Deploying Nagios in a Large Enterprise

Carson Gaspar
Goldman Sachs
carson.gaspar@gs.com

or...

If you strap enough rockets to a brick, you can make it fly

In the beginning...

- New Linux HPC skunkworks project
- Catastrophic success
- Need monitoring added yesterday

Looking for Solutions

- Why not use what we already had?
  - Stability problems
  - Resource utilization problems
  - Custom agents were hard
  - No support for our new Linux platform

Why Nagios?

- Purchasing a new commercial solution was politically difficult
- At the time (2003) nagios was the most mature of the open source solutions
- Good community support

The Naive Approach (and why it didn't work)

- Performance Problems
- Configuration Management Problems
- Availability Problems
- Integration / Automation Requirements

Performance Problems

- State check performance
  - Active checks: ~3 checks / second maximum
- Statistics performance
  - fork()/exec() for every sample
- Web UI performance
  - Large configurations take a long time to display (much improved in 2.x)

Configuration Management Problems

- Configuration files are very verbose (even with templates)
- Syntax errors are easy
- Keeping up with a high churn rate in monitored servers is expensive

Availability Problems

- Hardware / software failures
- Building power-downs
- Patches / upgrades
- Who watches the watchers?

Integration / Automation Requirements

- Alarms need to be dispatched to our existing alerting and escalation system
- Alarms need to be suppressed by existing maintenance tools
- Provisioning should flow from our existing provisioning system

