

Comparing Different CI/CD Pipelines



Degree Programme in Business Information Technology

Hämeenlinna University Center

Autumn, 2021

Joni Virtanen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli vertailla erilaisia jatkuvaan integraatioon ja jatkuvaan julkaisemiseen saatavilla olevia järjestelmiä. Työn aikana tarkasteltavia tuotteita olivat Jenkins, Atlassian Bamboo, Azure Pipelines, Google Cloud Build, AWS CodeBuild ja CodePipeline, GitLab Actions, GitHub CI/CD ja Bitbucket Pipelines. Opinnäytetyön toimeksiantajana oli Ambientia Oy.

Opinnäytetyön teoriaosa avaa jatkuvan integraation, jatkuvan julkaisemisen ja julkaisuputkien käsitteet. Tämän jälkeen teoriaosuudessa käsitellään yleisellä tasolla eri ratkaisujen tarvittavat komponentit julkaisuputken luomiseksi. Opinnäytetyö on tutkimuksellinen. Aineistoa kerättiin eri palveluiden omista dokumentaatioista. Käytännön osuudessa toteutettiin mahdollisimman identtinen julkaisuputki eri palveluihin teoriaosuudessa kuvattujen tietojen pohjalta.

Tutkimuksessa havaittiin erilaisten palvelujen olevan useimmiten toistensa kaltaisia ja saavuttavan niiltä odotetun putken tavoitteet. Tutkimustyön ja sen pohjalta toteutettujen käytännön toteutuksien osalta yksittäisen tuotteen suosittelu on lähes mahdotonta ja useimmiten tuotteen käyttäjän tehokkuus onkin pitkälti kiinni hänen kokemuksestaan kyseisen tuotteen kanssa.

Avainsanat Jatkuva integraatio, jatkuva julkaiseminen, julkaisuputki

Sivut 54 sivua ja liitteitä 1 sivua

Author Joni Virtanen

Year 2021

Subject Comparing different CI/CD Pipelines

Supervisors Lasse Seppänen

ABSTRACT

The purpose of this thesis was to provide a comparison between different available continuous integration and continuous deployment services. During the process different products that were examined were Jenkins, Atlassian Bamboo, Azure Pipelines, Google Cloud Build, AWS CodeBuild and CodePipeline, GitLab Actions, GitHub CI/CD and Bitbucket Pipelines. The thesis was commissioned by Ambientia Oy.

The theoretical part of this thesis aimed to explain the concepts of continuous integration, continuous delivery and CI/CD pipelines. Afterwards, in the theory part different services are addressed on a high level in regards of the needed components in order to create a working CI/CD pipeline. The thesis is research-based. The material was collected from each services supplier's documentation. The practical part of this thesis focused on reproducing a near identical pipeline for each environment, based on the information described in the theoretical part of this thesis.

During this study it was observed that implementations between the various services are most often similar, and they all should be able to accomplish the expectations of the simplistic pipeline. Based on the research work and the various implementations created, it is near impossible to consider one product better than the other, and often the efficiency of the user is more dependant on the familiarity of the product.

Keywords Continuous integration, Continuous delivery, Pipeline

Pages 54 pages and appendices 1 pages

Sanasto

Build	Software build, compiled software that is ready for use
Development Lifecycle	Process for application development including different lifecycle phases such as planning, implementing and maintenance
Development Env.	Development environment, collection of tools and processes used while developing the software
Integration Build	Automatically run build on recent modifications to source code
SCM	Source Code Management, version control, nowadays most often implemented via git. Used to manage source code versions and history
Branching Strategy	Defined process on how to utilize branches in SCM
Version Control	see SCM
Repository	Storage location, in this thesis for both source code and for integration builds
Atlassian Connect	Atlassian provided software technology for extending functionality of Atlassian software
Open-source	Source code that is freely available. Different licenses exist.
MIT license	Permissive license allowing commercial usage, distribution, modification and private use, while requiring preservation of license and copyright notice. Provides no warranty or liability.
Docker	OS-level virtualization technology.
Dockerfile	Text document that contains instructions for building a docker container image.
Docker image	A file that contains instructions to run a specific container.
Container	Isolated environment containing a software and its dependencies.
JRE	Java Runtime Environment, software layer that includes all resources that a specific Java application requires.
DSL-syntax	Domain-specific language, a specific language that applies to a specific piece of software.

Pipeline as code	Pipeline implementation stored in a text file. Allowing for storage within a repository, providing single source of truth for the pipeline.
AWS	Amazon Web Services, Amazon cloud computing platform
Azure	Microsoft cloud computing platform
AWS KMS	AWS Key Management Service, AWS managed service for managing cryptographic keys
AWS EBS	AWS Elastic Block Service, high performance block storage service.
AWS Secrets Manager	AWS managed service for managing secrets, providing rotation, encryption and API for interoperation with other services.
AWS S3	AWS Simple Storage Service, Amazon managed object storage.
IDE	Integrated development environment, software for development and building of applications.
IAM	Identity and Access Management, service for controlling users and groups access to resources.
Agent-software	Software making server resources available for CI/CD server. Remote agent describes a remote server, while local agent describes the CI/CD server itself.
Rotation policy	Policy, to rotate certain objects e.g., AWS secret manager resources. Rotating an object refers to refreshing the value of a certain object to provide additional protection against possible leaks.
Replication policy	Policy, to replicate certain objects e.g., AWS secret manager resources. Replicating the object refers to making the object more available by replicating it to another location.

Contents

1	Introduction.....	1
2	Continuous Integration and Continuous Delivery.....	3
2.1	Continuous Integration	3
2.2	Continuous Delivery.....	4
2.3	CI/CD Pipeline	4
3	CI/CD Systems.....	6
3.1	Jenkins.....	6
3.2	Atlassian Bamboo.....	7
3.3	AWS CodeBuild and CodePipeline	8
3.4	Azure Pipelines.....	9
3.5	Google Cloud Build.....	11
3.6	Bitbucket Pipelines.....	12
3.7	GitHub Actions	14
3.8	GitLab CI/CD	17
4	Implementation objective	19
5	Implementation	21
5.1	Jenkins.....	21
5.1.1	Integration with SCM	21
5.1.2	Managing credentials.....	21
5.1.3	Pipeline.....	22
5.1.4	Integrations	23
5.1.5	Costs	25
5.2	Bamboo	25
5.2.1	Integrating with SCM.....	25
5.2.2	Managing credentials.....	26
5.2.3	Pipeline.....	27
5.2.4	Integrations	28
5.2.5	Costs	30
5.3	AWS CodePipeline and CodeBuild	30
5.3.1	Integrating with SCM.....	30
5.3.2	Managing credentials.....	30
5.3.3	Pipeline.....	31
5.3.4	Integrations	32

5.3.5	Costs	33
5.4	GCP Code Build.....	34
5.4.1	Integrating with SCM.....	34
5.4.2	Managing credentials.....	35
5.4.3	Pipeline.....	35
5.4.4	Integrations	36
5.4.5	Costs	37
5.5	Azure Pipelines.....	38
5.5.1	Integrating with SCM.....	38
5.5.2	Managing credentials.....	38
5.5.3	Pipeline.....	39
5.5.4	Integrations	40
5.5.5	Costs	41
5.6	GitHub Actions	42
5.6.1	SCM	42
5.6.2	Managing credentials.....	42
5.6.3	Pipeline.....	43
5.6.4	Integrations	44
5.6.5	Costs	44
5.7	GitLab CI/CD	45
5.7.1	SCM	45
5.7.2	Managing credentials.....	45
5.7.3	Pipeline.....	46
5.7.4	Integrations	47
5.7.5	Costs	48
5.8	BitBucket Pipelines.....	48
5.8.1	SCM	48
5.8.2	Managing credentials.....	48
5.8.3	Pipeline.....	49
5.8.4	Integrations	50
5.8.5	Costs	51
6	Key findings	52
7	Summary.....	55
	References.....	56

