NWC 2011

Monitoring a Cloud Infrastructure in a Multi-Region Topology

Nicolas Brousse

nicolas@tubemogul.com

September 29th 2011

Nagios® World Conference North America

Introduction - About the speaker

- My name is Nicolas Brousse
- I previously worked for many industry leading company in France
  - From Web Hosting to Online Video services (Lycos, MultiMania, Kewego, MediaPlazza...)
  - Heavy traffic environment and large user databases
- I work as a Lead Operations Engineer at TubeMogul.com since 2008
- I help TubeMogul to scale its infrastructure
  - From 20 servers to +500 servers
  - Using 4 Amazon EC2 Regions + 1 Colo
  - Monitoring with Nagios over 6,000 actives services and 1,000 passives services
  - Collecting over 80,000 metrics with Ganglia
  - Managing over 300 TB of data in Hadoop HDFS
  - Billions HTTP queries a day
- Occasionally contribute to OpenSource projects
  - Ganglia (PHP and PERL module)
  - PHP Judy

Introduction - About TubeMogul

- Created in November 2006 by John Hughes and Brett Wilson
- Formerly a video distribution and analytics platform
- Acquire Illuminex - a flash analytics firm - in October 2008
- New platform call PlayTime™ :
  - TubeMogul is a Video Marketing Company
  - Built for Branding
  - Integrate real-time media buying, ad serving, targeting, optimization and brand measurement

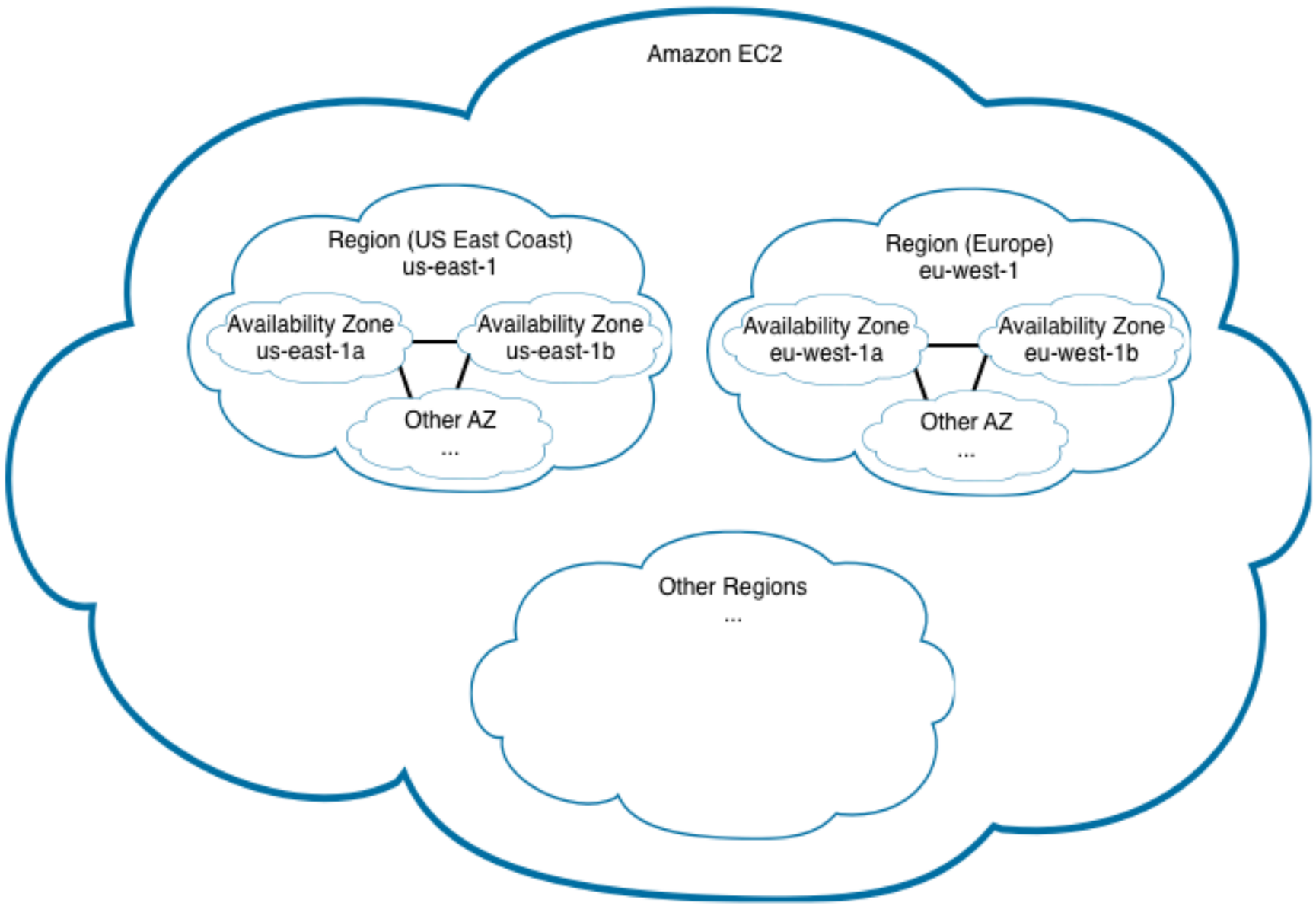
TubeMogul simplifies the delivery of video ads and maximizes the impact of every dollar spent by brand marketers

