UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Timofei Ganjusev

Comparison of Gitlab Runner implementations without Docker machine for cloud providers

Bachelor's Thesis (9 ECTS)

Supervisor(s): Shivananda Rangappa Poojara, MSc

Tartu 2022

Comparison of Gitlab Runner implementations without Docker machine for cloud providers

Abstract:
In recent years DevOps practices became more popular among all companies in the IT industry. Gitlab is a platform built on top of a code version control system and services to manage an application active development process.The automation of processes of application development lifecycle together with Agile practices allow to deliver faster software products to market. The objective of the thesis is to analyze existing Gitlab Runner implementations design including scalable solutions without Docker Machine because it was deprecated by Docker. The Gitlab utilizes its own auto scaling runner integration with popular cloud providers such as Google Cloud, AWS and Azure Cloud and has to maintain a forked version of Docker Machine. The thesis mainly describes aspects of different Gitlab Runner implementations, finding advantages and disadvantages of each solution and proving the theoretical problems via benchmarks. Using different cloud provider environments, were set up auto scale Gitlab Runners in Kubernetes Clusters, non-scalable on Linux server. Further, the runners were benchmarked with a pipeline that contains 3 stages with 7 jobs emulating basic project pipeline. According to the results, the best solution is runner in Azure Kubernetes Service with cache in Azure Cloud, but the popular cloud providers have instances with CPU on ARM architecture with better performance in some cases and different cache storage solutions could be tested.

Keywords:
Gitlab, Kubernetes, CI/CD, Docker, Podman, Kaniko, Azure, Hetzner, Cloud computing

CERCS: P170 Computer science, numerical analysis, systems, control

Erinevate pilve teenustepakkujate jaoks Gitlab Runner'ite implementatsioonide võrdlus ilma Docker masinata

Lühikokkuvõte:
Viimastel aastatel on muutunud DevOps'i praktikad populaarsemaks kõigi IT-valdkonnaga seotud ettevõtetes. Gitlab on platform, mis põhiteenus on koodi versioonihalduskeskkond ning sellega seotud teenused aktiivse arendusprotsessi haldamiseks. Tarkvara loomise protsesside automatiseerimine koos Agile praktikumidega võimaldab kiiremalt tuua turule lõpptoote. Lõputöö eesmärk on analüüsida olemasolevat Gitlab'i Runner'i rakenduste disaini, sealhulgas lahendused skaleerimise võimalusega ilma Docker masinata sest Docker lõpetas Docker masina haldamise. Gitlab kasutab automaatse skaleerimise runner'i integreerimise jaoks Docker masinat koos populaarsete pilveteenuse pakkujatega (näiteks Google Cloud, AWS ja Azure Cloud), mis peaksid haldama Docker masina. Lõputöös kirjeldatakse erinevate Gitlab Runner'ite implementatsioonide aspekte, eeliste ja puuduste leidmise iga lahenduse kohta ning praktilise võrdlusaluste abil teoreetiliste

probleemide tõestamine. Erinevate pilveteenuse pakkujate teenuste kasutamisel, paigaldati Gitlab Runner'i Kubernetes keskondadele ning Linuxi serverile. Edasi testiti Gitlab Runner'ite sooritusvõimekused konveieri abil, mis sisaldab 3 etappi 7 tööga ning omakorda jäljendab tava tarkvararakenduse projekti konveierit. Vastavate tulemuste põhjal parimaks lahenduseks osutus Gitlab'i Runner'i käivitanud Azure Kubernetes keskkond koos vahemäluga Azure pilves. Kuid populaarsetel pilveteenuste pakkujatel on olemas masinad protsessoriga ARM-arhitektuuril, mis võimaldavad saada paremaid tulemusi mõndadel juhtudel ja samuti on võimalik testida erinevaid vahemälu salvestuslahendusi.

Võtmesõnad:
Gitlab, Kubernetes, CI/CD, Docker, Podman, Kaniko, Azure, Hetzner, Pilvearvutus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

In the past decade the DevOps approach became more popular. The DevOps practices enable to connect development and operation teams in order to decrease the amount of time needed for application development, maintenance and delivering the end product to the customers. The DevOps consist of culture, set of practices and tools that allows to increase end product quality, reduce development time and automate application development and deployment processes.

The Gitlab platform is one of the tool sets that helps to automate the development process. It has many built-in features such as a repository manager based on Git, continuous integration (CI) and continuous delivery (CD) integrations with different cloud providers and different tools for project management.

The Gitlab CI/CD can automate the application build, test, containerization and deployment stages. These stages can be described in CI pipeline configuration. Each stage consist of one or more jobs. The pipeline is running on the Gitlab Runners - the application installed on the server that runs CI/CD jobs. The Gitlab provides shared runners, however, the customization of this type runners is not available. The Gitlab allow to connect own runners to the platform to use it within the project or group of projects. For the small projects or teams with basic pipelines it is enough to use Gitlab Runner on GNU/Linux OS servers, but the limitation of these runner instances is in limitation in available resources of the instance for running the job. For the projects with many development teams the Gitlab allows to use auto scaling runners that have integrations with popular cloud providers such as Azure Cloud, AWS and Google Cloud. Although this solution is available, this type of auto scalable runners based on Docker Machine that is used for orchestration runner instances, however, the Docker Machine is depricated [3] by Docker and Gitlab team maintaining [1] its own forked version of Docker Machine for auto scaling runners. Gitlab has already written the whitepaper and described a solution for future auto scalable runners without docker-machine and are planning to introduce it [2] in 4 quarter of 2024 year. As an alternative Gitlab Team has developed an auto scalable runner for Kubernetes Cluster with different integrations with popular cloud providers and recommends [8] to use this solution.

