Scale CI
from 20K to 140K builds per day

Alexander Akbashev
VelocityConf, London | Oct 18-20, 2017
Agenda

00. Intro
01. The history of CI
02. Building Common CI
03. Reality strikes back...
04. How to scale executions?
05. How Jenkins tends to die?
06. How to scale the rest?
07. Final check-list
Here Technologies

HERE, the Open Location Platform company, enables people, enterprises and cities to harness the power of location. By making sense of the world through the lens of location we empower our customers to achieve better outcomes – from helping a city manage its infrastructure or an enterprise optimize its assets to guiding drivers to their destination safely.

To learn more about HERE, including our new generation of cloud-based location platform services, visit http://360.here.com and www.here.com
The history of CI
The history of CI
The history of CI
The history of CI
The history of CI
The history of CI
The history of CI

Team A
The history of CI

Team A
The history of CI
The history of CI

• Jenkins master per team
• Self-hosted
• Everyone does “own CI”
• “Release team” makes “release”
Building Common CI
Building Common CI

• One Jenkins master (pre big product)
Building Common CI

• One Jenkins master (pre big product)
• AWS EC2 based (via EC2 Plugin)
Building Common CI

- One Jenkins master (pre big product)
- AWS EC2 based (via EC2 Plugin)
Building Common CI

- One Jenkins master (pre big product)
- AWS EC2 based (via EC2 Plugin)
- AWS S3 (via S3 Plugin)
Building Common CI

- One Jenkins master (pre big product)
- AWS EC2 based (via EC2 Plugin)
- AWS S3 (via S3 Plugin)
- Unified process
Building Common CI

- Gerrit
- Gerrit Plugin
- Pre-submit Trigger
  - Build
  - Build
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
  - Tests
Reality strikes back
Reality strikes back
Reality strikes back

- PSV leads to addiction
Reality strikes back

• PSV leads to addiction
• More tests -> more flakiness
Reality strikes back

- PSV leads to addiction
- More tests -> more flakiness
- More builds -> longer compilation
Reality strikes back

• PSV leads to addiction
• More tests -> more flakiness
• More builds -> longer compilation
• Misusing -> try commit first
Reality strikes back

100k+ builds per day
- Each “build” is execution of one build/test job
- Total number correlates with number of commits
- Number of builds is not so important as number of commits
- Big throughput is extremely important

~1.5k concurrent builds
- Morning commit
- Before lunch
- “Last attempt for today”

1.3-2.5k executors
- Raised on-demand
- Health checks
- Jenkins strategy is not optimized for cloud
04
How to scale jobs itself
How to scale jobs itself
How to scale jobs itself

• Dependencies in docker
How to scale jobs itself

• Dependencies in docker
• Distributed cache
How to scale jobs itself

• Dependencies in docker
• Distributed cache
  • ccache-memcache for gcc
How to scale jobs itself

- Dependencies in docker
- Distributed cache
  - ccache-memcache for gcc
  - clcache for MS VS
How to scale jobs itself

• Dependencies in docker
• Distributed cache
  • ccache-memcache for gcc
  • clcache for MS VS
• Cache for git
How to scale jobs itself

- Dependencies in docker
- Distributed cache
  - ccache-memcache for gcc
  - clcache for MS VS
- Cache for git
- Cache for docker images
How to scale jobs itself

• Dependencies in docker
• Distributed cache
  • ccache-memcache for gcc
  • clcache for MS VS
• Cache for git
• Cache for docker images
• Profile tests
How to scale jobs itself

- Dependencies in docker
- Distributed cache
  - ccache-memcache for gcc
  - clcache for MS VS
- Cache for git
- Cache for docker images
- Profile tests
- Split tests
What is "death" for Jenkins
How does it die?
How does it die?

• Queue is (only) growing
How does it die?

• Queue is (only) growing
• UI is unresponsive
How does it die?

• Queue is (only) growing
• UI is unresponsive
• It really dies
Why does it die?
Why does it die?

