" module, NPM, SAM, NCM, IPAM, UDT etc. An account on that working SolarWinds installation that can create and edit alerts
- Access (i.e. RDP) to the SolarWinds primary poller
- An account on the SolarWinds primary poller that lets you install software
- PowerShell installed on the SolarWinds primary poller (if it's not Windows 2008 or newer, where PowerShell is automatically included you may need to install it)
- Access to slack.com via port 80 from the poller where alerts will originate
- A Slack account
- A Slack group that has read/write permission to at least one channel

THE NITTY GRITTY

OVERVIEW

After all the fanfare, we're finally THERE. We're going to get it done. Like any good journey, the first thing you'll want to do is review the stops along the way, so that you don't get lost.

At a very high level, here's what we're going to do:

1. Configure a "webhook" on a Slack channel
2. Install and test cURL® on the SolarWinds primary poller
3. Create a new alert in SolarWinds Orion
4. Add an Orion alert action that calls cURL
5. Add an Orion alert action that calls a PowerShell script which wraps cURL
6. Add an Orion alert action that calls a PowerShell script that does it the Right Way
7. Bask in the glory of your own genius
8. Lather, rinse, and repeat for greater glory

Please note that step 7 is absolutely required, 8 is highly recommended.

Also note that steps 4, 5, and 6 are actually variations of the same step (set up an alert action) with increasing levels of sophistication. If you are REALLY confident in your skills — you know, the kind of person who does math problems and crossword puzzles in pen rather than pencil — then feel free to jump straight to 6.

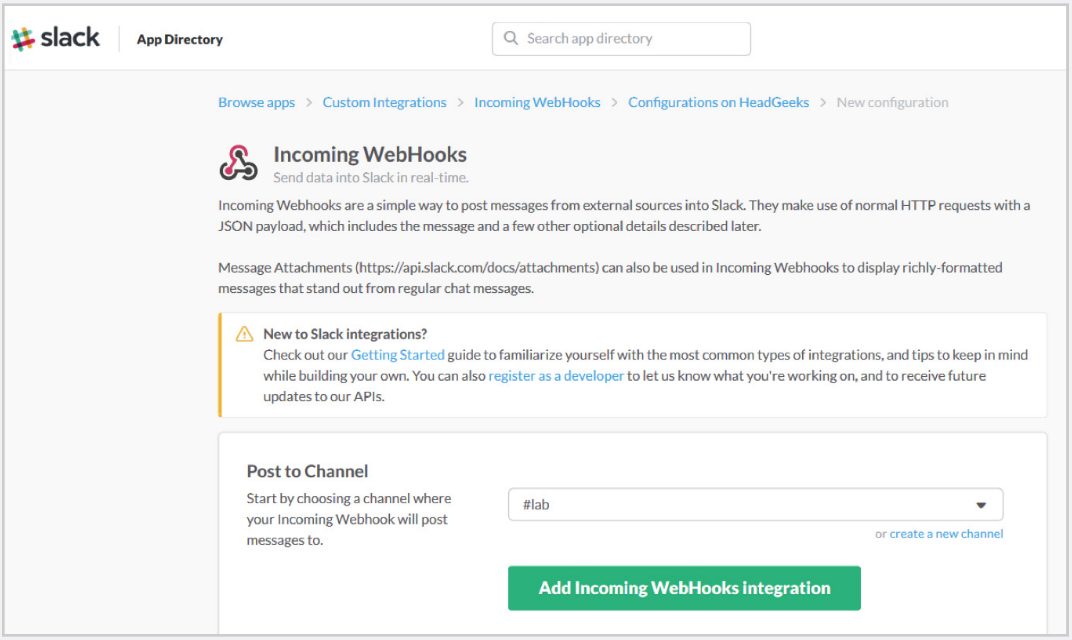
But we think that going through ALL the steps has value. Remember that it's the journey, not the destination that matters most.

STEP 1: INITIAL SETUP

Set up Slack

TRAP: *Remember, you need to have a Slack account that has admin (i.e. Read-Write) permission to the channel where you want to send messages.*

1. On an actual computer or VM – not a phone, iWhatever, or Android device- log in to Slack in your web browser. (go to https://slack.com and click ***"Sign In"***)
2. Next, go to the webhooks admin page https://api.slack.com/incoming-webhooks. You don't actually NEED to go to this page, you can just skip to the link in step 3. But this is an INSANELY useful page, so we wanted you to see this first. Our apologies for the subterfuge and occasional caps lock.
3. Click ***"incoming webhook integration"*** (https://my.slack.com/services/new/incoming-webhook)
4. From the drop-down, select the channel where you want SolarWinds alert messages to appear, and click the "***Add Incoming Webhooks Integration"*** button.


slack
App Directory
Search app directory
Browse apps > Custom Integrations > Incoming WebHooks > Configurations on HeadGeeks > New configuration
Incoming WebHooks
Send data into Slack in real-time.

Incoming Webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details described later.

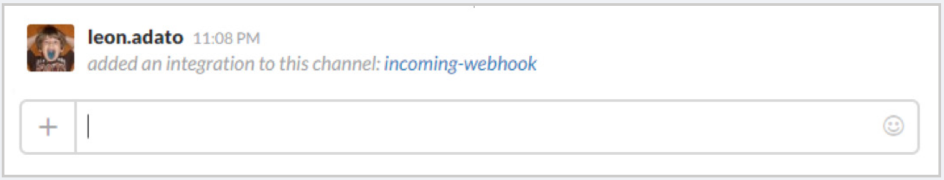
Message Attachments (https://api.slack.com/docs/attachments) can also be used in Incoming Webhooks to display richly-formatted messages that stand out from regular chat messages.

New to Slack integrations?
Check out our Getting Started guide to familiarize yourself with the most common types of integrations, and tips to keep in mind while building your own. You can also register as a developer to let us know what you're working on, and to receive future updates to our APIs.

Post to Channel
Start by choosing a channel where your Incoming Webhook will post messages to.

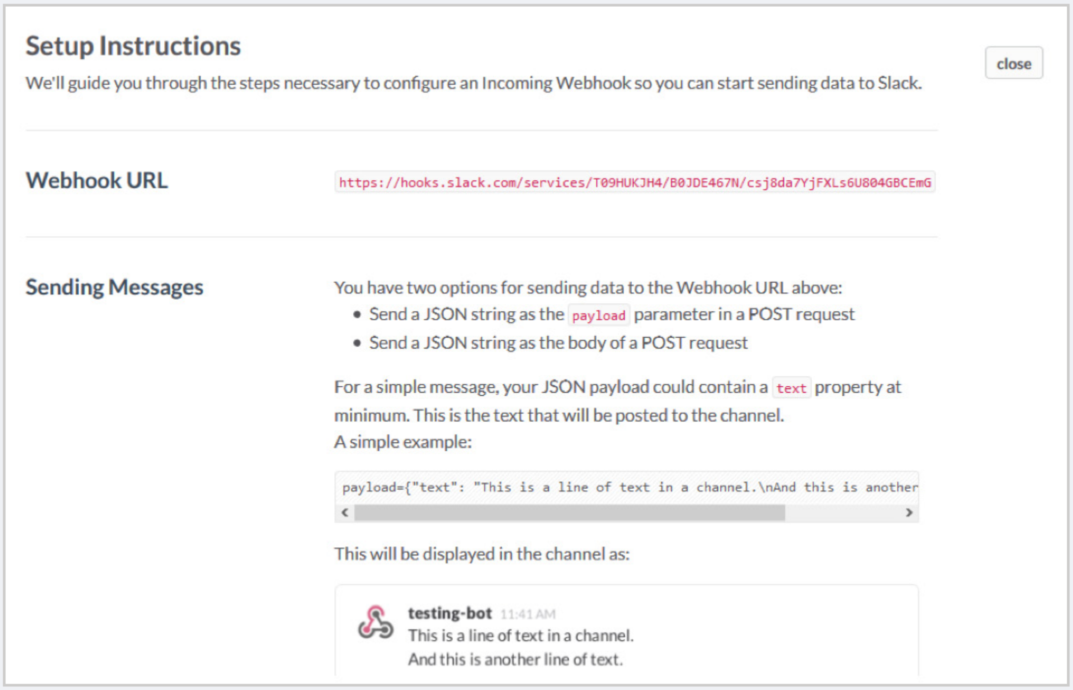
#lab

5. Slack will helpfully notify you either the web browser or the mobile app that a new webhook has been created. Notice a theme emerging here- instant notification in a stream you're already watching.



leon.adato 11:08 PM
added an integration to this channel: incoming-webhook

6. In the web page, you will see a screen that gives you a specific URL. This is effectively an endpoint you will send all your messages to, which will then inject them in the channel on Slack. Copy that URL and save it somewhere for later.



Setup Instructions

We'll guide you through the steps necessary to configure an Incoming Webhook so you can start sending data to Slack.

Webhook URL

```
https://hooks.slack.com/services/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
```

Sending Messages

You have two options for sending data to the Webhook URL above:

- Send a JSON string as the `payload` parameter in a POST request
- Send a JSON string as the body of a POST request

For a simple message, your JSON payload could contain a `text` property at minimum. This is the text that will be posted to the channel.
A simple example:

```
payload={"text": "This is a line of text in a channel.\nAnd this is another
```

This will be displayed in the channel as:

testing-bot 11:41 AM
This is a line of text in a channel.
And this is another line of text.

7. Examine the options on this page to customize your new webhook. You can give it a different name, change the icon that displays when a new message comes in, and lots more.

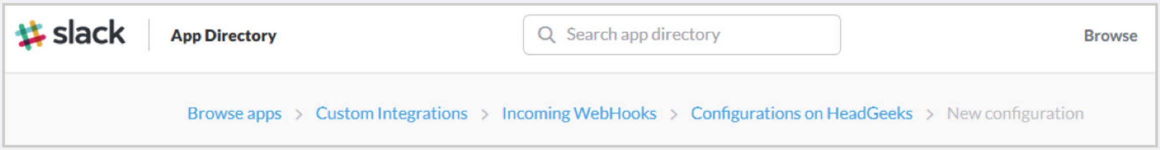
8. Once you've fiddled with those options a bit, click ***"Save Settings"*** at the bottom of the page.

Troubleshooting

Most of the troubleshooting at this point will be if you pick the wrong channel, or lose the URL, or something minor like that.

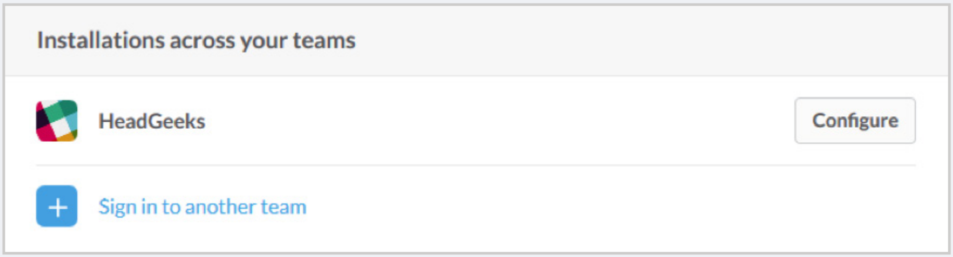
To fix any of these issues (or more), go back to https://my.slack.com/services/new/incoming-webhook.

Now look at the top of the page, where the ***"breadcrumbs"*** are located, and click the ***"Incoming Webhooks":***



Browse apps > Custom Integrations > Incoming WebHooks > Configurations on HeadGeeks > New configuration

You'll see a page with any of the Slack teams you are logged into. Click ***"Configure"*** next to the one we're setting up today.

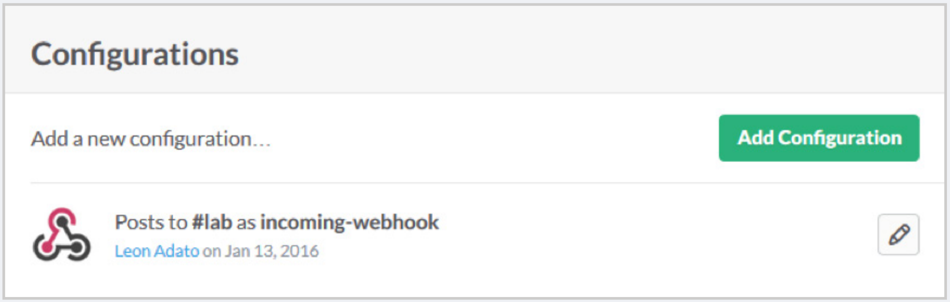


Installations across your teams

HeadGeeks — Configure

Sign in to another team

Now you'll see a list of all the webhooks currently in place for your Slack team. Click the pencil to edit the one which is broken.



Configurations

Add a new configuration... — Add Configuration

Posts to #lab as incoming-webhook
Leon Adato on Jan 13, 2016

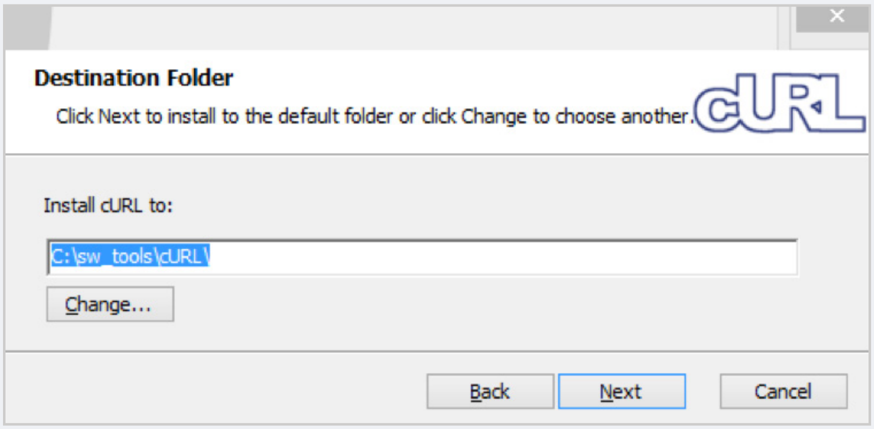
You are now BACK at the page where the webhook settings are located. You can Re-generate a new service URL, copy the existing one, change something that you don't like, delete the webhook URL in case you posed it to Facebook® or just enjoy the view. Ok, we know it sounds weird and slightly creepy, but we try not to judge.

TRICK: *Remember that this webhook is Open to the Public, so you may come back and regenerate the URL more often than you may think right now. If you intend for this service to remain private, and you don't want people sending unwanted messages to your Slack channel, you will probably go through a few URLs. Fortunately, there's a list of all URLs you've generated and clicking the edit pencil icon lets you remove any you regret. Regret!? There's no regret in DevOps!*

Set up cURL

The next thing you need to do is get cURL for Windows installed on the SolarWinds primary poller:

1. RDP to the SolarWinds primary poller
2. Open a browser and go to http://www.confusedbycode.com/curl/
3. Download the correct version of cURL. You need the one that runs with Administrator privileges. While we would like to insist that you install the 64-bit version, we leave it to your discretion. However, SolarWinds itself requires a 64bit system, so why you'd go for 32bit here is beyond us. NOTE: *If you need a copy of Microsoft C++ library, you will be prompted to download and install it here. Just go with the flow.*
4. During the install, select advanced and CHOOSE where cURL will be installed. Pick or create a folder that is a single word because lots of systems don't like spaces in path names. Make a note of this path, even go as far as to copy it into your notes file for ready reference later, for example in step 5.