In the aim of the thesis is to compare auto scalable Gitlab Runner performance in Azure Kubernetes Service, self hosted Kubernetes Cluster in Hetzner Cloud and Gitlab Runner on server with Linux OS in running basic pipeline that consist of 3 stages: installation stage, test stage and containerization stage and get true overview of advantages and limitations of different Gitlab Runner implementations proved by the benchmark results. Chapter 2 gives an overview about Gitlab platform services related to pipelines, describes the Docker Machine problem and provides information about used technologies. Chapter 3 describes workflow and architecture of Gitlab Runners implementations without Docker Machine using different cloud providers tools. The benchmarks environments, example application and results are described in Chapter 4

and chapter 5. In section 6 are this thesis conclusion and discusses about further work.

2 Background

This section provides an overview of Gitlab CI/CD pipelines architecture, integrations with different cloud providers and used technologies in the implementation of this thesis practical part.

2.1 Gitlab

Modern software development processes utilize different Git[1]-based repository hosting manager tools because it provides different management which is used for the collaborative workflow of software lifecycle. The aim of these tools is to manage development projects and related files since they are changing over time. Gitlab[2] is a repository manager which implements needed integrations and tools between software development and IT operations.

2.1.1 Gitlab Continuous Integration

Continuous integration (CI) is the practice of automation steps required to build, test and prepare applications to deploy. It is a DevOps practice that allows to frequently merge code updates into a central repository with guarantee that newly added code update correctness by running different kinds of tests such as unit tests, integration tests, end-to-end tests, security tests, automated code quality tests and syntax style review tools. The aims of continuous integration are to prevent bugs in a software, improve quality of end application and reduce time needed for validation and quickly deliver updates to customers. The Gitlab platform provides tools to create a continuous integration pipeline and visualize all the steps in the Gitlab web user interface.

2.1.2 Gitlab CI/CD pipelines

The pipelines are a top-level abstraction that implements continuous integration and continuous deployment flow. Pipeline itself is a series of stages that builds application code, runs different kinds of tests and is responsible for deployment. Moreover, pipelines are executed automatically according to the configuration, but there is also the possibility to interact with them manually. The configuration describes the pipeline workflow that includes different stages. The basic pipeline contains build, tests and deployment stages. The jobs are executed by Gitlab Runners. Parallel executed jobs require a concurrent amount of available runners to execute it at the same time. The pipeline configuration is stored in a file with YAML[3] extension.

[1] `https://git-scm.com/`
[2] `https://about.gitlab.com/`
[3] `https://yaml.org/`

2.1.3 Distributed jobs artifacts

The Gitlab Runner provides a cache mechanism with a variety of caching strategies where selected directories or files are saved and shared between subsequent jobs. The default cache mechanism stores cached artifacts in a Gitlab local path. The Gitlab Runner by default pulls the artifacts from the Gitlab local path at the beginning of the job step and pushes updated artifacts at the end of the job step. Although the up-to-date artifacts significantly reduce the job step completion time, the problem of that method could be the network bandwidth bottleneck between Gitlab instance and Gitlab Runner if they are located in different cloud providers. Gitlab provides the opportunity to customize artifact storage location and migrate artifacts to AWS S3-Compatible[4] storage service or similar Blob Storage[5] in Azure and make them available on different Gitlab Runners. In addition, it is possible to combine described methods to one - connect S3-Compatible object storage directly to the runner as local directory, create a unique directory for each branch, share and keep artifacts up-to-date between different job steps.

2.1.4 Autoscaling runners integrations

The Gitlab platform provides the interfaces to use computing resources of cloud providers in a specific schedule and dynamic way. It can autoscale the amount of runners as it needs for jobs to avoid job waiting queues and speed up the continuous integration process without resource limitations. This feature requires specific configuration of Gitlab Runner and it will act as an orchestrator for all virtual machines it creates. After autoscaling is configured properly on Gitlab Runner, virtual machines instances are created on demand and execute jobs on it. This technique allows optimizing the cost of virtual machine instances in different cloud providers.

2.2 Docker

Docker is an open source platform for building, shipping and running distributed applications across many machines, often with a variety of hardware and OS configurations. Docker provides an ability for a sandboxing environment capable of abstracting the specifics of the underlying host, without requiring editing the application source code and performance overhead.

2.2.1 Docker Machine problem

Docker Machine is a tool for preparing and managing Docker Engine. It provides methods to provision remote Docker hosts on different operating systems. In the role

[4] `https://aws.amazon.com/s3/`

[5] `https://azure.microsoft.com/en-us/services/storage/blobs/`

of hosts could be cloud providers or servers inside a data center. Docker creates virtual machines, inside which it installs Docker and automatically configures local Docker instances to use it with the remote Docker Machine manager [7]. However, the Docker Machine is deprecated [3] because the Docker team manages to replace it with Docker Desktop. Gitlab is currently maintaining its own fork of Docker machine for auto scaling runners.

2.3 Technologies

Before starting preparing testing environments, it is essential to choose the correct technology stack that will cover all functional and not-fuctional requirements of the pipelines. The right technology stack ensures rapid development, support and security of the end solution. The technologies required to be up-to-date and maintained by the software developer community in order to avoid security breaches and critical bugs in the applications.

2.3.1 Podman

Podman[6] is an open source Linux native tool for building, running and sharing distributed applications under the Open Container Initiative (OCI) compliant container standard runtime. The Podman is developed by RedHat and is used as an alternative to Docker. It is one of a set of command-line tools from RedHat designed only for developing and managing containers. The tool uses Buildah for building container images via Dockerfiles. The main advantages of Podman are daemonless [10] and rootless container runtime compared to Docker. The Podman command line interface (CLI) implements all the core Docker CLI commands. The architecture differences between Podman and Docker are illustrated in the Figure 1.

2.3.2 Buildah

Buildiah[7] is another tool set developed from RedHat which is designed for creating containers from the Open Container Initiative (OCI) image format or the upstream docker image format named Dockerfile.