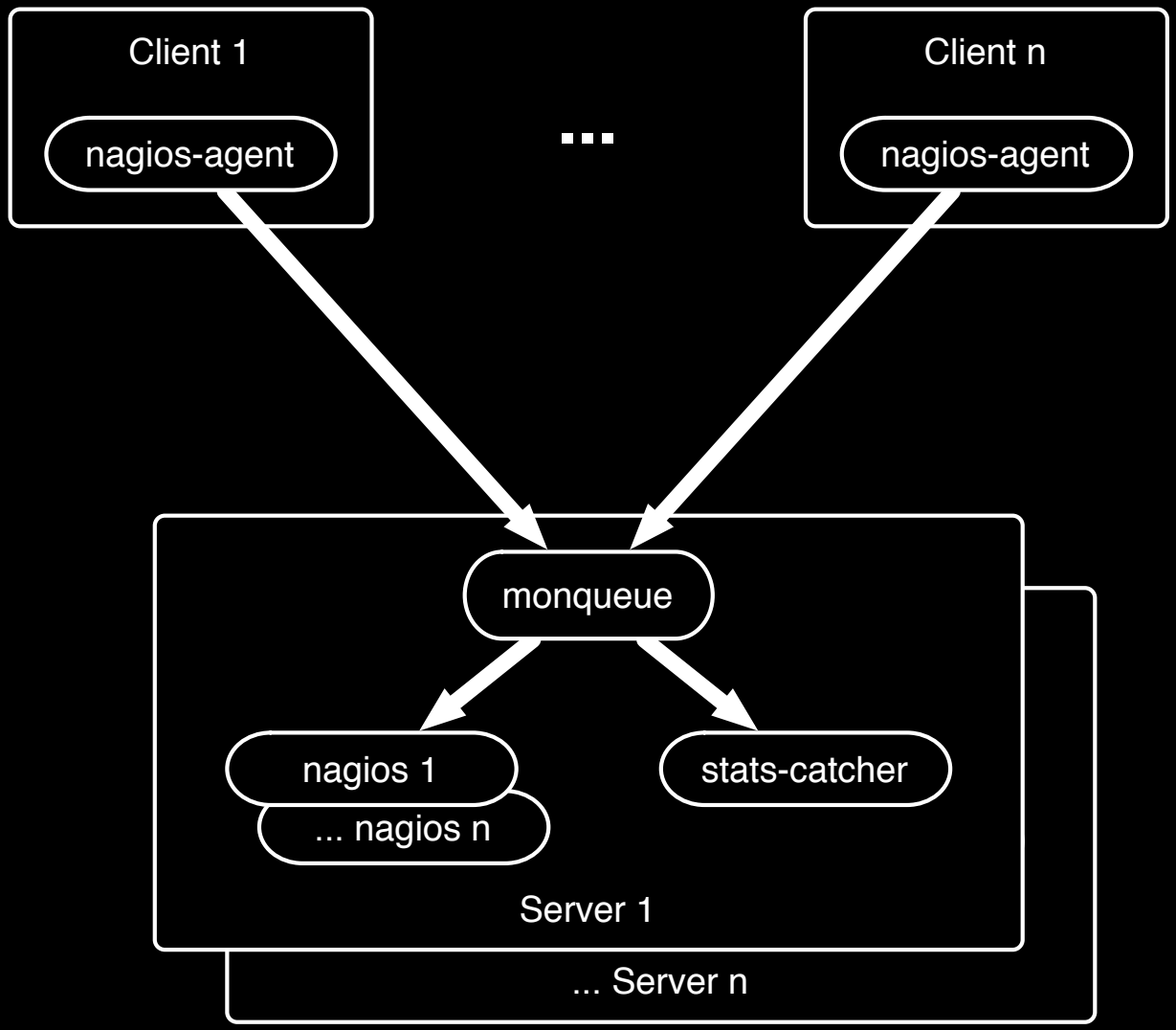
Solving the Problems

- Move to passive checks
- Run multiple nagios instances
- Deploy HA nagios servers
- Use data-driven configuration file generation
- Create a custom notification back end

Passive Checks

- Move most of the work to the clients
- Batch server updates unless a state change occurs
- Randomize server update times to avoid spikes
- Queue the results on the server
- Send statistics to a different back end