Destination Folder

Click Next to install to the default folder or click Change to choose another.

Install cURL to:

C:\sw_tools\cURL\

5. Test to make sure cURL is running correctly by going to the DOS prompt, and yes, it IS called a DOS prompt, not ***"the command prompt"*** you baby-faced youngster! Now hush while we take a sip of Ensure®. Typing the command ***"<your path>\curl.exe --help"***. For example, if you used our suggestion and put cURL in ***C:\sw_tools\cURL\*** then you would type:

   ```
   C:\sw _ tools\curl\bin\curl –help
   ```

6. Finally, test to make sure cURL running on your system can actually connect to Slack. At the DOS prompt, type the following line

   ```
   C:\sw _ tools\curl\bin\curl.exe -X POST --data-urlencode
   "payload={\"text\":\"This is a line of text.\nAnd this is
   another one.\"}" https://YOUR _ SLACK _ WEBHOOK _ GOES _ HERE!
   ```

   Obviously you need to replace "***https://YOUR_SLACK_WEBHOOK_GOES_HERE!!"*** with your actual webhook. But if all is right with the universe, you should see a simple message pop up in your Slack channel.

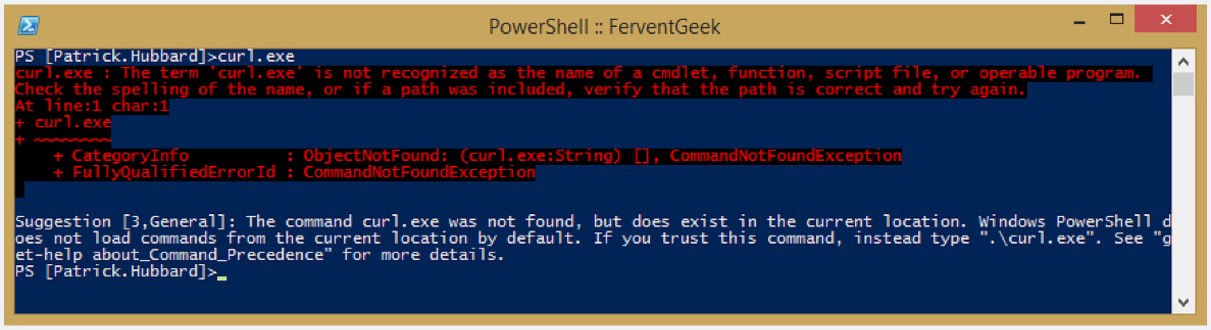
Troubleshooting

There shouldn't be any troubleshooting during this step. We're not saying nothing could possibly go wrong, just that anything that happens is really abnormal, probably specific to your environment, and unfortunately you are pretty much on your own to fix it. But, a great first step is to hit something small with cURL first to ensure it's not the bits, path or the way you're calling it.

For that you can certainly use the DOS command prompt, or because we're going to be doing all this the Right Way thank-you-very-much by the end of this tutorial, use a real admin's tool, PowerShell.

Start a PowerShell prompt, cd to your cURL install folder's ***\bin*** directory and at the command prompt type ***curl google.com*** [Enter]

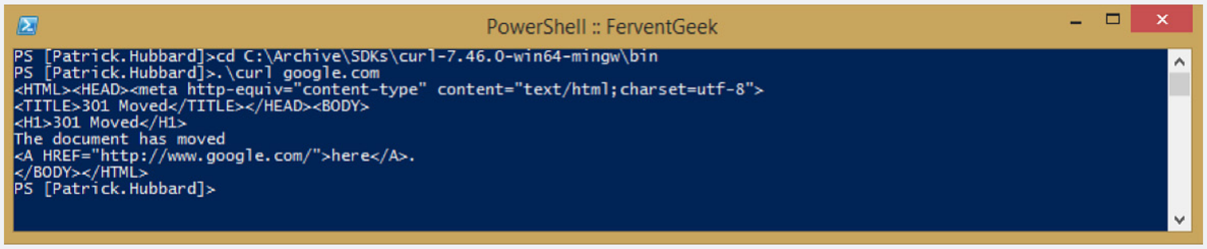
Ack! What the heck is that?



PowerShell :: FerventGeek

```
PS [Patrick.Hubbard]>curl.exe
curl.exe : The term 'curl.exe' is not recognized as the name of a cmdlet, function, script file, or operable program.
Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ curl.exe
+ ~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (curl.exe:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException


Suggestion [3,General]: The command curl.exe was not found, but does exist in the current location. Windows PowerShell d
oes not load commands from the current location by default. If you trust this command, instead type ".\curl.exe". See "g
et-help about_Command_Precedence" for more details.
PS [Patrick.Hubbard]>_
```

Ahh PowerShell, keeping you out of trouble and ensuring you don't have ambiguity in your execute path. And look at the last line, it even tells you it thinks it knows where it is and suggests adding **.** before ***curl.exe***. So friendly, that PowerShell. Let's try that again. Type ***.\curl google.com*** [Enter]



PowerShell :: FerventGeek

```
PS [Patrick.Hubbard]>cd C:\Archive\SDKs\curl-7.46.0-win64-mingw\bin
PS [Patrick.Hubbard]>.\curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
PS [Patrick.Hubbard]>
```

Ahh much better. If this works for you then cURL is working, and the issue is with your syntax. See the Everything You Always Wanted To Know About the CURL Command troubleshooting section for more details.

STEP 2: SET UP YOUR FIRST ALERT

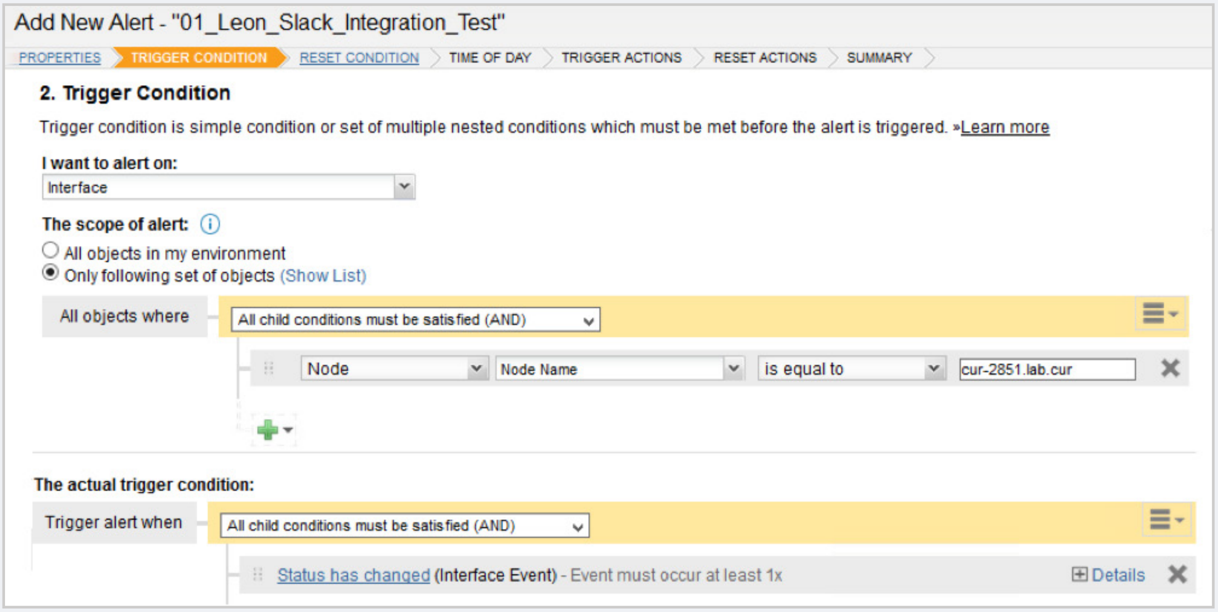
In this step we're going to set up a fairly simple alert. Because we want to test this alert a lot, we're looking for something that we can trigger and then fix, easily and frequently, with extreme prejudice. A monitor which checks your primary internet router is exactly the worst choice you could make.

In this book, we're going to go with a simple ***"interface status has changed"*** alert because we have a lab and some gear nobody is using and no one will freak out if we shutdown interfaces. You could also do something like monitor a single switch port which you can enable/disable/unplug, or a whole device which you can shut off/restart, or any other simple up/down type trigger.

Remember, the point here is not the sophistication of the alert, it's all about getting the alert action working.

To set up our alert, we are going to:

1. Go to the ***Orion web console***
2. Go to ***Settings***
3. Go to ***"Manage Alerts"***
4. Start a new alert
5. Set up the name, description, etc. on the first screen
6. In the second screen, we're going to set a specific node (***cur-2851.lab.cur***) and a trigger looking for any time an interface status changes


Add New Alert - "01_Leon_Slack_Integration_Test"

PROPERTIES > TRIGGER CONDITION > RESET CONDITION > TIME OF DAY > TRIGGER ACTIONS > RESET ACTIONS > SUMMARY

2. Trigger Condition

Trigger condition is simple condition or set of multiple nested conditions which must be met before the alert is triggered. »Learn more

I want to alert on: Interface

The scope of alert:

( ) All objects in my environment
(●) Only following set of objects (Show List)

All objects where: All child conditions must be satisfied (AND)

Node | Node Name | is equal to | cur-2851.lab.cur

The actual trigger condition:

Trigger alert when: All child conditions must be satisfied (AND)

Status has changed (Interface Event) - Event must occur at least 1x — Details

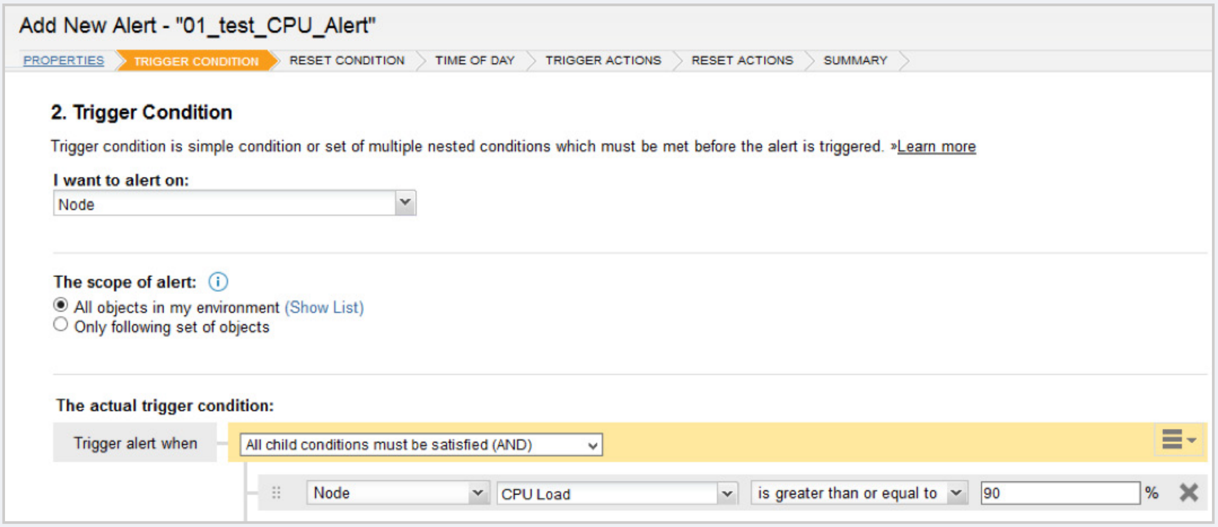
For NOW, you can click ***"next"*** past the rest of the screens and save the alert. There is obviously more to do, but we'll get to that shortly.

A brief word about the "test" button

The test button included in the SolarWinds Orion web-based alert manager is an amazingly convenient option that calls to many monitoring engineers with the seductive power of the sirens in "The Odyssey." And in many cases, the test button works. However, it has a few drawbacks and we need to make them clear so you don't troubleshoot problems that aren't a problem.

As the name implies, the test button will generate a test alert for the trigger you've set up. But the trick here is that there isn't really a problem to be alerted on, so some parts of this test alert inevitably go missing or are incomplete and for once as and admin, that's perfectly ok.

For example, there's no "Acknowledge" URL. Why? Because there was no actual alert triggered, so there's nothing to acknowledge! If your alert lists "components that are down," that list will be blank because as a test alert, none of the components are actually down. And so on. The goal here is to get something in Slack and test the integration. Later on we'll verify the details. Assuming that skeeves you out, how do you get around this? Step 1, don't use the test button. Step 2, set up your alert so that you can actually trigger it. Step 3, profit. For example, let's say you want to test ***"high CPU"*** alert that triggers when CPU utilization is over 90%.



Add New Alert - "01_test_CPU_Alert"

PROPERTIES > TRIGGER CONDITION > RESET CONDITION > TIME OF DAY > TRIGGER ACTIONS > RESET ACTIONS > SUMMARY

2. Trigger Condition

Trigger condition is simple condition or set of multiple nested conditions which must be met before the alert is triggered. »Learn more

I want to alert on: Node

The scope of alert:
[✓ All objects in my environment] (Show List)
Only following set of objects

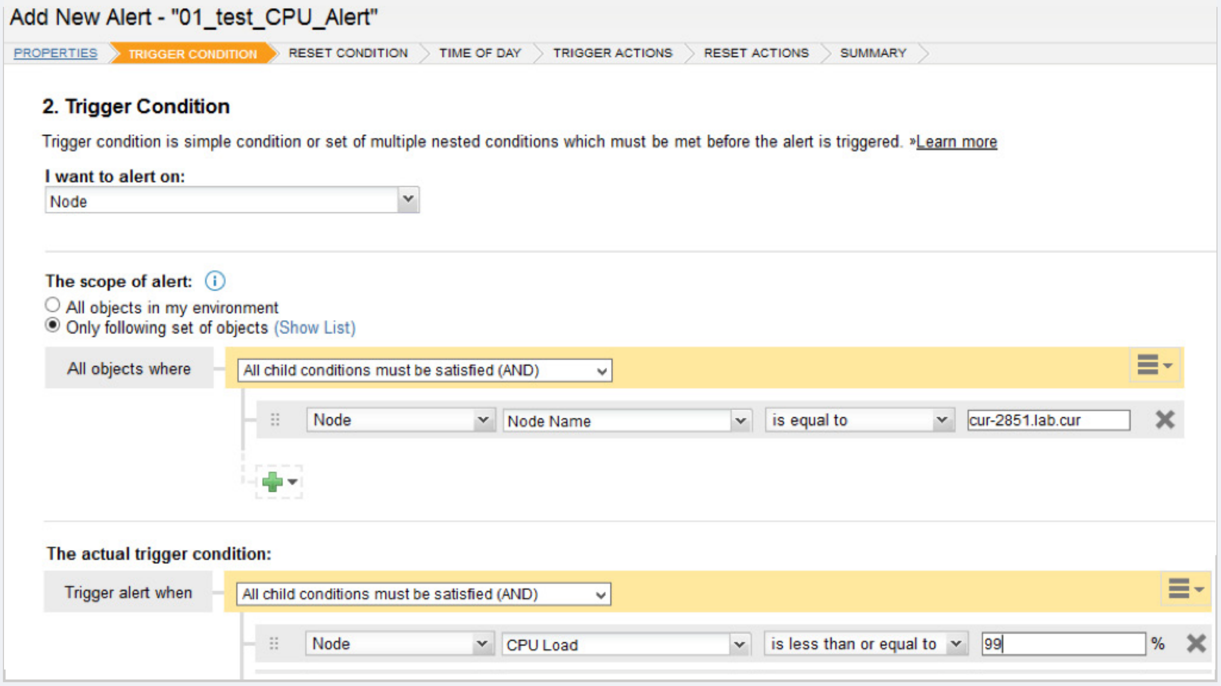
The actual trigger condition:

Trigger alert when: All child conditions must be satisfied (AND)

Node | CPU Load | is greater than or equal to | 90 %

First, add criteria that limits your alert to a single machine. That way you don't end up triggering 9,000 alerts by accident (don't laugh, we've done it. Twice.).