2.3.3 Kaniko

Kaniko[8] is an open-source tool for building container images from Dockerfile developed by Google that is designed to execute image building from Dockerfile inside Kubernetes

[6] https://podman.io/
[7] https://buildah.io/
[8] https://github.com/GoogleContainerTools/kaniko

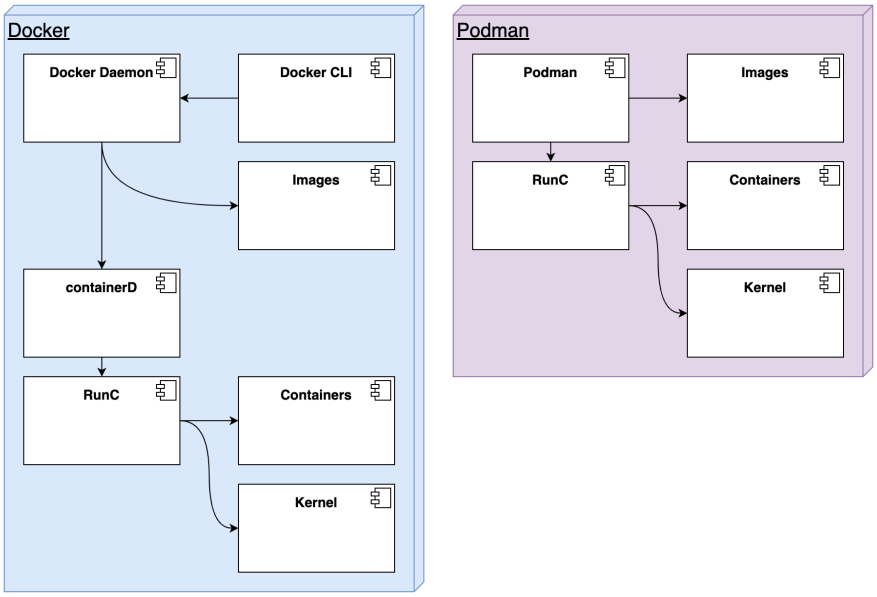

Figure 1. Architecture differences between Docker and Podman.

Cluster in an unprivileged environment.

2.3.4 Hetzner cloud provider

Hetzner Online GmbH[9] is a cloud hosting provider and data center operator. It offers Web, Dedicated and Virtual Private Server (VPS) hosting products. Variety of cloud servers with flexible packages, high availability and quick set up time makes this provider an ideal alternative to popular cloud providers. Affordable prices and large community and variety of open-source tools for it makes Cloud provider suitable as an environment for Gitlab Runner instances.

2.3.5 Azure cloud

Azure cloud[10] is a cloud computing platform maintained by Microsoft. The cloud provides many products for scalable solutions. Azure is chosen because of simplicity in setting up Kubernetes[11] environment and flexible payment options.

[9] `https://www.hetzner.com/`
[10] `https://azure.microsoft.com/en-us/`
[11] `https://kubernetes.io/`

3 Proposed design

This section gives an overview of Gitlab Runner implementations without Docker Machine that could be used in different popular cloud environments. The basic Gitlab Runner execution workflow is shown in the Figure 2.

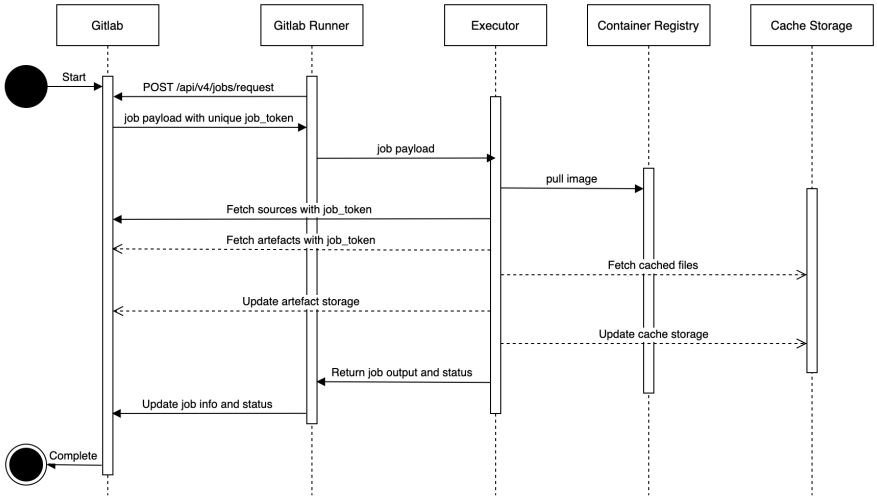

Figure 2. Gitlab Runner execution workflow.

3.1 Gitlab Runner on GNU/Linux

The Gitlab Runner on Linux system is a basic Gitlab Runner implementation that runs pipelines developed by Gitlab. It allows the use of different executors: docker, shell and others. Full list of available executors can be found on the Gitlab Documentation page. The main advantage of this runner is simplicity in set up, variety of code executors and small overhead in performance because the Gitlab Runner application is running directly on server instance.

However, the limitation of this solution is in instance resources and by default single jobs can be executed at once - it supports parallel job execution [9] that could significantly speed up pipeline with parallel, but runner itself operate as isolated processes and does not manage the server resources between the jobs that could potentially fail the jobs due to limit in resources. Another solution for running jobs in parallel on this type of runners is to use other runners if there are available runners connected to the Gitlab platform.

