Testing In The Cloud

How we reduced testing time by 50% and eliminated waiting time entirely

Kenneth Falck <kennu@sc5.io>

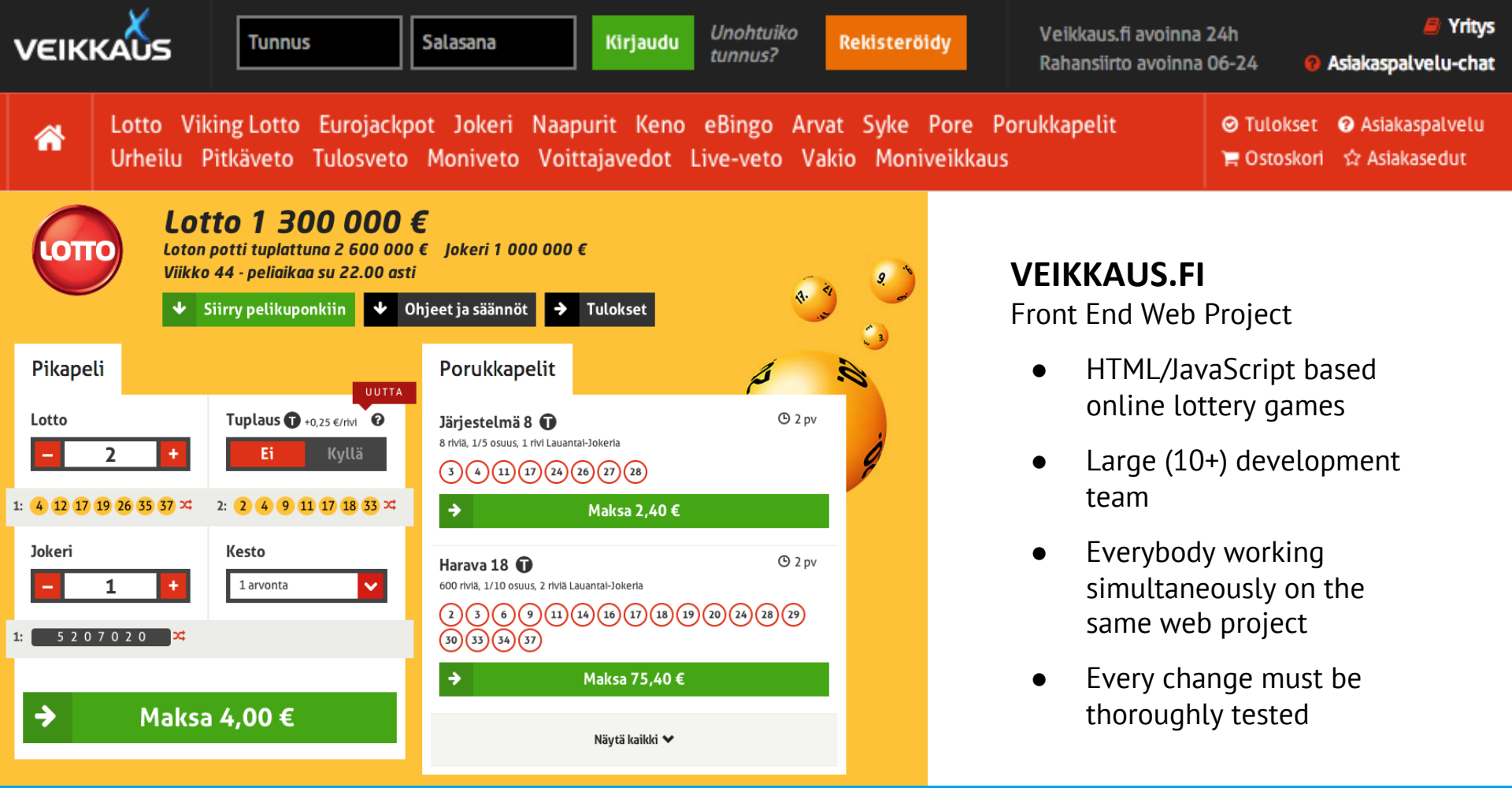
4.11.2014

Presentation Overview

Case Veikkaus.fi

- The Problem
- Background
- Our Solution
- How Did It Go?

