TAL TECH
IT KOLLEDŽ

SPARROW CI

Scope of work in Distributed Systems project

Student:	Gert Vesterberg
Student Code:	175871IDDR
Supervisor:	Andres Käver

Tartu 2020

Table of contents

Introduction

There are many CI tools available, but most of them operate as SaaS models. Therefore it is quite expensive for a single developer or student to afford such a subscription. Although some services allow us to use a limited amount of free build-minutes, in some cases it might be not sufficient.

There are also downloadable CI tools available, which can be run locally. The most popular tool is Jenkins [1]. Jenkins was initially intended for building Java applications. To build some other projects, relevant plugins need to be installed. In addition to outdated UI and UX, Jenkins is quite difficult to set up properly and lacks essential features for building modern cloud-based applications.

The goal of this project is to create a lightweight open-source, cloud native distributed continuous integration (CI) tool, which can be easily run in locally or in Kubernetes cluster.

Domain-specific language (DSL) for describing pipelines will be simple YAML [2], which is part of project's GIT repository.

Scope of work

Project scope is divided into multiple milestones:

Must have

1. Design and define database entities
2. Define controllers needed managing database entities
3. User authentication and authorization
4. Executing shell-commands and capturing results from Application (Proof-of-concept)
5. Github integration for repositories discovery
6. Incoming WebHooks from Github
7. Commit, Branch and Pull Request discovery
8. DSL parser (Proof-of-concept)
9. Initiating a build on WebHook notification, storing the results and artifacts locally

Nice to have

1. DSL visualization in UI
2. Storing artifacts to S3 [3] bucket
3. Multi-node build executors on Kubernetes [4] cluster

The scope of minimum viable product (MVP) is to develop following microservices:

Main services

Frontend

Frontend single-page web application will be built using Angular [5] framework and ngx-admin UI visual framework.

Frontend application uses standard JSON Web Token (JWT) [6] flow for authentication and token-refreshing. Transpiled static files of frontend application will be served by NGINX [7] web-server.

Server

Server processes HTTP request, publishes the build tasks to message queue and listens to the task results from queue.

Server project has following services:

- **UserService** - user management and authentication
- **BuildOrcestratorService** - publishes build messages to queue; listens for results
- **BuilderService** - listens the build messages, executes build, sends the response
- **BuildExecutorService** - executes the build via running shell command
- **DSLService** - validates and parses YAML DSL build pipelines
- **IntegrationService** - integrates with repositories on Github
- **IncomingWebHookService** - manages incoming webhooks

Middleware

MySQL Database

MySQL Database is used in Server components to store the user authentication data, projects and their integrations and build results. MySQL was chosen because of its performance and the author's previous experience with such RDBMS.

For the project demo, the system used DigitalOcean’s managed database cluster.

Kubernetes

Frontend and Server components will be deployed to the Kubernetes cluster. For the demo project, System runs in 3-node DigitalOcean managed Kubernetes cluster. NGINX ingress will be configured as internal load balancer.

Message Broker

In order to BuildExecutorService to know when is the right time to execute the build, a message broker will be used. The exact message broker will be chosen during the implementation phase, the potential candidates are RabbitMQ, Apache Kafka, ActiveMQ [7].

File storage

Build artifacts (build logs and artifacts) cannot be stored in the Kubernetes cluster due to it’s distributed nature. Therefore separate file storage layers will be used. Exact service will be chosen during the implementation. Potential candidates are: Amazon S3 and DigitalOcean Spaces.