Next, change the ***"greater than or equal to 90%"*** to ***"less than or equal to 99%."***



Add New Alert - "01_test_CPU_Alert"

PROPERTIES > TRIGGER CONDITION > RESET CONDITION > TIME OF DAY > TRIGGER ACTIONS > RESET ACTIONS > SUMMARY

2. Trigger Condition

Trigger condition is simple condition or set of multiple nested conditions which must be met before the alert is triggered. »Learn more

I want to alert on: Node

The scope of alert:
All objects in my environment
[✓ Only following set of objects] (Show List)

All objects where: All child conditions must be satisfied (AND)

Node | Node Name | is equal to | cur-2851.lab.cur

The actual trigger condition:

Trigger alert when: All child conditions must be satisfied (AND)

Node | CPU Load | is less than or equal to | 99 %

Unless the machine you've specifically selected is already having a rough day, this alert should trigger every single time.

NOW all you have to do is hit the ***"reset"*** button, and it will trigger again and again, until you get your alert actions just the way you want them.

Troubleshooting

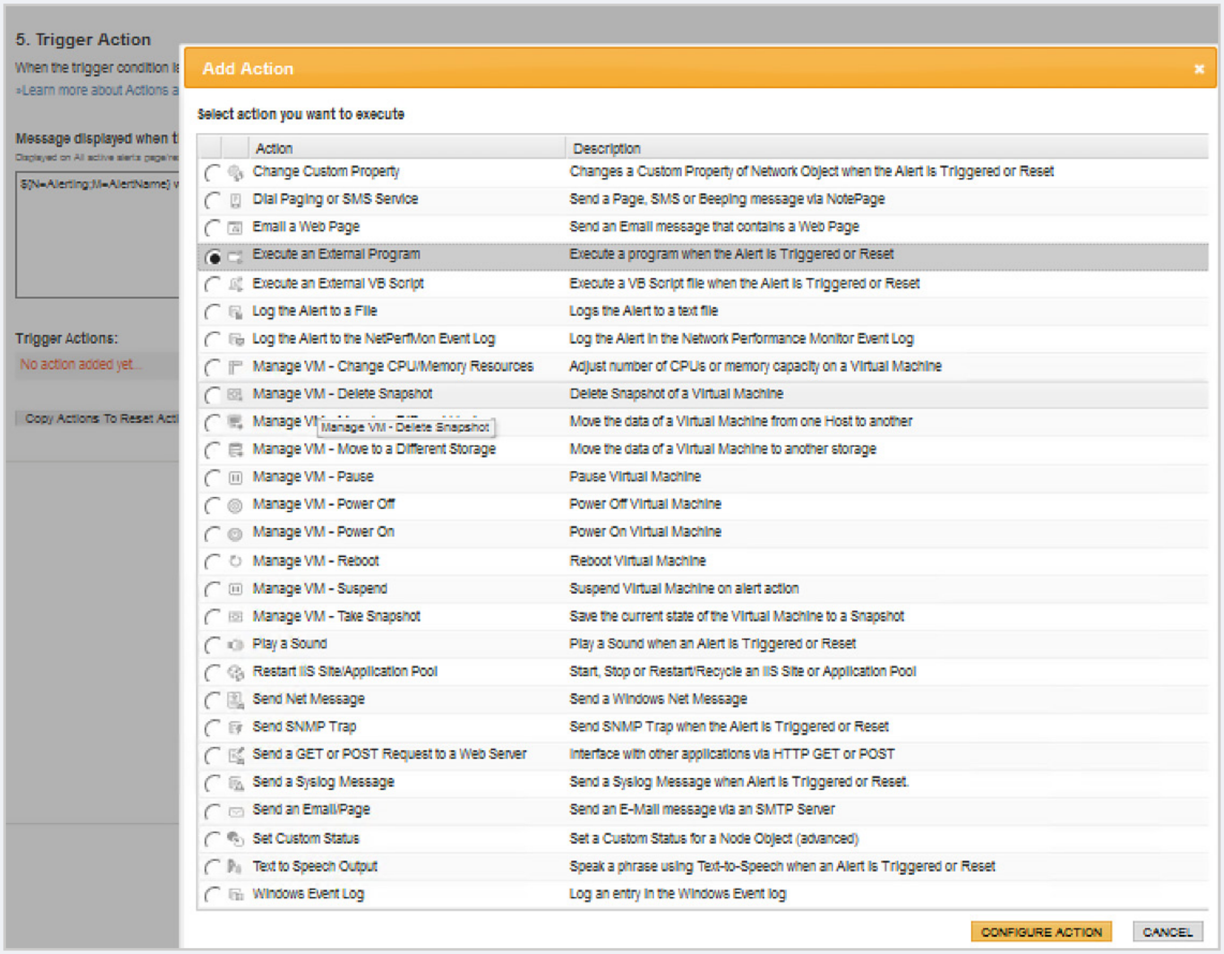
While there are a number of things that could potentially go wrong at this phase, none of them are relevant to the integration of SolarWinds and Slack. For that reason, we're going to leave you to your own devices (no pun intended). If you are really stuck, we strongly recommend going over to thwack.com and posting a question. Trust us, ev-eryone there is very willing to help out.

STEP 3A: ADD AN ACTION USING cURL

And now we finally get down to it – adding the special sauce that will send your alert messages sailing gracefully into your Slack session.

In this first example, we're going to use cURL (remember how we installed it back in step 1?) which is specially designed to programmatically interact with web pages and then report the results back to the calling program.

In plain English, this means cURL does the same web browser clicking you would do, and then sends the results back to you as the output. Edit the alert you started before, and go to the ***Alert Actions*** tab. Then click ***"Add Action"*** and select ***"Execute an External Program"***.


5. Trigger Action
When the trigger condition is
»Learn more about Actions a
Message displayed when t
Displayed on All active alerts page/re
${N=Alerting;M=AlertName} v
Trigger Actions:
No action added yet...
Copy Actions To Reset Acti
Add Action
×

Select action you want to execute

	Action	Description
○	Change Custom Property	Changes a Custom Property of Network Object when the Alert is Triggered or Reset
○	Dial Paging or SMS Service	Send a Page, SMS or Beeping message via NotePage
○	Email a Web Page	Send an Email message that contains a Web Page
●	Execute an External Program	Execute a program when the Alert is Triggered or Reset
○	Execute an External VB Script	Execute a VB Script file when the Alert is Triggered or Reset
○	Log the Alert to a File	Logs the Alert to a text file
○	Log the Alert to the NetPerfMon Event Log	Log the Alert in the Network Performance Monitor Event Log
○	Manage VM - Change CPU/Memory Resources	Adjust number of CPUs or memory capacity on a Virtual Machine
○	Manage VM - Delete Snapshot	Delete Snapshot of a Virtual Machine
○	Manage V[illegible] Manage VM - Delete Snapshot	Move the data of a Virtual Machine from one Host to another
○	Manage VM - Move to a Different Storage	Move the data of a Virtual Machine to another storage
○	Manage VM - Pause	Pause Virtual Machine
○	Manage VM - Power Off	Power Off Virtual Machine
○	Manage VM - Power On	Power On Virtual Machine
○	Manage VM - Reboot	Reboot Virtual Machine
○	Manage VM - Suspend	Suspend Virtual Machine on alert action
○	Manage VM - Take Snapshot	Save the current state of the Virtual Machine to a Snapshot
○	Play a Sound	Play a Sound when an Alert is Triggered or Reset
○	Restart IIS Site/Application Pool	Start, Stop or Restart/Recycle an IIS Site or Application Pool
○	Send Net Message	Send a Windows Net Message
○	Send SNMP Trap	Send SNMP Trap when the Alert is Triggered or Reset
○	Send a GET or POST Request to a Web Server	Interface with other applications via HTTP GET or POST
○	Send a Syslog Message	Send a Syslog Message when Alert is Triggered or Reset.
○	Send an Email/Page	Send an E-Mail message via an SMTP Server
○	Set Custom Status	Set a Custom Status for a Node Object (advanced)
○	Text to Speech Output	Speak a phrase using Text-to-Speech when an Alert is Triggered or Reset
○	Windows Event Log	Log an entry in the Windows Event log

Everything You Always Wanted to Know About the cURL Command

Once you are there, you are going give the Alert Action a name, and add a network path value that looks something like this:

```
PATH _ TO _ CURL\bin\curl.exe -X POST --data-urlencode
"payload={\"channel\": \"#YOURCHANNEL\", \"username\": \"OrionBot\",
\"text\": \"Alert - Interface: ${N=SwisEntity;M=Caption}
status changed to ${N=SwisEntity;M=Status}\
nView: <${N=Alerting;M=AlertDetailsUrl}|Details>,
<${N=SwisEntity;M=Node.DetailsUrl}|Node>\nTo acknowledge click
<${N=Alerting;M=AcknowledgeUrl}|here>\"}",
https://YOUR _ SLACK _ WEBHOOK
```

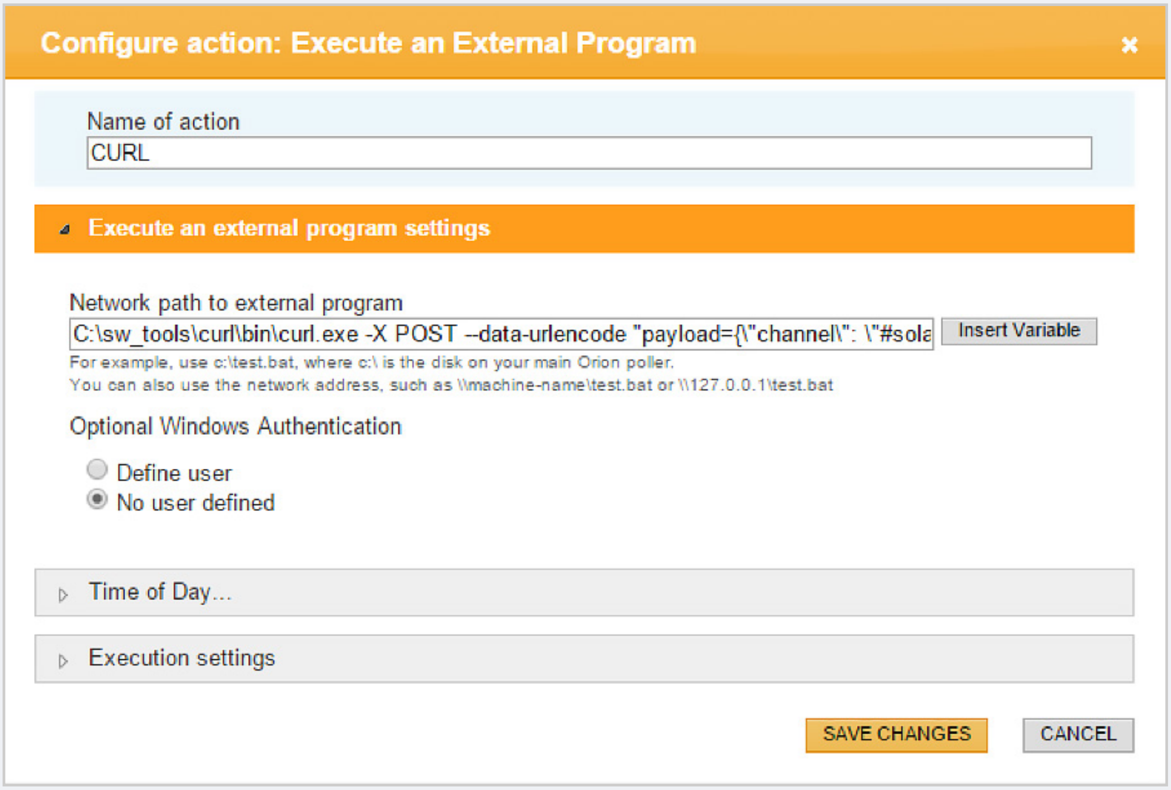
OK, that's a whole lotta stuff, so let's break this down:

- **PATH_TO_CURL\bin\curl.exe** – This is quite literally, the path to ***curl.exe*** on your system. So if you put it in ***C:\sw_tools*** like we did, this line would read: ***C:\sw_tools\curl\bin\curl.exe***
- **-X POST** – eXecute a "POST" action (meaning send some stuff out, rather than a "GET" action which pulls web information back.
- **--data-urlencode** – send the information in the same way it would work if you had clicked a web option on a page.
- **"payload={** – This tells cURL that the next part (everything between the quotes) is literally what is going to get sent.
- **\"channel\": \"#YOURCHANNEL\",** – this is a variable that Slack wants: the channel you are sending this into. Plus the actual channel name. Don't use ***"#YOURCHANNEL"*** unless you have some very weird Slack channel names.
- **\"username\": \"OrionBot\",** – When the alert information pops up, this is the user it will appear to be from. It doesn't have to be an actual Slack user, so you can call this anything you like.
- **\"text\": \"Alert – Interface:** – This marks the beginning of the text of the message. Everything after ***\"text\"***: is what will show up as the alert. So this alert starts by saying ***"Alert – Interface…".***

- **${N=SwisEntity;M=Caption}** – This has nothing to do with Slack and everything to do with SolarWinds. It's a SolarWinds SWQL (SolarWinds Query Language) variable. In this case, it pulls the interface name out of the SolarWinds database, and sends it.
- **status changed to** – more literal text
- **${N=SwisEntity;M=Status}** – another SolarWinds variable, giving the status of the interface
- **\n** – This is a "hard return". It means that the message in Slack will start a new line.
- **View:** – more literal text
- **<${N=Alerting;M=AlertDetailsUrl}|Details>** – This passes a SolarWinds variable for the URL of the alert details link, but uses the word "Details" as the clickable part. So in Slack you will simply see the word "Details" but it will be click-able.
- **<${N=SwisEntity;M=Node.DetailsUrl}|Node>** – Similar to the Acknowledge link, this one says "Node" but is clickable to take you to the Node Details page.
- **\nTo acknowledge click** – More literal text
- **<${N=Alerting;M=AcknowledgeUrl}|here>\",** – A link to acknowledge this alert in SolarWinds
- **https://YOUR_SLACK_WEBHOOK** – This is your Slack webhook URL, which tells cURL where to send all of this lovely SolarWinds-y goodness.