http://www.tubemogul.com/company/about_us

Our Environment

- +10 servers hosted at LiquidWeb
- Few VPS on Linode
- +500 instances on Amazon EC2
  - Over 50 different servers configurations
- Our technology stack :
  - JAVA, PHP
  - Hadoop : HDFS, MapReduce, HBase, Hive
  - Membase
  - Memcache
  - MySQL
  - And more...
- Monitoring with Nagios
  - Using NSCA when possible
- Graphing and Trending using Ganglia with Python plugins
  - Some legacy servers using Munin
- Configuration Management using Puppet

Amazon Clound Environment



Amazon Clound Environment

- We like it because....
  - We can quickly start new servers/clusters
  - We can quickly start new servers/clusters **in many regions**
    - US East (Virginia)
    - US West (North California)
    - Europe (Dublin)
    - Asia Pacific (Tokyo & Singapore)
  - We can use different type of instances (RAM, CPU, Disks, etc.)
  - It's easy to automate with EC2 API
  - It's easy to plug to a configuration management tool

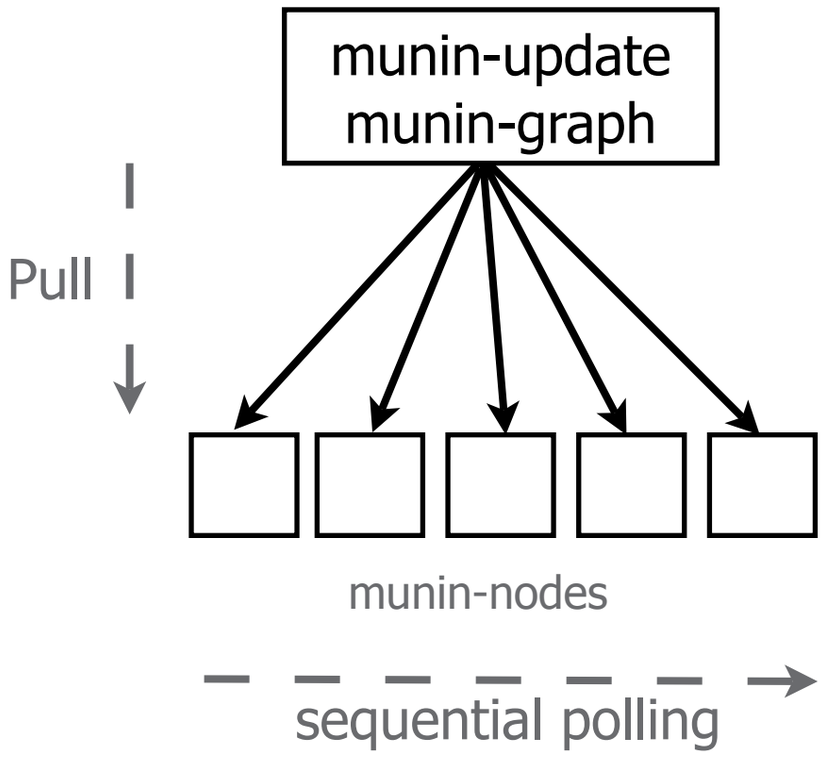
- But...
  - It can be hard to troubleshoot some failures or network problems
  - Occasionally being notified of hardware failures after the facts
  - No Multicast (Though, possible with Amazon VPC)
  - Bandwidth cost between regions can get expensive