Figures, codes and tables

Figure 1 Different phases of a pipeline illustrated (Red Hat, n.d.).....	5
Figure 2 Atlassian Bamboo build plan overview. (Atlassian, n.d.)	7
Figure 3 Azure Pipeline execution overview (Microsoft, 2021b)	10
Figure 4 Structural overview of Bitbucket Pipelines (Atlassian, n.d.-g)	13
Figure 5 GitHub Actions workflow file structure visualized (GitHub, n.d.-c)	15
Figure 6 GitHub Actions workflow structure (GitHub, n.d.-c).....	15
Figure 7 GitLab CI/CD pipeline status visualized. (GitLab, n.d.-a).....	17
Figure 8 CI/CD Pipeline structure for this thesis' implementation	19
Figure 9 OAuth credential creation for Jenkins plugin in Jira Cloud	24
Figure 10 Jenkins configuration for Jira integration.....	24
Figure 11 Configuration for Slack notification plugin.....	25
Figure 12 Bamboo tasks declarations.....	28
Figure 13 New Webhook template	29
Figure 14 Defining new build notification for the Pipeline	29
Figure 15 AWS ChatBot integration in #aws-test channel	33
Figure 16 Azure Pipelines Slack Integration installation	40
Figure 17 Azure Pipelines notifications about the status changes of the pipeline.	41
Figure 18 Slack message notifying about the successful subscription to repository.	44
Figure 19 GitLab Integration test post using Incoming Webhook Slack app.	47
Figure 20 Description of Slack notification settings within a repository	51
Code 1 Basic YAML structure of GitLab CI/CD.....	17
Code 2 Definition Jenkins Pipeline, agent and environmental variables	22
Code 3 Definition of the pipelines pre-build stage.	23
Code 4 Jenkins Pipeline executing Docker image build and push to DockerHub	23
Code 5 Cloudbuild.yml step to install npm dependencies inside a container	36
Code 6 Reading environment secrets in Google Cloud.....	36
Code 7 npm package.json script to print environment variables to .env file	36
Code 8 Description of a task to install Node.js version 12.....	39

Code 9 Task to run <code>npm install, npm test</code> using the secret variables	39
Code 10 Task to build and push the container image to Docker Hub container registry	39
Code 11 GitHub Actions trigger configuration	43
Code 12 Basic structure of GitHub Actions workflow	43
Code 13 Configuration to cache <code>node_modules</code> directory between different containers.	46
Code 14 Job structure for GitLab CI/CD	46
Code 15 YAML structure for Bitbucket Pipelines	50
Code 16 Step to build and push docker image to Docker Hub container registry.....	50
Table 1 Build minute multipliers for different operating systems (GitHub, n.d.-a)	16
Table 2 Jenkins project credentials configurations	22
Table 3 Description of variables stored within Bamboo	26
Table 4 Description of Credentials stored in Secret Manager	35
Table 5 Pipeline cost breakdown in Google Cloud Platform environment.....	37
Table 6 Description of secrets used in GitHub	42
Table 7 Description of variables in GitLab	45
Table 8 Description of Variables for Bitbucket Pipelines	49

Liitteet

Liite 1	Aineistonhallintasuunnitelma
---------	------------------------------

1 Introduction

Nowadays, software projects tend to grow and get more complicated, and the integration process becomes more error-prone as the time goes on. To combat such a situation, the feedback loop must be tight. Possible errors should surface as early as possible and be as easily pinpointable as possible. This leads to the scenario, where we may want to fail fast, and fail often. If possible, we would like to fail in an environment that is as close of a replication of the production environment, for the software we are aiming to run, as possible. This is where Continuous integration and continuous delivery step in. The aim of these services is to provide an error free, repeatable process automatically, to provide consistency.

The main point of interest for this thesis is to produce a brief overlook of the currently available CI/CD products for the authors employer. In this study, we will focus on creating a simple CI/CD workflow and attempt to reproduce it within each of the major service providers selected for this thesis; Jenkins, Bamboo, AWS CodePipeline, Azure Pipelines, Google Cloud Build, GitHub Actions, GitLab CI/CD and Bitbucket Pipelines. These CI/CD implementations will be implemented in a way that should allow for safe storage of sensitive credentials or variables, provide notifications on the build statuses to Slack and update related ticket statuses on Jira Cloud. Costs should be assessed and compared, while trying to get a sense of the user-friendliness of the services. The thesis is commissioned by Ambientia Oy, authors current employer.

In the theory chapters of this thesis, we will briefly touch on the subjects of continuous integration, continuous delivery and take a brief overlook at the service providers implementations. This is in order to provide a general understanding on the matter we are about to delve into. By no means is this thesis meant to be considered an absolute description of the implementation in a particular environment, as there are multitude of variations that could be utilized in e.g., AWS or GCP to complete the same job.

This thesis is a research-based study on differences of different products. Materials for this thesis were gathered mostly from the service providers documentation, as there is no better

place to learn of the components than from the creator itself. This information was then built upon the basic principles of the CI/CD workflows to build a minimalistic prototype process.

The research questions are:

- How do the different products differ from each other?
- How could the Pipelines be used in Ambientia?
- Is there a single product that is clearly superior to the rest?

2 Continuous Integration and Continuous Delivery

This chapter discusses the concepts of the Continuous Delivery, Continuous Integration, and the possible Pipeline implementations on a high level. The aim is to provide an understanding on the concepts that are frequently used in the later chapters.

Fowler states that one of his first experiences with a large software project that had been in the development for a couple of years and was in course of integration for several months now, with no clear completion date known. He then states learning that integration was a “long and unpredictable process” even though it does not need to be, and that the integration can be treated as a non-event. (Fowler, n.d.)

2.1 Continuous Integration

Continuous Integration (later CI) is a software development process, that provides agile development teams with the benefit of making consistent builds and finding possible issues as early as possible in the development lifecycle. CI achieves this by automating the build process, thus providing a consistent development lifecycle with minimal user interactions. (Nikhil, 2017)

Applying effective CI requires team’s commitment, as it is fundamentally just a set of guidelines that in the end produce a build. To achieve this, teams must work consistently throughout the development lifecycle of the software, as CI servers usually pull the software source code from a single repository or a branch within that repository. Pulled revision of the source code is then build into an integration build. Integration builds should occur frequently to detect integration errors as early as possible. (Duvall et al., 2007a)

Working CI enables teams to achieve better project visibility and have greater confidence on their product by providing trends about the current situation of the project based on the build’s overall quality. This also builds confidence in the software product as the tests are run on every build automatically, to verify the correct state of the software prior to build succeeding, thus reducing the overall risks. (Duvall et al., 2007b)

2.2 Continuous Delivery

Continuous Delivery (later CD) is a software development process that extends CI by automatically preparing software for release to production environment. Often CD pipelines deploy successfully build software to pre-production environments automatically. This allows development teams to run more versatile tests, compared to unit tests, before deploying the software to production, with more confidence in the overall performance of the software systems performance. This procedure allows the development teams to have a production ready software ready for deployment. Continuous Delivery can be further extended into Continuous Deployment. Continuous Deployment aims to make sure that the production environment is always running the latest version of the software, with the latest features and improvements. (Amazon Web Services, n.d.-g)

2.3 CI/CD Pipeline

Pipeline is an abstraction, or an execution plan, containing the steps required for both CI and CD processes. Pipelines are used to improve the overall quality of releasing software. Pipelines most often provide statistics on how the status of the projects builds, thus providing monitoring and valuable information on the status of the project on integration and test phases to the development team. (Red Hat, n.d.)

Red Hat describes an example of CI/CD Pipelines as consisting of three different phases, these are further demonstrated in Figure 1. As described by Red Hat, in an example of the Pipeline (Figure 1) starting with continuous integration phase, that consists of build, test and merge steps. After success on these steps, the continuous delivery phase starts. At the end of CD phase, the product should be at a state where it is deemed stable and should be deployable to production at any convenient time, or automatically, to achieve continuous deployment. (Red Hat, n.d.)

Figure 1 Different phases of a pipeline illustrated (Red Hat, n.d.)



3 CI/CD Systems

GitLab's 2020 Global DevSecOps Survey implies that finding the right toolset for the teams working methodology results in more efficiency during projects ongoing development, while ensuring high quality standards for the work as well as simultaneously reducing the workload on individual project personnel. (GitLab, n.d.-a)

An article by JFrog also supports the ideology behind finding the right tools, practices and culture for your team. Article includes a list of worthy CI/CD tools to consider for beginners, from which GitLab and GitHub Actions are also tested within this thesis' scope. (JFrog, n.d.)