• Single point of failure
Why does it die?

• Any plugin can subscribe to any event
• Any plugin can do whatever maintainer wants
Why does it die?

"Handling GET / from 192.168.1.1 : RequestHandlerThread[#1871] View/sidepanel.jelly"

"Executor #3: executing someJob #777"

"jenkins.Queue [#1000]"
Why does it die?

"Handling GET / from 192.168.1.1 : RequestHandlerThread[1871] View/sidepanel.jelly": waiting to lock monitor 0x7fa5544b49a8 (some.plugin.ClassA), which is held by "Executor #3: executing someJob #777"

"Executor #3: executing someJob #777"

"jenkins.Queue [#1000]"
Why does it die?

"Handling GET / from 192.168.1.1 : RequestHandlerThread[#1871] View/sidepanel.jelly":
waiting to lock monitor 0x7fa5544b49a8 (some.plugin.ClassA),
which is held by "Executor #3: executing someJob #777"

"Executor #3: executing someJob #777":
waiting to lock monitor 0x7fa5641ee128 (some.plugin.ClassB),
which is held by “jenkins.Queue [#1000]”

"jenkins.Queue [#1000]"
Why does it die?

"Handling GET / from 192.168.1.1 : RequestHandlerThread[#1871] View/sidepanel.jelly":
  waiting to lock monitor 0x7fa5544b49a8 (some.plugin.ClassA),
  which is held by "Executor #3: executing someJob #777"

"Executor #3: executing someJob #777":
  waiting to lock monitor 0x7fa5641ee128 (some.plugin.ClassB),
  which is held by “jenkins.Queue [#1000]”

"jenkins.Queue [#1000]":
  waiting to lock monitor 0x7fa5544b49a8 (some.plugin.ClassA),
  which is held by "Executor #3: executing someJob #777"
Why does it die?

- Single point of failure
- Deadlocks
Why does it die?

- Java manages memory for you
- Usually it works fine
- Sometimes application needs a lot of memory
- Full GC (or any other stop-the-world pause in JVM)
Why does it die?

• Single point of failure
• Deadlocks
• Stop-the-world in JVM
06
What’s about scaling
Artifacts

- S3 Plugin
  - Publish from slave only
  - Keep logs in S3 and show directly from Jenkins UI
Build rotation

- Use Build Rotator Plugin
- Use date based rotation only
Nginx in front

- Static content
- Direct access to logs
- Caching
- Limits
Dangerous buttons

• Remove access to some buttons
Logs

• Logs to artifacts
• Redirect to file and scan file on agent
• Minimise verbosity
• Remove not really needed plugins
07
Anti-patterns
Read entire log instead of only needed part

- Timestamper
- Jenkins core (Email-notifier, etc.)
Read entire history instead of only needed part

- Jenkins core (forever-red build)
- Robot Framework Plugin
Other

• Build Failure Analyzer (thread pool per failed build)
Other

- Build Failure Analyzer (thread pool per failed build)
- EC2 Plugin (mis-using AWS API)
Other

- Build Failure Analyzer (thread pool per failed build)
- EC2 Plugin (mis-using AWS API)
- Groovy Event Listener Plugin (compile groovy all the time)
Other

- Build Failure Analyzer (thread pool per failed build)
- EC2 Plugin (mis-using AWS API)
- Groovy Event Listener Plugin (compile groovy all the time)
- Jenkins core (heavy queries from UI)
Other

• Build Failure Analyzer (thread pool per failed build)
• EC2 Plugin (mis-using AWS API)
• Groovy Event Listener Plugin (compile groovy all the time)
• Jenkins core (heavy queries from UI)
• etc.
Check list
Final checklist

Monitoring
Nginx in front
Change log rotation
Remove unused/not-really-needed plugins
Minimise/redirect logs

Publish artifacts from agent
Thank you

Contact
Alexander Akbashev

GitHub: Jimilian
alexander.akbashev@here.com

HERE
Invalidenstraße 116
10115 Berlin