What's the plan ?

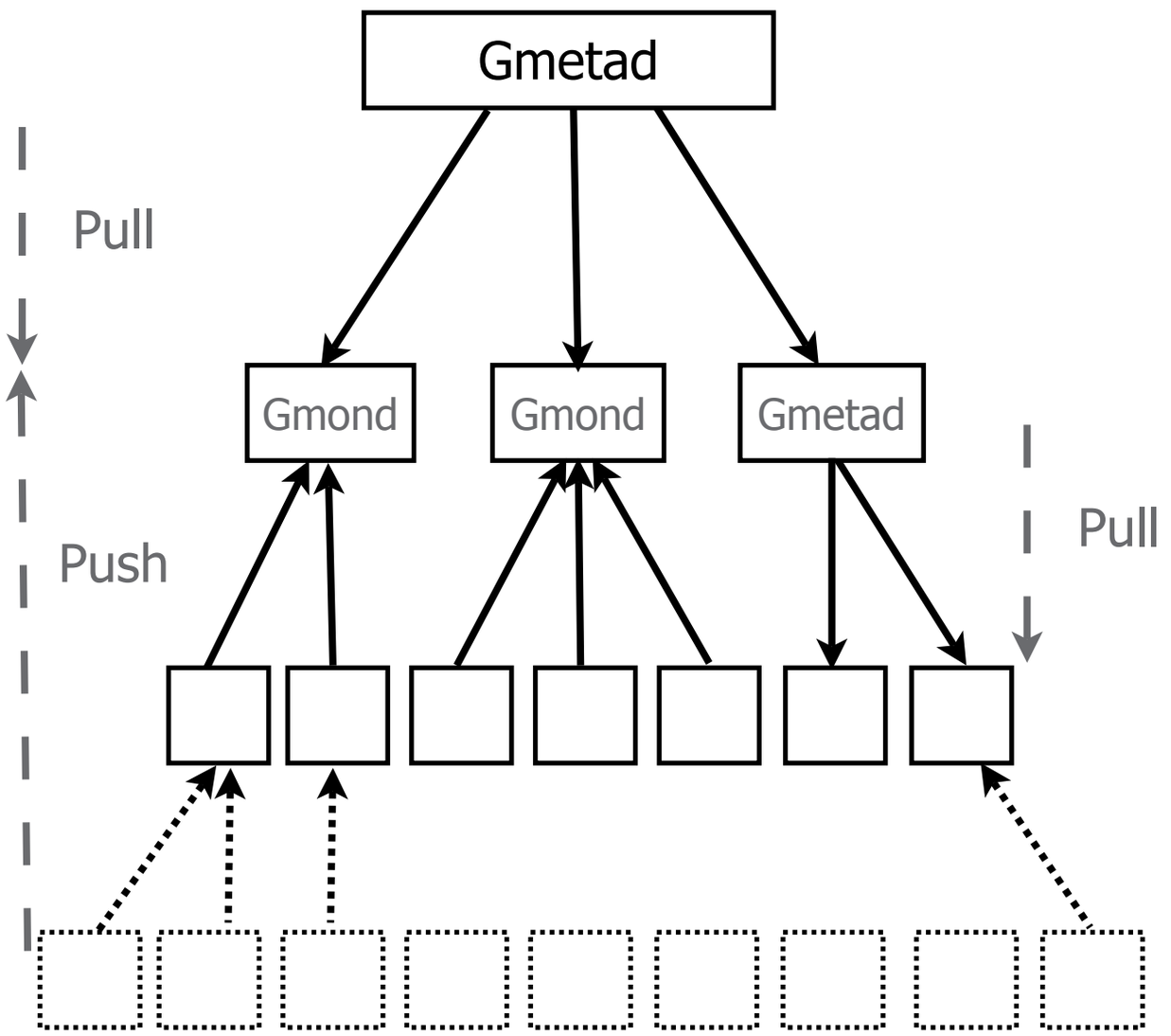
- Our monitoring must be able to scale
- We need a better Graphing/Trending solution
- Our monitoring configuration must be automated
  - How to monitor a cluster of servers with variables number of servers every hours ?
  - How to change configuration in multiple regions without missing something ?
- A failure in one region shouldn't impact other regions
- We want to be wake-up only when it really matter
- We have limited resources
  - Can't spend big bucks for monitoring
  - Small operation team