CASE VEIKKAUS.FI



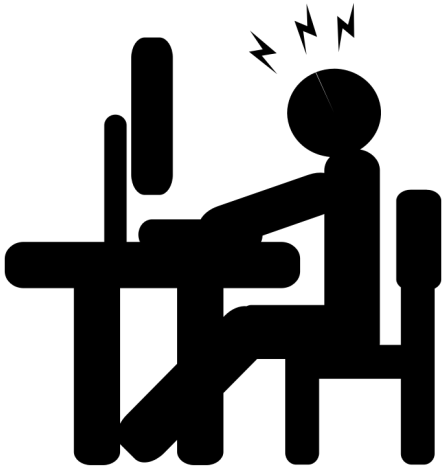
VEIKKAUS.FI

Front End Web Project

- HTML/JavaScript based online lottery games
- Large (10+) development team
- Everybody working simultaneously on the same web project
- Every change must be thoroughly tested

THE PROBLEM

Testing Is Slow



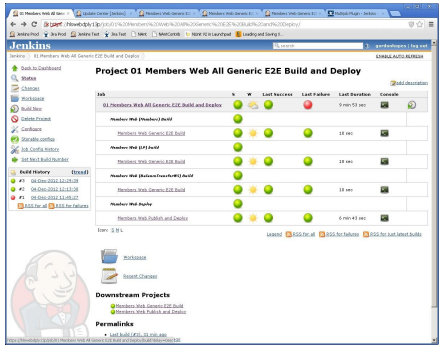
- Developers need to run tests several times a day
- One test run can take 30 minutes
  - First wait 10 min for a free slot
  - Then wait 20 min for tests to run
- As a result, developers become less productive and time is wasted

BACKGROUND

What is Jenkins?

Jenkins is a testing and deployment automation platform.

Developers use Jenkins to test a new version of their project, and to deploy it to production.



Jenkins Web UI

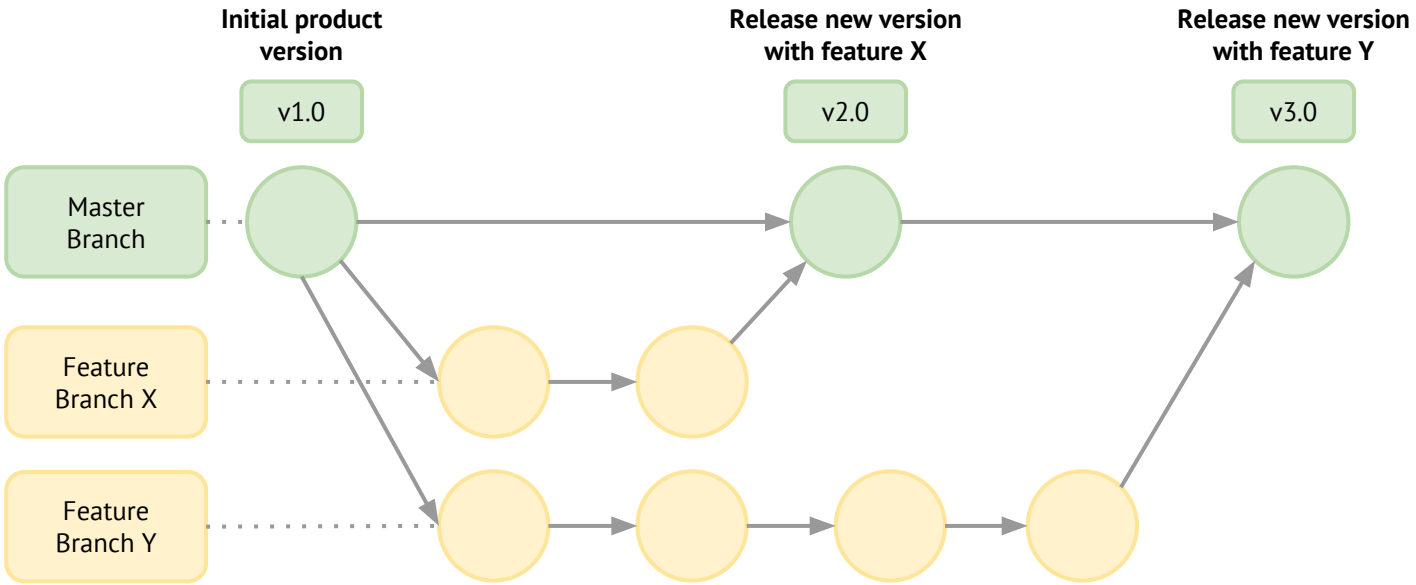
Developers don't need to know any details - They click "Test" and wait for the green light.