3.2 Gitlab Runner on Kubernetes Cluster

The Gitlab Runner on Kubernetes is an application developed by Gitlab that runs inside the Kubernetes cluster and uses its infrastructure to run pipelines. The architechture of the Runner inside Kubernetes is shown in the Figure 3. It creates a single temporary runner inside pod with a defined image via Kubernetes API for every job, runs code and destroys it after completing the job. The advantage of this runner is parallel job execution and auto scaling runner support. In addition, popular cloud providers have a native storage integration as cache service which could speed up job execution. Nevertheless, this type of runner requires minimal knowledge of Kubernetes for setting up, creating overhead for jobs that require Docker CLI.

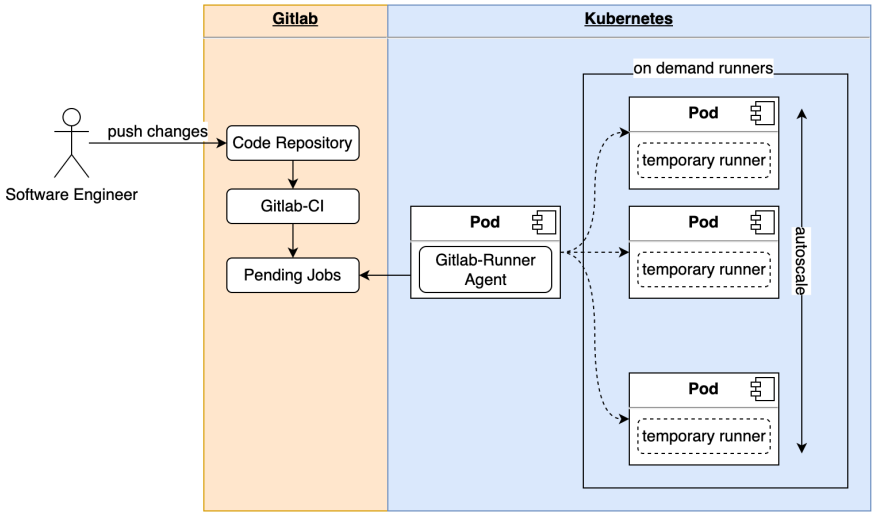

Figure 3. Gitlab Runner on Kubernetes architecture.

3.3 Gitlab custom executor for Amazon Fargate

The Gitlab community provides a driver for Gitlab Runners Custom Executor which supports Amazon AWS Fargate[12] instances for autoscaling runners implemented on golang. The instances themselves are running containers inside Elastic Container Service or Elastic Kubernetes Service without a VM layer which gives advantage in startup time and simplicity of maintenance. The arhitecture of this runner is shown in Figure 4. The project is under development and not recommended for production usage. The source code is available from the Gitlab repository[13].

[12] `https://aws.amazon.com/fargate/`
[13] `https://gitlab.com/gitlab-org/ci-cd/custom-executor-drivers/fargate`

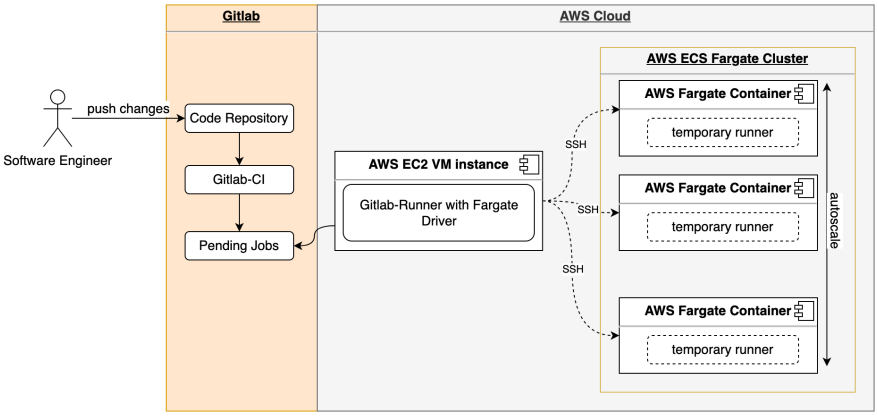

Figure 4. Architecture of auto scale Runner on AWS Fargate.

3.4 Gitlab custom executor for Openstack

The Red Hat Quality Engineering team provides a custom driver based on the Openstack platform implemented on Python. The solution is presented as an alternative to cloud providers since Openstack is an open source solution, it has an API for VM management and could be set up on a private server environment. The project is created as a proof of concept and is not production ready. The source code is available from the Red Hat Github repository[14].

[14] https://github.com/RedHatQE/ocp-gitlab-runner

4 Experiments

This sections gives an overview of different approaches of Gitlab Runner that contains autoscalable implementation Kubernetes Operator in Azure Kubernetes Service[15] and self hosted Kubernetes in Hetzner Cloud, and self hosted Gitlab Runner on virtual machine with lower abstraction complexity.

The pipeline benchmark of 3 stages and 7 jobs aimed to measure for each runner:

- every job average execution time - represent the runner and runner's configuration performance
- every job queue average duration time - represents the runner auto scaling performance
- pipeline average execution time - represent needed time to complete pipeline
- jobs execution in a single pipeline total duration time - represent the runner and runner's configuration performance

The main idea of using different environments is to compare simple in setting up and auto scalable approaches without a docker machine with a single runner instance on a dedicated virtual machine in pipeline execution time. The reason to benchmark different environments is to get a true overview of each solution advantages and disadvantages and prove it by using presented Gitlab Runner implementations.

4.1 Environments

In order to create a true overview of different runner implementations in a variety of environments with different cache configurations and benchmark them after, the description of configuration of each runner is provided in the subsections.

4.1.1 Gitlab Runner Operator in Azure Kubernetes Service

The Azure Cloud is chosen as an example of a prepared Kubernetes Cluster environment with scalable resources and different native integrations with the Gitlab platform. In the test environment kubernetes cluster is located in the Sweden region with auto scalable instances type Standard D2s v3[16] with 2 vCPU, 8 GiB Memory, 16 GiB of temporary storage. The cluster Kubernetes version is 1.22.11.