Graphing, Trending...

Munin



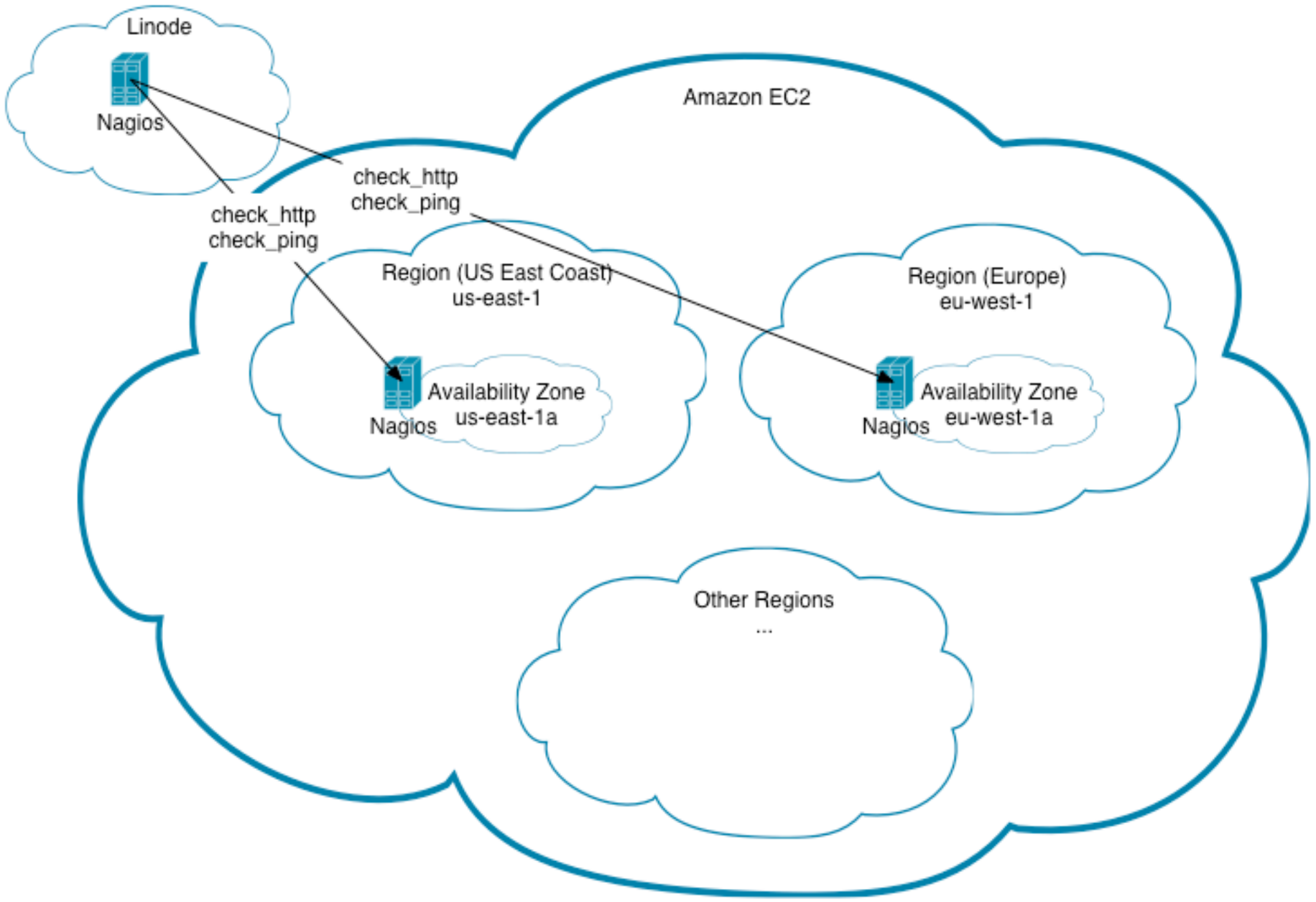
Ganglia



Graphing, Trending...

- Why we switched from Munin to Ganglia ?
  - Pretty much : Pull vs Push
    - Munin server fetch data from Munin Clients (munin-nodes)
      - Can quickly overload the Munin server in disk I/O and CPU
      - Data collected in sequential order impacted by previous run time and server load
    - Ganglia Client send data to representative clusters nodes. Data get federated periodically by a Gmetad process.
      - Lighter on the aggregation side
      - Clients push data at defined interval
      - Can use threshold to send data only when it make sense
        - » using **time_threshold** and **value_threshold** in the metric
  - Ganglia is designed for Clusters and Grids
    - You can use multiple layer of gmond/gmetad process
    - You don’t need to manually add servers to your configuration

Monitoring with Nagios



Automating Nagios configuration

- Puppet will configure our monitoring instance in each Region
  - We use Nagios regex : **use_regexp_matching=1**
  - But we don’t use true regex : **use_true_regexp_matching=0**
  - We use NSCA with Upstart

```
# Nagios NSCA

description     "Nagios NSCA Daemon"

start on network
stop on runlevel [!2345]

respawn
respawn limit 10 5

exec /opt/nagios/bin/nsca -c /opt/nagios/etc/nsca.cfg --daemon
```

  - We don’t use the perfdata
  - We includes our configurations from 3 directories
    - objects => templates, contacts, commands, event_handlers
    - servers => contain a configuration file for each server
    - clusters => contain a configuration file for each cluster

```
# OBJECT CONFIGURATION FILE(S)
cfg_dir=/opt/nagios/etc/objects
cfg_dir=/opt/nagios/etc/servers
cfg_dir=/opt/nagios/etc/clusters
```

Automating Nagios configuration

Process of event when starting a new host and add it to our monitoring:

1. We start a new instance using Cerveza and Cloud-init

2. Puppet configure Gmond on the instance

3. Our monitoring server running Gmetad get data from the new instance