What are Branches?

Master is what gets published to production

Developer 1 works on feature X in her own branch

Developer 2 works on feature Y in her own branch



How Development Works

Developer 1: “Test branch X”

(Wait for 30 min...) Fail

Developer 2: “Test branch Y”

(Wait for 30 min...) Fail

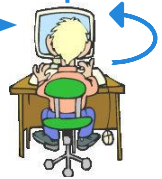
Developer 3: “Test branch Z”

(Wait for 30 min...) OK

New version is published

OK

Developer 4: “Test and publish the master branch”

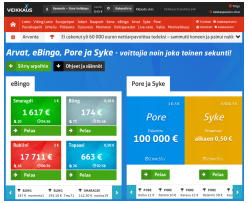


Developer 3 makes a Pull Request to merge branch Z

Developer 4 accepts Pull Request, merges branch Z to the master branch

Why Is Testing Slow?

PhantomJS is the virtual web browser that runs our automatic JavaScript tests without human interaction.

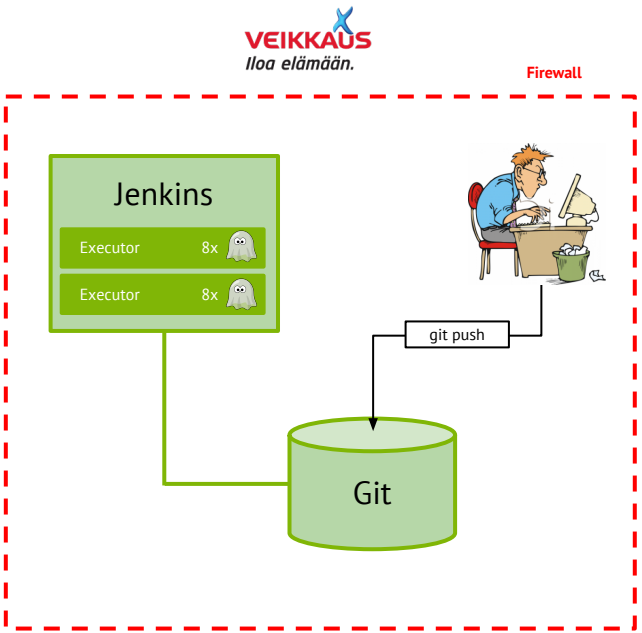


One PhantomJS requires one CPU core. To run 8 tests in parallel, 8 cores are needed. This makes testing 8 times faster.

To maximize testing speed, 400+ cores would be needed (every test would run simultaneously).

OUR SOLUTION TO SPEED THINGS UP

Original Setup at Veikkaus



A physical 16-core Jenkins server (no cloud)

- Scarce resource shared by many coders, no scaling
- Only two parallel job executors, 8 PhantomJS each
- Full test suite needs 400+ PhantomJS runs