The Gitlab Runner configuration is declared in config.toml[17] that which could be rewriten. The Runner in the Azure Cloud Kuberentes (AKS) is configured to work with

[15] https://azure.microsoft.com/en-us/services/kubernetes-service/

[16] https://docs.microsoft.com/en-us/azure/virtual-machines/dv3-dsv3-series

[17] https://toml.io/en/

Azure Storage blob instance as Cache in pipeline between jobs to speed up execution time by caching compiled files and throwing cache to all jobs in this pipeline. The cache is located in the same region as the Kubernetes instance which also reduces job duration because of minimal network latency between cache and runner instances.

4.1.2 Gitlab Runner Operator in self hosted Kubernetes

Despite the popularity of cloud providers such as Azure Cloud, AWS and Google Cloud and other cloud providers with Kubernetes, there are organizations that could not use these solutions due to security policies that require them to use applications in isolated environments. In addition, the cost of instance performance of these types of providers is higher [11] than in smaller cloud providers with virtual private or dedicated servers.

Hetzner cloud providers have instances[18] with the same performance, but cost is multiple times smaller than in popular cloud providers. The community of these cloud providers have many popular open source tools for working with resources.

The Hetzner Kube[19] is an open source tool for setting up Kubernetes clusters in the Hetzner environment. The tool is written with Terraform script language. It supports mostly all features needed for production Kubernetes environments based on K3s[20] lightweight Kubernetes distribution developed by Rancher. The cluster is installed on Hetzner CPX21 instances with 3 vCPU, 4 GiB Memory, 80 GiB Disk storage and Kubernetes version is 1.24.3.

Gitlab Runner Operator is configured for using docker daemon as a service in pipelines. However, due to higher complexity of self hosted Kubernetes environment, there are issues with setting up self hosted distributed S3-Compatible storage for reducing execution time. Instead of using cache, in this environment is used artifact storage which is located on Gitlab platform service.

4.1.3 Gitlab Runner on a virtual private server

This environment is chosen as a generic Gitlab Runner solution with low level abstraction. The virtual private server is located in the Hetzner Cloud provider. The specifications of Hetzner CX31 instance: 2 vCPU, 8 GiB Memory, 80 GiB Disk storage. Operation system installed on instance is Ubuntu 22.04.

The Runner is configured to use a cache located on the same instance which dramatically reduces cache updating duration since the latency between cache and runner is minimal. The runner uses docker daemon natively running virtual server without overhead and stores job images locally without pulling them on every job run.

[18] `https://www.hetzner.com/cloud`
[19] `https://github.com/kube-hetzner/terraform-hcloud-kube-hetzner`
[20] `https://k3s.io/`

4.2 Backend application

The example of backend application is implements integration with SWAPI[21] - Star Wars API maintained by Pipedream. The application have one 1 rest API endpoint, 1 basic unit test and 1 integration test and is written on JavaScript using Node.js[22] framework. The application realises the API gateway pattern [6] and architecture of application is shown on Figure 5.

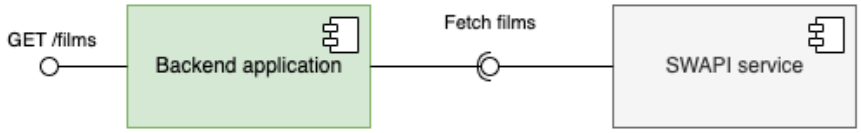

Figure 5. Backend application integration architecture diagram.

4.3 Pipeline

The pipelines are mostly the same in all test environments. The main difference is Docker service in Kubernetes test environments for building images. Pipeline have most common stages in all development projects: dependency installation, test and building docker image for deployment. (Figure 6)

- install stage - runs backend application dependency installation job.
- test stage - includes unit and integration test jobs of Node.js application.
- build stage - it has 4 different approaches to building images: kaniko, buildah, docker and docker multi architecture image build.

The install stage runs the dependency installation of backend application written on Node.js via npm package manager inside alpine container with 18-alpine version.

The test stage runs in parallel the integration and unit tests written on Mocha.js framework[23]. The dependencies needed in this stage are cached in Gitlab Artifactory storage or other cache storage.

The build stage runs 4 jobs in parallel with different container builders. The idea of using different builders is to compare different approaches for creating images with application inside and compare the runners performance with these image building jobs.

[21] `https://pipedream.com/apps/swapi`
[22] `https://nodejs.org/en/`
[23] `https://mochajs.org/`

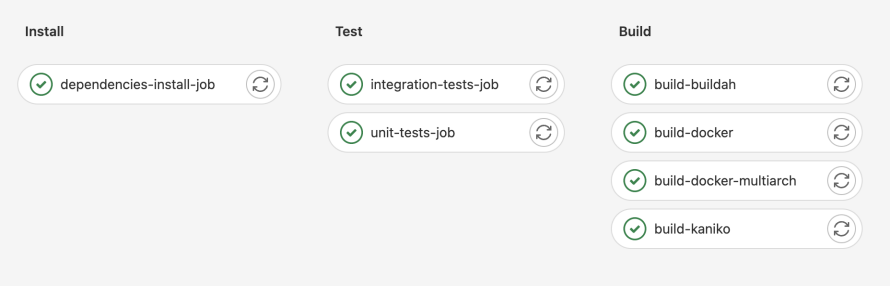

Figure 6. Benchmark pipeline in Gitlab Web UI.

5 Results

In order to receive correct pipeline duration time and corresponding jobs execution time, the Gitlab pipelines were running 5 times in every environment with code change in the application. The information about jobs and corresponding pipelines is retrieved from Gitlab via REST API calls[24].

The benchmark result of runners in different enviroments are shown in Figures 7, 8, 9, 10.