3.1 Jenkins

Jenkins is an open source-based automation server software, licensed under MIT-license. Jenkins is available as native system packages, Docker images and runnable as standalone by using Java Runtime Environment. (Jenkins.io, n.d.-a)

Jenkins Pipeline is a collection of plugins that support the implementation and integration of CI/CD pipelines into Jenkins. Pipeline allows organizations to define their integration and delivery pipelines via Pipeline DSL-syntax, by creating a text-file named "Jenkinsfile" (Jenkins.io, n.d.-b). "Pipeline-as-code" allows processing the pipeline as part of the project, storing it in version control and reviewing it as any other code. Using Jenkinsfile offers several benefits including, but not limited, to creating automatic builds for specific actions in all branches, audit trails for the pipeline and single source of truth for the pipeline that is stored in SCM along with other project files. Pipelines can be defined in a declarative or scripted format via Jenkinsfile. (Jenkins.io, n.d.-e)

Jenkins credential management is provided by default via Jenkins' Credentials Binding plugin. Credentials stored by Credential Binding plugin can then be accessed from anywhere within the application (Global credentials) or access to the credentials can be specified per pipeline item or Jenkins user or group. Credentials stored within Jenkins are stored, in an encrypted format, on the controller Jenkins instance. Credentials can be accessed by using

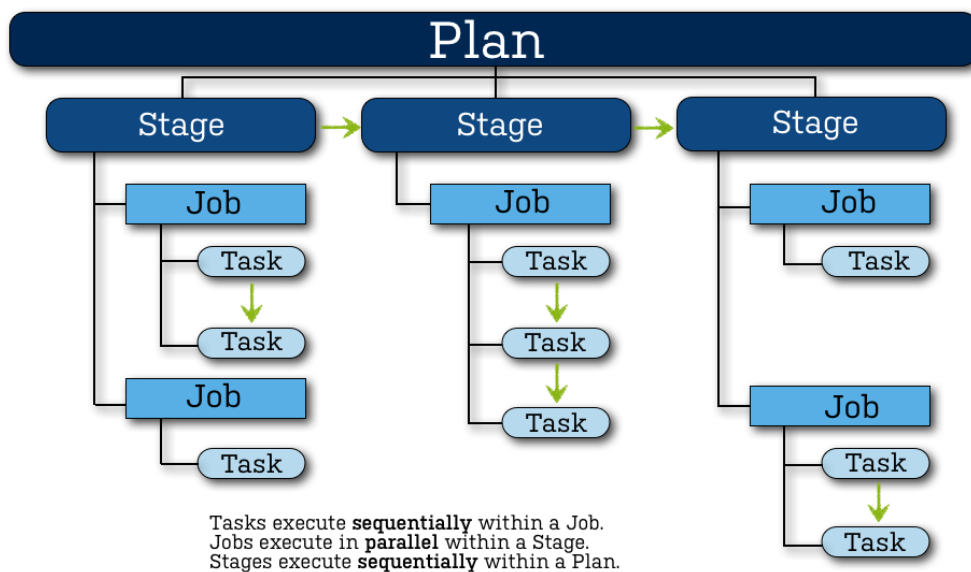
their variable names, thus minimizing the possibility of accidentally revealing sensitive secrets to end users. (Jenkins.io, n.d.-f)

Jenkins allows integration with Jira and Slack by plugins. Jira plugin allows for integration between Jenkins and Jira to allow for build status to be communicated to relevant Jira issues. Slack Notifications plugin provides the ability to send messages to certain channels or certain persons about the status of build pipeline. (Jenkins.io, n.d.-c, n.d.-d)

3.2 Atlassian Bamboo

Atlassian's closed source CI/CD server, offering development teams with reporting tools and visibility and control over release artifacts and environments on top of the usual CI/CD options. Bamboo is only available as an on-premises installation. Bamboo schedules and coordinates the work that is required for the integration builds to complete. Bamboo supports multiple agents and parallelism between them to provide fast and efficient builds (Figure 2). Bamboo Data Center license cost is calculated based on the number of remote agents to be installed (Atlassian, n.d.-b). (Atlassian, n.d.-o, n.d.-n)

Figure 2 Atlassian Bamboo build plan overview. (Atlassian, n.d.-n)



Bamboo Specs allows for users to manage their build pipelines by configuration-as-a-code methodology. Bamboo Specs provides support for YAML-syntax configurations, as well as a Java-based configuration language. Bamboo Specs in YAML-syntax works by defining the pipeline in `bamboo.yaml` file in the `<repository-root>/bamboo-specs` directory. (Atlassian, n.d.-e, n.d.-c)

Atlassian Bamboo uses the word 'plan' to describe pipelines. Plans define the triggers for the integration pipeline to start, notifications, variables and the source repository for the project. Plan can have multiple stages; each stage can have multiple jobs. Jobs then group up tasks, that should contain a singular command to run on the pipeline. Jobs can be completed in parallel within a stage. (Atlassian, n.d.-h)

Credentials can be managed by using the plan variables. These variables are only accessible from within the projects plan and can be overwritten while manually running the build, should the user wish to do so. Credentials for Bamboo Specs can be provided by using the Bamboo Specs encryption, this allows for the credentials to be encrypted. For credentials to be considered as secrets, they will need to include one of the following phrases in the variable name: "password", "passphrase", "secret", "sshkey". (Atlassian, n.d.-d)

Bamboo can be integrated with Jira by using an application link. This integration allows for example Bamboo to fetch linked Jira issues from Jira or provide information on build statuses or deployment information to Jira. Slack Notifications for Bamboo provides an integration to deliver similar status reports on build status or jobs that are timing out or hung up to Slack channels. (Atlassian, n.d.-m, n.d.-l)

3.3 AWS CodeBuild and CodePipeline

AWS CodeBuild (later CodeBuild) is AWS managed CI service. CodeBuild scales automatically based on the need of the project. CodeBuild allows source code to be pulled from AWS CodeCommit (later CodeCommit), GitHub, Amazon ECR, or Amazon S3. CodeBuild is designed to run the integration phases steps: compile source, run test and produce deployable product. After successful build, changes can be deployed using AWS CodePipeline (later CodePipeline) that integrates multiple managed services for deploying

the product such as AWS CodeDeploy, AWS Elastic Beanstalk, Amazon Elastic Container Service, or AWS Fargate. (Amazon Web Services, n.d.-a)

Pipelines are fully customizable through workflow modelling, a graphical tool that lets you configure series of stages, each consisting of jobs. CodePipeline allows users to manage their AWS infrastructure using AWS CloudFormation. This can also be applied to the serverless environment running in AWS. Users can trigger an AWS Lambda function from CodePipeline at any point of Pipeline execution, thus allowing for greater variation of different operations during the pipeline execution. AWS provides open-source agent that allows for user to hook their own servers to the execution of pipeline. (Amazon Web Services, n.d.-c)

AWS provides wide variety of IAM policies and complete customisability of IAM roles and policies to provide access control mechanisms for inter service operability, e.g., the connectivity between CodeBuild and AWS S3 service (Amazon Web Services, n.d.-f). To manage credentials within the pipeline AWS Systems Manager Parameter Store can be used. AWS Systems Manager Parameter Store provides secure and scalable way of storing and accessing secrets within AWS. AWS IAM policies can be used to provide easily auditable and granularly configurable access control within Parameter Store. (Amazon Web Services, n.d.-e)

Costs for AWS CodePipeline and CodeBuild have many variables that influence the final pricing of the pipeline. As for the CodePipeline itself there is a cost for an active pipeline to be billed monthly. CodeBuild is billed depending on the instance size, operating system type and build minutes. On top of these, additional charges may be generated for example by the usage of S3 to store the artifacts of the pipeline, Parameter Store to store the credentials, or for data transfer out of the Amazon Web Services backbone network. (Amazon Web Services, n.d.-d, n.d.-b)

3.4 Azure Pipelines

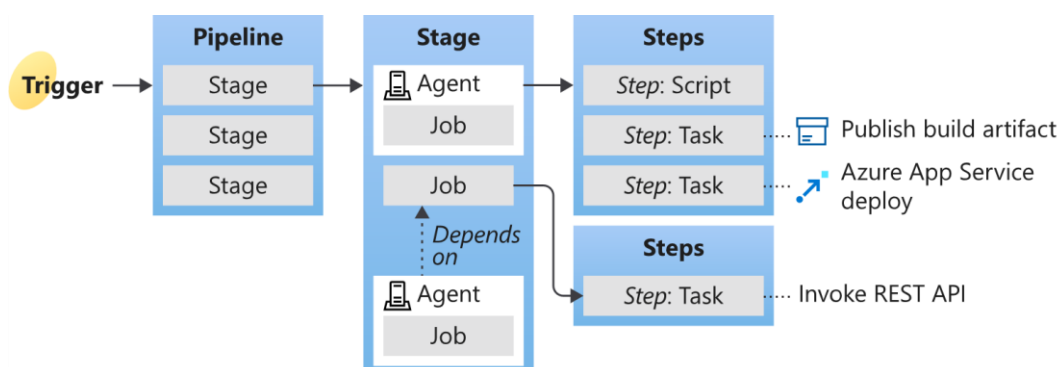
Microsoft managed implementation combining both CI and CD Pipelines into one service. Azure Pipelines allows source code to be pulled from GitHub, GitHub Enterprise, Bitbucket Cloud, Subversion, or any other Git repository. It enables the user to run full CI/CD Pipeline

through the steps from integration all the way to the continuous deployment phase. Deployments can be targeted to multiple different target types, including virtual machines, whether on-premises or in the cloud, containers and PaaS services (Microsoft, n.d.-f).

