IT-Automation: It's not that scary! - Cumulative Project

Objectives:

- Configure monitoring and alerting on Prometheus for a simple load balanced website
- Configure a Chef server
- Automatically deploy Apache2 to multiple servers
- Monitor the Apache2 service across the cluster
- Configure alerts for the Apache2 service

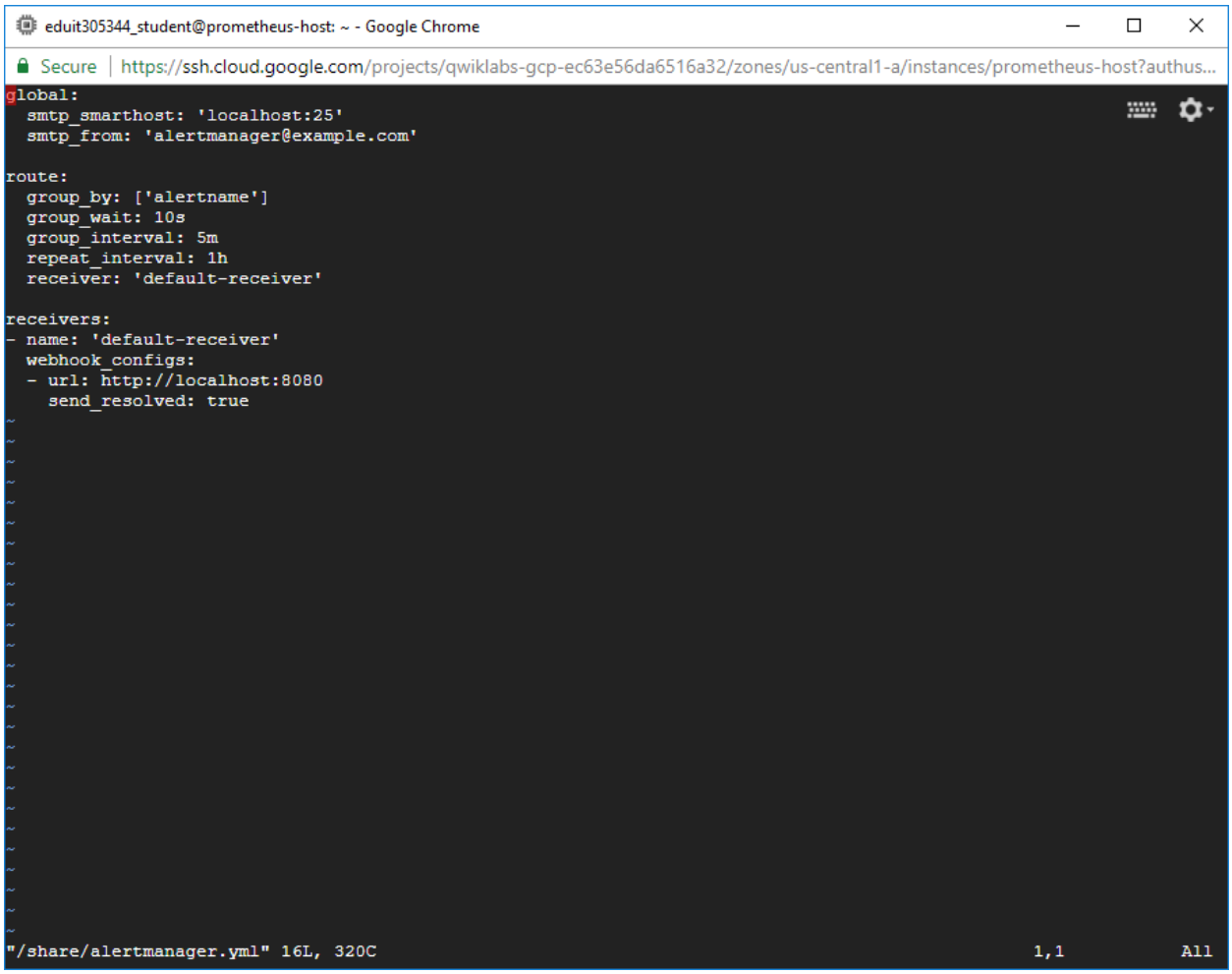
Assumptions:

- Prometheus Server Configured with Alertmanager
- Chef Workstation Configured
- Chef Nodes Configured (x5 for our example)

Configure the Prometheus Server

1. First, we need to set up our Prometheus Server utilizing Alertmanager

   We will be using a pre-configured Alertmanager configuration file that looks like this:



We will use the above file to start the alertmanager using this command:

```
alertmanager --config.file=/share/alertmanager.yml &
```