The Figure 7 represents the average duration time of different job execution. The duration of dependency installation on runners on Linux and in AKS have shown approximately the same result because the runner on VM utilize local cache and the runner in AKS use Azure Blob storage as a cache while runner in self hosted Kubernetes is utilizing Gitlab Artifactory storage as cache. From the Figure 9 can be seen that by summarizing all the job durations the fastest runner is runner installed on a server with Linux OS. However, this runner does not support parallel job execution and it is not automatically scalable, so the runner installed in AKS showed the best result with minimal pipeline duration time as can be seen from Figure 10. The worst result in job execution performance showed the runner in self hosted Kubernetes since it does not have any cache storage between temporary runners, but on the other hand, according to the Figure 8 it shown the better auto scaling performance than the runner in AKS. The minimum time (Figure 9) of all jobs execution is shown by Gitlab Runner on Linux OS server because it has the minimum system abstraction overlay compared to other runners.

[24] `https://docs.gitlab.com/ee/api/jobs.html`

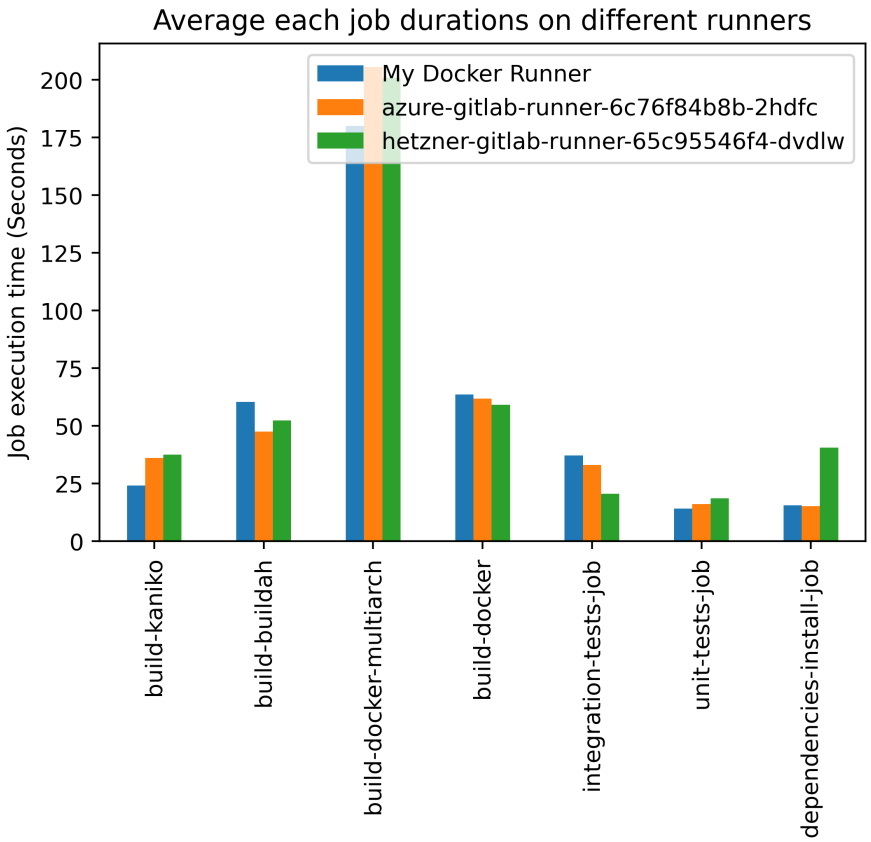

Figure 7. Average job duration on different runners.

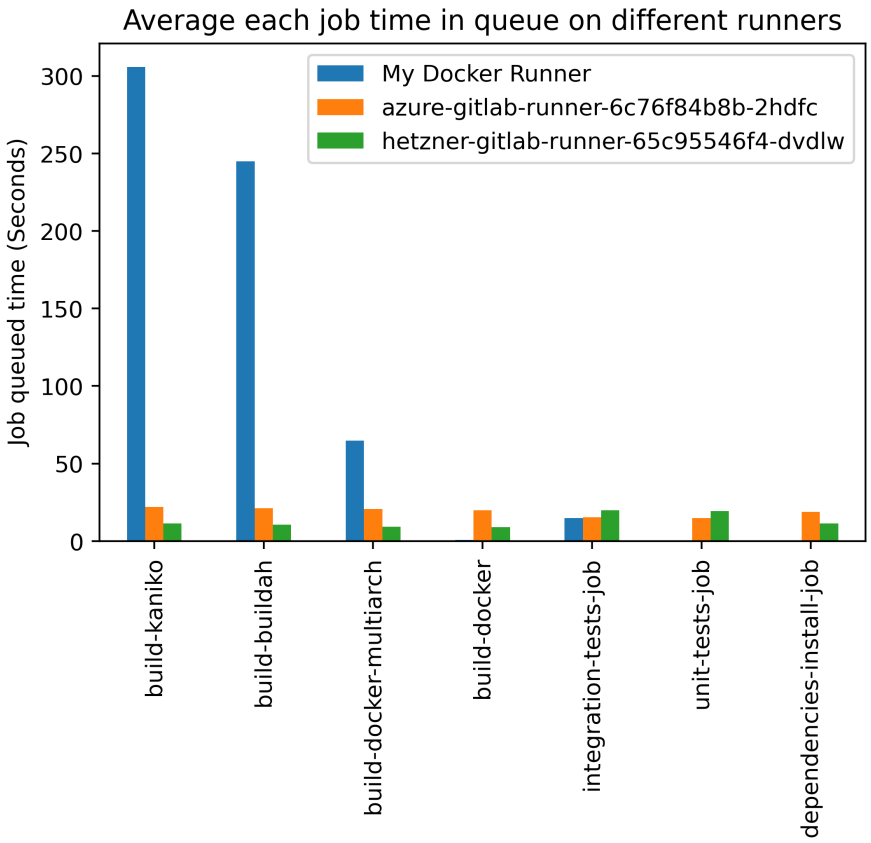

Figure 8. Average jobs queued time on different runners.