Microsoft uses an agent software to build and deploy packages according to the pipeline rules. As the pipeline runs, agent is processing different jobs from the pipeline. Agents can be either Microsoft-hosted or self-hosted. Microsoft-hosted agents are maintained by Microsoft, hence eliminating the need for time allocation to maintain the system. Microsoft-hosted agents can run jobs either directly on the virtual machine or in a container. Microsoft-hosted agents exist for the duration of a single job, after which a new one is created. Self-hosted agents give users more control over the environment and can persist through multiple jobs, possibly speeding up the Pipeline execution. (Microsoft, n.d.-b)

Pipeline execution (Figure 3) starts from a trigger, e.g., push or pull request on certain branch on repository, or user started pipeline execution. Pipeline stage is then started, and a job is sent to the agent. The agent runs the steps within the the job and reports on the status of the job. Depending on the job run result and conditionals for the jobs results, the agent will then start processing a new job or return erroneous report and the pipeline execution is stopped. (Microsoft, n.d.-e)

Figure 3 Azure Pipeline execution overview (Microsoft, n.d.-e)



Azure Pipelines allows managing of named variables, thus enabling storing sensitive information within the Pipeline, without the risk of exposing it to users. Variables can also be encrypted and set as secret. Secret variables need to be explicitly referenced for the pipeline

to decrypt the variable. Variable groups can be used to make variables available across multiple projects. (Microsoft, n.d.-d)

Azure Pipelines offers integrations with both Slack and Jira. Azure Pipelines app for Slack offers for easy integration to monitor and interact with your pipeline. After the installation user can interact with the pipeline from Slack client and notifications can be sent to specific channels or users when the pipeline has an ongoing event (Microsoft, n.d.-c). Azure Pipelines for Jira offers bidirectional linking for the pipeline and Jira issues; however, it currently lacks the ability to track build statuses and does not yet support YAML defined pipelines.(Atlassian, n.d.-a; Bansal, n.d.-a, n.d.-b)

Azure Pipelines pricing depends on the number of required services and user licenses. Open-source projects can be run on Microsoft-hosted agents for free. The amount of Microsoft-hosted Azure Pipelines agents and Azure Artifacts will determine the number of individual services required for the project. On top of this users will need licenses for the project. (Microsoft, n.d.-a)

3.5 Google Cloud Build

Serverless CI/CD service managed by Google, that executes builds on Google Cloud Platforms infrastructure. Import source code from external repositories or cloud storage spaces and execute builds as specified. Cloud Build produces artifacts and stores them as specified. (Google, n.d.-c)

Builds are specified using Google Build Config. Build configs provide instructions for different stages and jobs to run. Builds can be used to run normal CI/CD pipeline required tasks, such as fetch dependencies, produce builds and run unit tests. Build configs are written in YAML or JSON syntax. Cloud Build also provides the support to build Docker images from Dockerfile (Google, n.d.-e). Cloud Build pipelines can be run locally, to test run the Build Config written for the pipeline, using “cloud-build-local” tool. (Google, n.d.-c)

Access control between different components can be adjusted using service accounts and Google Clouds IAM. This allows granular access control to different resources or services in

Google Cloud. IAM permissions are not granted to singular user, group or service account, instead permissions are added to roles and roles are assigned to users or service accounts. This allows effective usage of the principle of least privilege. Service accounts can be used to grant roles to specific services in Google Cloud, thus providing access control within your Google Cloud project for your computing resources or other services (Google, n.d.-g). (Google, n.d.-b)

Google Secret Manager allows users to store and access credentials based on IAM permissions. Secrets managed by Google, and Cloud KMS allows you to manage the cryptographic encryption keys. Secrets are project global objects containing metadata of the object and the secret itself. IAM permissions can be used to grant access to secrets to certain user entities. Secrets have versions, rotation policies and replication policies. (Google, n.d.-f)

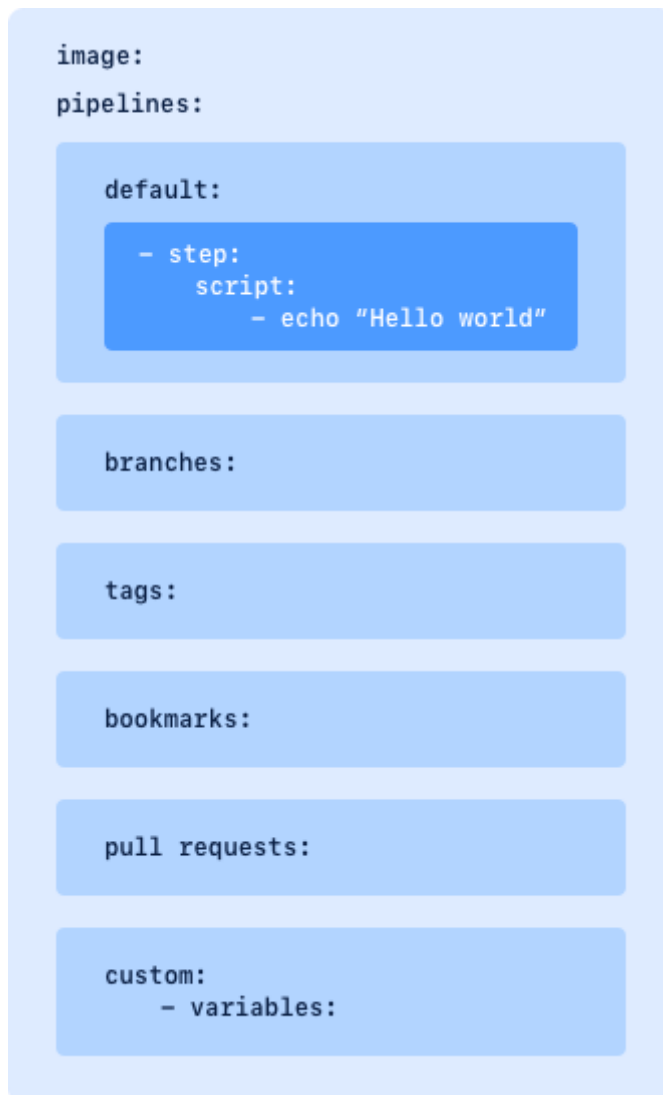
CI/CD implementation pricing on Google Cloud depends on at least the number of build minutes on Google Cloud Build. On top of this costs on stored artifacts and used network capacity may incur. Secrets Manager also adds to the costs (Google, n.d.-d). (Google, n.d.-a)

3.6 Bitbucket Pipelines

Bitbucket Pipelines is an Atlassian managed, software as a service -model, application that is integrated in to Bitbucket Cloud instance. Bitbucket Pipelines allows for CI/CD pipeline functionality, from integration and testing to deployment. Build jobs are run within containers deployed by Atlassian to the cloud. Bitbucket Pipelines requires the repository to be stored in Bitbucket Cloud instance. (Atlassian, n.d.-i)

Pipelines are defined using YAML file called “bitbucket-pipelines.yml”. Bitbucket Cloud provides a UI Wizard that can also be used to create YAML from predefined templates. The basic structure of pipeline is demonstrated in Figure 4. Pipelines must have at least one (1) step with one (1) script inside it. Each step is run in a separate Docker container. This allows for usage of different, more use case specific, containers between each step. Containers are assigned limited RAM. Caches can be defined to provide inter-container data sharing and avoiding duplication of steps. (Atlassian, n.d.-g)

Figure 4 Structural overview of Bitbucket Pipelines (Atlassian, n.d.-g)



Credentials can be stored within pipelines as variables. Variables within the pipeline can be managed on multiple levels, workspace, repository, deployment. Each having different scopes and the order of preference being from the latest to the first. Variables can be set to secure mode, which provides user with the ability to reference the variable from pipeline but masks it from the logs. (Atlassian, n.d.-p)

Bitbucket Pipelines and Jira integration makes it possible to automatically communicate the statuses of builds and deployments to Jira issues, thus increasing the efficiency of the project team and providing more complete view of the project status (Atlassian, n.d.-c). Bitbucket Pipelines and Slack integration allows for notifications about the status of the pipeline and

the deployments to be communicated to specific channels in Slack and the restart of the Pipeline in case of a failed build (Atlassian, n.d.-d).