Epic Copy – Paste

Last you're going to squeeze a lot of cURL call into a little text box, and appreciate editing big chunks of text in an editor not the web. Of course it lets you back them up and test more easily, and everyone backs up their work correct?. When you've got your cURL call looking the way you want, paste it into the ***"Network path to external program"*** text box.



Configure action: Execute an External Program

Name of action: CURL

Execute an external program settings

Network path to external program

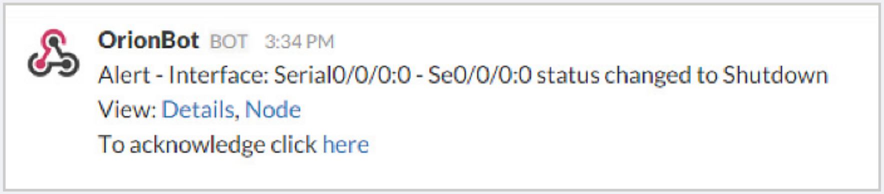
```
C:\sw_tools\curl\bin\curl.exe -X POST --data-urlencode "payload={\"channel\": \"#sola
```

For example, use c:\test.bat, where c:\ is the disk on your main Orion poller.
You can also use the network address, such as \\machine-name\test.bat or \\127.0.0.1\test.bat

Optional Windows Authentication

- ○ Define user
- ● No user defined

Save your changes, click the box to the left of your new execute action in the action list, click the test button and Hello World, you should see this in the Slack channel of your choice:



OrionBot BOT 3:34 PM
Alert - Interface: Serial0/0/0:0 - Se0/0/0:0 status changed to Shutdown
View: Details, Node
To acknowledge click here

Whew! We know that seems like a lot, but once you run through this a couple of times it will become very familiar.

A Picture is Worth...

But we want to kick it up just a bit, and add some style. Let’s say that you want your OrionBot Slack user (who doesn’t really exist) to have a really cool icon. Something like this, and for the record those are particles not a flame.

And let's say you have that icon accessible on the internet here:

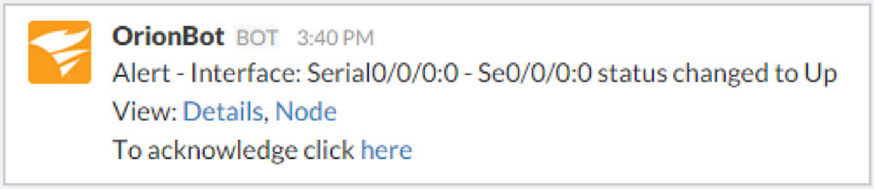
http://cool.example.com/slack/orionbot/img/particles.png

You COULD add just one extra variable to make each message show up with that user icon, like this:

```
\"icon _ url\": \" http://cool.example.com/slack/orionbot/img/
particles.png"
```

Put it all together, and your command would look something like this. For the sake of argument, we're going to say the path to cURL is ***C:\sw_tools***, and the channel is ***"#labbot"***.

```
C:\sw _ tools\curl\bin\curl.exe -X POST --data-urlencode
"payload={\"channel\": \"#labbot \", \"username\": \"OrionBot\",
\"text\": \"Alert - Interface: ${N=SwisEntity;M=Caption}
status changed to ${N=SwisEntity;M=Status}\
nView: <${N=Alerting;M=AlertDetailsUrl}|Details>,
<${N=SwisEntity;M=Node.DetailsUrl}|Node>\nTo acknowledge
click <${N=Alerting;M=AcknowledgeUrl}|here>\", \"icon _ url\":
\http://cool.example.com/slack/orionbot/img/particles.png\"}"
https://YOUR _ SLACK _ WEBHOOK
```



OrionBot BOT 3:40 PM
Alert - Interface: Serial0/0/0:0 - Se0/0/0:0 status changed to Up
View: Details, Node
To acknowledge click here

Test Early, Test Often

After you've gotten all of this set up (we recommend putting it all together in a simple text editor like notepad, or better still in a programmer's editor like Notepad++, Sublime, BlueFish®, or whatever you prefer). The next thing you want to do, like any good developer, is TEST.

What we mean is, go to the DOS prompt, and execute EXACTLY your command. Doing this from the DOS prompt will show you key error messages that you wouldn't see if you went straight to copy-and-pasting into the SolarWinds alert trigger box.

You didn't just jump ahead to that step, did you? Nah, we didn't think you did. Honest.

Now, when you test this from the DOS prompt, you aren't going to get the actual variables – you know, things like the ACTUAL node name. In Slack, you will see ***"${nodename}"*** instead. But that's ok because now you know there are no critical errors.

NOW you should go ahead and paste your code into the SolarWinds alert action, and save everything.

Depending on how you set things up, either use the test button or (if you read carefully in the "A Brief Word About the Test Button" section) trigger your alert, and then use the reset button to keep re-triggering it until everything is working the way you want.

Troubleshooting

The main problems you may encounter here and going forward will be related to two issues:

1. Basic syntax and general fat-finger-foo
2. Bad values passed to Slack

For syntax issues the DOS or PowerShell prompts are your friends if you're sorting out a double-wrapped, twice-escaped, and once Url encoded CURL call. The challenge is that even if you turn on verbose logging in Orion, the execute CURL command will eat a lot of the error detail. The only way to catch it is when you run the command in a command window. (And be sure to remember the handy-dandy Log Adjuster in

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\SolarWinds Orion\Documentation and Support\Log Adjuster.exe).

If you're getting errors in the Orion Alert test window, or success but no messages appearing in Slack, first take a look at Orion's actions execution alert log:

C:\ProgramData\Solarwinds\Logs\Orion\ActionsExecutionAlert.log.

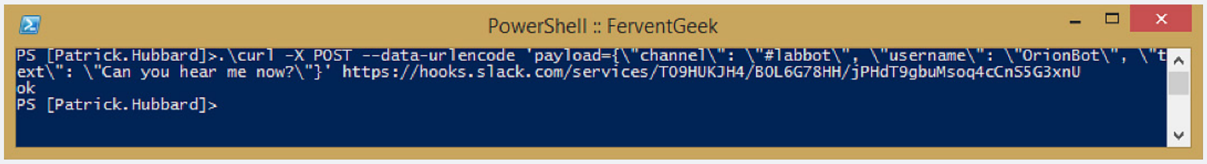
If it's something basic like a firewall block, Slack webhook root reachability or other basic transport issues it will be captured there. And if you're getting "Host not found" or "Host unreachable", then Hooray! We're network admins and that's super easy to troubleshoot. Book it, done.

But usually that's not the problem and what you'll be limited to is something like this:

```
2016-02-03 10:17:40,560 [17] ERROR AlertingLogger - Action [Action:
ID: 122, ActionType: SendHttpRequest, Title: Slack Webhook POST
Action, Description: Post to https://hooks.slack.com/services/
T09HUKJH4/B0L6G78HH/jPHdT9gbuMsoq4cCnS5G3xnU, Enabled: True,
Order: 2 , Context: SolarWinds.Orion.Core.Models.Actions.Contexts.
AlertingActionContext, EnviromentType: Alerting, ExecutionMode:
Trigger, EntityType: Orion.NPM.Interfaces, EntityUri: swis://dev-
aus-lada-01./Orion/Orion.Nodes/NodeID=101/Interfaces/InterfaceID=75,
AlertContext: AlertName: 00 _ Slack update on interface status
change, CreatedBy: , AlertActiveId: , AlertObjectId: ] execution
has failed.

System.Exception: Failed to execute HTTP request ---> System.Net.
WebException: The remote server returned an error: (500) Internal
Server Error.

at System.Net.HttpWebRequest.GetResponse()

at SolarWinds.Orion.Core.Actions.Impl.SendHttpRequest.
SendHttpRequestExecutor.ExecuteInternal()

 --- End of inner exception stack trace ---

at SolarWinds.Orion.Core.Actions.Impl.SendHttpRequest.
SendHttpRequestExecutor.ExecuteInternal()

at SolarWinds.Orion.Core.Actions.ActionExecutorBase.
Execute(IServiceProvider serviceProvider, ActionDefinition
definition, ActionContextBase context, CancellationToken
cancellationToken)
```

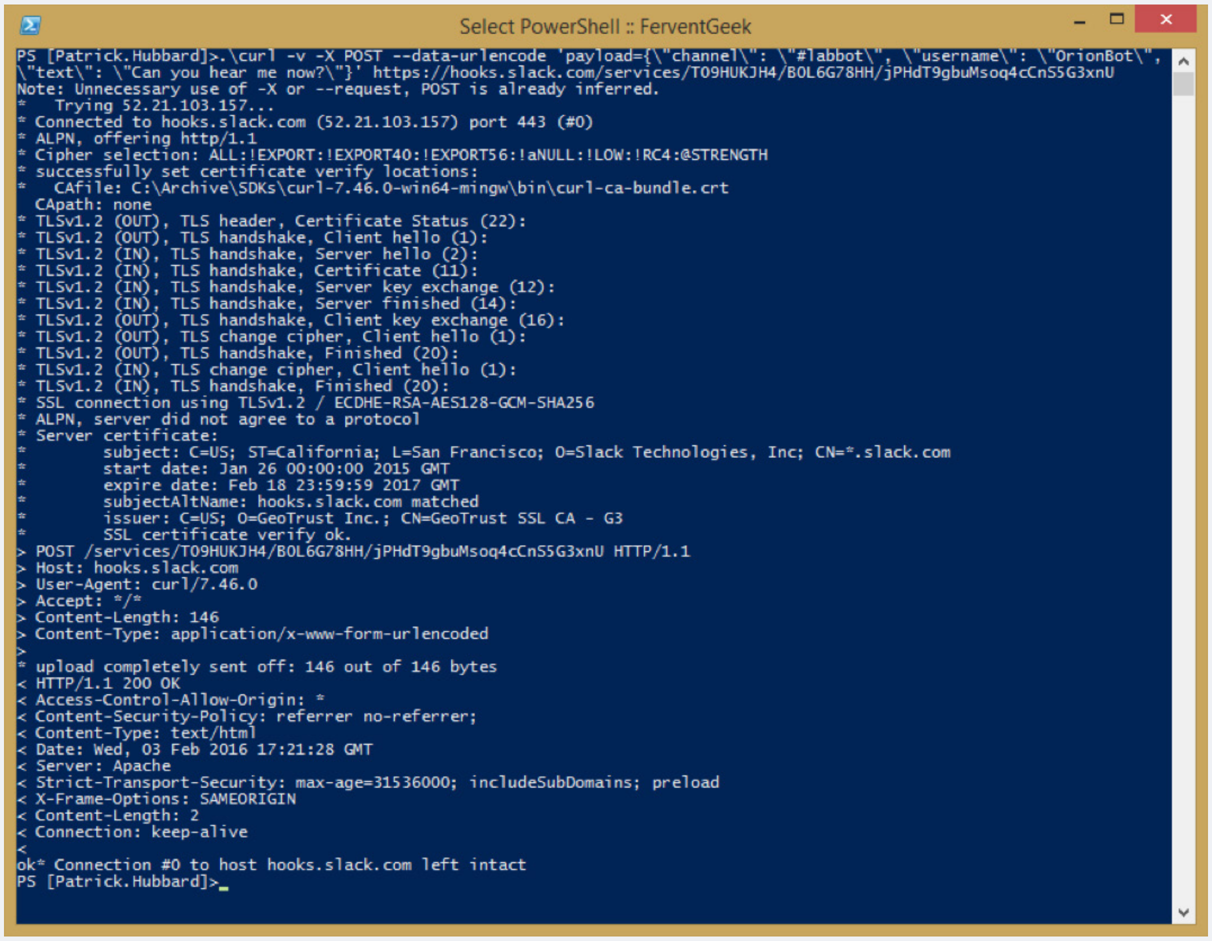
“Stack traces?!” you say, “I hate reading stack traces!” <Sighs, gets glasses, reads stack trace>. Ah, “remote server returned an error: (500) Internal Server Error.” Easy, Slack is down, right? Nope. Slack is just saying it barfed on what you sent over to it.

So, to get the real nitty gritty from Slack on why it balked, the first place to start debugging is always the command line, not in the Alert Action box, and of course we recommend PowerShell. And because you’re debugging with PowerShell, you get to take advantage of another handy feature – single-quote strings. That crazy double-escaped string you’re passing to cURL? Just wrap it in single quotes and it works great. If Slack accepts your POST you’ll see this:



```
PS [Patrick.Hubbard]>.\curl -X POST --data-urlencode 'payload={\"channel\": \"#labbot\", \"username\": \"OrionBot\", \"t
ext\": \"Can you hear me now?\"}' https://hooks.slack.com/services/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
ok
PS [Patrick.Hubbard]>
```

Slack, would it hurt ya to just send back a little meta besides HTTP 200 OK? Slack webhooks are terse, no joke. If you really want to see what cURL is up to, add the verbose –v parameter and you’ll get this:

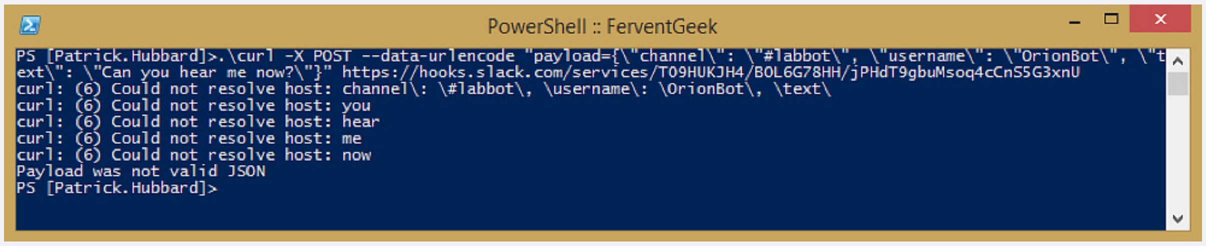


```
PS [Patrick.Hubbard]>.\curl -v -X POST --data-urlencode 'payload={\"channel\": \"#labbot\", \"username\": \"OrionBot\",
\"text\": \"Can you hear me now?\"}' https://hooks.slack.com/services/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
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 52.21.103.157...
* Connected to hooks.slack.com (52.21.103.157) port 443 (#0)
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: C:\Archive\SDKs\curl-7.46.0-win64-mingw\bin\curl-ca-bundle.crt
  CApath: none
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server did not agree to a protocol
* Server certificate:
*        subject: C=US; ST=California; L=San Francisco; O=Slack Technologies, Inc; CN=*.slack.com
*        start date: Jan 26 00:00:00 2015 GMT
*        expire date: Feb 18 23:59:59 2017 GMT
*        subjectAltName: hooks.slack.com matched
*        issuer: C=US; O=GeoTrust Inc.; CN=GeoTrust SSL CA - G3
*        SSL certificate verify ok.
> POST /services/T09HUKJH4/B0L6G78HH/jPHdT9gbuMsoq4cCnS5G3xnU HTTP/1.1
> Host: hooks.slack.com
> User-Agent: curl/7.46.0
> Accept: */*
> Content-Length: 146
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 146 out of 146 bytes
< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Content-Security-Policy: referrer no-referrer;
< Content-Type: text/html
< Date: Wed, 03 Feb 2016 17:21:28 GMT
< Server: Apache
< Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
< X-Frame-Options: SAMEORIGIN
< Content-Length: 2
< Connection: keep-alive
<
ok* Connection #0 to host hooks.slack.com left intact
PS [Patrick.Hubbard]>_
```

cURL is not at all terse, but confirms that http://hooks.slack.com most certainly is.