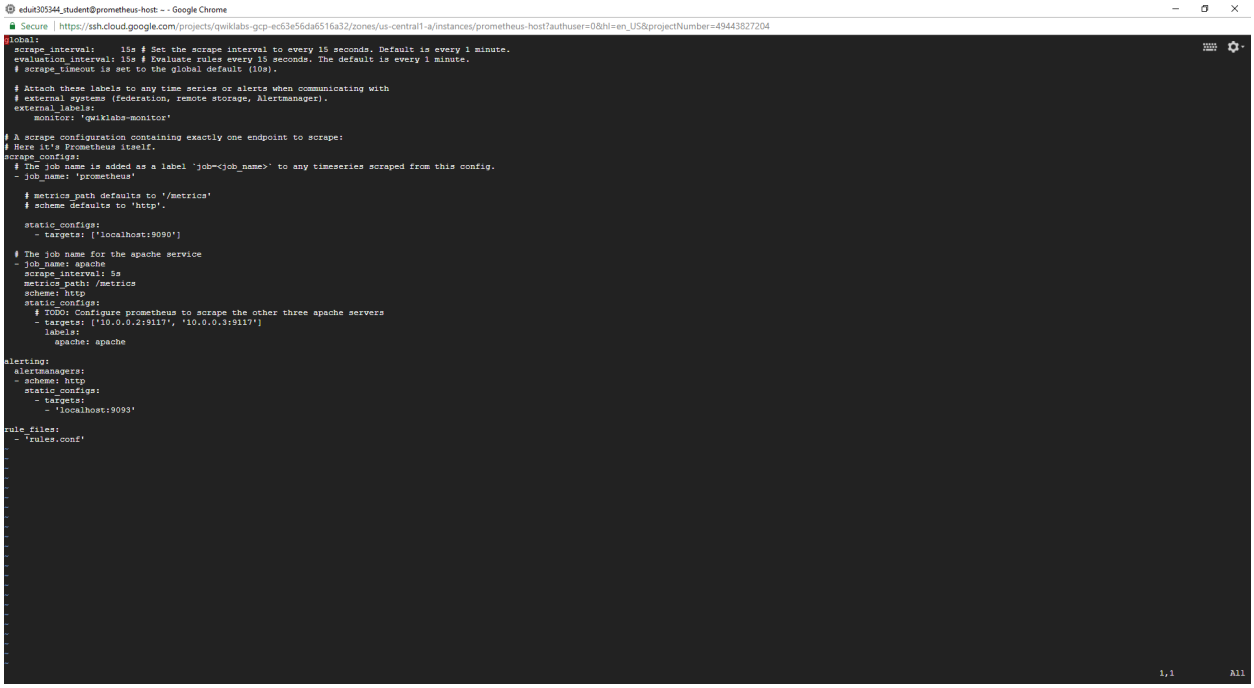
We passed the path to the configuration file and sent it to the background by using **&** so that we can continue to use the console.

To get back to the standard prompt after sending alertmanager to the background, just hit enter one more time.

Now let's start Prometheus using a pre-configured configuration file, tell it to use our local installation of Alertmanager and send it to the background.

```
prometheus --config.file=/share/prometheus.yml --alertmanager.url=http://localhost:9093 &
```

Here is what the prometheus.yml configuration file looks like:



With Prometheus and Alertmanager running, we can navigate to the Prometheus external IP running on port 9090 in a browser to see our alerts page.

```
EXTERNAL_IP_ADDRESS:9090
```

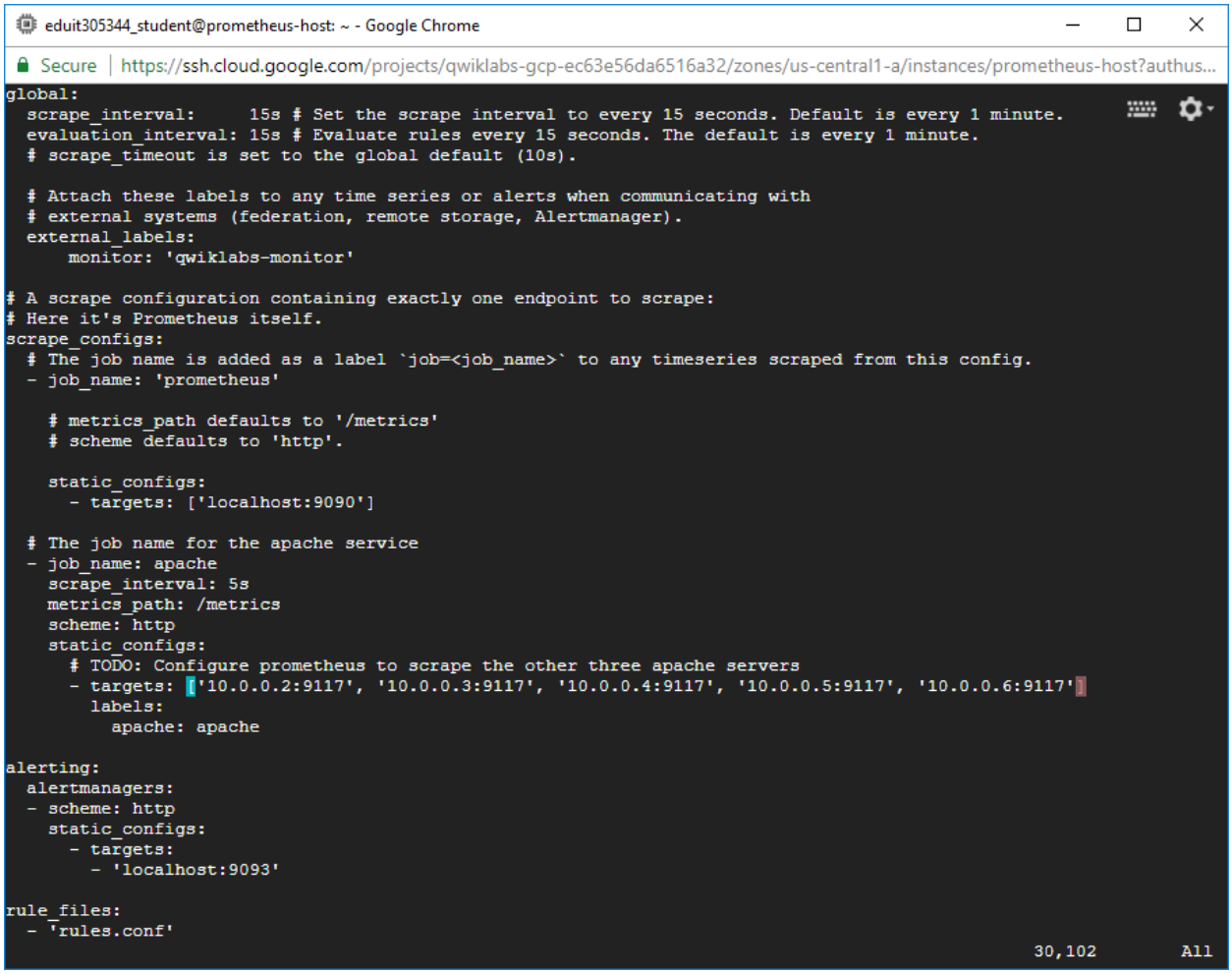
Oh no! We only have 2 nodes running out of the 5 we want to configure.