Bitbucket Pipelines requires Bitbucket Cloud subscription. Bitbucket Cloud pricing varies depending on the number of users and the needed features. Build minutes are bundled within subscription to a certain cap, after which incur more costs. Runners allow user to run Pipelines in users own infrastructure, without being charged for the minutes consumed by the Pipeline (Atlassian, n.d.-c). (Atlassian, n.d.-f)

3.7 GitHub Actions

GitHub Actions is a GitHub hosted software as a service implementation of automating tasks on your development cycle on certain events and as such it can also be used to implement CI/CD pipelines. GitHub workflows requires for a repository in GitHub. Workflows consist of jobs; jobs have steps and steps have actions. Workflows are run on runners (Figure 6). Runner is an application that is run on servers, whether physical or virtual. Runners can be hosted and managed by GitHub or on self-hosted machines. (GitHub, n.d.-c)

Workflows are defined using YAML file that is stored in a directory path “.github/workflows”. Workflow’s structure is roughly demonstrated in (Figure 5). YAML structure is defined to contain a job that is launched on event, such as a pull request on a certain branch or user requested launch of workflow. Jobs can include multiple steps that can use a GitHub action. Actions are predefined functionalities that can be combined to produce a desired workflow. Users can reuse Actions created by the GitHub community org create their own from scratch. (GitHub, n.d.-c)

Figure 5 GitHub Actions workflow file structure visualized (GitHub, n.d.-c)

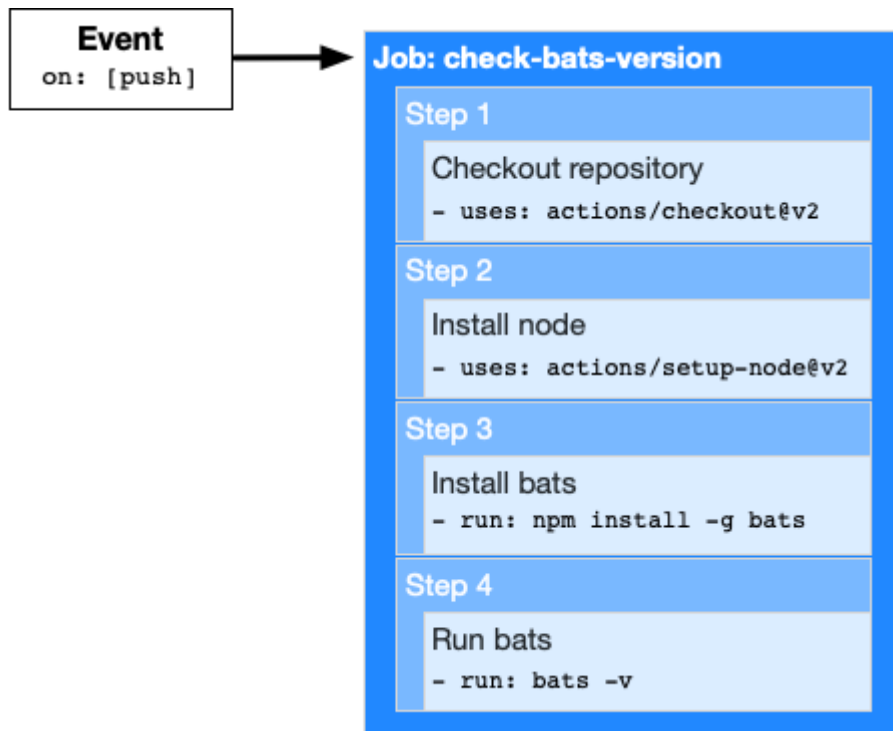
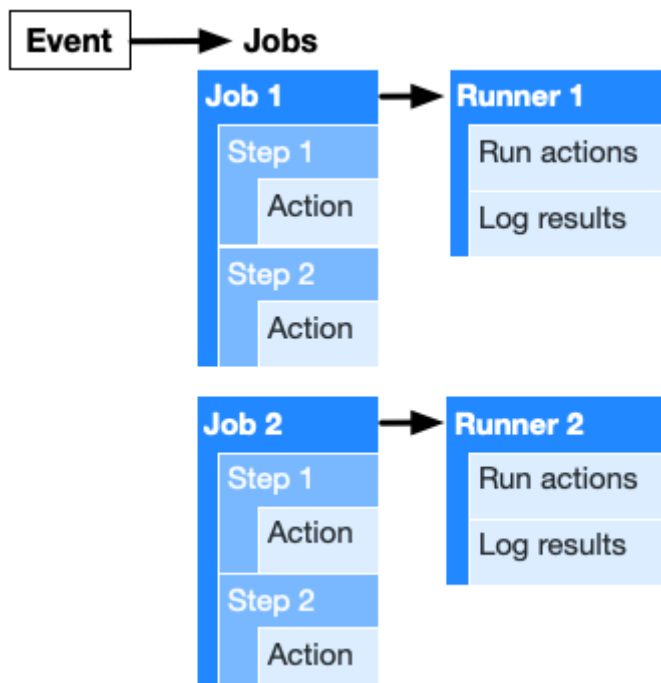


Figure 6 GitHub Actions workflow structure (GitHub, n.d.-c)



Credentials can be stored as secrets in organization, repository or repository environment. Secrets are encrypted in rest and in transit, as they only need to be unencrypted when the workflow is running. Users access to secrets can be further restricted by using access control

methods provided by GitHub. Secrets can then be referenced from the Workflow to allow access to the stored content with GitHub masking the contents from the logs. (GitHub, n.d.-b)

GitHub can be integrated into Jira using the Atlassian created GitHub for Jira integration. GitHub for Jira is currently compatible between GitHub and Jira Cloud instances. GitHub for Jira allows for the integration to manage the workflow of the project in your stead, including linking pull requests, commits, branches, builds and deployments. Integration also allows for the usage of specific comment messages to perform certain tasks on the issues, such as close the issue or add a comment. (Atlassian, n.d.-j, n.d.-k)

GitHub for Slack provides an integration between GitHub and Slack. GitHub for Slack allows for the application to provide status updates on multitude of actions happening on your repository, such as new issues, commits or deployments. (Slack, n.d.)

GitHub actions provides a certain amount of Storage and free minutes that depend on the level of contract with GitHub. For example, GitHub Free includes only 500 megabytes of storage and 2 000 minutes of run time for jobs per month. Minutes are also multiplied depending on the operating system of the runner used, as can be seen from Table 1. (GitHub, n.d.-a)

Table 1 Build minute multipliers for different operating systems (GitHub, n.d.-a)

Operating system	Minute multiplier
Linux	1
macOS	10
Windows	2

3.8 GitLab CI/CD

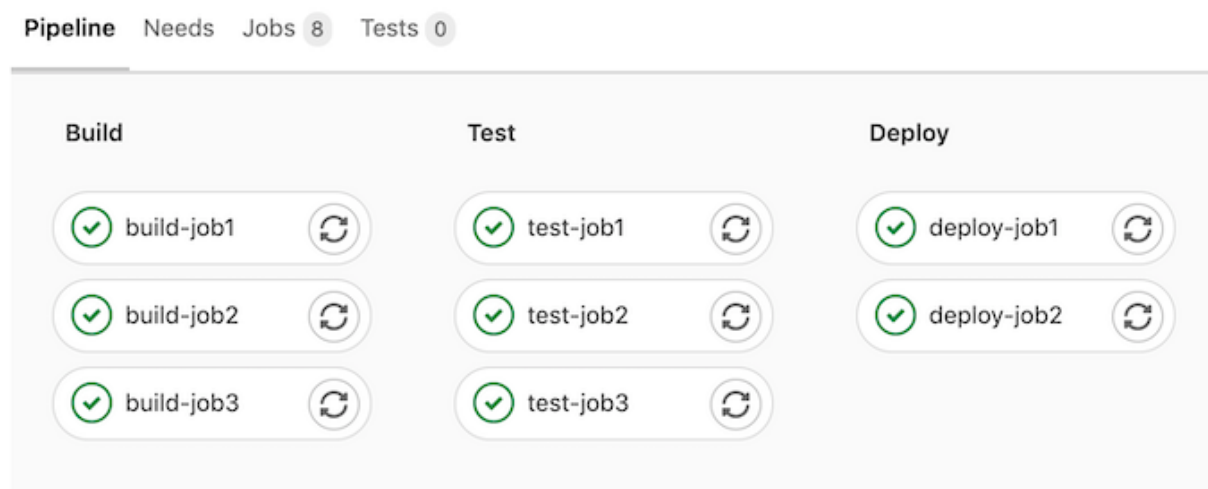
GitLab CI/CD is a functionality within GitLab, to use it user will need to have repository in GitLab. GitLab can be self-hosted and managed or bought as a subscription-based SaaS-application. To use GitLab CI/CD runner needs to be available, runners can be hosted by GitLab or self-hosted. GitLab runner is software that runs on a server and communicates with GitLab instance, sole purpose of runner is to run jobs in a pipeline (GitLab, n.d.-e). (GitLab, n.d.-c)