Passive Data Flow



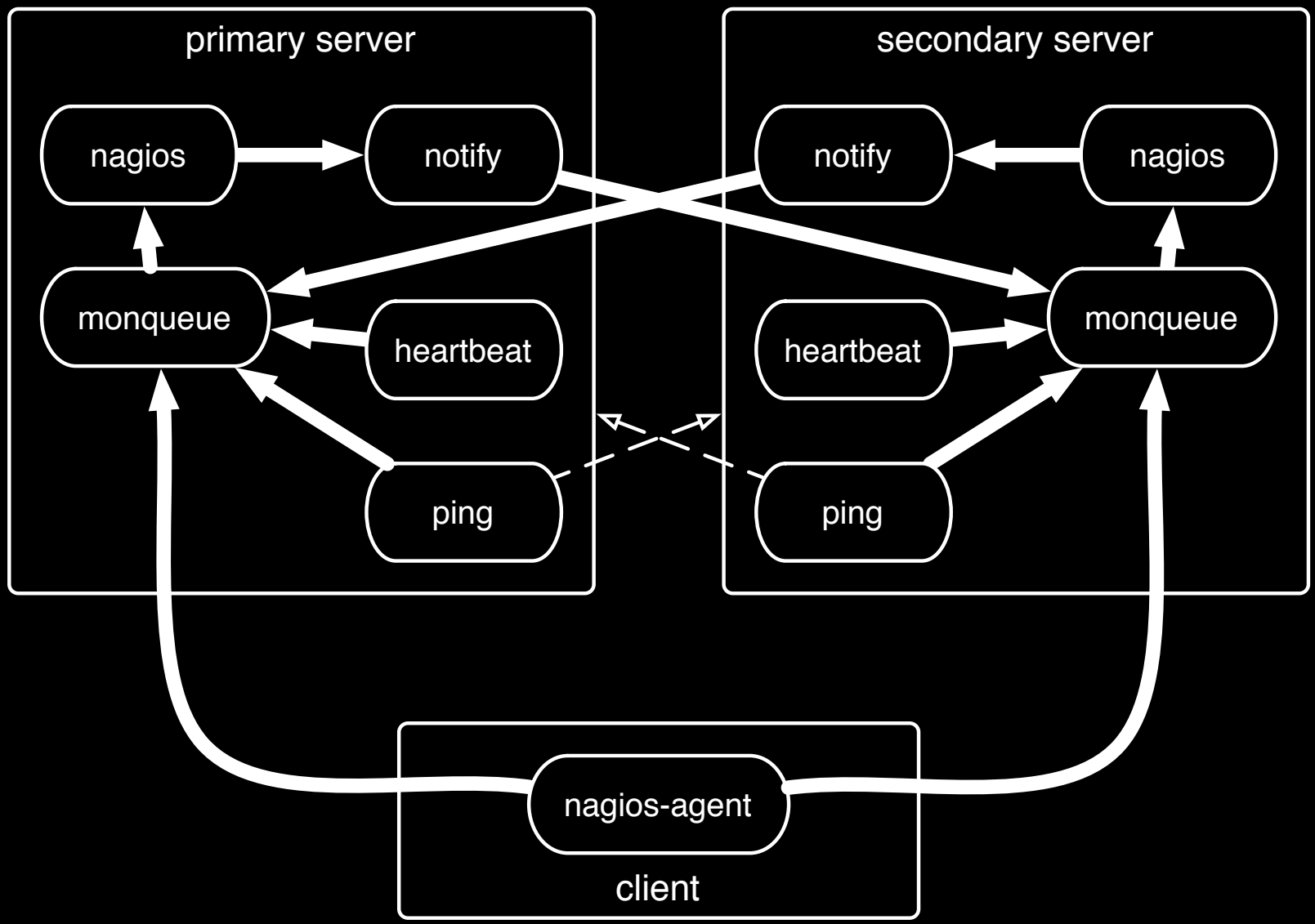
Multiple Nagios Instances

- Run many copies of nagios on one server
- Improve web UI performance
- Show each group only their own servers, so the top level dashboard is more useful
- Allow per-group customizations

HA Nagios Servers

- Run multiple nagios servers on different machines in different buildings
- All clients update all servers
- A heartbeat is published through each server to its counterpart
- Notifications are suppressed on slaves if the heartbeat service is OK
- Partitioned masters can cause duplicate alerts

HA Data Flow


primary server
secondary server
nagios
notify
notify
nagios
monqueue
heartbeat
heartbeat
monqueue
ping
ping
nagios-agent
client

Data-driven Configuration File Generation

- Leverage our existing host database and provisioning tools
- Generate client and server configurations via cfengine based on templates and database lookups
- Mostly driven by database data, with some per-server threshold overrides managed in cfengine master files

Custom Notification Back End

- Custom code integrates with our Netcool infrastructure
- Alarm suppression based on external criteria
- Also supports email alerts, optionally batched