To add the 3 missing nodes, we will have to modify the prometheus.yml configuration file and add 3 more targets.

Open the prometheus.yml file in a text editor and add the target IPs as shown below.

```
- targets: ['10.0.0.2:9117', '10.0.0.3:9117', '10.0.0.4:9117', '10.0.0.5:9117', '10.0.0.6:9117']
```

This example used vim:



Now that we changed our prometheus.yml configuration file, we need to reload the file so that the changes are reflected in our Prometheus service.

To accomplish this, we can use this command in the console:

```
curl -X POST http://localhost:9090/-/reload
```

After allowing a few seconds for the change to be reflected, we can refresh our alerts page and see that 5 hosts are showing.

2. At this point, we can configure our Chef server to automatically deploy Apache2 to each node in the load-balanced cluster.

   We will use our `chef-workstation` to push the changes to Chef server.

   To get permissions and authenticate with the Chef server we need to get the `chefadmin.pem` key.

   Conveniently, a script to automate the process of acquiring the key has been provided. Just run `/share/get_chef_key.sh`

   To see what the script is doing, here is what it looks like:

edult305344_student@chef-workstation: /share - Google Chrome

Secure | https://ssh.cloud.google.com/projects/qwiklabs-gcp-ec63e56da6516a32/zones/us-central1-a/instances/chef-workstation?authus...

```
#!/bin/bash
#
# This script copies over the chefadmin.pem file to the localhost.

set -e

function metadata_value() {
  curl --retry 5 -sfH "Metadata-Flavor: Google" \
    "http://metadata.google.internal/computeMetadata/v1/$1"
}

function server_name() {
  metadata_value "instance/attributes/server-name"
}

function project_name() {
  metadata_value "project/project-id"
}

function copy_chef_key() {
  scp -o StrictHostKeyChecking=no -i /share/project_key "$(server_name).c.$(project_name).internal:/share/chefadmin
.pem" /share/.chef/chefadmin.pem
}

copy_chef_key || exit $?
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/share/get_chef_key.sh" 24L, 580C                                    1,1           All
```

SSL certificates need to be copied from an HTTPS server to our `trusted_certs_dir`.

The **knife** tool will help with this.

Run: `knife ssl fetch`

Then to verify this worked, run: `knife ssl check`

Now we are allowed to push changes to our Chef-server!

3. Let's upload a cookbook to our Chef-server so that we can deploy Apache2 to our cluster.

   The Apache2 cookbook we will upload contains 3 files.

   - A metadata file that looks like this:

eduit305344_student@chef-workstation: /share - Google Chrome

Secure | https://ssh.cloud.google.com/projects/qwiklabs-gcp-ec63e56da6516a32/zones/us-central1-a/instances/chef-workstation?authus...

```
name 'chef_apache2'
description 'Installs and configures apache2'
long_description 'Installs and configures apache2'
version '1.0'
/share/cookbooks/chef_apache2/metadata.rb (END)
```