GitLab CI/CD is defined using the YAML format file called "gitlab-ci.yml". A job within pipeline is to be defined using a predefined structure, as declared in Code 1. In this example the name of the job is provided, after which the stage of the job is defined. Script section defines the commands to be ran on this step. Pipeline can also be visualized (Figure 7) to allow for more complete understanding of current state (GitLab, n.d.-b). (GitLab, n.d.-e)

Code 1 Basic YAML structure of GitLab CI/CD

```
test-job1:
  stage: test
  script:
    - echo "This job tests something"
```

Figure 7 GitLab CI/CD pipeline status visualized. (GitLab, n.d.-b)



Credentials can be stored securely as variables in project settings. Credentials stored in this way can be masked and protected. Masked variables appear masked in pipelines logs, thus

providing additional protection. Protected variables allow for the variable to be used only from certain branches or tags. Access control roles can be used to allow users to update or delete variables. (GitLab, n.d.-d)

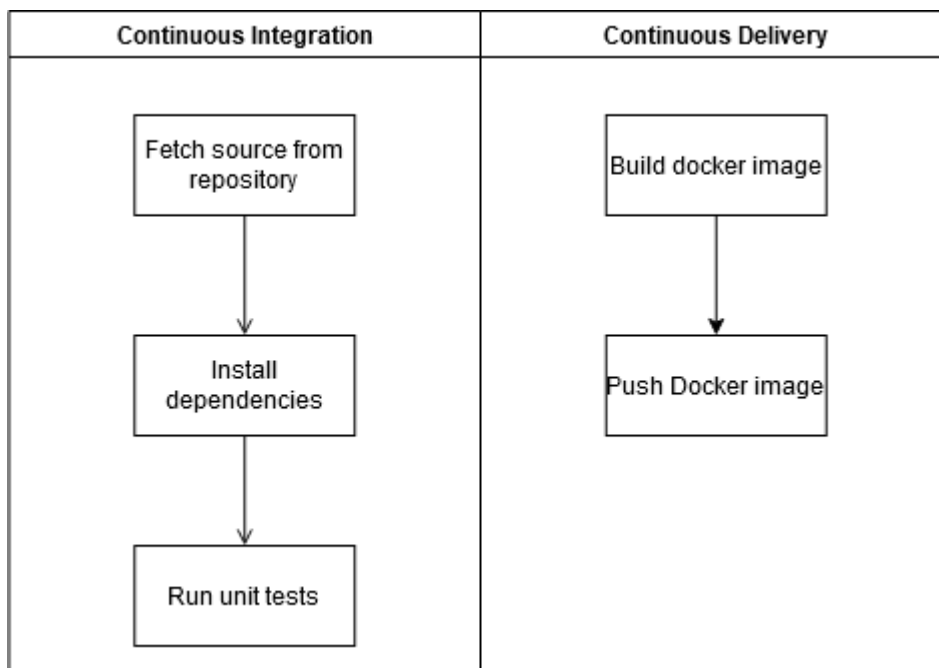
Jira integration provides functionalities, such as creating link to Jira issue, allowing for visibility to commits or merge requests and creating comments on Jira issues displaying commit messages on commits and merge requests. Integration can also move the issue between transitions on Jira issues to provide better understanding of the current status of the project. (GitLab, n.d.-g)

Slack provides an application for communicating with GitLab; however, this application only works with GitLab SaaS implementation (GitLab, n.d.-f). Slack notifications service provides integration between GitLab and Slack, allowing for communication from GitLab to certain channels on Slack and controlling the GitLab from Slack by using certain commands. Integration can send messages on events, including but not limited to, such as merge request, push, pipeline status change or deployment status change. (GitLab, n.d.-h)

4 Implementation objective

Implementations between different pipelines should follow a somewhat similar structure, in order to provide some level of comparison between the different systems. Figure 8 explains the different jobs defined in the CI/CD pipeline. Building of the application usually takes place during the integration part of the pipeline, as the focus of the continuous integration is to provide fast feedback loop on integration builds. However, in this project Docker is used to provide containerized images of the software and its dependencies in a pre-packaged environment. This also seems more suitable fit for the delivery phase of the pipeline as Docker Hub provides a convenient storage location for the image.

Figure 8 CI/CD Pipeline structure for this thesis' implementation



The first job of the pipeline is to fetch the source code from the repository to make sure that the latest changes are included in the source code of the current pipeline run. This job is should be equivalent to a `git clone` command issued against the target repository.

After the source code has been successfully cloned node.js dependencies will need to be installed. This can be achieved by running `npm install` in the projects working directory. This job will produce **node_modules** directory in the project root as the end product.

When the node.js dependencies have been successfully installed, our source code should be runnable. This gives us the ability to run unit tests, to verify that the code is working as expected. Within this project we can use `npm test` to run the unit tests.

The delivery phase of the pipeline is simple: the only goal is to build a Docker image based on the provided Dockerfile and the push it to the container registry, Docker Hub in this case, to complete the job.

When the pipeline is running it should be able to communicate with Jira and Slack to notify project teams users on the state changes of the project. Integrations that allow this kind of functionality with minimal configuration should be higher prioritized than those that require more work. Considering Atlassian products integrations, this thesis will not take into consideration the possible integration implementations made possible by Atlassian Connect.

5 Implementation

In this chapter and its sub-chapters this thesis will aim to complete the pipeline described in Chapter 4. This implementation will be done in each of the Cloud Platforms and SaaS products described in Chapter 3. Each product will be processed as subchapters for this chapter. The implementations should be as similar as possible to give the possibility to compare the different products.

5.1 Jenkins

Jenkins was installed on a virtual machine running CentOS 8 operating system. Installation was done using `dnf` package manager from Jenkins' repository. After this a short post-installation setup wizard was completed using a web browser. During installation process Jenkins was installed using suggested plugins.

5.1.1 Integration with SCM

Pipeline was created using the Jenkins UI and configured to use GitHub repository for the source control management. Jenkins was configured to scan a branch 'jenkins' from the repository.

Pipeline was set to use the pipeline script, file with a name 'Jenkinsfile', from the pulled source code. This allows for pipeline-as-code approach, effectively granting visibility for the whole history of the pipeline.

5.1.2 Managing credentials

Credentials and variables were configured on project level thus providing access to users that are part of the project. Variables were created as displayed in Table 2. Both kinds of credentials used within this project are stored as encrypted on Jenkins' controller instance, and only unencrypted on use.

Table 2 Jenkins project credentials configurations

ID	Kind	Encrypted
github-personal-token	Username with password	YES
docker-hub-creds	Username with password	YES
MONGODB_URI	Secret text	YES
SECRET	Secret text	YES

5.1.3 Pipeline

The pipeline is fully configured using the DSL-syntax by providing Jenkinsfile with the repository. Pipeline starts with the definition of Pipeline, immediately followed by definitions on which agent to run the pipeline on and environmental variables, as seen in Code 2. As can be seen the credentials needed for the successful run of the pipeline are also loaded into environmental variables. These could also be more narrowly scoped by defining the environment values at specific stages.

Code 2 Definition Jenkins Pipeline, agent and environmental variables

```

pipeline {
  agent any
  environment {
    TEST_PORT = 3001
    TEST_MONGODB_URI = credentials('MONGODB_URI')
    SECRET = credentials('SECRET')
  }
  ...
}

```

Code 3 demonstrates the Pipeline entering the 'pre-build' stage, in which it utilizes a nodejs plugin to function with a npm executable installed on the host system. In this stage nodejs dependencies are installed and the unit tests are run.

Code 3 Definition of the pipelines pre-build stage.

```
...
stages {
  stage('pre-build') {
    steps {
      nodejs(nodeJSInstallationName: 'Node 12.22.1') {
        sh 'npm install'
        sh 'npm test'
      }
    }
  }
}
...
```

Next two stages provide the interaction with Docker CLI, as demonstrated in Code 4.

