Jenkins Scaling and Organization for an Efficient CI

Andrea Giardini
@GiardiniAndrea

Camunda Services Gmbh
camunda.com

DevOps Pro Europe - Vilnius - 27/03/2019

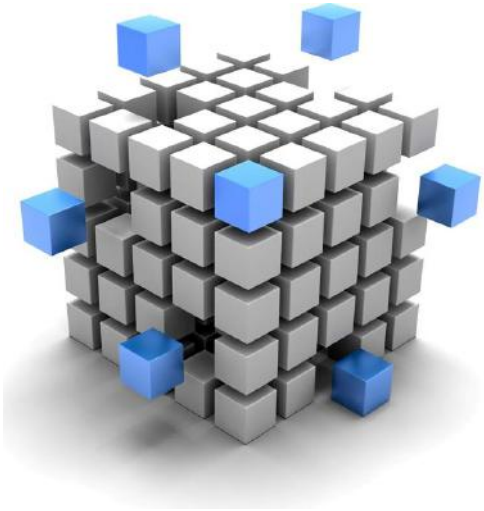
How Camunda's CI differs from others

Main Product: Camunda BPM, an Open Source platform for workflow and decision automation that brings business users and software developers together.

Embed as a Java library or use as a standalone service through REST.
Requires a traditional RDBMS.

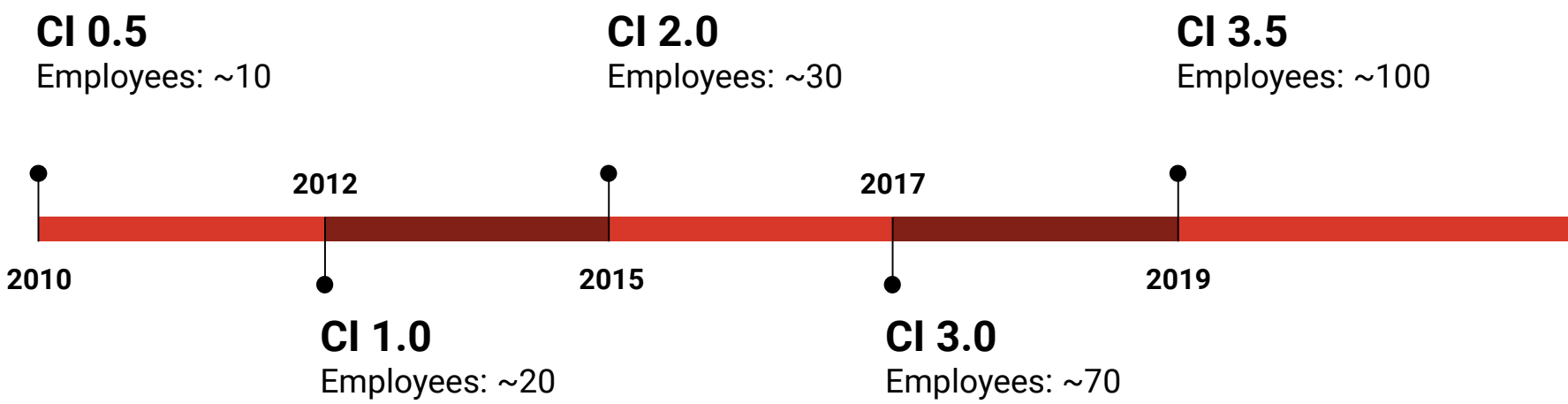
Supports

- 7 database vendors
- 4 Java vendors
- 7 server runtimes



Bi-annual release of a new minor version

CI@Camunda - Scale with the company



CI 0.5 - Age of Bare-Metal

Single Hudson instance running on bare-metal in Camunda HQ

Everything was provisioned manually: Databases, CI Jobs

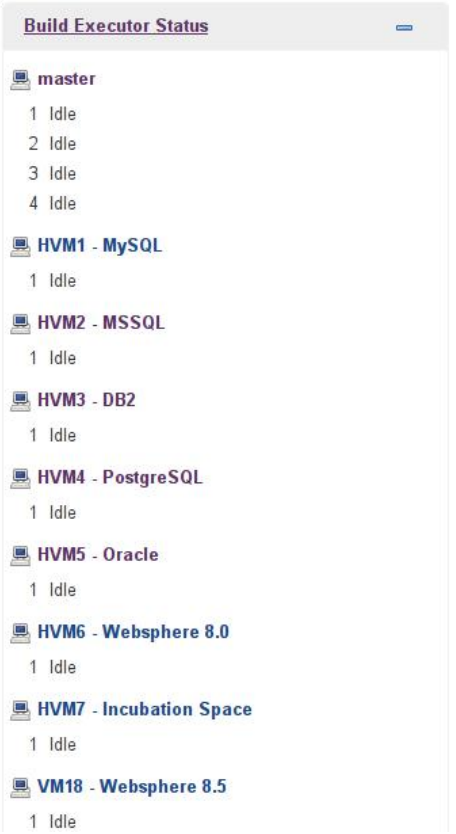
Problems:

- Reliability -> Jenkins UI became unresponsive when running lots of tests requiring CPU / Disk IO
- Scalability -> CI jobs piled up in queue
- Maintainability -> Major changes regarding configurations was painful, eg. common setting for all jobs changed

CI 1.0 - The Age of Virtual Machines

Single Jenkins Master in DataCenter

- 8 static Jenkins Agent VMs
- ~1000 Jobs in total manually managed consisting of:
  - 4 Camunda Versions
  - Community Projects
  - Websites
  - Operational Tooling



CI 1.0 - Problems

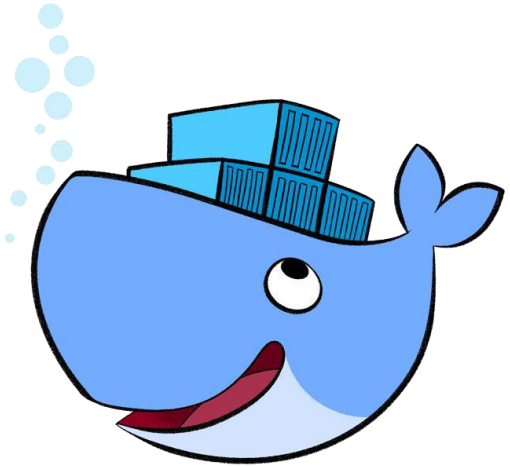
- Maintainability
  - Manually managed configuration -> CI Jobs, Databases and Servers
  - Unable to test updates to Jenkins and its plugins
- Resource isolation and Scalability
  - 1 instance per database
  - 1 instance for enterprise-grade application servers like Weblogic and WebSphere
- Reproducibility
  - Devs and QA could not easily recreate CI environment

Anecdote:
Use of an excel sheet to track port configurations of application servers and databases across CI jobs

CI 2.0 - The Age of Containers

Rethinking CI in terms of:

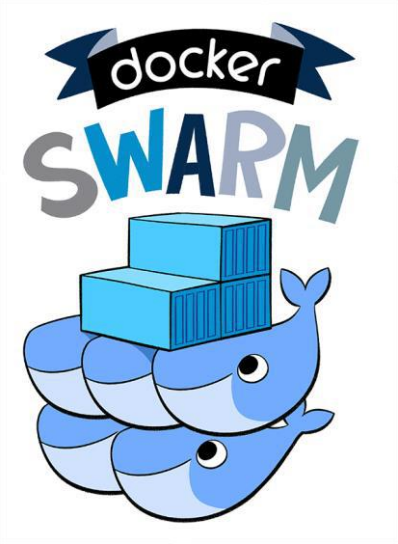
- Maintainability and Reproducibility
  - Infrastructure as Code -> Everything is in SCM and immutable
  - Easy to support new environments
- Resource isolation
  - Everything runs isolated in its own context
- Scalability and resource management
  - Orchestration
- How to achieve maximum efficiency with limited resources and costs
  - Split use cases in performance vs importance
  - Run important services in DC, the rest on commodity hardware to keep costs low

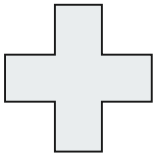


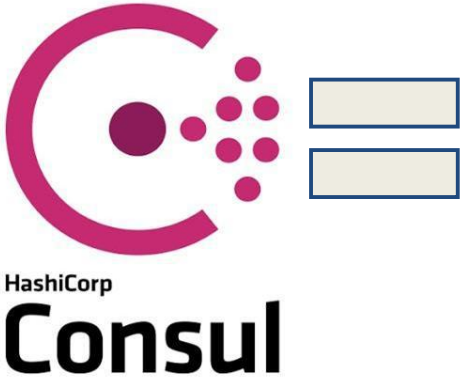
CI 2.0 - The Age of Containers

Solving the problem of scalability, resource management and allocation?

Welcome to Container orchestration!





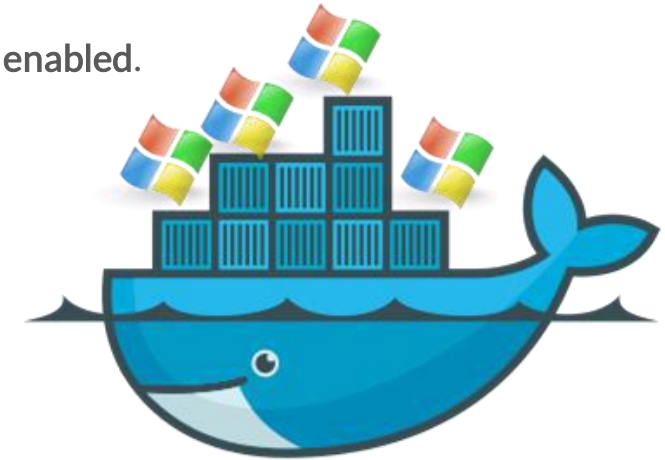


CI 2.0 - Reproducibility for Devs and QA

CI 2.0 - Anecdotes - Microsoft SQL Server

Scaling Microsoft SQL Server

1. Create a QEMU image by using Chef cookbooks and Packer
2. Create a Docker image to bundle QEMU and the image.
3. Make sure to run it on **bare-metal** or
   on VMs with **nested hardware virtualization cpu flag enabled**.
4. Win!