Long waiting and running times

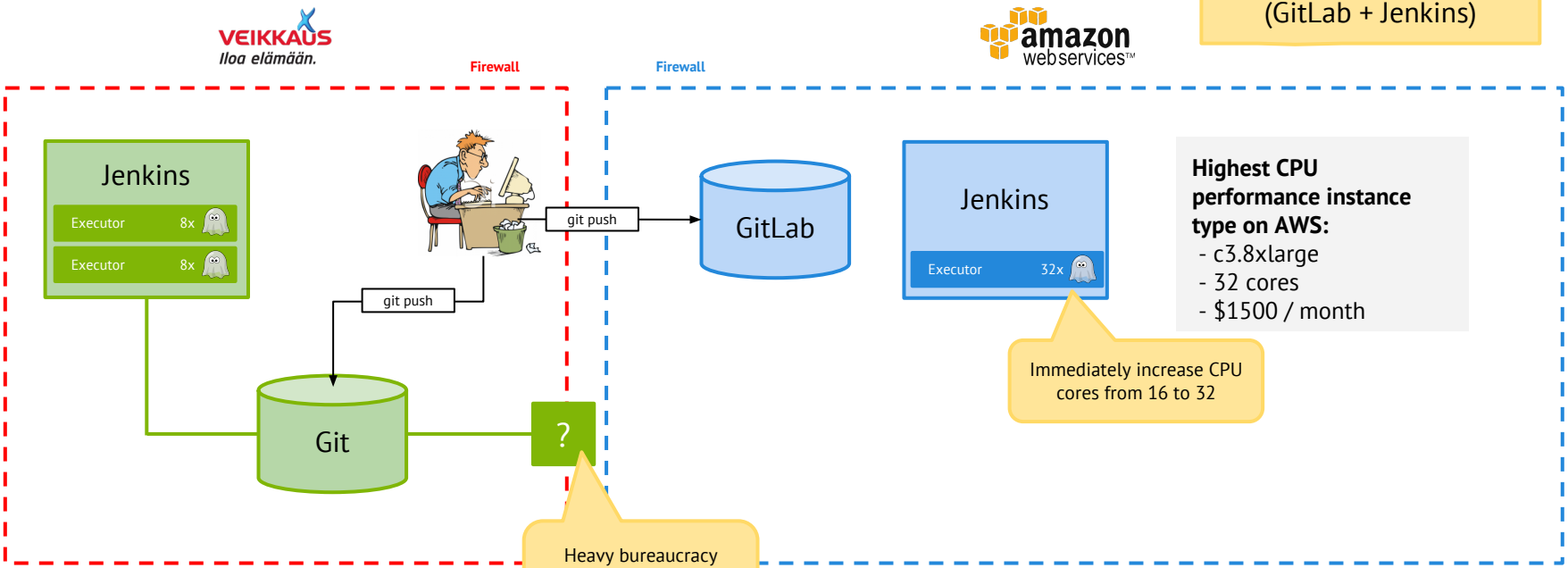
- First wait 10 minutes for the previous job to end
- Then wait 20 minutes for your own job to finish

How could we improve this?

- Let's move it to the cloud (Amazon AWS)
- We made a proposal of a scalable architecture and an analysis of the security implications (accepted)