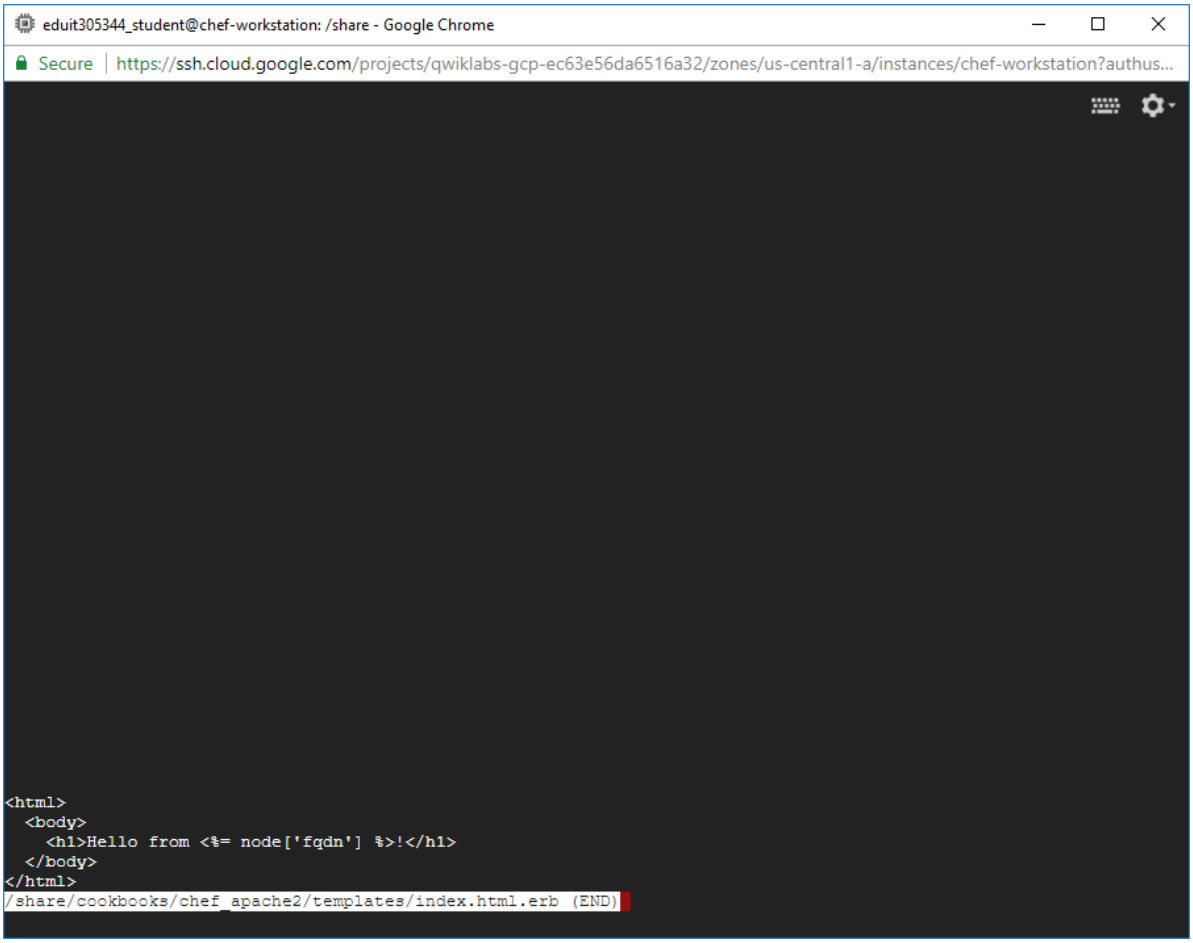
- The recipe itself:

eduit305344_student@chef-workstation: /share - Google Chrome

Secure | https://ssh.cloud.google.com/projects/qwiklabs-gcp-ec63e56da6516a32/zones/us-central1-a/instances/chef-workstation?authus...

```
apt_update 'Update the apt-cache daily' do
  frequency 86_400
  action :periodic
end

package 'apache2'

service 'apache2' do
  supports status: true
  action [:enable, :start]
end

template '/var/www/html/index.html' do
  source 'index.html.erb'
end
/share/cookbooks/chef_apache2/recipes/default.rb (END)
```

- And finally the index.html template:



We upload this cookbook using the knife tool by running:

`knife cookbook upload chef_apache2`

Then verify the upload succeeded: `knife cookbook list`

4. It's time to deploy the cookbook to all our nodes.

   Without this script, we'd have to run this process manually for each node.