But what Slack is great at, and what cURL, (and later on ***Invoke–WebRequest***) will expose for you, is helpful debugging information when the data you're sending is bad. There are lots of ways to mess this up as you experiment but they're easy to fix, and then sin no more.

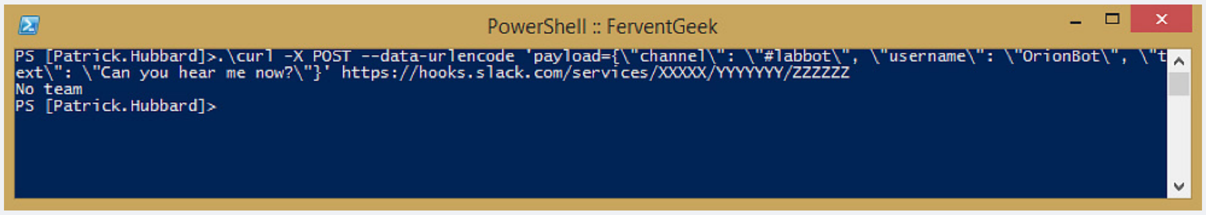
Bad escape syntax often looks like this:


PowerShell :: FerventGeek

```
PS [Patrick.Hubbard]>.\curl -X POST --data-urlencode "payload={\"channel\": \"#labbot\", \"username\": \"OrionBot\", \"t
ext\": \"Can you hear me now?\"}" https://hooks.slack.com/services/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
curl: (6) Could not resolve host: channel\: \#labbot\, \username\: \OrionBot\, \text\
curl: (6) Could not resolve host: you
curl: (6) Could not resolve host: hear
curl: (6) Could not resolve host: me
curl: (6) Could not resolve host: now
Payload was not valid JSON
PS [Patrick.Hubbard]>
```

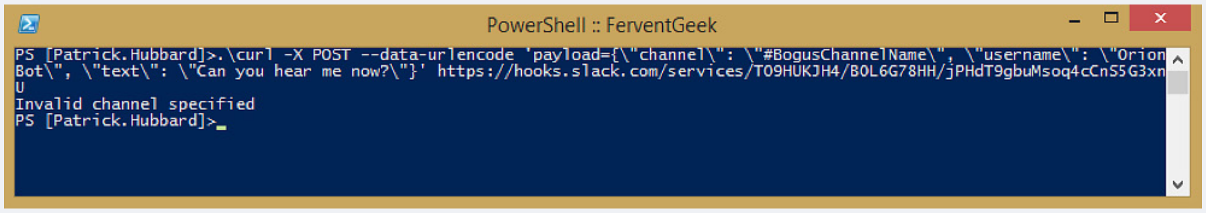
You'll even see cURL trying to resolve parts of the test in real time, as cURL can't even reach Slack. (In this example we forgot to switch our outer POST payload enclosing double quotes to singles.)

When your call is good but you pass incorrect data to Slack, it tells you what to fix. Here's a bad webhook as indicated by "No Team":


PowerShell :: FerventGeek

```
PS [Patrick.Hubbard]>.\curl -X POST --data-urlencode 'payload={\"channel\": \"#labbot\", \"username\": \"OrionBot\", \"t
ext\": \"Can you hear me now?\"}' https://hooks.slack.com/services/XXXXX/YYYYYYY/ZZZZZZ
No team
PS [Patrick.Hubbard]>
```

And here's an invalid channel name:


PowerShell :: FerventGeek

```
PS [Patrick.Hubbard]>.\curl -X POST --data-urlencode 'payload={\"channel\": \"#BogusChannelName\", \"username\": \"Orion
Bot\", \"text\": \"Can you hear me now?\"}' https://hooks.slack.com/services/T09HUKJH4/B0L6G78HH/jPHdT9gbuMsoq4cCnS5G3xn
U
Invalid channel specified
PS [Patrick.Hubbard]>_
```

Slack will also send specific errors for invalid text characters, general JSON property name problems, and more. In all these cases refer to the Slack developer pages and you'll be able to fix them in a jiffy from the command prompt. When you're good to go, paste the fixed syntax back into the Alert Execute Action.

If you're sick of messing with escape characters and wonky string builds, that's perfectly normal and why we really recommend using a PowerShell script for this where you can create JSON the right way, with no cURL, using objects. Read on for an example.

STEP 3B: ADD AN ACTION USING POWERSHELL (USING cURL)

Why This Matters

Doing direct cURL commands from within SolarWinds alerts is... fine. It gets the job done. HOWEVER, if you later decide you want to add additional fields, or change the channel, or whatever, you will need to manually change (and test! Never forget you must test!) each and every alert.

So in this section we're aiming to accomplish two things:

1. Get this running on PowerShell, because PowerShell is cool and all the cool kids are doing it and you should be doing it too because you want to be cool, right?
2. Create a "wrapper" (usually called a "shim" by programmer-types). This is what you paste into every alert action. The shim has minimal information that it passes to the real script. The shim will not change.

The real script (which is called by the shim) is a simple text file and can be changed separately. Changing the script once will affect every single alert that uses it with no need to update the Orion alert configuration. It's also extra handy because multiple alerts can re-use the same Alert Action. Reuse is a beautiful thing.

This is why programmers say that

"Laziness is a virtue."

Frame that and put it up over your desk.

Let's Do This

We're going to work backwards on this. First, we're going to create the PowerShell script and make sure it works, then we're going to update our alert to use the shim.

1. If you didn't create that ***C:\sw_tools*** folder when you installed cURL on your primary poller do it now.
2. In that folder, create a new file called ***"MySlackOrionAlertSender.ps1."***
3. In that file, put (copy/paste or type if you are OCD and/or a masochist) the following:

```
#Define Params to make life easy
param([string[]]$Caption,[string]$Status,[string]$De-
tails,[string]$Node,[string]$AckUrl)

# Define varilables
$curl = 'C:\sw _ tools\curl\bin\curl.exe'$icon = 'http://cool.example.com/
slack/orionbot/img/particles.png'
$webhook = 'https://YOUR _ SLACK _ WEBHOOK'
$channel = 'labbot'
$username = 'OrionBot'

$myarg = '-X POST --data-urlencode "payload={\"channel\": \"#' + $chan-
nel + '\", \"username\": \"' + $username + '\", \"text\": \"Alert - ' +
$Caption + ' status changed to ' + $Status + '\nView: <' + $Details +
'|Details>, <' + $Node + '|Node>\nTo acknowledge click <' + $AckUrl +
'|here>\", \"icon _ url\": \"' + $icon + '\"}" ' + $webhook

Start-Process $curl -ArgumentList $myarg
```

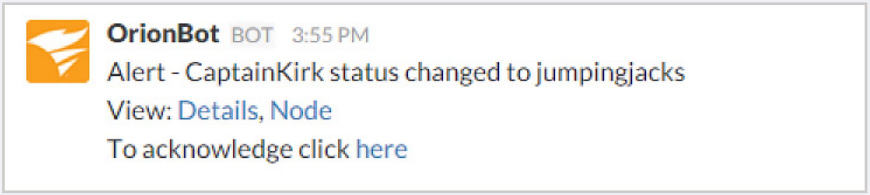
4. Now let's break down the code so you understand
    - Anything that starts with a hash/pound symbol (#) is a comment. I'm not commenting on comments.
    - $curl = 'C:\sw_tools\curl\bin\curl.exe' – This sets the location for the curl command.
    - $icon = 'http://cool.example.com/slack/orionbot/img/particles.png'– This sets the image you want to use as the avatar for your Slack user
    - $webhook = 'https://YOUR_SLACK_WEBHOOK' – This is your Slack webhook URL
    - $channel = 'labbot' – This is the name of the Slack channel where your messages should appear
    - $username = 'OrionBot' – This is the (fake) ***"username"*** that the alerts will appear to come from
    - $myarg = – This sets up ALL of the arguments that will be passed to the cURL command. Everything between here and the ending single quote are argument variables.
    - '-X POST --data-urlencode – Just like with the cURL command in the last section, this tells cURL to POST the data as a web page.

- "payload={ – Everything from here to the ending french brace (}) is the actual set of options that is going to be sent to slack.
- \"channel\": \"#' + $channel – The channel, using the variable we set earlier
- + '\", \"username\": \"' + $username – The username, using the variable we set earlier
- '\", \"text\": ' – From here on is the actual text message that will appear in the Slack window.
- \"Alert - ' + $Caption – The word ***"Alert"*** plus the name of the element that is having a problem.
- + ' status changed to ' + $Status – The words ***"status changed to"*** and the actual status.
- + '\nView: <' + $Details + '|Details>, – The word ***"View"*** and a clickable link to the alert details.
- <' + $Node + '|Node> – A clickable link to the Node details.
- \n – start a new line
- To acknowledge click <' + $AckUrl + '|here>\", - The words ***"To acknowledge, click"*** and a clickable link to the acknowledge URL
- \"icon_url\": \"' + $icon – This sets the icon that appears for the "orionbot" Slack user.
- + $webhook – This is your slack webhook URL
- Start-Process $curl -ArgumentList $myarg – And this is the ACTUAL command that is run by Powershell.

5. Save this file.
6. Now TEST this file. No really, take a minute and test it by just running:

```
Powershell.exe -File C:\sw_tools\MySlackOrionAlertSender.ps1
CaptainKirk jumpingjacks http://www.google.com
http://www.xkcd.com http://www.solarwinds.com
```

7. What you should see in slack is a message from OrionBot with the orionbot icon, that


OrionBot BOT 3:55 PM
Alert - CaptainKirk status changed to jumpingjacks
View: Details, Node
To acknowledge click here

8. Now, back in SolarWinds, you need to go back to that alert

TRICK: *We actually think that copying the first alert is better than editing, so that you have a reference you can use to teach others later.*

9. In the alert action, paste the following:

```
Powershell.exe -File C:\sw _ tools\MySlackOrionAlertSender.
ps1 "${N=SwisEntity;M=Caption}" "${N=SwisEntity;M=Status}"
"${N=Alerting;M=AlertDetailsUrl}" "${N=SwisEntity;M=Node.
DetailsUrl}" "${N=Alerting;M=AcknowledgeUrl}"
```

10. And once again, let's break this down:

- Powershell.exe – This invokes PowerShell
- -File C:\sw_tools\MySlackOrionAlertSender.ps1 – This tells PowerShell to run the ***"MySlackOrionSender.ps1"*** script, with the following variables.
- "${N=SwisEntity;M=Caption}" – The name of the system
- "${N=SwisEntity;M=Status}" – The status of the system
- "${N=Alerting;M=AlertDetailsUrl}" – The alert details URL
- "${N=SwisEntity;M=Node.DetailsUrl}" – The…

You know what? You're probably pretty sharp and we bet you can figure out the rest from here. You are sending the variables into the script, which then get processed and placed into a coherent message for Slack, right?

Right. Let's move on.

Actually, there isn't much "on" to move. Once you've pasted that line into your alert, you should test it.

A lot.

Troubleshooting

The biggest issue you're likely to encounter is calling the script correctly inside the Alert action. Make sure that these three things are working from the ***<cough>*** DOS command window.

1. The path to PowerShell is correct. If it's ***Powershell.exe*** searching the system path incorrectly or with a broken definition, it will cause problems. Try using the full path to the exact copy of PowerShell you prefer on the system. This is also a great way to select a specific version when there are several installed.
2. The path to the script is correct. If ***Powershell.exe*** doesn't know where to look, that's a problem.
3. You're not passing a wonky parameter to the script. This is a simple position based example, but there are other ways, including named parameter passing that may help. Google® passing parameters to PowerShell scrips for tips and examples.

STEP 3C: ADD AN ACTION USING POWERSHELL (THE RIGHT WAY)

i.e. NOT using cURL

So what do we have against cURL?

Nothing. cURL is a perfectly lovely, a perfectly functional way to get stuff out of a program and into a web-exposed REST service.

But it's not the most efficient, flexible, or secure option. What it is, is simple. And that's why we started with it. Once you get the hang of it, cURL is extremely straightforward.

But CURL is soooooo 5 pages ago.

You Sold Me. Where Do I Sign?

Before we even get into the actual steps, we want to clarify something: Nothing changes in SolarWinds. The alert trigger action can remain exactly the same, because we're still just passing variables to a PowerShell script, and letting the script do all the heavy lifting.

HOWEVER... because this is a learning experience and not exactly real life, we ARE going to change things a little.

1. First, copy the alert from the last section. Once again, this is so you can look back at how this whole thing has progressed.
2. In the SolarWinds alert trigger action change ***MySlackOrionAlertSender.ps1*** to ***MySlackOrionAlertSender2.ps1***

That's it for SolarWinds. Everything else will be taken care of in the script:

3. Go to ***C:\sw_tools***
4. Create a new file: ***MySlackOrionAlertSender2.ps1*** (of course!)
5. Enter the following:

```
# Define params to keep it clean when called by the Orion Alert
param([string[]]$Caption,[string]$Status,[string]$De-
tails,[string]$Node,[string]$AckUrl)
# Set the webhook endpoint on Slack, and format a tight datestamp
$webhook = 'https://YOUR _ WEBHOOK _ URL
```

```
$ftime = Get-Date -format “yyyy.MM.dd@HH:mm:ss”
# Set an emoji for specific status
$emoji = ‘’ # default empty
switch ($Status)
 {
       Unknown {$emoji = ‘:thought _ balloon: ’}
       Up {$emoji = ‘:green _ heart: ’}
       Down {$emoji = ‘:poop: ’}
 }
# build the JSON payload for the web request
$slackJSON = @{}
$slackJSON.channel = ‘#labbot’
$slackJSON.username = ‘OrionBot’
$slackJSON.icon _ url = ‘http://cool.example.com/slack/orionbot/img/
particles.png’

# Munge together the Slack formatted text string, (See Slack API
page for details)
$slackJSON.text = $emoji + $ftime + ‘- `’ + $Caption + ‘` status
changed to *’ + $Status + ‘*’ + “`n” + ‘View: <’ + $Details + ‘|De-
tails>, <’ + $Node + ‘|Node>’ + “`n” + ‘To acknowledge click <’ +
$AckUrl + ‘|here>’

# Build the web request
$webReq=@{
 Uri = $webhook
 ContentType = ‘application/json’
  Method = ‘Post’
 body = ConvertTo-Json $slackJSON
}

# Send it to Slack
Invoke-WebRequest @webReq

# Uncomment the following line(2) to debug the final send
# write-output $slackJSON
# write-output $slackJSON.body
```

6. Next, let's break down the key elements of this script. Even though it's fairly well commented, this is a teaching guide so we're not going to leave you hanging with just "look through the code and it will all make sense" (even though we hope it actually does!)