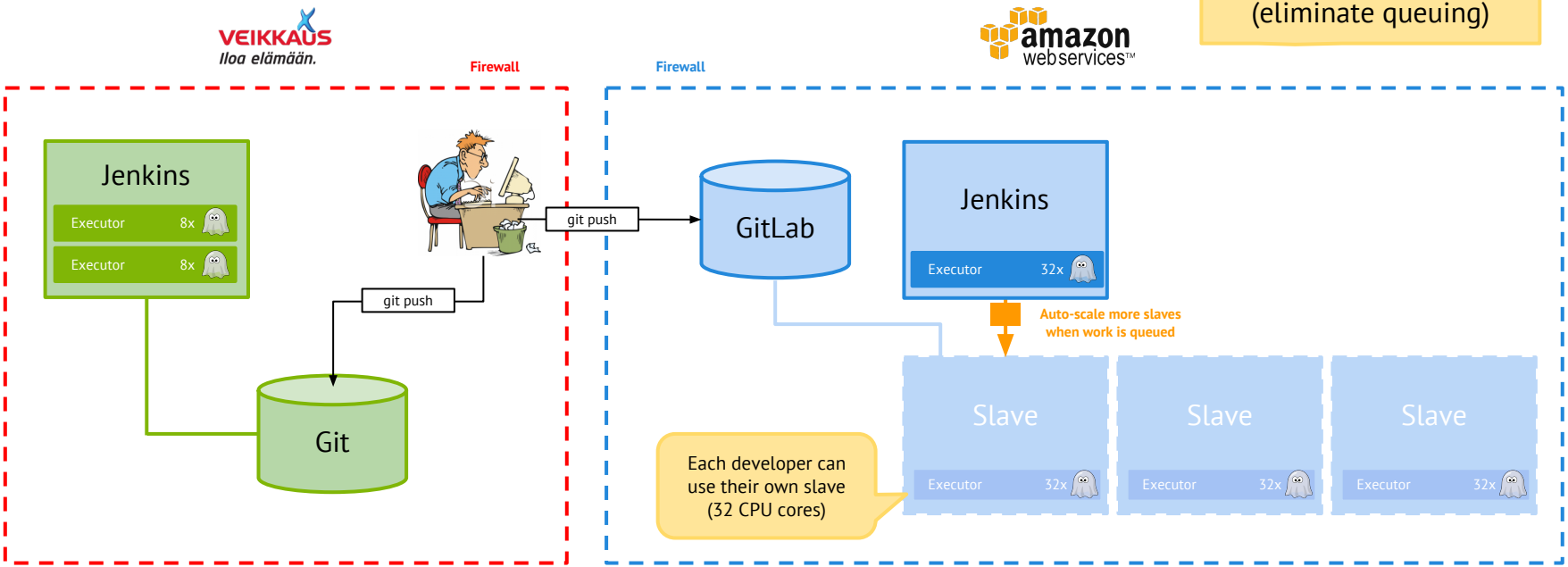
Phase 1: Jenkins on Amazon

Goal: Setup basic cloud infrastructure (GitLab + Jenkins)



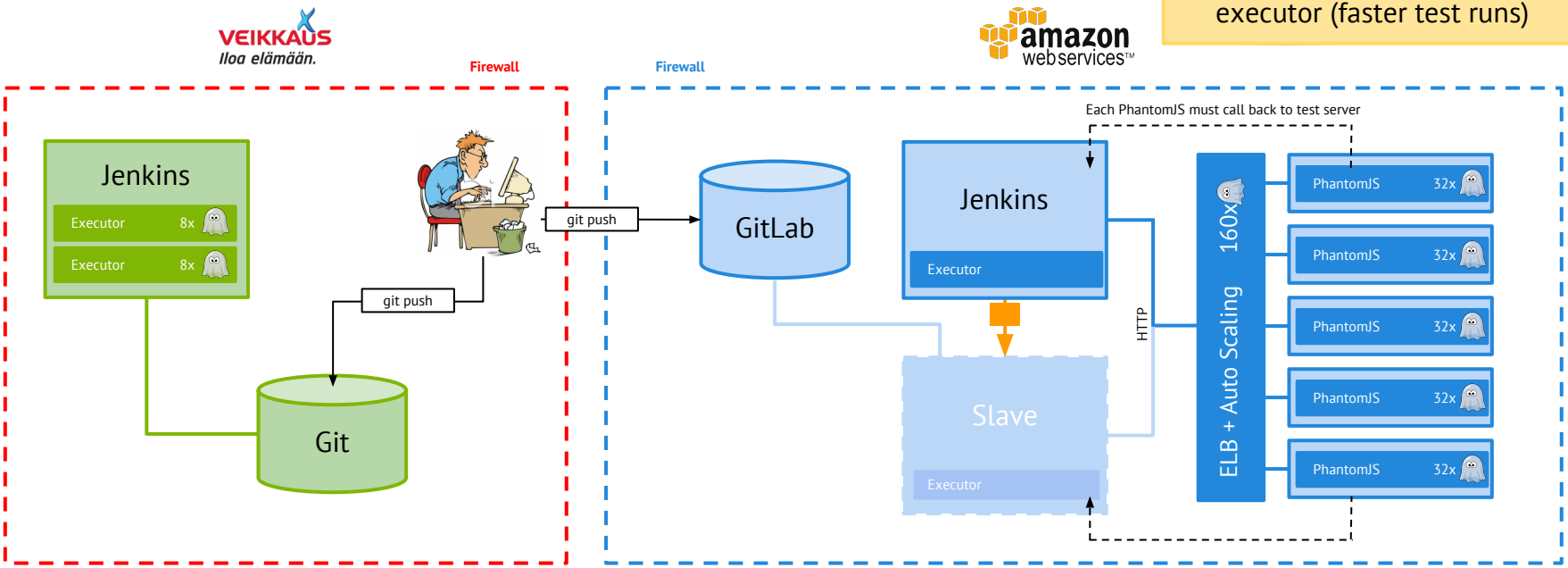
Phase 2: Scalable Jenkins

Goal: Scale out to N parallel job executors (eliminate queuing)