Diagrams

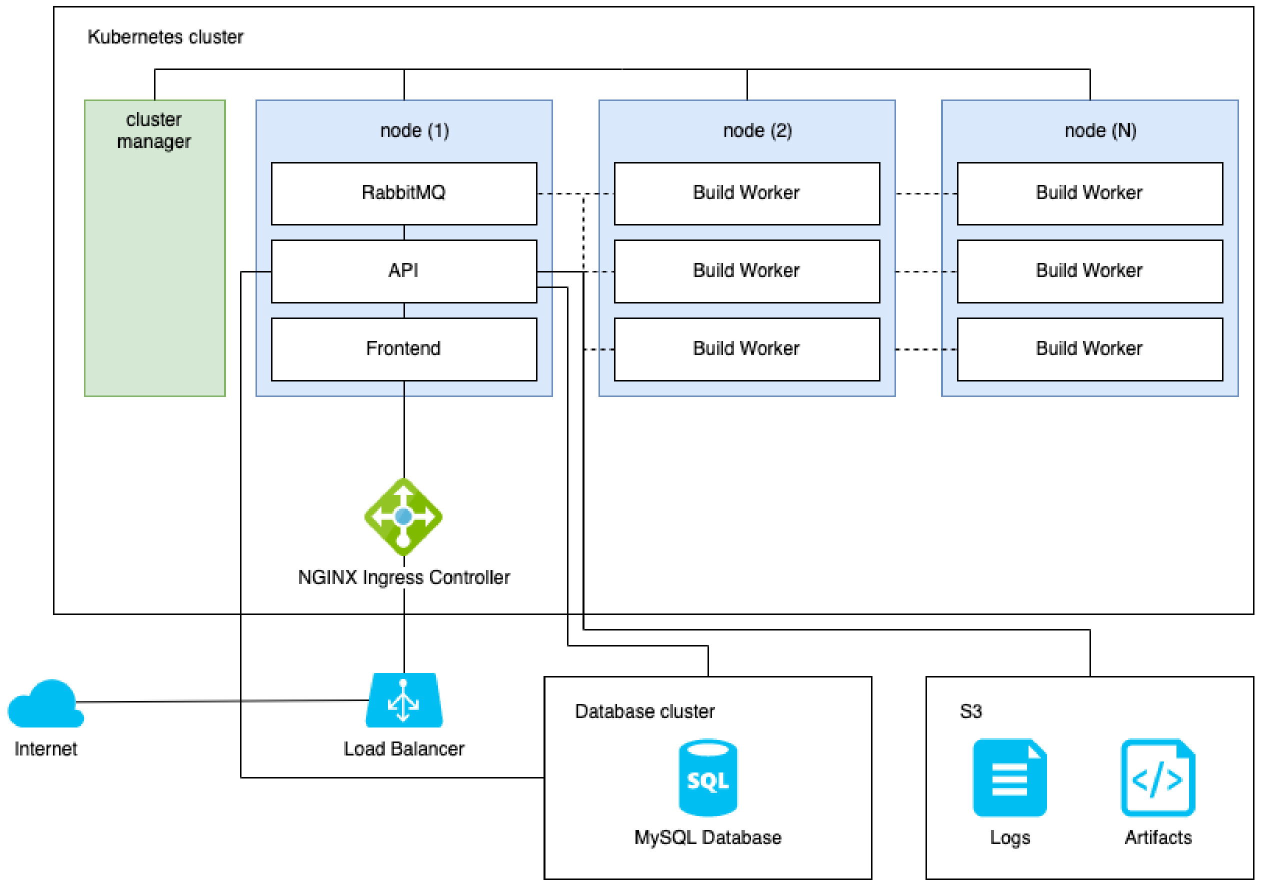

Diagram 1: **System Architecture**

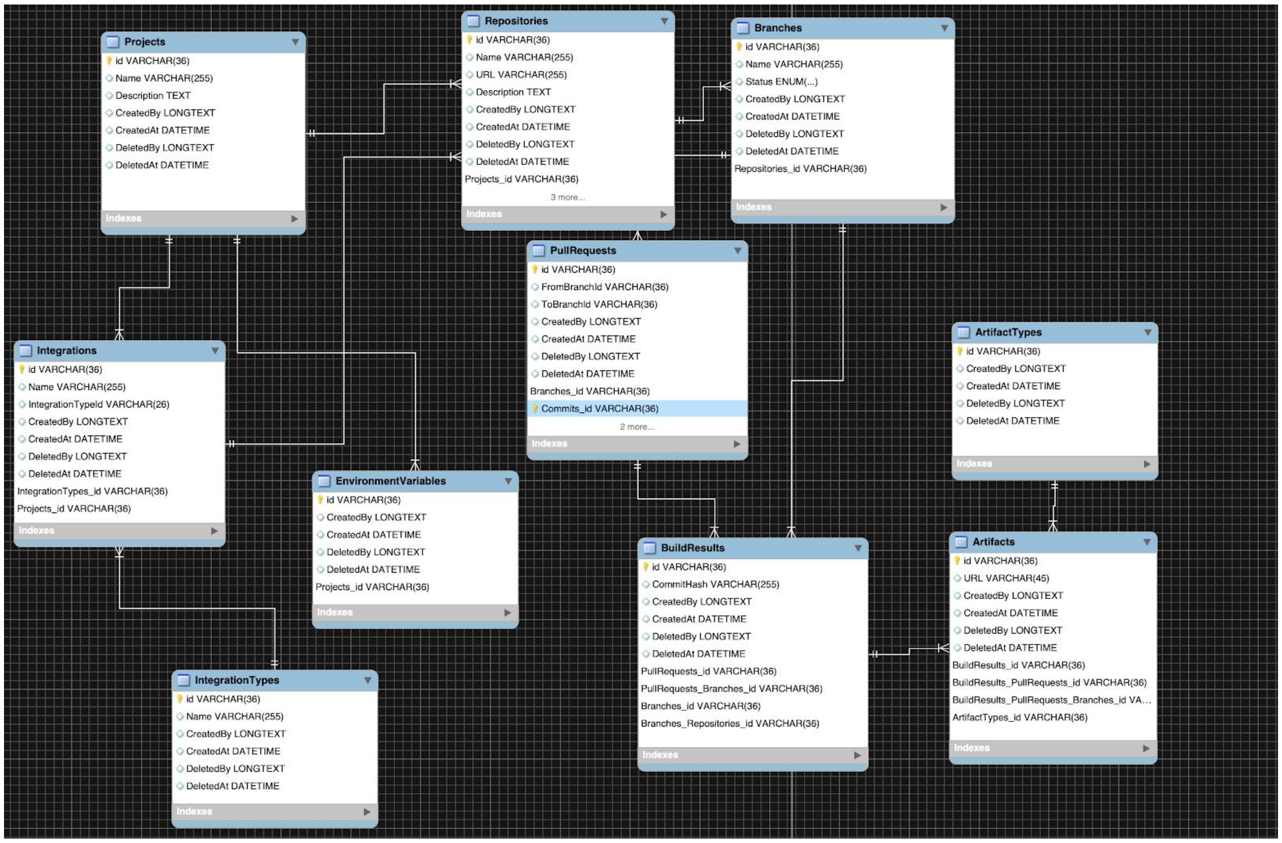

Diagram 2: **ERD** *(subject to change during the project)*

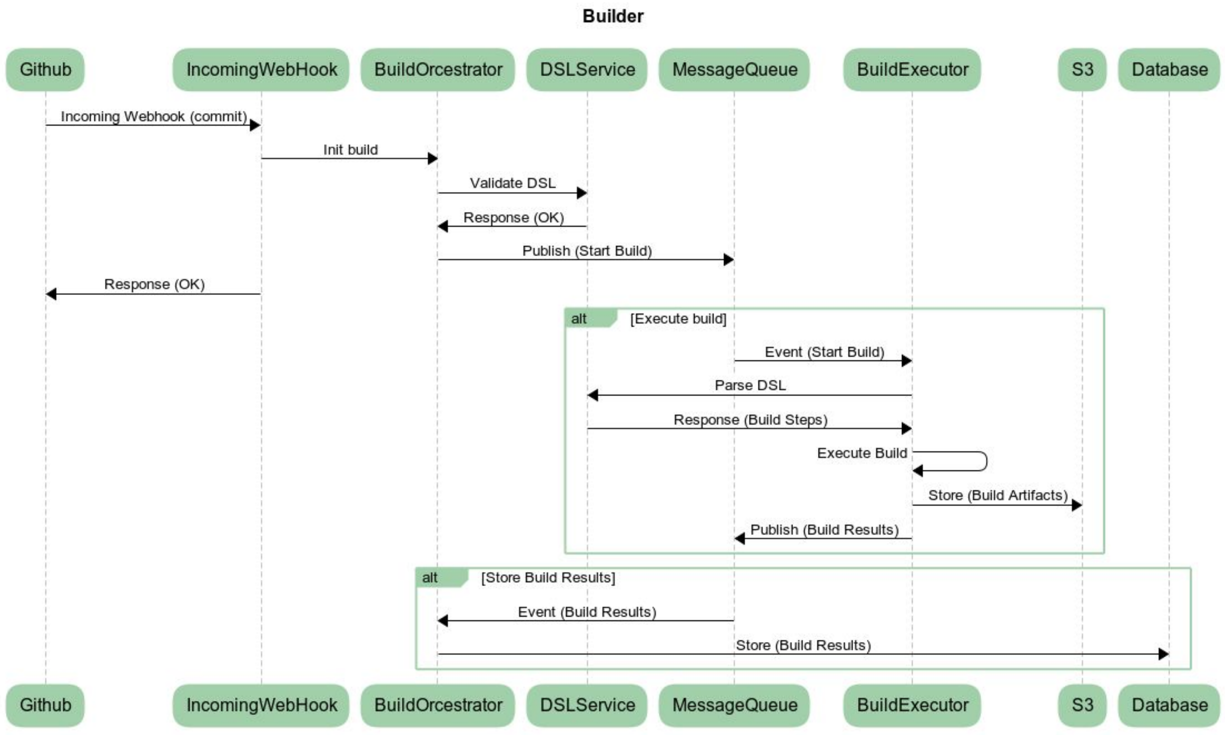

Diagram 2: **Sequence Diagram - main flow** *(subject to change during the project)*

References

[1] **Jenkins CI -** https://jenkins.io/
[2] **YAML** - https://yaml.org/
[2] **Amazon S3** - https://aws.amazon.com/s3/ ; **DigitalOcean S3 analogue** - https://www.digitalocean.com/products/spaces/
[3] **DigitalOcean Managed Kubernetes** - https://www.digitalocean.com/products/kubernetes/
[4] **NGINX** - https://www.nginx.com/
[5] **Angular** - https://angular.io/ ; **ngx-admin**: https://github.com/akveo/ngx-admin
[6] **JSON Web Token** - https://jwt.io
[7] **Message Brokers**: Apache Kafka - https://kafka.apache.org/ ; RabbitMQ - https://www.rabbitmq.com/ ; ActiveMQ - https://activemq.apache.org/