   Thankfully, this process can be automated by utilizing a script that looks like this:

```
#!/bin/bash
#
# Automate the chef node bootstrap process.
_BLUE="\e[34m"
_RED="\e[31m"
_WHITE="\e[0m"
PROJECT_KEY="/share/project_key"
function error_message() {
  # Echo an error message to the screen
  # Args:
  #   $1: String, The message to echo
  if [[ "${1}" == "" ]]; then
    echo "error_message requires one argument"

  else
```

```
    echo -e "${_RED}[ERROR]${_WHITE} ${1}"
  fi
  exit 1
}
function info_message() {
  # Echo an error message to the screen
  # Args:
  #   $1: String, The message to echo
  if [[ "${1}" == "" ]]; then
    error_message "info_message requires one argument"
  else
    echo -e "${_BLUE}[INFO]${_WHITE} ${1}"
  fi
}
function usage() {
  printf "USAGE: $(basename "${0}") [options]
  This script is used to manage chef nodes. Please provide one of the available flags to
execute.
  Options:
  -b, --bootstrap    bootstrap the chef nodes
  -k, --kill         kill the apache2 services on the chef nodes
  -u, --update       update the chef nodes\n"
}

function validate_config() {
  # Ensure knife is installed
  info_message "Checking for knife..."
  if [[ $(knife --version) =~ 'Chef' ]]; then
    info_message "Checking for knife ssl..."
  else
    error_message "Knife is not properly installed, please check the chefdk installation"
  fi

  # Check the knife ssl configuration
  message=$(knife ssl check)
  if [[ $? -eq 0 ]]; then
    info_message "Checking for cookbooks..."
  else
    error_message "ssl configuration has an error: ${message}"
  fi

  # Check for a successful cookbook list
  if [[ $(knife cookbook list) =~ 'chef_apache2' ]]; then
    info_message "Checking for the private ssh key..."
  else
    error_message "Knife cannot find the cookbooks please ensure you have uploaded the
'chef_apache2' cookbook"
  fi

  # Check for the private key
  if [[ -f $PROJECT_KEY ]]; then
    info_message "All checks are complete."

  else
```

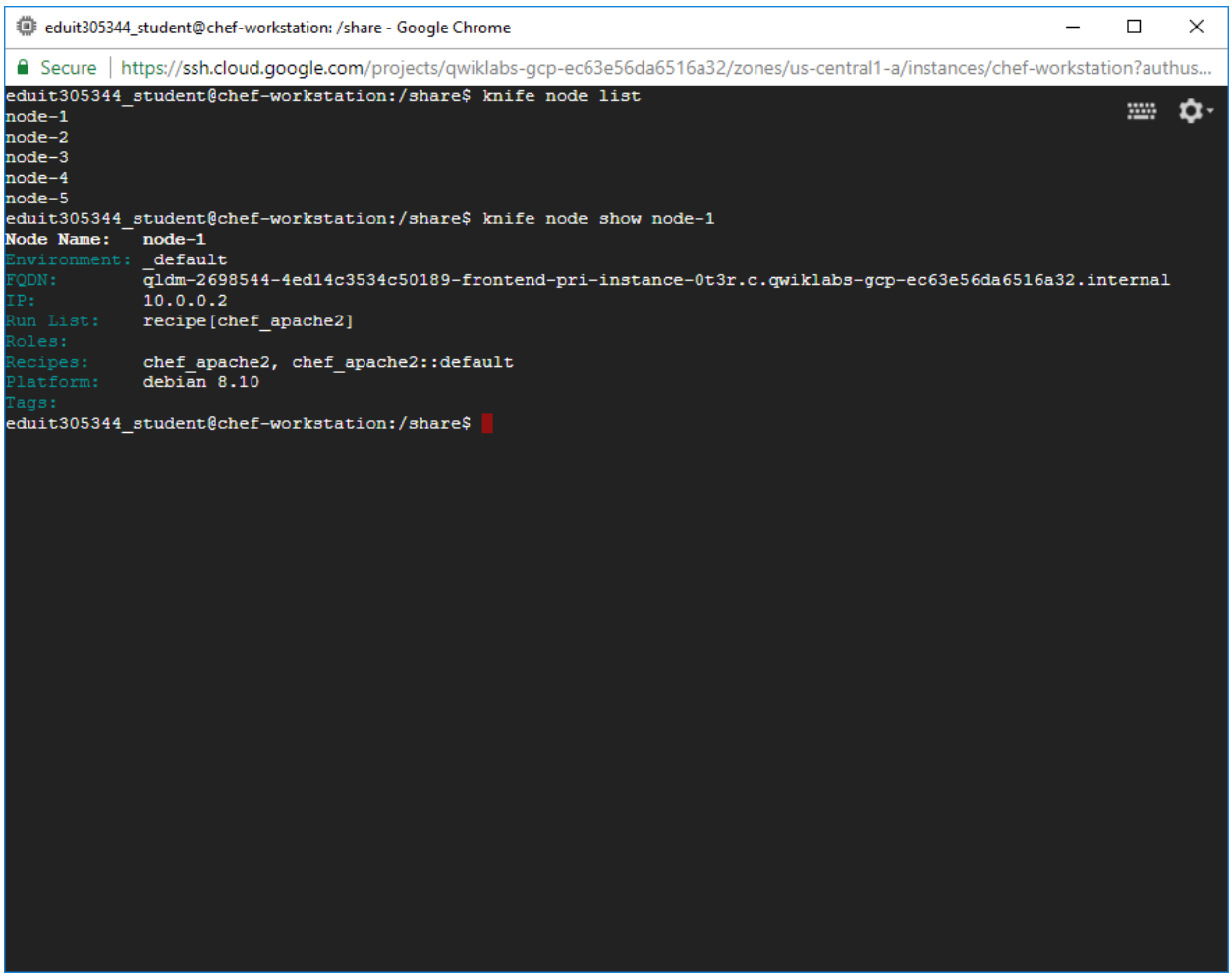
```
    error_message "The private key: ${PROJECT_KEY} does not exist"
  fi
}

function bootstrap() {
  validate_config

  info_message "Time to bootstrap!"

  for i in $(seq 1 5); do
    NODE_IP=$i
    (( NODE_IP += 1))
    info_message "Bootstrapping 10.0.0.${NODE_IP} as node-${i}"
    knife bootstrap 10.0.0.$NODE_IP --ssh-user $USER --sudo --identity-file $PROJECT_KEY --
node-name node-$i --run-:
list 'recipe[chef_apache2]'
  done
}

function update() {
  validate_config

  info_message "Time to update!"

  for i in $(seq 1 5); do
    info_message "Updating node-${i}"
    knife ssh name:node-$i sudo chef-client --ssh-user $USER --identity-file $PROJECT_KEY -
-attribute ipaddress
  done
}

function kill_apache() {
  for i in $(seq 1 5); do
    NODE_IP=$i
    (( NODE_IP += 1 ))
    info_message "Killing apache2 on node-${i}"
    ssh -i $PROJECT_KEY 10.0.0.$NODE_IP -t 'sudo pkill apache2; exit'
  done
}

if [[ $# -gt 0 ]]; then
  case $1 in
    -b|--bootstrap)
      bootstrap;
      exit 0
      ;;
    -k|--kill)
      kill_apache;
      exit 0
      ;;
    -u|--update)
      update;

      exit 0
```

```
      ;;
    *)
      usage;
      exit 1
      ;;
  esac
else
  usage
  exit 1
fi
```