Phase 3: PhantomJS Cluster

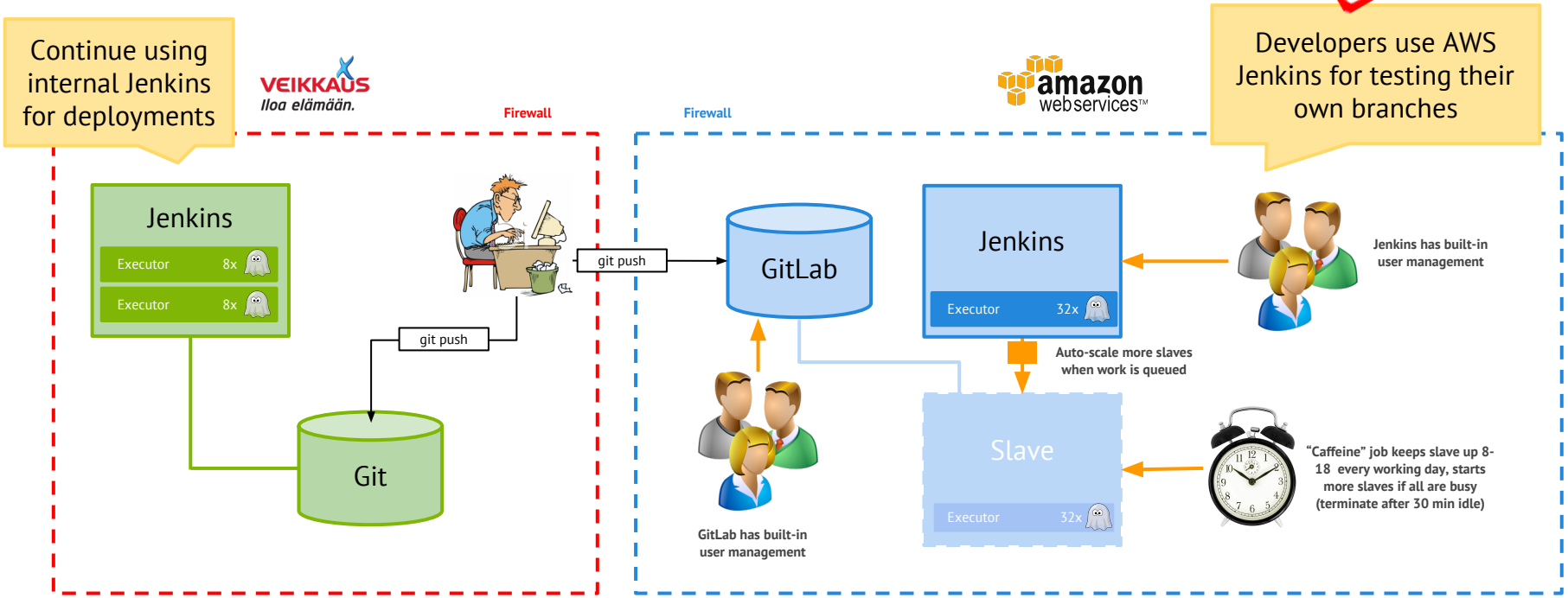
Goal: Scale out to more than 32 parallel Phantoms per job executor (faster test runs)



HOW DID IT GO?