4. A Nagios check run every minute and look for new hosts in Ganglia

5. If a new host is found, the check script rebuild the Nagios config and reload Nagios

6. If the config is corrupt, the check script will send a critical alert

Automating Nagios configuration

- Each server configuration is generated from a template
- Our nagios plugin “check_tm_clusters”, goes over the RRD files generated by Ganglia
- If a new host is found, it simply copy the template to the servers config directory and replace the variables as reported by Ganglia and looking at DNS entries

```
##############################################################################
##############################################################################
#
# HOST DEFINITION - Config managed via puppet - DO NOT CHANGE MANUALLY !!
#
##############################################################################
##############################################################################

# Define a host for the local machine

define host{
        use                     linux-server
        host_name               #HOSTNAME#
        hostgroups              #CLUSTERNAME#
        alias                   #FQDN#
        address                 #IP#
        _DNSVAL                 #IP#
        display_name            #CLUSTERNAME# #HOSTNAME#
        _PAGING                 yes
        notes                   #HOSTNAME# is part of #CLUSTERNAME#. Health check using ping.
        }


##############################################################################
##############################################################################
#
# SERVICE DEFINITIONS
#
##############################################################################
##############################################################################

define service{
        use                     generic-service
        host_name               #HOSTNAME#
        servicegroups           network-status
        service_description     INTERNAL DNS
        max_check_attempts      3
        check_command           check_dns_value!#FQDN#!$_HOSTDNSVAL$
        }
```