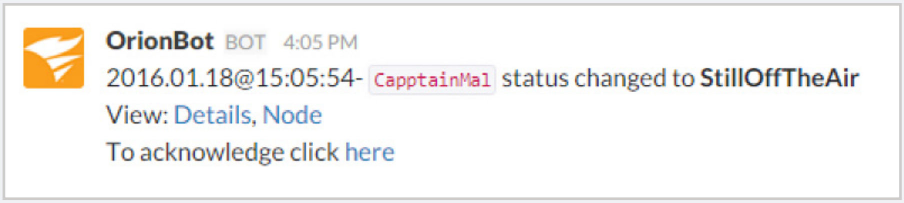
    - param([string[]]$Caption,[string]$Status,[string]$Details,[string]$Node,[string]$AckUrl) – This is where we define our variables. Why should we bother? Ask Corey Adler (https://thwack.solarwinds.com/community/solarwinds-community/geek-speak_tht/blog/2015/12/01/moar-coding-tip-and-lots-of-caps)
    - $webhook = 'https://YOUR_WEBHOOK_URL – This is your Slack webook
    - $ftime = Get-Date -format "yyyy.MM.dd@HH:mm:ss" – This gets the current time (you'll see how it fits in later)
    - $emoji = '' – This (and the lines that follow it) set up a cute little enhancement we couldn't do with cURL. It will set an emoji icon to match the status. A good status gets you a green heart. A bad status looks like poop (literally!), and an unknown status is a thought bubble.
    - $slackJSON = @{} – This sets up the array that will hold all the JSON variables. You'll see it later. Right now, this is just another declared variable.
    - $slackJSON.channel = '#labbot' – The Slack channel the alert will show up in.
    - $slackJSON.username = 'OrionBot' – The Slack user name the message will appear to come from.
    - $slackJSON.icon_url = 'http://cool.example.com/slack/orionbot/img/particles.png' – As with all the cURL examples, this is the good old avatar that will show up as the Slack user's image.
    - $slackJSON.text = – This block sets up the content of the actual alert message.
    - + $emoji – The emoji for the status (that we set up earlier in the script)
    - + $ftime – The current time of the alert (again, set up earlier)
    - + '- `' + $Caption – The name of the device, interface, etc.

- + '` status changed to *' + $Status – The literal words ***"Status changed to *"*** plus the status (down, up, unknown, etc.)
- + '*' + "`n" + 'View: <' + $Details + '|Details>, <' + $Node + '|Node>' – An asterisk (*) to finish off the status line, plus clickable links for Alert Details and Node Details.
- + "`n" + 'To acknowledge click <' + $AckUrl + '|here>' – A hard return plus a clickable link for the alert acknowledge.
- $webReq=@{ – This puts everything together into a nice neat usable bundle. The URL plus the type of content and method (json and POST, respectively), and the text of the message converted into JSON format.
- Invoke-WebRequest @webReq – Believe it or not, everything up until now has been setup. This starts the ACTUAL command.
- # write-output $slackJSON – This section is for error checking. If things aren't working, you can uncomment these two lines to see on screen what is going on.

7. Now test the code. At the regular old DOS prompt, type the following command:

```
Powershell.exe -File C:\sw _ tools\MySlackOrionAlertSender2.ps1
CapptainMal StillOffTheAir http://www.google.com http://www.xkcd.
com http://www.solarwinds.com
```

You should now see the following message in Slack:


OrionBot BOT 4:05 PM
2016.01.18@15:05:54- `CapptainMal` status changed to **StillOffTheAir**
View: Details, Node
To acknowledge click here

HelloScript, we see what you did there.

8. Assuming everything works, you are done. Why? Because we already changed the shim in the alert trigger area.

Troubleshooting

The most likely problems you'll encounter will be, wait for it, PowerShell syntax errors. No biggie, just fix them. Ok, seriously, use Google to find the appropriate TechNet and stackoverflow articles, tweak and test from the command line and you're all set. Even if you're currently a config jockey by day, programming is fun, and unavoidably necessary to keep a gig as we move to a software defined infrastructure. Experimentation, even if you're new to programming won't hurt a bit.

Now then if you really want to make it easy, check out the end of the Advanced REST and Orion Alert Integration section for an example of using modularity and reuse to reduce script errors. Now scat!

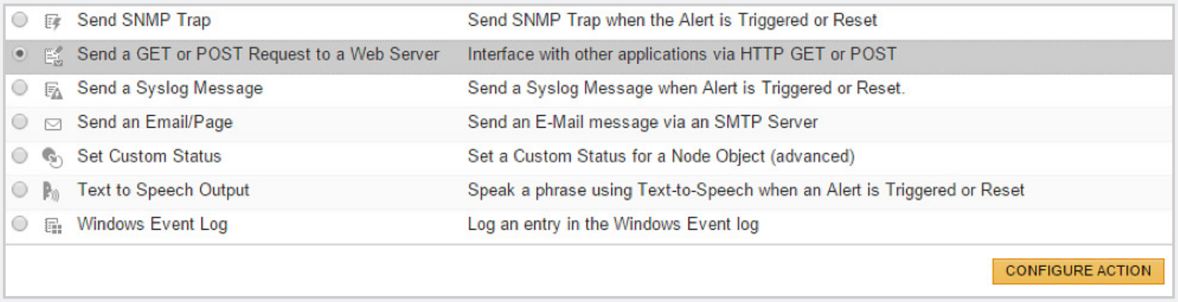
THE CASE AGAINST ORION'S NATIVE GET OR POST REQUEST ACTION

So what do we have against the built-in POST action?

Orion ships with another built-in action that's technically the quickest way to send alerts to Slack. You don't need to install cURL, string escapes are easier and you can still use Orion variables. But with easy comes limited troubleshooting and more maintenance. You can't do authentication, there's no error reporting, (see reading the Action log above), and no logic. That means if you have more than one, you'll have to configure each one individually. You can't change behavior in a script and you can't interactively debug in a command window.

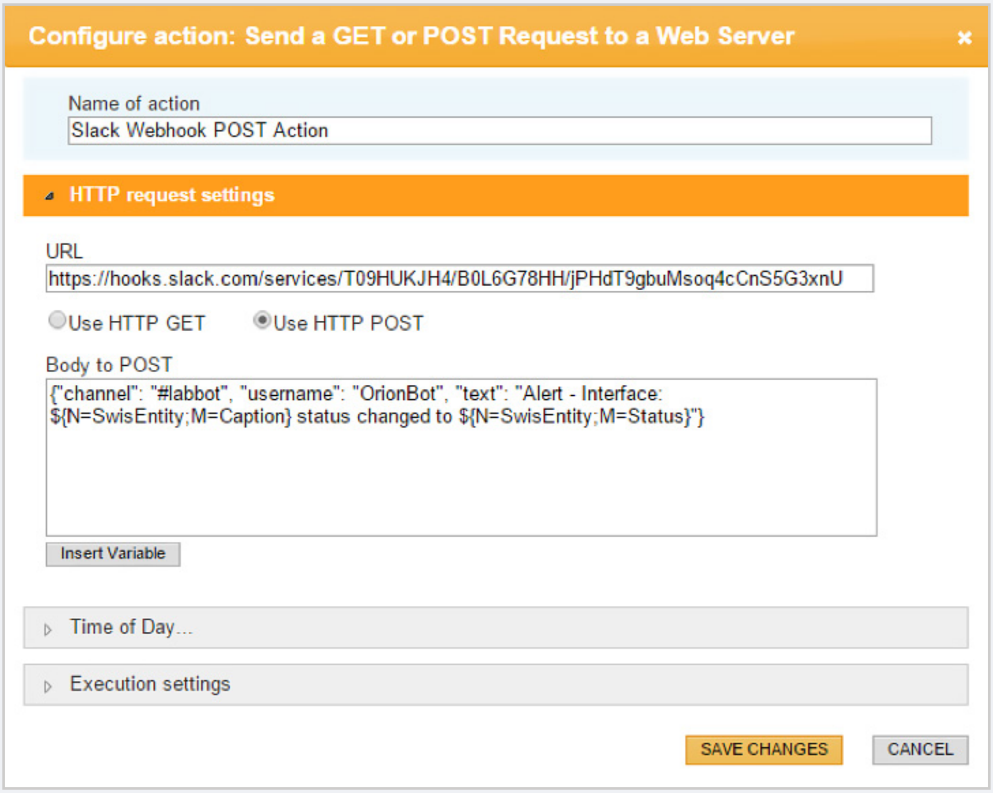
But for the sake of completeness, here's how to do it. You'll initially find it easy, might like it and it may perfectly suit your needs. But you'll be a bad person for using it, and worse, a less accomplished geek.

First, edit the Trigger Actions section of an Alert, and add a ***"Send a GET or POST Request to a Web Server"*** Action:



○ Send SNMP Trap	Send SNMP Trap when the Alert is Triggered or Reset
● Send a GET or POST Request to a Web Server	Interface with other applications via HTTP GET or POST
○ Send a Syslog Message	Send a Syslog Message when Alert is Triggered or Reset.
○ Send an Email/Page	Send an E-Mail message via an SMTP Server
○ Set Custom Status	Set a Custom Status for a Node Object (advanced)
○ Text to Speech Output	Speak a phrase using Text-to-Speech when an Alert is Triggered or Reset
○ Windows Event Log	Log an entry in the Windows Event log

Click ***"Configure Action"*** and configure the action. Give it a name, paste your webhook into the URL text box, select ***"Use HTTP POST,"*** and then post the JASON you want to send to Slack in the ***"Body to POST"*** box.


Configure action: Send a GET or POST Request to a Web Server

Name of action: Slack Webhook POST Action

HTTP request settings

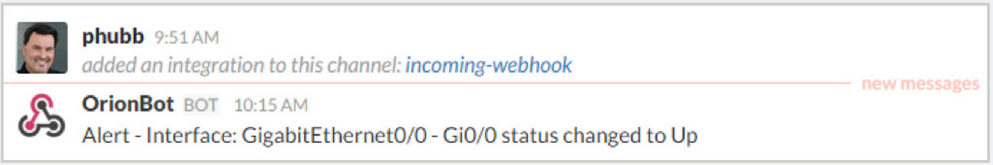
URL: https://hooks.slack.com/services/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

Use HTTP GET [✓ Use HTTP POST]

Body to POST:

```
{"channel": "#labbot", "username": "OrionBot", "text": "Alert - Interface:
${N=SwisEntity;M=Caption} status changed to ${N=SwisEntity;M=Status}"}
```

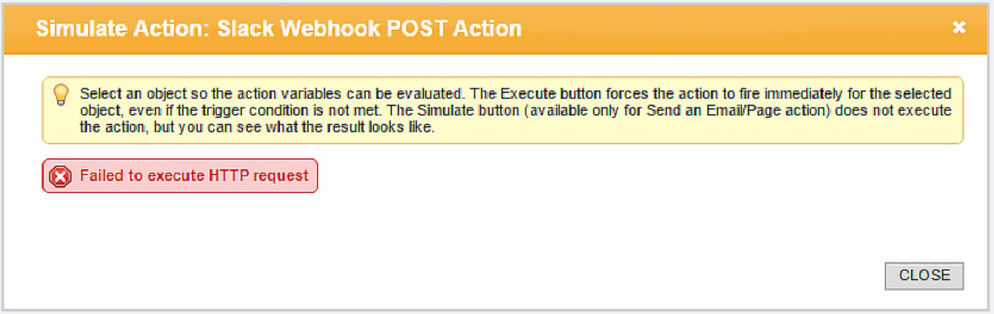
Save your changes and test using the ***Alert Action*** test button.


phubb 9:51 AM
added an integration to this channel: incoming-webhook

new messages

OrionBot BOT 10:15 AM
Alert - Interface: GigabitEthernet0/0 - Gi0/0 status changed to Up

And voila! It's working, you're all done. Call it a day and knock off early. Unless this happens:


Simulate Action: Slack Webhook POST Action

Select an object so the action variables can be evaluated. The Execute button forces the action to fire immediately for the selected object, even if the trigger condition is not met. The Simulate button (available only for Send an Email/Page action) does not execute the action, but you can see what the result looks like.

Failed to execute HTTP request

What caused this? You may never know. If it's a transport or HTTP issue you will get details in the log file, but if there is any issue with your JSON, you'll only get "HTTP 500 you stink, go away," and won't be able to quickly debug the payload. This is especially true with the really cool messages we're sure you'll come up with using the very flexible Slack formatting options.

Ok, so how about a compromise rather than wholesale condemnation of this method? If you want to set up a basic, quick alert to a Slack channel that you can use to get your team interested and your boss to give you time to configure rich integration, it's an ok way to start.

There, we said it. Now, on to the advanced stuff.

Troubleshooting

There's not really a lot of options- that's the good and the bad. If you get a "***Failed to execute HTTP Request"*** message when you test it, look for transport and HTTP protocol issues in

C:\ProgramData\Solarwinds\Logs\Orion\ActionsExecutionAlert.log.

For anything else, you'll just have to experiment with the "Body of POST" JSON and cross your fingers.

ADVANCED REST AND ORION ALERT INTEGRATION

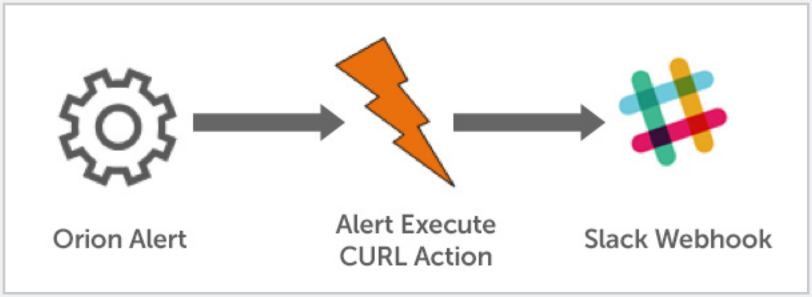
Aw yeah.

ROBUST & MAINTAINABLE CODE

Like it or not, we're talking about code here, people. Scratch that. Like it people, we're talking about code! Code is how we manage infrastructure by putting logic onto disk so we don't have to constantly reconfigure everything. It's also how we go beyond basic layouts and simple data integration, to advanced integration. We get rich data that tells the admin on the other end of the notification everything they need to quickly close tickets. Who likes to quickly close complex tickets? You!

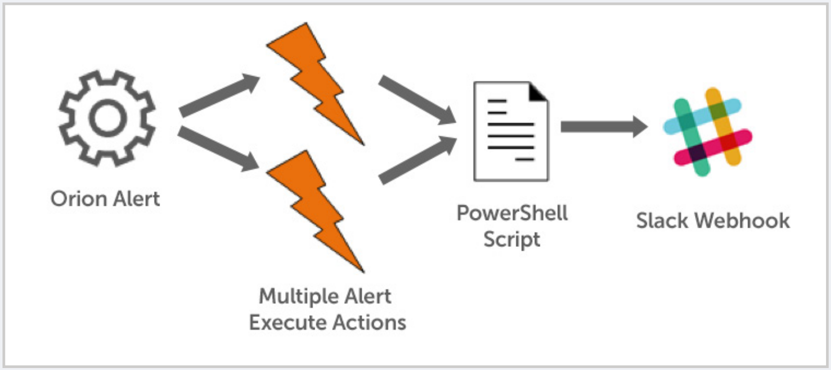
Consider these possibilities using Execute Program Alert Actions and PowerShell scripts:

In the first example of our guide you configured a direct cURL integration:



The Orion Execute Program Alert Action called cURL on the command line and cURL sent an escaped, formatted POST directly to Slack. The downside is you're stuffing a lot of long parameters on the DOS command line and all edits must happen in the Alert config screen.