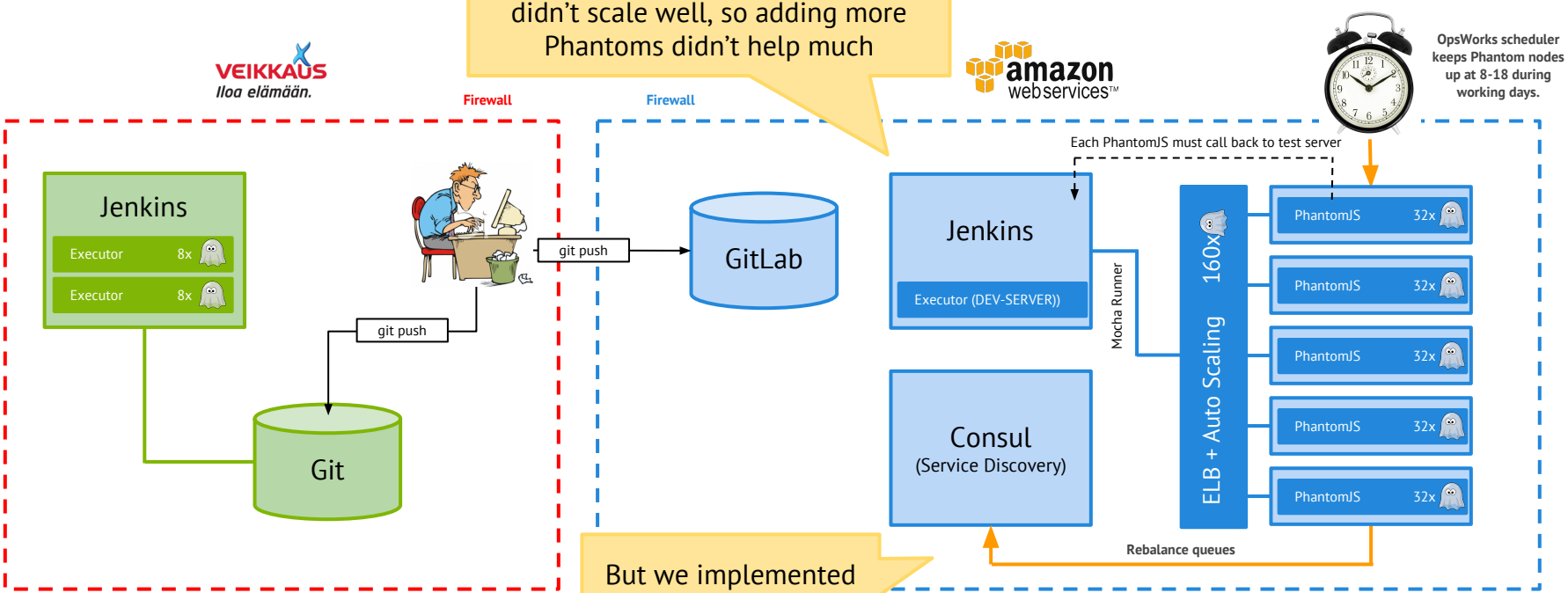
Jenkins & GitLab on AWS

MISSION ACCOMPLISHED



PhantomJS Cluster

We found that this part (dev-server) didn't scale well, so adding more Phantoms didn't help much



But we implemented the PhantomJS cluster anyway

Clean AWS Architecture

STACK 1

Chef Recipe

OpsWorks

Jenkins

Run Command | Stack Settings | Delete Stack

A stack represents a collection of EC2 instances and related AWS resources that have a common purpose and that you want to manage collectively. Within a stack, you use layers to define the configuration of your instances and use apps to specify the code you want to deploy. Learn more.

Instances

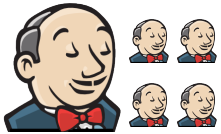
1 Instances | 1 online | 0 launching | 0 shutting down | 0 stopped | 0 error

Layers 1

Jenkins

Apps 0

No apps. Add an app.



Jenkins + slaves | GitLab | Consul | Manager UI

STACK 2

Chef Recipe

OpsWorks

Phantom Cluster

Run Command | Stack Settings | Delete Stack

A stack represents a collection of EC2 instances and related AWS resources that have a common purpose and that you want to manage collectively. Within a stack, you use layers to define the configuration of your instances and use apps to specify the code you want to deploy. Learn more.

Instances

9 Instances | 1 online | 0 launching | 0 shutting down | 8 stopped | 0 error

Layers 1

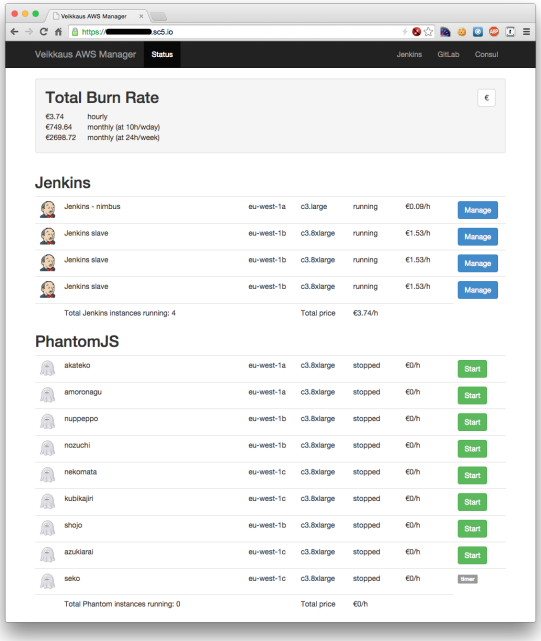
Node.js App Server

Apps 1

Phantom Server deploy

PhantomJS Cluster Nodes (custom Node.js app)