Reducing noise and false positive

- We disable most notification and only care of a cluster status

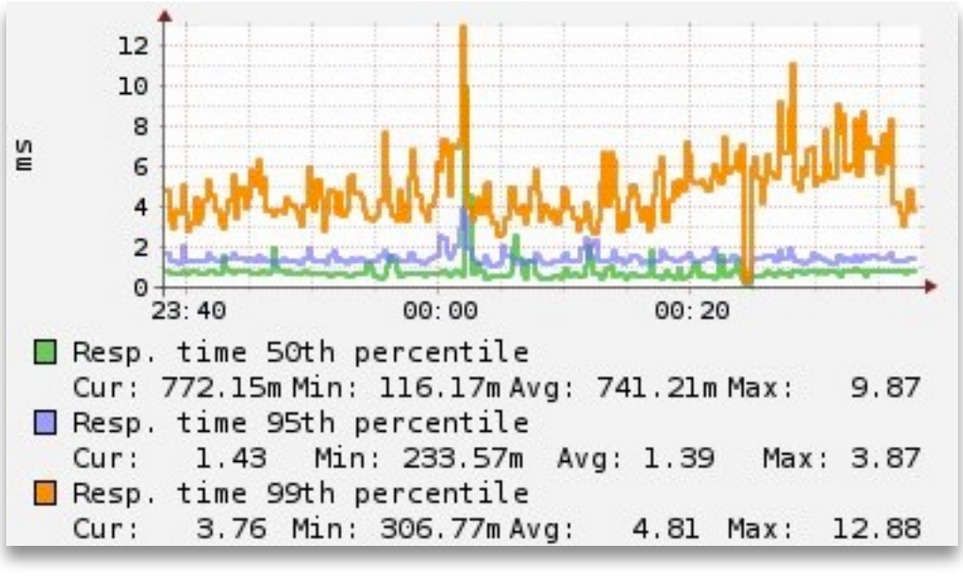
```
define service{
        use                             local-service
        hostgroup_name                  ^[a-z0-9_-]+-cluster
        service_description             PING
        servicegroups                   network-status
        check_command                   check_ping!100.0,20%!500.0,60%
        max_check_attempts              10
        notifications_enabled           0
        }
define service{
        use                             generic-service
        host_name                       <%= hostname %>
        servicegroups                   cluster-service
        service_description             Cluster - PING
        check_command                   check_cluster_service!^.+!^PING$$
        contact_groups                  noc
        }
```

- Most of our checks are based on Ganglia RRD files

```
define service{
        use                             generic-service
        hostgroup_name                  ^[a-z0-9_-]+-cluster
        servicegroups                   system-status
        service_description             mnt disk used
        check_command                   check_rrd!$USER3$/$HOSTGROUPALIAS$/$HOSTALIAS$/mnt-disk_used.rrd!80!90
        notifications_enabled           0
        }
define service{
        use                             generic-service
        host_name                       <%= hostname %>
        servicegroups                   cluster-service
        service_description             Cluster - mnt disk used
        check_command                   check_cluster_service!^.+!^mnt disk used$$
        contact_groups                  noc
        }
```

Reducing noise and false positive

- It become really easy to monitor any metrics returned by Ganglia




```
define service{
        use                             generic-service
        hostgroup_name                  [redacted]-<%= ec2_placement_availability_zone %>-cluster
        servicegroups                   services-status
        service_description             [redacted]-http-response-time
        display_name                    Check [redacted] HTTP response time
        check_command                   check_rrd!$USER3$/$HOSTGROUPALIAS$/$HOSTALIAS$/http_valid_99th_percentile_response_time.rrd!90000!120000
        contact_groups                  [redacted]
        notifications_enabled           0
        }
define service{
        use                             generic-service
        host_name                       [redacted]
        servicegroups                   cluster-service
        service_description             cluster-[redacted]-http-response-time
        display_name                    Cluster - Check [redacted] HTTP response time
        check_command                   check_cluster_service!^[redacted][0-9]+!^[redacted]-http-response-time$$!--warning=1 --critical=30
        contact_groups                  [redacted]
        }
```