Utilizing the context that Jenkins docker plugin provides, the pipeline first builds an image and then on the next stage pushes it to the Docker Hub container registry using credentials defined with id **docker-hub-creds**.

Code 4 Jenkins Pipeline executing Docker image build and push to DockerHub

```
stage('build docker image') {
  steps {
    script {
      app = docker.build('vijoni/bloglist-backend-cicd:jenkins')
    }
  }
}
stage('push docker image') {
  steps {
    script {
      docker.withRegistry('https://registry.hub.docker.com',
'docker-hub-creds') {
        app.push('jenkins')
      }
    }
  }
}
```

5.1.4 Integrations

Jenkins integration to Jira Cloud was started by creating OAuth Credentials for the integration in the Jira Apps settings page. The necessary information was provided as can be seen from Figure 9. OAuth credentials were provided access to build and deployment

information. Jenkins was then configured to contact the Jira Cloud instance as can be seen from Figure 10. It was mandatory to enter the address for the Jira Cloud instance and the ClientID and Secret of the freshly created credentials. Secret was stored within Jenkins credentials. Connection was tested and the credentials deemed working.

Figure 9 OAuth credential creation for Jenkins plugin in Jira Cloud

Create OAuth credentials

App name *

Server base URL *

Logo URL

Permissions *
 Deployments
 Builds
 Development information
 Feature flags

[Learn more](#)

Figure 10 Jenkins configuration for Jira integration

Jira Software Cloud Integration

Jira Cloud Sites

Jira Cloud Site

Site name

ClientID

Secret

Successfully validated site credentials

Integrating Jenkins Pipeline into the Slack was done using the Slack Notification plugin (ID: slack, in Jenkins Plugin manager). After the installation the plugin was configured, related plugin configuration can be found from “Configure System” options, under the headline: “Slack”. Configuration was done as follows and connection test was run successfully, as can be seen from Figure 11.

Figure 11 Configuration for Slack notification plugin

The screenshot shows the configuration page for the Slack notification plugin. It features several input fields and a checkbox:

- Workspace:** A text input field containing the value "test".
- Credential:** A dropdown menu showing "slack" with an "Add" button next to it.
- Default channel / member id:** A text input field containing the value "#jenkins".
- Custom slack app bot user:** A checkbox that is currently unchecked.
- Buttons:** "Advanced..." and "Test Connection" buttons are located at the bottom right of the form.
- Status:** A "Success" message is displayed at the bottom left.

5.1.5 Costs

Jenkins is open-source software and as such does not carry any license costs with it.

Infrastructural costs do exist and do depend largely on the size of the Jenkins instance and the amount and size of the projects being built. Due to its open-source nature, there is also very limited support available, which in turn means that the management of the systems cost more money.

For this thesis there were no costs associated with running Jenkins as KVM was used to virtualize the systems running the Jenkin servers.

5.2 Bamboo

Bamboo was installed on a virtual machine running CentOS 8 operating system. PostgreSQL database server was installed using `dnf`. Bamboo was downloaded from Atlassian download portal as `tar.gz` archive and extracted to proper folder. Bamboo application user was created and home directory, application installation directory rights were set. After this a short post-installation setup wizard was completed using a web browser. During installation process Bamboo was defined to use PostgreSQL database.

5.2.1 Integrating with SCM

Setting up the source repository is easily implemented while configuring the plan (aka. Pipeline) on "Link repositories" phase. In this thesis' case the repository host was defined to

be GitHub repository. As the project used in this thesis is publicly available, the only information needed for creation of the plan was a display name and GitHub username. After this information is provided it is possible to load all available repositories, select the correct one from drop down list and decide which branch to use for the plan.

5.2.2 Managing credentials

Most of the credentials, and or variables, were stored in plan variables as documented in Table 3. Initially Docker credentials were tried to store within the plan variables as well, but this did not work as the Docker that was installed on the virtual machine was not on high enough version to support such an operation that Bamboo was trying to use. As a result of this Docker credentials were set up on the host running Bamboo agent. Depending on the method Docker is configured to use to store credentials, they can either be stored in a json file using hashes or in the operating systems keychain in encrypted form.

Table 3 Description of variables stored within Bamboo

Variable	Encryption	Location
TEST_PORT	NO	Plan variables
MONGODB_SECRET	YES	Plan variables
SECRET	YES	Plan Variables
Docker credentials	YES*	Host system

5.2.3 Pipeline

The pipeline, or plan as they are called in Bamboo, was created using the Bamboo Web UI. New stage, “Integration”, was created and within it a “Integration run” job. This job was configured to be run on host environment and not in a Docker container.

Bamboo offers a nice and intuitive way of working with the jobs and tasks in the pipeline, as can be seen from the Figure 12. The first task of the job was to checkout the source code, to pull the latest changes from the source repository. After this a job was created to install Nodejs package dependencies, for this a new executable was added with the name of “npm – agent”, using a node binary installed on host system `npm install` is run by the pipeline. The following task runs the unit tests invoking the command `npm test`, thus making sure the specific version of the software is going to function properly when built into Docker image. In this task we include environmental variables on the configuration page, as seen in the Figure 12. On the next step the Docker image is built, and after this pushed into the Docker Hub container registry. Docker Hub credentials were provided from the host system by logging in to the user that runs the agent and entering `docker login` command. This provides the needed concept for the user running the bamboo agent to be able to communicate with Docker Hub using docker command-line interface.

The whole pipeline was built into a singular step for demonstration purposes. This implementation would fail, should there be more than one agent available to run the jobs, as jobs are run in parallel. More logical representation of the pipeline would have been to use a single stage to pull the source code, second one to install the dependencies, third stage to run the tests and so on.

Bamboo-spec file can be generated from the already existing pipeline.

Figure 12 Bamboo tasks declarations

Job details Docker **Tasks** Requirements Artifacts Other

Tasks

A task is a piece of work that is being executed as part of the build. The execution of a script, a shell command, an Ant Task or a Maven goal are only few examples of Tasks. [Learn more about tasks.](#) You can use [runtime](#), [plan](#), [project](#) and [global variables](#) to parameterize your tasks.

- Source Code Checkout
Checkout Default Repository
- npm
NPM - Install dependencies
- npm
NPM - Run tests
- Docker
Build Docker image
- Docker
- Final tasks Are always executed even if a previous task fails
Drag tasks here to make them final
- Add task

npm configuration

Task description

Disable this task

Add condition to task

Node.js executable*
 [Add new executable](#)

Command*

Command that npm should run (e.g. 'install', 'test')

Advanced options

Use isolated cache
A temporary directory will be used as cache folder. Might slow down task execution.

Environment variables

Extra environment variables. e.g. JAVA_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working subdirectory

Specify an alternative subdirectory as working directory for the task.

[Save](#) [Cancel](#)

5.2.4 Integrations

Due to utilizing Jira Cloud in this thesis in combination with running a locally virtualized environment, it was decided not worth the effort to create the necessary infrastructural changes to get Bamboo Application Linking to work. Marketplace did not provide the necessary apps to communicate with Jira Cloud instance, for the 8.0.0. version of Bamboo that was in use during the writing of this thesis.

Slack integrations were also not available for the version of Bamboo that was used, so these had to be substituted with a simple webhook implementation. Webhooks can be configured from the administration menu, from “Webhook templates” configuration page.

New template was created with a name of “Slack webhook” and a type of “POST” with payload and headers as described in Figure 13. The payload consists of just an informative, proof of concept, type message that could be highly customized if needed. After this the plans notification settings can be customized to utilize the newly created webhook template

to send a message to a specific webhook endpoint for Slack (Figure 14), this will then result in the incoming webhooks application to deliver the message to the requested channel.

Figure 13 New Webhook template

Edit a webhook template

HTTP method

Name*

Payload

```
1 {
2   "text": "Pipeline state changed."
3 }
```

For example json, xml Here you can use variables in the following format: \${bamboo.variableName}

Headers

```
1 Content-type: application/json
```

Figure 14 Defining new build notification for the Pipeline

Edit a notification

Edit build notification

Event

Notification sent for every job that finishes building

Recipient type

Webhook name

URL

5.2.5 Costs

Costs of running Bamboo are defined by the license costs and the costs of running the hardware. For this thesis there were no costs associated with running Bamboo, as an evaluation license was used and KVM was used to virtualize the systems running the Bamboo servers.

5.3 AWS CodePipeline and CodeBuild

A new organization account was created to AWS Cloud Platform for this thesis. Billing information was provided to AWS to understand the cost structure of the service. AWS search functionality provided an easy way to navigate between different services available in the platform.