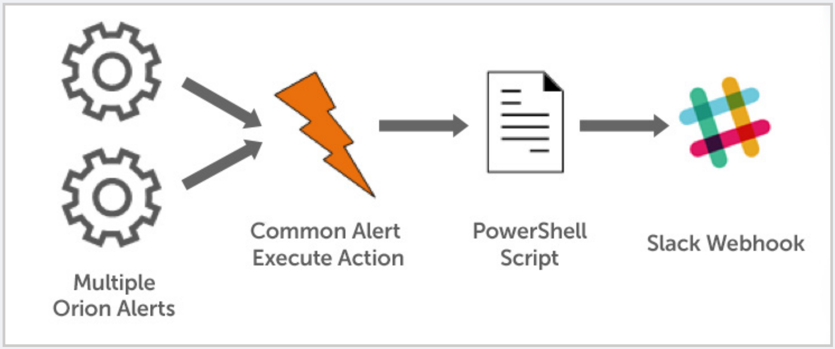
Now consider what happens when you instead create a script that sends the alert to Slack:



In this scenario, we have an extra "thing" to manage, but management is easier. First the configuration of the Run Command Alert Action is much, much easier to maintain. ***{Path to PowerShell} {path to script} {parameters from Orion variables}***. That's much easier for development. Second, notice that you can have multiple Alert Actions call the same script. Maybe you want to pass different variables based on the type of alert, (Node, Interface, Application, Virtual Machine etc.), but only maintain one script that does the actual send to Slack. Aaaahh.

As an added bonus, you set up your alerts only once in Orion, and then fiddle with the presentation details as much as you like, easily. Instead of editing Alert Actions, you edit a script, save, test from the command line, and you're done. Oooooh, Ahhhh.

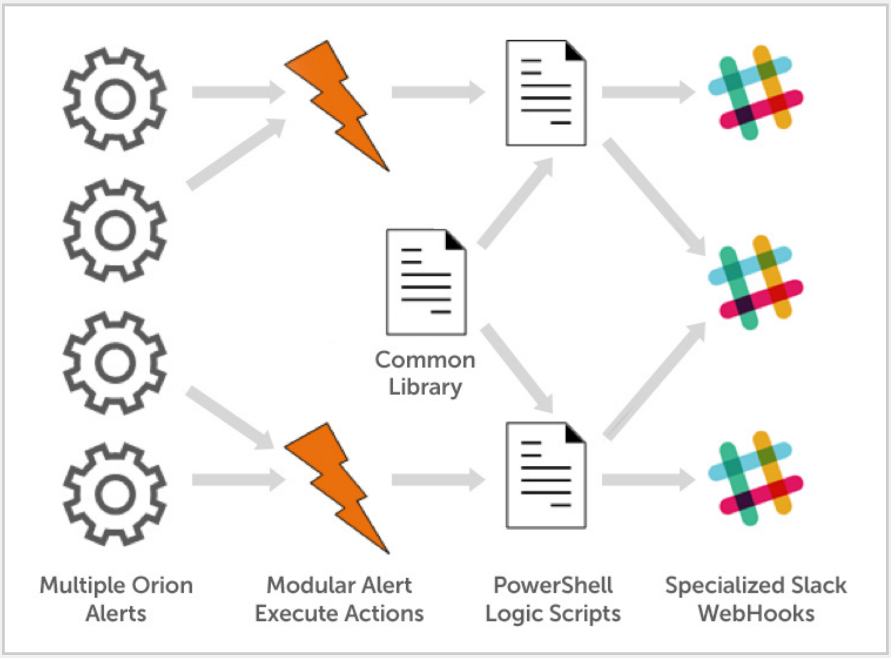
You can also do this:



Reuse is a beautiful thing. You can configure the Alert Action only once then reuse it for many Alerts. It's a great way to do Alert escalation, thresholding and more. Because you can pass Alert instance data in the Alert Action config to the script, the script can reformat based on the Alert ID, rather than the Action definition. Good golly Miss Molly.

A FULLY ARMED AND OPERATIONAL BATTLE STATION

The real beauty of code comes in cleaning up the quick-typed spoor of your first version to produce something that's not brittle. It happens when you realize what you can do with Orion Alerts and Slack. For example, what about this?



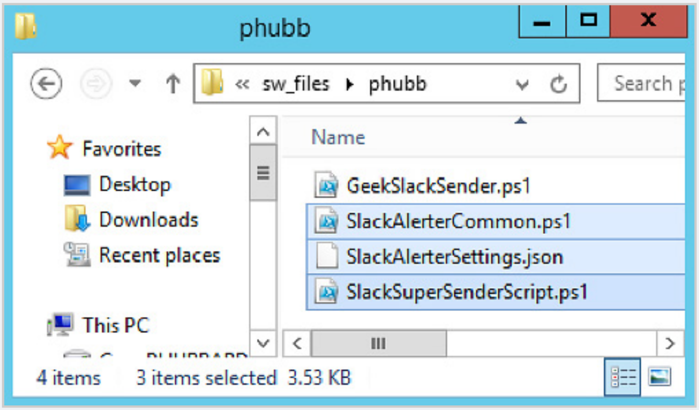
Don't freak out! We'll explain what's going on here. You may find you want to send certain alerts to specific Slack channels to reach different audiences. You may end up with lots of Alerts using differentiated Alert Actions based on advanced rules. You may have a selection of specialized sender scripts that contain specific advanced logic or formatting, used in different situations. And in those cases the diagram above is actually pretty straightforward.

This design takes advantage of a small number of PowerShell scripts and lots of reuse to scale out as far as your inspiration and perspiration take you. For example, you can name your channels in a common config script, and set properties for their webhooks. Then if you need to change the webhook, you set it in a single file, and every other integration gets updated. Want to reuse complex logic as a function across multiple PowerShell scripts? W00t, you can do that too, in time for lunch.

IT'S WAY EASIER THAN YOU THINK

Enough telling, more showing. Let's walk through a setup using nothing but SolarWinds Orion, and Slack and you'll see what we're nattering on about.

First, pick a folder on the Orion poller where the alert runs and create three files.



GeekSlackSender.ps1
SlackAlerterCommon.ps1
SlackAlerterSettings.json
SlackSuperSenderScript.ps1

4 items 3 items selected 3.53 KB

Two are PowerShell .ps1 scripts and the other is a JSON config .json file. Slack***SuperSenderScript.ps1*** has an include for ***SlackAlerterCommon.ps1*** which knows how to read ***SlackAlerterSettings.json***. Is your head exploding yet? Of course not, because this is easy now that you've followed this oh-so-handy guide.

Why did we put the config in JSON instead of XML or text, well let's take a look at SlackAlerterSettings.json, while you copy and paste it onto your server. (Filenames are important here but PowerShell will helpfully barf hints if you make a typo.)

```
{       "channels" : [
                { "labbot" : {
                        "webhook" : "https://hooks.slack.com/services/
T09HUKJH4/B0L6G78HH/jPHdT9gbuMsoq4cCnS5G3xnU",
                        "username" : "OrionBot"
                }},
                { "foobarbaz" : {
                        "webhook" : "kjshfkjsdf/sdksjdhfsdaf/sadfdfsdsdfdsaf",
                        "username" : "OrionBot"
                }}
        ],
        "icon _ url" : "http://cool.example.com/slack/orionbot/img/
particles.png"
}
```

First, it defines an array of ***"channels,"*** named ***"labbot"*** and ***"foobarbaz."*** Each has properties like the webhook and username to use. This means you can have as many webhooks as you want shared in many scripts, with single point update. Yea verily. And second, it also has a global ***"icon_url"*** property, and I'll bet you can guess what that's for.

So how do we read this file? Easy, we add our second file, ***SlackAlerterCommon.ps1***, which you can also copy from right here and tweak.

# Load the config file into an global object and add a couple of convenience vaial-bles, note you may need the fill path to the .json config file here too.

```
$OrionSlackConfig = Get-Content -Raw -Path SlackAlerterSettings.json
| ConvertFrom-Json
$SlackChannels = $OrionSlackConfig.channels
$ftime = Get-Date -format "yyyy.MM.dd@HH:mm:ss"
function NewSlackPayload($channel, $username, $icon _ url)
{
      # Creates the payload object as a hashtable
      $slackJSON = @{
            channel = '#' + $channel
            username = $username
            icon _ url = $icon _ url
            text = $null
      }
      return $slackJSON
}
function GetStatusColor($status)
{
      # Gets colors for Critical, Down, Unknown, Up, and Warning
status
      switch ($status)
      {
            Unknown {return '#808080'}
            Up {return '#008000'}
            Warning {return '#FFFF00'}
            Critical {return '#800000'}
            Down {return '#FF0000'}
            default {return "#808080"} # unknown
      }
}
```

```
function SendToSlack($webhook, $payload)
{
      # Build the web request and sends it off to Slack
      $webReq=@{
            Uri = $webhook
            ContentType = ‘application/json’
            Method = ‘Post’
            body = ConvertTo-Json $payload -Depth 100
      }

      # Send it to Slack
      Invoke-WebRequest @webReq
}
```

Remember this file is designed to be reused by all of your actual sender scripts and offload the common work like making the config data available, building some base objects to make life easier and less error prone and also does the actual work of sending the data to Slack. Sorry cURL, there’s no place for you here – because you’re one more layer to install and debug, that’s why.

So last we have the script that does the actual creation and formatting of the message we’re sending to Slack. Notice it doesn’t have any configuration details needed to send the message, the common file takes care of that for us. Everything here is either receiving the parameters from the Orion Alert Action of formatting output. Create one more file called SlackSuperSenderScript.ps1 and paste this in. Tweak if needed for your system, like adding the full paths to the include file.

```
# Define params to keep it clean when called by the Orion Alert and
allow for named passing rather than positional
Param(
 [string]$objectCaption,
 [string]$status,
 [string]$nodeDisplayName,
 [string]$detailsUrl,
 [string]$nodeUrl,
 [string]$ackUrl,
 [string]$alertTriggerCount,
 [string]$downTime
)

# Include common config, loads the config .json to objects
# This will work on your desktop as is, but you'll want the full path to
this file in
# production or Orion won't be able to find it.
. .\SlackAlerterCommon.ps1

# call a method in the common file to create the slack payload stub
$payload = NewSlackPayload "labbot" "OrionBot" $OrionSlackConfig.icon _ url

# And now the magic of building JSON from hashtables in PowerShell. Look
how beautiful this is!
# Use the Slack formatting guide for ideas https://api.slack.com/docs/
formatting and the attachments
# guide for overall structure https://api.slack.com/docs/attachments.
$attach = @{
      title = $objectCaption + ' status changed to ' + $status
      title _ link = $alertUrl
      color = GetStatusColor $status
      fallback = $objectCaption + ' status changed to ' + $status + ' @'
+ $ftime
      text = '<' + $ackUrl + '|Acknowledge>'

      fields = @(
            @{
                  title = "Time"
                  value = $ftime
                  short = $true
            },
            @{
                  title = "Node"
                  value = '<' + $nodeUrl + '|' + $nodeDisplayName + '>'
                  short = $true
            },
            @{
                  title = "DownTime"
                  value = $downTime
                  short = $true
            },
            @{
                  title = "Trigger Count"
                  value = $alertTriggerCount
                  short = $true
            }
      )
}
```

```
# Add them to the Slack payload wrapped in an array
$payload.attachments = @($attach)

# Un-comment the following line if you want to get the output of
the JSON on the command line for debug
# ConvertTo-Json $payload -Depth 100 | write-output
# Send it off using a reusable function in the include library
SendToSlack $SlackChannels.labbot.webhook $payload
```

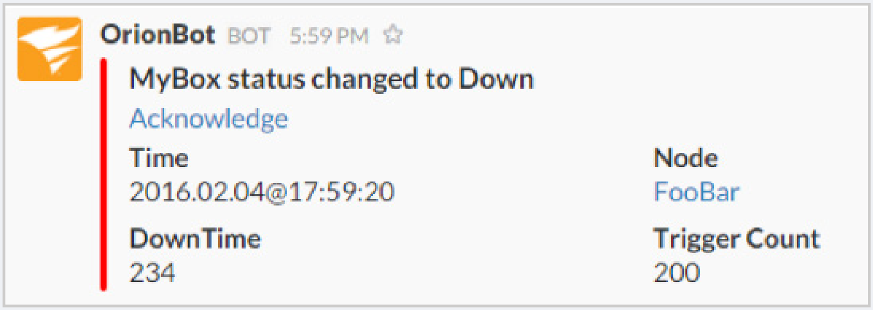
We bet you're getting the picture by now. Notice that all the icky, yucky and evil URL escaping and other shenanigans are completely gone, and happily forgotten. It's easy to read and much easier to debug than one big file. And the best bit is if you want to completely reformat other messages, you can just clone this one into something completely different. For extra credit you might even pass the layout definition in on the Alert Action and have multiple layouts in a single file. The possibilities are endless.

TIME TO TEST – HIT IT!

We're almost done. The last step is to test it on the PowerShell command line thusly. (Note we're checking using incomplete parameter names but don't have to type them all out- another neat PowerShell trick.)

```
PS [Patrick.Hubbard]>.\SlackSuperSenderScript.ps1 -s "Down" -o
"MyBox"-ac "http://solarwinds.com" -nodeUrl "http://google.com
" -nodeD "FooBar"-alert 200 -down 234
```

And what should happen is something like this over in your Slack channel:



OrionBot BOT 5:59 PM

MyBox status changed to Down

Acknowledge

Time
2016.02.04@17:59:20

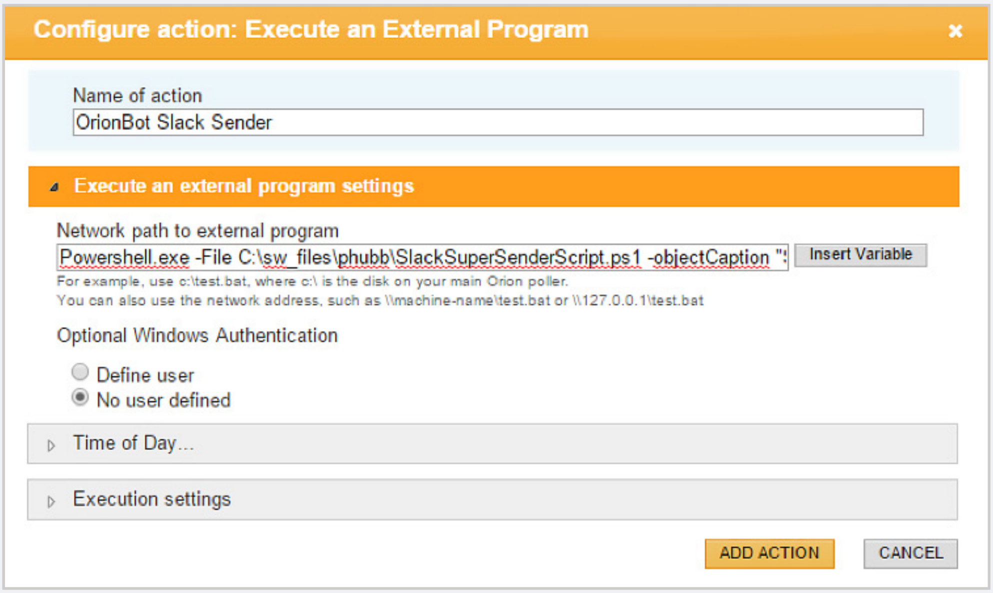
Node
FooBar

DownTime
234

Trigger Count
200

Wowzers, look at all that slick Slack formatting. It's using attachments to set a colored bar for the status, includes an action header, and organizes the sub-elements using fields. The cool thing is that this is not custom code in Slack, we're just using their online guide and hacking coolness on the fly. Also notice in the test above that we're passing named parameters into the script. This is a really helpful upgrade when it comes to debugging, because positional parameters stink on ice and are prone to bit rot.