Reducing noise and false positive

- We can check cluster status by hosts/services but also per returned messages !

```
# nagios status file check for hosts
define command{
    command_name    check_cluster_service
    command_line    $USER1$/check_nagios_status --host-regex=$ARG1$ --service-regex=$ARG2$ $ARG3$
}

# nagios status file check for services msg
define command{
    command_name    check_cluster_service_msg
    command_line    $USER1$/check_nagios_status_msg --host-regex=$ARG1$ --service-regex=$ARG2$ --msg-filter=$ARG3$ $ARG4$
}
```

```
Usage: check_nagios_status [options]

Options:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose logging. (default: False)
  --status-file=STATUS_FILE
                        Path to the Nagios status file. (default:
                        /opt/nagios/var/status.dat)
  --host-regex=HOST_REGEX
                        Regex used to filter host name.
  --service-regex=SERVICE_REGEX
                        Regex used to filter service description. (default:
                        none)
  -w WARNING, --warning=WARNING
                        Warning threshold in percent. (default: 30)
  -c CRITICAL, --critical=CRITICAL
                        Critical threshold in percent. (default: 60)
  -u UNKNOWN, --unknown=UNKNOWN
                        Unknown threshold in percent. (default: none)
```

```
Usage: check_nagios_status_msg [options]

Options:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose logging. (default: False)
  --status-file=STATUS_FILE
                        Path to the Nagios status file. (default:
                        /opt/nagios/var/status.dat)
  --host-regex=HOST_REGEX
                        Regex used to filter host name.
  --service-regex=SERVICE_REGEX
                        Regex used to filter service description. (default:
                        none)
  -w WARNING, --warning=WARNING
                        Warning threshold in percent. (default: 30)
  -c CRITICAL, --critical=CRITICAL
                        Critical threshold in percent. (default: 60)
  --msg-filter=MSG_FILTER
                        Regex used to filter plugin output and mark it as
                        error.
```

Reducing noise and false positive

- We extensively use our “check_cluster” plugin
- We limit as much as possible email notification
- We use a custom variable _PAGING to identify pageable services
- Paging **ONLY** on **Critical** alerts for services/hosts with **_PAGING=yes**
- Use different contacts and time periods to send alerts to the **right** person
- We use Nagios Checker for FireFox and Chrome

Host	Host (alias)	Service	Flags	Attempt	Status	Duration	Information
TM EC2 US-EAST-1B #1							
[redacted]hive-master01	-	[redacted]hdfs-usage		3/3	warning	1h 47m 47s	WARNING: DFSUsed Warning (90.11 %),
TM EC2 US-WEST-1A							
[redacted]hive-master01	-	[redacted]hdfs-replica		3/3	warning	4d 6h 31m 2s	WARNING: 34 under-replicated blocks.
TM EC2 AP-SOUTHEAST-1A							
[redacted]hive-master01	-	[redacted]hdfs-replica		3/3	warning	6d 9h 52m 13s	WARNING: 11 under-replicated blocks.
TM EC2 EU-WEST-1A							
[redacted]hive-master01	-	[redacted]hdfs-replica		3/3	warning	16d 9h 28m 59s	WARNING: 18 under-replicated blocks.
TM EC2 US-EAST-1B #2							
[redacted]hbase-master01	-	[redacted]hbase-cluster [redacted]ealth		3/3	warning	6d 21h 15m 22s	WARNING: Node count for [redacted]hbase-

YSlow | 16 unknown services | 5 service warnings | 8 critical services | S3Fox

Thank You...

TubeMogul is Hiring !

http://www.tubemogul.com/company/careers
jobs@tubemogul.com

Follow us on Twitter

@tubemogul

@orieg