Deploy utilizing the above script with this command: `/share/manage_nodes.sh --bootstrap`

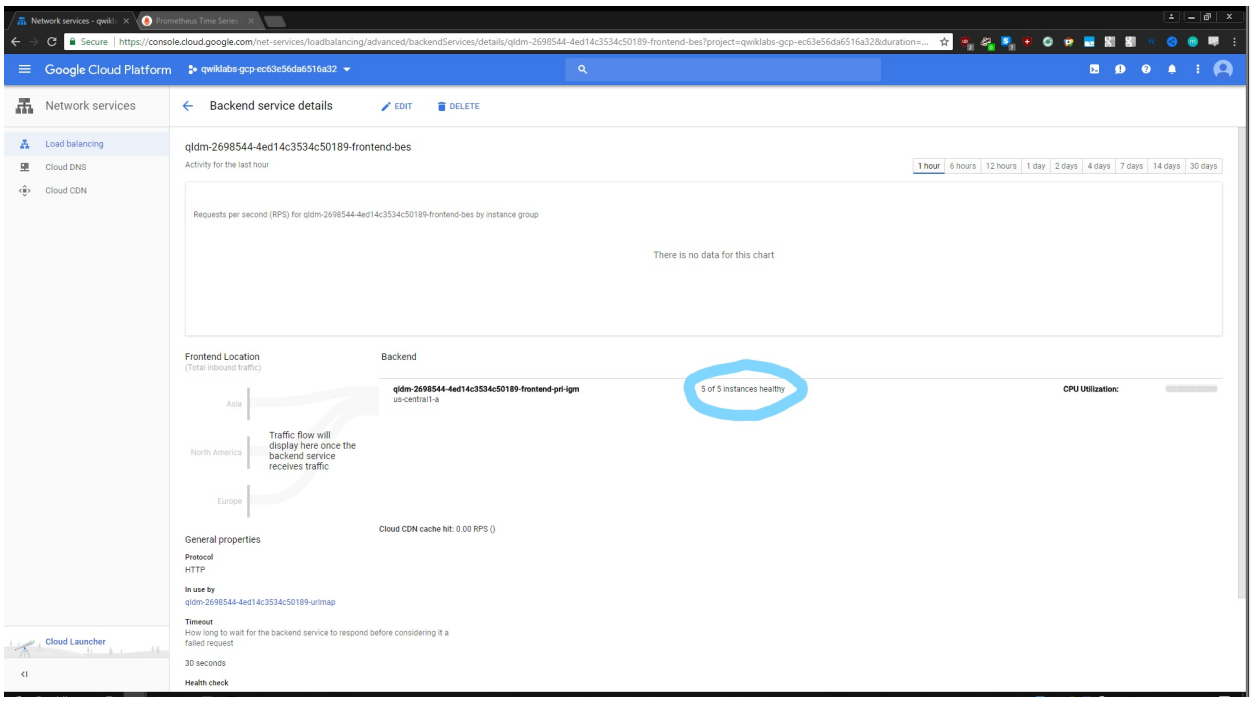
Once the script completes, verify everything is good to go by using, once again, the knife tool:

```
knife node list
knife node show node-1
```