CI 2.0 - Lessons learned

Building Docker images:

- Evaluate downloading third party binaries vs installing by package manager
- Rebuilt base images + downstream images from time to time if you do not host an own package manager mirror to be aware of changes in upstream repositories
- Running more than one processes in a Docker container by eg. using something like supervisord is not so bad
- Having a unified interface to interact with Docker images is a big bonus, eg. by using a Makefile describing the tasks like build, run ...

CI 2.0 - Problems

- Maintainability
  - Compatibility: Jenkins Docker plugin vs Docker versions vs Operation system
  - Requirements increased: More projects, more environments, more testing

- Scalability
  - limited hardware capacity
  - Buy vs Rent

- Resource allocation:
  - Still a hard problem, especially with multiple Java processes in a CI context
  - Java resource limits

CI 3.0 - The Cloud Age

Want to get rid of scalability restrictions by limited hardware? => Go Cloud!

More advanced Container orchestration than Docker Swarm? => Go Kubernetes!

Wanna reduce maintainability? => Use managed services, eg. GKE!

kubernetes

Our philosophy for CI3

Everything should be configured as code

Reusability across different projects is a must

Maintenance and upgrades should be low-effort

No more snowflakes

Jenkins - Kubernetes plugin

Moving from commodity machines to the Cloud

- No more limitations in term of resources
- Resources
  - Possibility to have large servers for a fraction of the cost
- Scalability
  - Few machines during the night
  - Large cluster during the day
- Everything happens on-demand

Jenkins - Configuration as Code Plugin

This plugin allows you to define your Jenkins configuration with YAML.

- We got rid of many custom groovy scripts
- Portability across different instances
- YAML is a better format than XML
  - Compact
  - Easier to read/edit

Disadvantages:

- Not everything is configurable using the CasC plugin
  - Some groovy scripts are still needed
- Not all the plugins support it
- Still young, but very promising

Jenkins - Configuration as Code Plugin

We use the base Jenkins image, no custom images:

- Plugins are downloaded (if necessary) by an init-container
- Jenkins is configured by the CasC plugin. The configuration is mounted using ConfigMaps
- Jenkins's jobs are configured using JobDSL

This makes our setup:

- Easy to replicate, even locally
- Portable and re-usable

How does it work: Environments

Every supported environment has its own repository.
This repository contains:

- Dockerfile
- Configurations files
- Tests for docker image

Repository is shared across versions:

- Minimal configuration changes (most of the time)
- Forces us to rebuild old images and verify that they still work
- Reusable code

IBM **Db2**

How does it work: Jobs

Job DSL plugin is used to maintain all our Jobs:

- Job dependencies
- Pipelines structure
- Parameters
- Configuration

Over the year we built a Groovy project that we use to template our JobDSL files

- Makes extending job easier
- Applying the same change to multiple jobs is much easier

What's the situation today?

Let me show you something...

Our build infrastructure in numbers

Currently, the Camunda CI has:

~ 2500 individual jobs (only for the Camunda BPM Platform project)

~ 22k builds in the past 30 days

- 19 Databases supported
- 8 Java versions
- 14 Application servers

But let's take a step back ...

How does it work: Autonomy

Smaller Jenkins instances are managed by the teams:

- We provide a ready-to-use and updated instance
- They provide the jobs

Every project has a folder that Jenkins uses to bootstrap all the jobs.

- Code and CI are in the same repo
- Autonomy for small change
  - Create new jobs, modify existing ones
- Right degree of trust
  - Jenkins configuration is protected
  - Credentials can't be accessed

Conclusions

Where we are today is the result of years of work and failed experiments

- Migration to Kubernetes improved the quality (and the speed) of the service noticeably
- CasC plugin is the way-to-go for large deployments
- Find the right degree of control and autonomy for your company

Giving more autonomy to our developers allowed us to:

- Avoid interruptions for easy fixes and reduce wait time
- Developers are responsible about maintaining the pipeline
    - Changes to the pipeline are testable in a branch, like code changes

Moving forward… Developer access to Kubernetes namespaces

Questions?

Slido.com - #devops2019

Thank you for the attention

Andrea Giardini - @GiardiniAndrea - https://andreagiardini.com

Camunda Services Gmbh - @Camunda - https://camunda.com