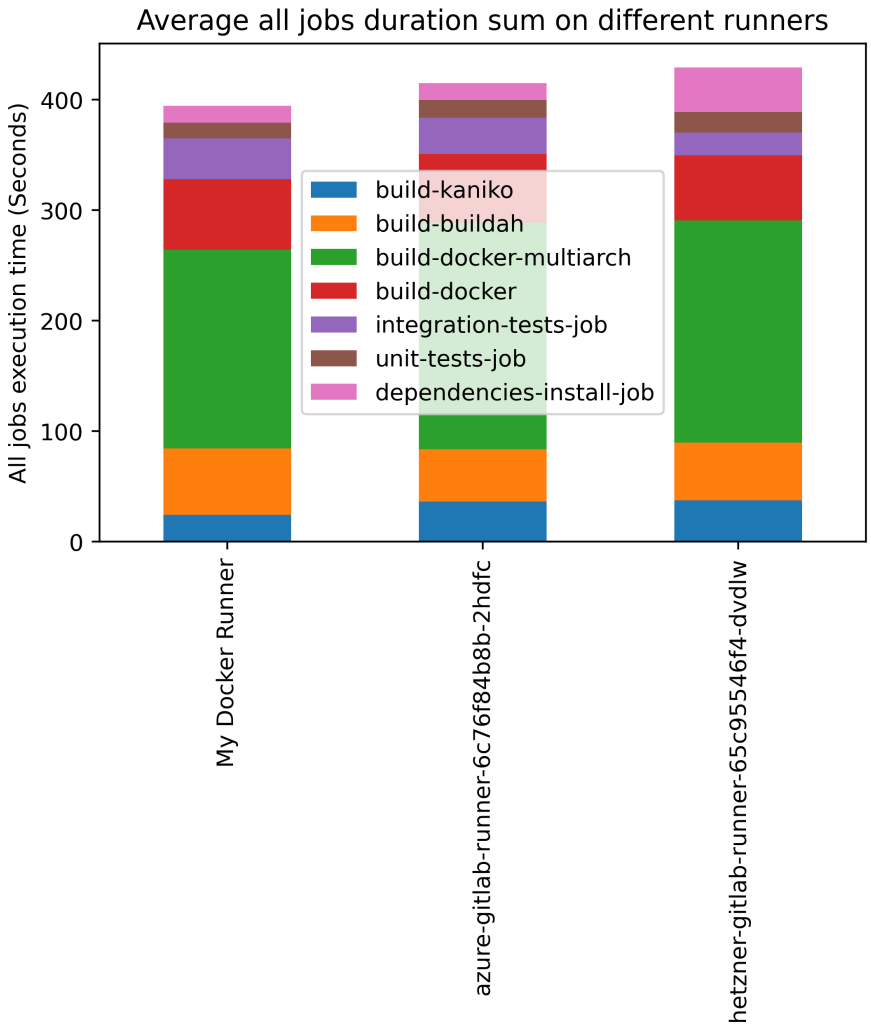

Figure 9. All jobs execution duration on different runners.

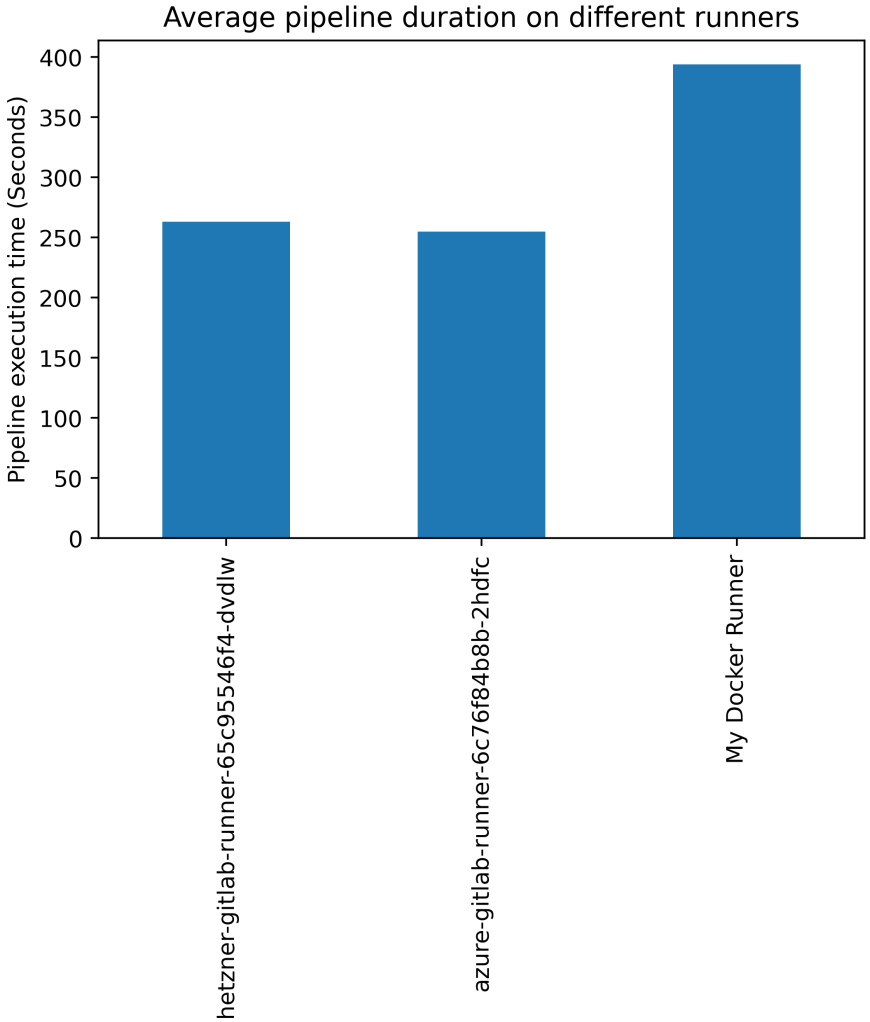

Figure 10. Average pipeline duration on different runners.