If that worked for you, then you're ready to complete the last step, configuring your Alert Action. On your alert, create a new Execute External Program Action and give it a cool name.



Configure action: Execute an External Program

Name of action
OrionBot Slack Sender

Execute an external program settings

Network path to external program
Powershell.exe -File C:\sw_files\phubb\SlackSuperSenderScript.ps1 -objectCaption "
For example, use c:\test.bat, where c:\ is the disk on your main Orion poller.
You can also use the network address, such as \\machine-name\test.bat or \\127.0.0.1\test.bat

Optional Windows Authentication

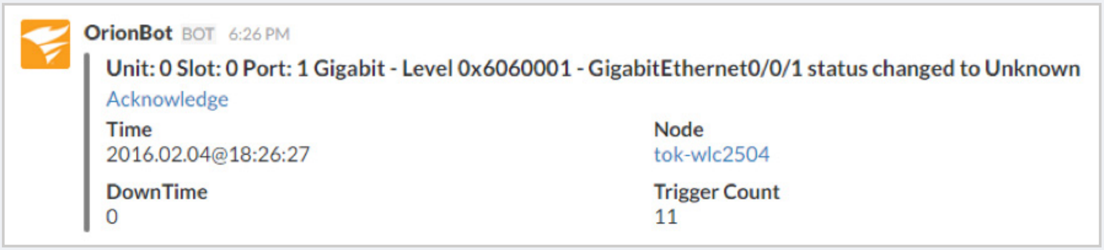
Define user [✓ No user defined]

Next paste in the following template into the Network path to external program box, editing the path to your ***SlackSuperSenderScript.ps1*** script:

```
Powershell.exe -File {script path on Orion poller server}
-objectCaption "${N=SwisEntity;M=Caption}" -status
"${N=SwisEntity;M=Status}" -nodeDisplayName "${N=SwisEntity;M=Node.
DisplayName}" -detailsUrl "${N=Alerting;M=AlertDetailsUrl}"
-nodeUrl "${N=SwisEntity;M=Node.DetailsUrl}" -ackUrl
"${N=Alerting;M=AcknowledgeUrl}" -alertTriggerCount
"${N=Alerting;M=AlertTriggerCount}" -downTime
"${N=Alerting;M=DownTime}"
```

Notice in this example that instead of trying to remember which parameter goes where, we name all of them. Check the ***parms*** definition at the top of your script and there they are ready for copy–paste. You can even stick a copy of this right in your script file as a multi-line comment block.

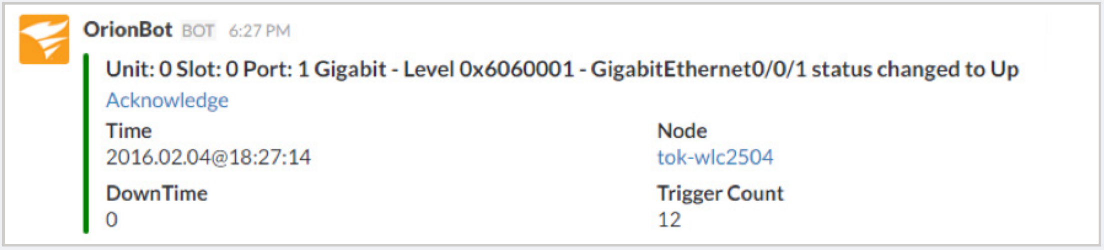
Click add action, finish saving the parent Alert and you're ready to test. For this one we test by un-managing and re-managing an interface because we're triggering on status change. You'll want to make sure your polling interval for your test interface is 2-3 minutes so you won't sit forever. What should happen is this:



OrionBot BOT 6:26 PM
Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Unknown
Acknowledge

Time	Node
2016.02.04@18:26:27	tok-wlc2504
DownTime	Trigger Count
0	11

First the alert query fires and notices that you previously set it into unmanaged which means for the poller it's unknown. The side bar is gray just like the status icon in Orion. Within a couple of minutes the poller fires and the status changes from Unknown to Up, which causes the Alert to fire again and bada-bing-bada-boom:



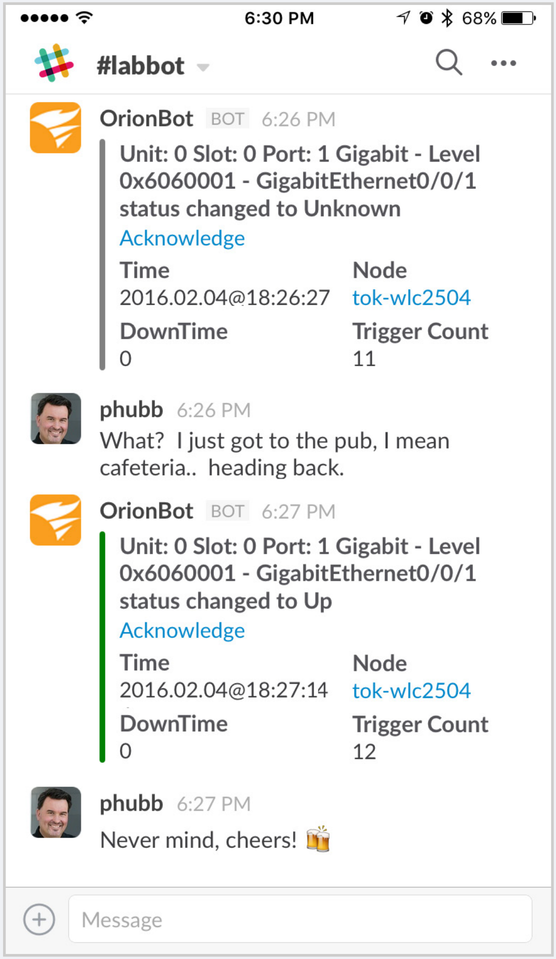
OrionBot BOT 6:27 PM

Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Up

Acknowledge

Time 2016.02.04@18:27:14	Node tok-wlc2504
DownTime 0	Trigger Count 12

You're green again indicating it's back up. If you happen to have the Slack mobile app you're super green:



OrionBot BOT 6:26 PM

Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Unknown

Acknowledge

Time 2016.02.04@18:26:27	Node tok-wlc2504
DownTime 0	Trigger Count 11

phubb 6:26 PM

What? I just got to the pub, I mean cafeteria.. heading back.

OrionBot BOT 6:27 PM

Unit: 0 Slot: 0 Port: 1 Gigabit - Level 0x6060001 - GigabitEthernet0/0/1 status changed to Up

Acknowledge

Time 2016.02.04@18:27:14	Node tok-wlc2504
DownTime 0	Trigger Count 12

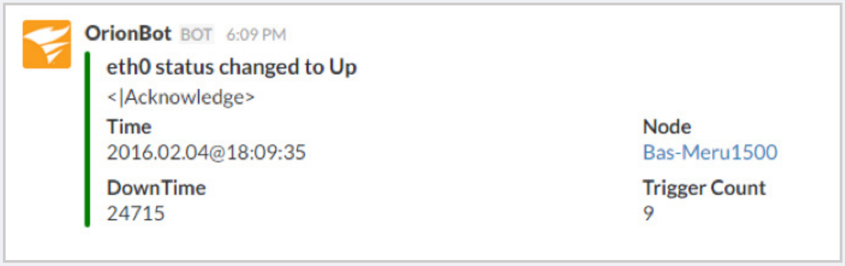
phubb 6:27 PM

Never mind, cheers! 🍻

Hopefully that wasn't difficult at all and now everyone on your team is singing your praises, buying your lunch and urging you to take a three day weekend. OK, probably not, but one can dream.

TROUBLESHOOTING

Assuming you got the above script to work from the first command line test, the most likely issue you may encounter is fat fingering the Alert action script call or selecting an incorrect SolarWinds Alert variable substitution. The quickest way to check that is to test the template in Action using the Orion, Manage Alerts Test Action button. You should see something like this:



OrionBot BOT 6:09 PM

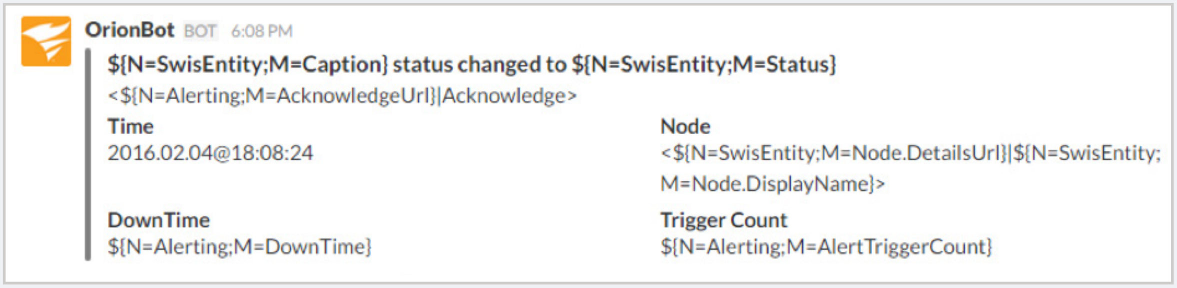
eth0 status changed to Up

<|Acknowledge>

Time	**Node**
2016.02.04@18:09:35	Bas-Meru1500
DownTime	**Trigger Count**
24715	9

It won't be fully populated, but it will pick up most of the information from the test interface you selected in the web test panel. Notice also that the ***<|{tag}>*** is visible because if a URL is blank, Slack shows an empty control.

If that doesn't work, then go back and paste the template into your PowerShell command window and you should see something like this:



OrionBot BOT 6:08 PM

${N=SwisEntity;M=Caption} status changed to ${N=SwisEntity;M=Status}

<${N=Alerting;M=AcknowledgeUrl}|Acknowledge>

Time	**Node**
2016.02.04@18:08:24	<${N=SwisEntity;M=Node.DetailsUrl}\|${N=SwisEntity;M=Node.DisplayName}>
DownTime	**Trigger Count**
${N=Alerting;M=DownTime}	${N=Alerting;M=AlertTriggerCount}

Notice that all the elements have been replaced with the SWIS API definitions for each of the variables you're passing to the script. It's really handy when you get to dozens of parameters where one is missing or out of place because you can test both the Alert engine layer and the script->Slack layer. And who doesn't like layers

A FINAL WORD ON ALERTS

LET'S BE SMART, HERE, PEOPLE!

This section has nothing to do with Slack, very little to do with SolarWinds, and everything to do with stopping the "noise" that everyone always complains about with regard to monitoring and alerting.

Intelligent alerting is not about more programming, better macros, etc. It's about putting enough information into each alert that recipients understand what happened and why they received a message. With that information, they're able to communicate with you (the monitoring engineer) to fine tune the trigger, timing, thresholds, etc.

It all starts with what information you include. At one end of the scale, you have the ugly (but all too common) message:

```
Your server has had a problem. Please investigate and resolve if necessary.
```

Sure thing, buddy. I'll get right on that.

Better (but still not great) is what we usually see in most monitoring environments:

```
The interface "GigabitEthernet 0/1" on "ROUTER017" is down. A ticket has been opened.
```

OK, at least we know what device and element it is, but it's still a long way from what any of us who have worked the helpdesk (or on-call) would consider to be truly useful.

Let's get an inventory of the information we'd like to see:

1. The problem category (CPU, disk, network, up/down, Exchange™, etc.)
2. The device affected
3. The IP of the device affected
4. The device vendor
5. The device model
6. The device OS

7. The "owner" of the device (group, location, etc.)
8. The sub-component affected
9. The current status of the sub-component
10. The threshold or trigger condition
11. The monitoring server that detected the issue
12. The time of the problem
13. The time this alert was sent
14. The name of alert that is sending this message

What that looks like might be something like this:

```
CPU ALERT: On ROUTER017 (IP: 10.199.3.5, Cisco 2851 running 12.4(13c),
RELEASE SOFTWARE (fc2)) the CPU is 98% and the threshold is 90%.
The NETOPS team has been informed via the ticket system and email.
This problem was detected by SWPOLLER05 at 3:17am. This alert was
sent at 3:18am. (ALERTNAME: NETOPS_CPU_CRIT)
```

These values are easily extracted from most monitoring systems. What they give the recipient is a far more accurate view of what happened (even with something as simple as a CPU alert).

And at the end of the day, that is what matters most. Not whether you send to Slack or Jabber® or email. Not whether you have fancy icons and emoji's.

What matters is that the right people get the right information in time to take decisive action to correct (or better still, avert) a problem that will impact your environment, and therefore impact your business.

WRAPPING UP

TAKE A MOMENT TO REFLECT

This has been an amazing journey for us, and we hope you feel the same sense of excitement over the possibilities these techniques open up for you and your team.

You are now able to get data and interactive links out of SolarWinds and into a collaborative online communication platform like (Slack), where teams can be notified on multiple devices (without having to set up multiple alerts); where you can create "FYI" messages without wondering which team is actually working the ticket and which have mentally decided they're "NOT IT!"; and where you have an ongoing historic archive of events and the conversations around those events that doesn't depend on email, log files, or locally-accessible ticket systems.

Looking beyond SolarWinds and Slack for just a moment, you've learned about cURL and JSON and REST-ful function calls, and how to use those via PowerShell. That's not a bad set of skills to have, with or without the monitoring and workgroup features.

WHAT'S NEXT?

Without really tipping our hand, we've opened the door for you to the wide world of "ChatOps." This is a whole new category of automation that allows teams to interact with systems within a text-based communication system (like Slack) using "chatbots."

Chatbots can do everything from sending information from remote systems into the chat stream (which we've already done) to listening for commands and acting on them in ways that are amusing (like "Find me a cute cat video"), informative ("Tell me what the stock price is for a particular company" or "What is the CPU on SERVER123"), to downright useful ("restart the IIS service on SERVER123").

The only limits are your willingness to code, and your imagination.

DEDICATIONS

*"For Debbie. Caring for someone who loves their job is clearly more difficult than caring for someone who hates it, but you have stood by me as I have savored this dream job called "Head Geek." Your patience and love have been a constant through decades of time and generations of family. I **still** love you more than everything."*

–Leon

"To my wife Mandy, the greatest automation engineer I've ever known. I'm truly honored to pass your white box tests."

–Patrick

BR-1602