AWS Management UI



Custom Angular.js/Node.js App

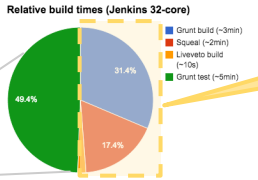
- Simple UI to check cloud status
- Calculate monthly cost of running instances
- Manually start/stop PhantomJS cluster nodes

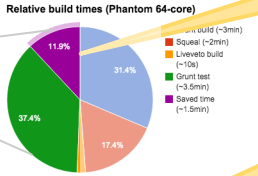
Test Speeds and AWS Pricing

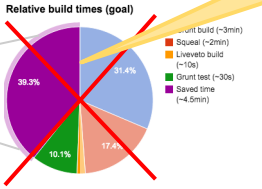
Setup	Total time	Test time	At 10h/workday	Always on 24/7
Jenkins 32-core	~10 min	5 min 7 s	$469 / month	$1,464 / month
Phantom 32-core	~10 min	4 min 38 s	$852 / month	$2,840 / month
Phantom 64-core	9 min 12 s	3 min 35 s	$1,234 / month	$4,217 / month
Phantom 96-core	9 min 33 s	3 min 55 s	$1,617 / month	$5,594 / month
Phantom 128-core	9 min 17 s	3 min 38 s	$2,007 / month	$6,770 / month
Phantom 160-core	9 min 17 s	3 min 38 s	$2,381 / month	$8,347 / month

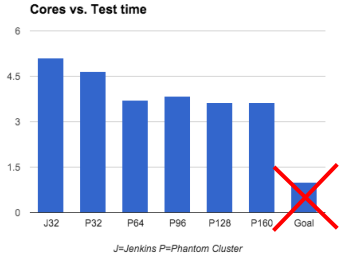
Not exact measurements, because PhantomJS often crashes and causes variations in test time

But still, end results were **50%** of the original time (~20 min)







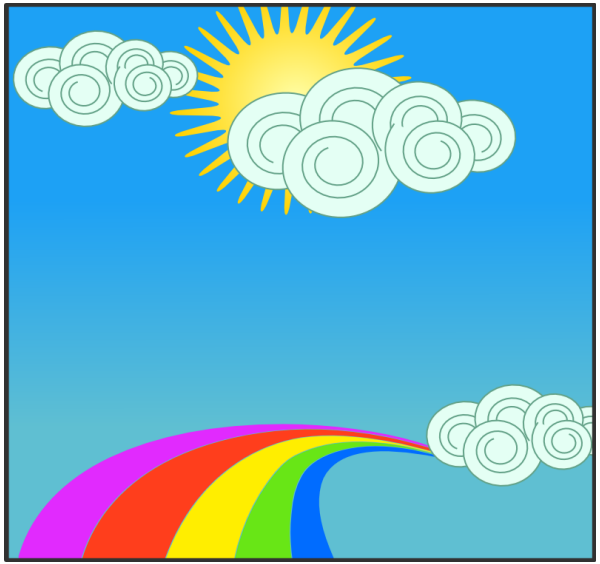


Half of the spent time was fixed (build and init time)

Violet sector = saved time

Goal couldn't be fully reached because dev-server did not scale

Bonus: Tests Are More Robust



Fixed many issues in tests

- Catch failures and error situations
- Eliminate dependencies on timing

Fixed PhantomJS crashing

- Retry tests when PhantomJS crashes

Made tests more universal

- Run locally on developer computers
- Run remotely in the cloud

Future: How To Be Even Faster

Get 192+ cores to work

- Tests started to fail after adding 160 cores
- Bottleneck in the development server
- The first test immediately generates 5000+ HTTP requests

Split test files that are too big

- If a test file takes 2 minutes to run, that sets the hard limit on speed (parallelization doesn't help any further)

Optimize PhantomJS execution

- Reduce startup time (pool / prefork Phantom processes)
- Disable image, font and CSS requests

THANKS!

Twitter: @kennu

Email: kennu@sc5.io

SC5