6 Conclusion

This section summarizes the theoretical and practical part of the thesis, describes possible improvements of Gitlab Runner benchmarking in different environments and limitations of tools which are used in comparison tests.

6.1 Summary

This thesis describes Gitlab Runner auto scaling implementations without Docker Machine and compares the performance of autoscaling Gitlab Runner in different Kubernetes environments and on a Linux virtual server. During the research, different open source solutions are found and the workflow and architecture of those solutions is described in detail. The limitations and advantages of each approach are highlighted and described.

The technologies used in the benchmark are described and the best practices for that solutions are used in order to provide a true overview of different environment performance. The example application was implemented on Node.js framework and implements Gateway Api pattern. For the benchmark purpose a pipeline was created with 7 jobs in 3 stages where 4 jobs were related to different docker image building approaches.

In the performance benchmark are used environments in Hetzner Cloud and Azure Cloud for setting up auto scaling Gitlab runner in Kubernetes and on Linux OS virtual private server. The instance specifications for Kubernetes cluster is chosen approximately with the same performance in each cloud provider. The Virtual Private Server instance is located in Hetzner Cloud Provider. The manuals are written for set up runners in every tested environment in order to reproduce the benchmark results and reduce the time needed for configuring Gitlab Runner.

During the research were uncovered limitations of using Gitlab Proxy service used for caching images used in the Gitlab pipeline jobs. The main limitation is authentication to different private container registries such as Google Container Registry or Quay Container Registry by providing credentials in Gitlab UI or via configuration file and the instance of proxy service cannot be automatically autoscale to avoid network limitation bandwidth.

6.2 Future work

As for future work, there are a lot of things that still need more research. Firstly, the tests pipeline could be populated with integration tests services such as databases or other services that would be run in the isolated environment specially for the pipeline.

In addition, the Gitlab Runner on Kubernetes configuration allows to specify CPU architecture of cluster nodes for popular cloud providers. In 2nd quarter of 2022 year Azure introduces own instances [5] with ARM CPUs. In 3rd quarter of 2022 year Google

announces own virtual machines [4] based on ARM CPUs. This could reduce the pipeline completion time since CPUs on ARM architecture sometimes are more powerful than CPUs on x86. Also, these instances could be useful for reducing time of building images for multiple architecture platforms.

Furthermore, different cache implementations can be also compared to each other. The Gitlab Runner currently supports S3-Compatible storages, Azure Blob Storage, Google Container storage.

Last but not least, the job performance consists of many factors - one of them is the performance of the runner where the jobs are running. There could be written a manual for choosing instance types with suitable performance for the nodes of Kubernetes Cluster with Gitlab Runner.

References

[1] Use gitlab's docker-machine fork in docker images. URL: https://gitlab.com/gitlab-org/gitlab-runner/-/issues/4916, Nov. 2019. Accessed 29.07.2022.

[2] Autoscaling provider for gitlab runner to replace docker machine. URL: https://gitlab.com/groups/gitlab-org/-/epics/2502, Jan. 2020. Accessed 12.07.2022.

[3] Chris Crone. Deprecate docker machine. URL: https://github.com/docker/roadmap/issues/245, Aug. 2021. Accessed 26.07.2022.

[4] Dylan Martin. Arm in the cloud definitely a trend now with google cloud's embrace. URL: https://www.theregister.com/2022/07/14/arm_cloud_trend/, Jul. 2022. Accessed 25.07.2022.

[5] Timothy Prickett Morgan. The looming arm server battle between aws and microsoft. URL: https://www.nextplatform.com/2022/04/05/the-looming-arm-server-battle-between-aws-and-microsoft/, Apr. 2022. Accessed 25.07.2022.

[6] Chris Richardson. Pattern: Api gateway / backends for frontends. URL: https://microservices.io/patterns/apigateway.html. Accessed 05.08.2022.

[7] Elton Stoneman. *Learn Docker in a Month of Lunches*, chapter 15. Configuring Docker for secure remote access and, pages 272–293. Manning Publications Co, 2020.

[8] Brian Wald and Darwin Sanoy. An sa story about hyperscaling gitlab runner workloads using kubernetes. URL: https://about.gitlab.com/blog/2022/06/29/a-story-of-runner-scaling/, Jun. 2022. Accessed 29.07.2022.

[9] James Walker. How to manage gitlab runner concurrency for parallel ci jobs. URL: https://www.howtogeek.com/devops/how-to-manage-gitlab-runner-concurrency-for-parallel-ci-jobs/, Mar. 2022. Accessed 05.08.2022.

[10] Daniel Walsh. Podman in action: The next generation of container engines. Chapter 6. Rootless containers, 2022.

[11] Peter Wayner. 9 low-rent cloud providers to challenge aws, azure, and gcp. URL: https://www.infoworld.com/article/3656688/9-low-rent-cloud-providers-to-challenge-aws-azure-and-gcp.html, Apr. 2022. Accessed 27.07.2022.

Appendix

I. Repository

This Appendix contains a link to Gitlab repository with enviroment setup and manuals to reproduce benchmark results.

Source code can be accessed from here: `https://gitlab.com/timofei.ganjushev/gitlab-runners`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Timofei Ganjusev**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Comparison of Gitlab Runner implementations without Docker machine for cloud providers,

   supervised by Shivananda Rangappa Poojara.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Timofei Ganjusev
5/8/2022