5. Our nodes should be running our Apache2 configuration.

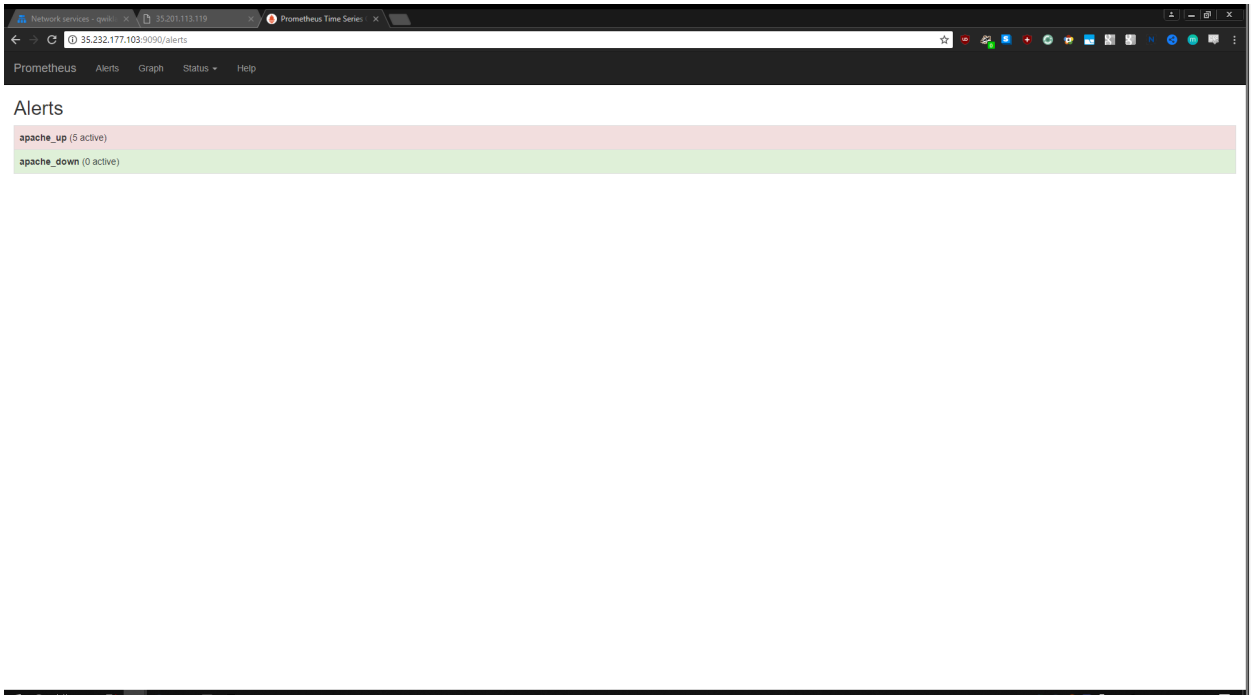
   Let's check by viewing our load balancing:



All 5 instances are healthy.

We can check our Apache2 servers are displaying correctly by visiting the global external IP address:

Confirm our alerts are working by revisiting the alerts page of our Prometheus server:

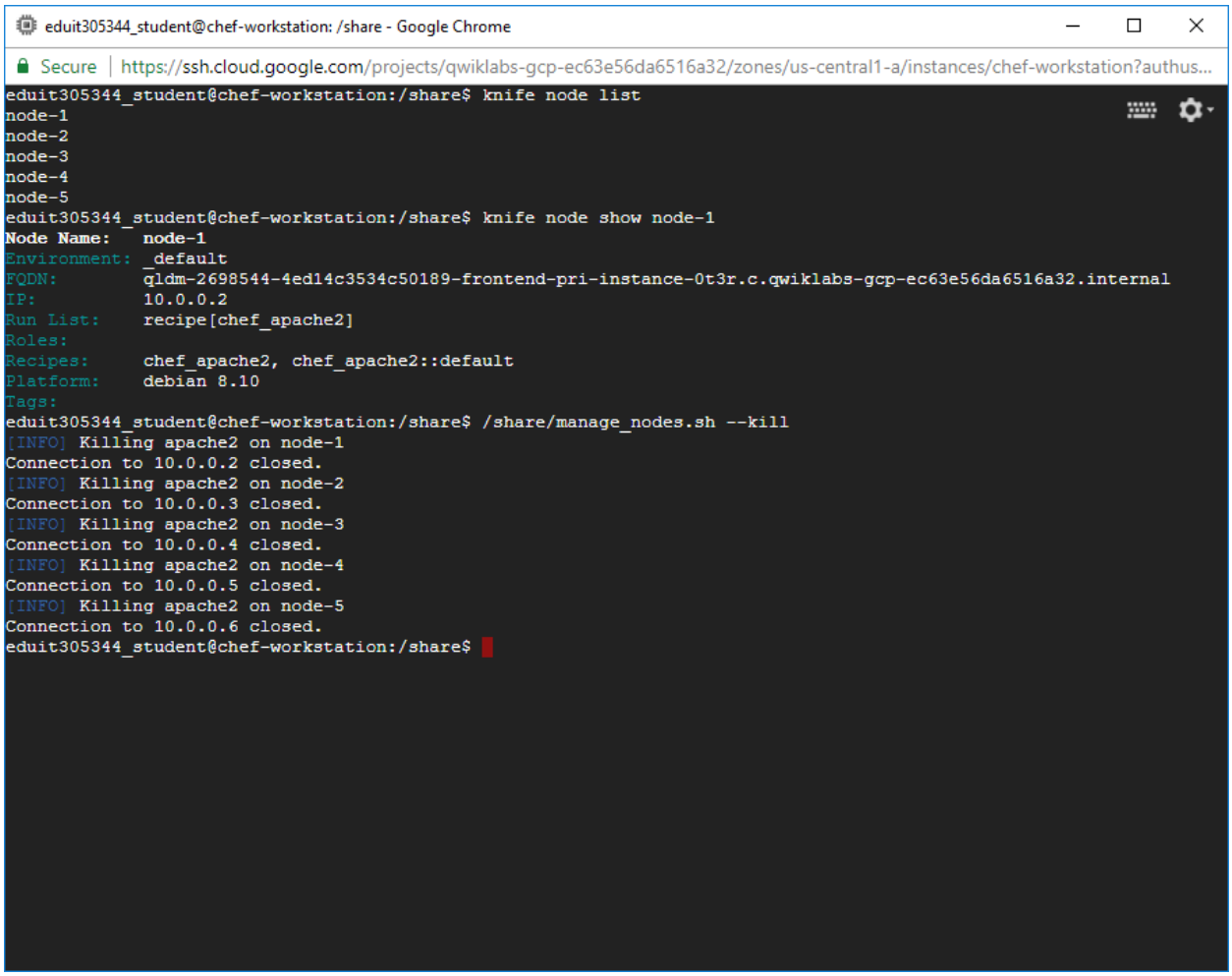


Great! Everything is working properly.

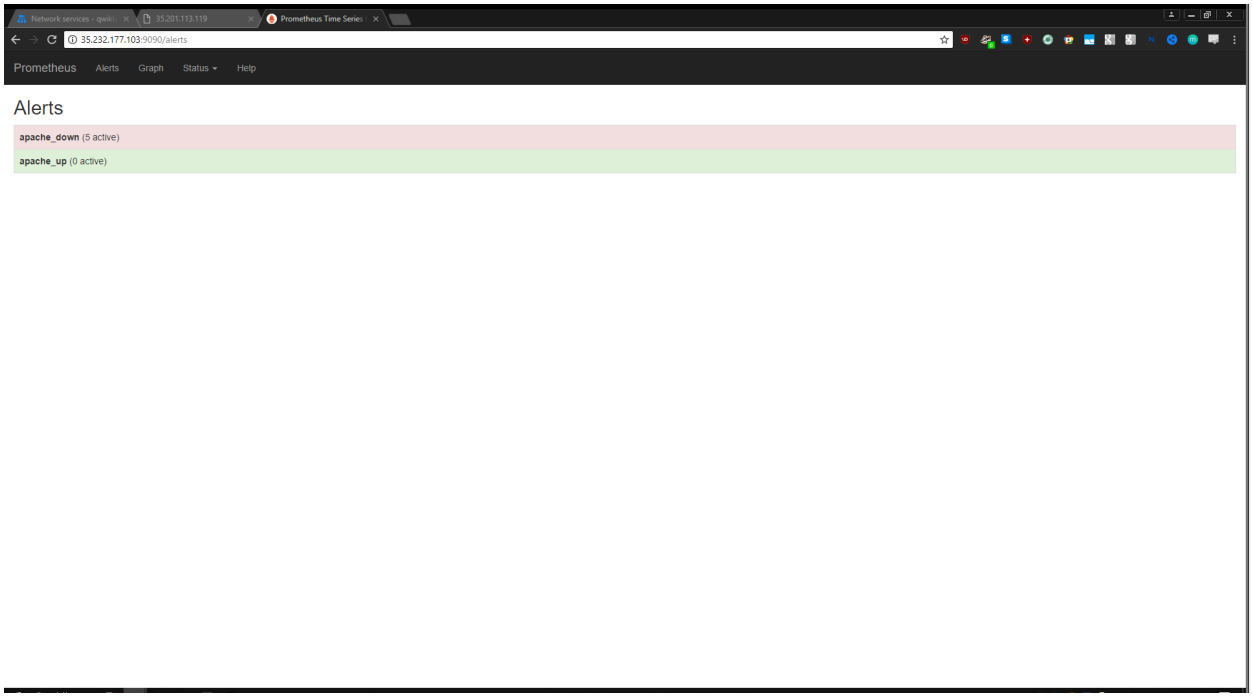
6. Let's take down apache and see the changes on our alerts page

   We can use the manage_nodes.sh script again to do this by passing in the flag --kill

   `/share/manage_nodes.sh --kill`



Refresh the alerts page and you should see:



If this were to occur in production, a PAGE alert would probably be appropriate.

Conclusion

We have successfully:

- Configured monitoring and alerting on Prometheus for a simple load balanced website by editing .yml configuration files
- Configured a Chef server by authenticating with the chefadmin.pem key, configured our SSL certificates, and utilized the **knife** tool to push changes
- Automatically deployed Apache2 to multiple servers utilizing a cookbook and script to use the cookbook
- Monitored the Apache2 service across the cluster
- Configured alerts for the Apache2 service
- Simulated apache going down