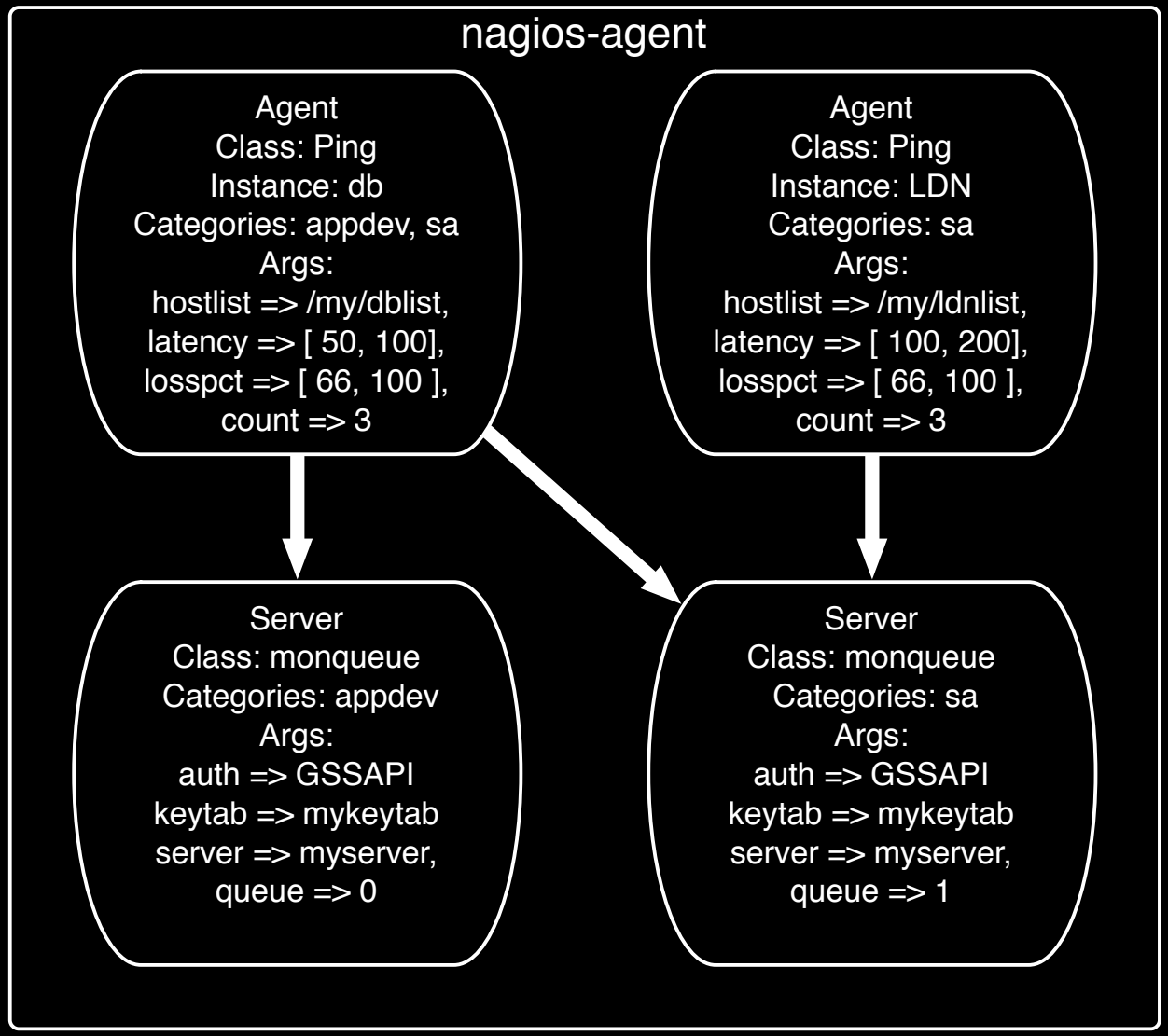
Design Trade-offs

- Batch updates mean slow detection of "zombie" hosts (ping-able, but not running user processes)
- nagios' notification escalation doesn't work well without active checks, especially if updates are batched
- Requires configuration management
- More complexity = more bugs

nagios-agent Design Criteria

- Lightweight
- Easy to write and deploy additional agents
- Avoid fork()/exec() where possible
- Support agent callbacks to avoid blocking
- Support “proxy” agents to monitor other devices where we can’t deploy nagios-agent
- Evaluate all thresholds locally and batch server updates

nagios-agent



monqueue Design Criteria

- Fast
- Secure
- Accept data from clients, and dispatch to multiple output queues
- Supply heartbeats to nagios
- Supply queue depth stats to stats-catcher

Client Evolution

- nagios-agent slowly grew features as they became required
  - multiple agent instances
  - agent instance to server mapping
  - auto reload of configuration, modules on update
  - auto re-exec of nagios-agent on update
  - stats collection
  - SASL authentication to monqueue

Server Evolution

- Started as one monolithic instance
- As deployment spread, split into multiple instances based on administrative domain
- added HA
- added SASL authentication and authorization
- added monitoring of monqueue itself, and service dependencies (so a monqueue failure didn't trigger alarms for all services)

Kudzu

- Originally for one project, fewer than 200 hosts
- Eventually used for large sections of the environment
- Documentation and internal consultancy are critical for user acceptance

One of Our Servers

- A single HP DL385 G1 (dual 2.6GHz Opteron, 4GB RAM), running RHEL4 U4, nagios 2.9
- 11 nagios instances
- 27,000+ services (mostly 2 minute intervals)
- 6,600+ hosts
- ~10% CPU
- ~500 MB RAM

Still to Come

- Source code release
- Encryption of nagios-agent / monqueue communications
- Support for “pulling” status from nagios-agent to better support DMZ environments
- Statistical analysis of multiple data samples to determine service status
- Yet more agent plugins
- nagios-agent support for traditional nagios plugins

Deploying Nagios in a Large Enterprise

Carson Gaspar
Goldman Sachs
carson.gaspar@gs.com