5.3.1 Integrating with SCM

Process was straightforward and could be set up while configuring the pipeline on the second stage of the pipeline creation, “source stage”. GitHub (Version 2) was selected as the source provider. New connection to GitHub was created by clicking on “Connect to GitHub” button. On the next dialog, the name for the connection was requested, and upon submitting the form GitHub redirect hook was used to provide login to GitHub application.

New GitHub app was installed to provide integration between GitHub and AWS CodePipeline to selected repositories. Applying desired configurations and clicking connect established the connection through the application integration.

5.3.2 Managing credentials

In this thesis the AWS Parameter Store was used to manage secrets. Secrets were created for the following variables Docker Hub username and Docker Hub password, URI for the

MongoDB to connect to in order to pass the tests and secret variable (a sort of a salt used to provide individual hashes for the environment).

The process was straightforward and could be completed in AWS Parameter Store Web UI. Choosing to create parameter and providing required information such as the name of the variable. Standard tier for the variable as the parameter size was less than 4KB and type of the parameter as SecureString to provide encryption using KMS keys from specified for the AWS account. AWS managed KMS keys were used for this thesis' use case. Finally, the value of the parameter was set, and the creation process completed.

5.3.3 Pipeline

Pipeline was created from AWS CodePipeline Web UI. On the first page of the pipeline creation process, basic pipeline related information such as pipeline's name and service role were created. This role could then be used to grant this specific Pipeline granular access to different services provided by AWS. Artifact store was left to default settings, which means that the Pipeline will store artifacts in S3 Bucket. Encryption keys were left to default, Amazon managed keypair.

In Source stage source provider was specified to be GitHub, new connection was set up for GitHub application and access to the repositories was granted for the application, via GitHub. Repository to be used was specified as "jonivirtanen/bloglist-backend-cicd" and the branch to be used was selected. Pipeline was set to be triggered on source code change. Pipeline's build stage was set to output artifacts in the CodePipeline default format (zip).

Build stage was defined to use AWS CodeBuild as the build provider and region was set to Stockholm. New project was created using default AWS managed image, with Ubuntu operating system using the "aws/codebuild/standard:5.0" image and the latest version of it. Privileged mode was toggled as the build pipeline ultimately builds a Docker image and as such requires larger privileges. Build was defined to use, the default setting, "buildspec.yml" file as the source of the build script. CloudWatch logs were not enabled, all though these might offer a way to provide needed integration towards Jira, using AWS SNS, Lambda

functions or AWS ChatBot. After defining build type as a single build, skipping the deployment stage and reviewing the settings, the Pipeline was created.

Next the build steps of the pipeline were defined in the “buildspec.yml file at the root of the repository. Understanding the YAML sections was not too complicated, but it did take some time to find the proper variables to execute the pipeline properly. MongoDB URI and SECRET environment values were saved in AWS Parameter Store to provide the functionality of only referencing them in the public repository and mask them in the runtime logs to minimize the possibility of leaking the secrets. Using AWS Parameter store also grants the possibility of granular access management through the usage of AWS IAM rules.

5.3.4 Integrations


Integrations between Jira Cloud and AWS should be possible to be created using Atlassian Connect. However, this does fall out of the scope of this thesis, and as such will not be described in this thesis.



To integrate CodePipeline with Slack AWS ChatBot (later ChatBot) application was added to Slack workspace. After this new ChatBot client was setup for Slack in the AWS Console. Slack redirect will be provided in order for the user to agree on the permissions for the Slack app on the selected workspace. After this the configuration for AWS ChatBot was set up using the AWS Console again, providing necessary information such as AWS ChatBot configuration name and Slack channel type and name.

On the next page AWS IAM permissions are provided. In this thesis we used the default AWS provided IAM role ‘AWSChatBot-role’ with default settings. After this AWS will take care of configuring necessary requirements for the role such as SNS topic.

As the final step the AWS ChatBot is invited to join the Slack channel, as can be seen from the Figure 15. From this point on the ChatBot integration will provide information to the Slack channel as configured.

Figure 15 AWS ChatBot integration in #aws-test channel

 **vijoni91** 7:25 PM
joined #aws-test.

 **aws** APP 7:41 PM
 **AWS Chatbot Notification | Test Message | Account: 863573546124**

This is a test message from Chatbot configuration: **aws-test-configuration** sent to SNS topic: **CodePipeline**.

If the configuration has more than one SNS topic specified, you should see a test message for each one. If any are missing, visit [Troubleshooting](#) in the AWS Documentation.

5.3.5 Costs

Table 2 describes the accumulation of costs from the pipeline implemented in this chapter and the amount of resources consumed during the implementation. These numbers are highly volatile, as they do change based on Amazon region, instance types, number of requests and amount of resources consumed.

Table 2 Cost breakdown of Pipeline implementation in AWS

Component	Units	Price / Unit	Price	Free tier
CodeBuild	32 (minutes)	-	-	100 minutes
CodePipeline	1 (Pipeline)	-	-	1
DataTransfer (Out bytes)	0.000026 (GB)	0	0	-
EBS	9.663 (GB-Mo*)	\$ 0.1045	\$ 1,01	30GB 2m I/Os

KMS	113 (Requests)	\$ 0.00	\$ 0.00	20000 API Requests
AWS Secrets Manager	0.987 (Secrets)	\$ 0	\$ 0	-
	7 (API Requests)	\$ 0	\$ 0	-
S3	98 (Requests, PUT / COPY / POST / LIST)	\$0.005 / 1000 (Requests)	\$ 0	2000 PUT
	221 (Requests, GET and others)	\$0.004 / 10000 (Requests)	\$ 0	20 000 GET
	0.001 (GB-Mo)	\$0.023 / GB (First 50TB)	\$ 0	5GB

5.4 GCP Code Build

Authors personal Google credentials were used for this thesis. New project was created on Google Cloud Platform and billing information was provided and attached to the project. This was done to understand the cost structure of the service. GCP provides a search functionality to allow an easy way to navigate between different services available in the platform.

5.4.1 Integrating with SCM

Google Cloud Build was configured via web user interface to connect to GitHub. Process was started by creating a new trigger and connecting the source repository to it. “Push to a branch” was chosen as the trigger event to run the pipeline. The pipeline can still be manually invoked to run at any point in time. For the source a new connection to repository was created and configured to follow “gcp” branch.

As the repository also contains a Dockerfile and GCP provides a “native” functionality to build container images using Dockerfiles, configuration type was forced to use Cloud Build file (cloudbuild.yml), located at the root of the repository, as the source for the Pipeline.

5.4.2 Managing credentials

Secrets were managed using Secret Manager service. To access Secret Manager from Cloud Build, a service account was created and required IAM privileges were granted. Credentials were created as described in the Table 4:

Table 4 Description of Credentials stored in Secret Manager

Secret name	Encryption	Replication	Rotation
docker-password	Google Managed	Automatically replicated	Not scheduled
docker-username	Google Managed	Automatically replicated	Not scheduled
Mongodb	Google Managed	Automatically replicated	Not scheduled
Secret	Google Managed	Automatically replicated	Not scheduled

5.4.3 Pipeline

Google offers no Web based editor to process the cloudbuild.yml file, so it was configured locally and updated to repository, which triggered a Cloud Build run. YAML syntax is logical, and all the steps are run within containers. Basic job syntax starts with “name” key and is

followed with parameters concerning the job. Code 5 snippet displays the cloudbuild.yml syntax for installing npm dependencies inside a “node:12” image based container.

Code 5 Cloudbuild.yml step to install npm dependencies inside a container

```
steps:
- name: node:12
  entrypoint: npm
  args: ["install"]
```

Because of the way Google Cloud Build functions, the Secret Manager and its secrets are only accessible on the *args* parameter. For this reason in this thesis’ workflow we need to import environment variables prior to running `npm test`. These modifications are declared in Code 6 and consist of launching a node image based container to read and import the environment variables. This also required minor modifications to the package.json file of the source repository, as described in Code 7, as it is necessary to define the script for the functionality.

Code 6 Reading environment secrets in Google Cloud

```
- name: node:12
  entrypoint: npm
  args: ["run", "create-env"]
  secretEnv: ["SECRET", "TEST_MONGODB_URI"]
```

Code 7 npm package.json script to print environment variables to .env file

```
{
  ...
  "scripts": {
    ...
    "create-env": "printenv > .env"
  },
  ...
}
```

5.4.4 Integrations

Google Cloud Build offers the flexibility of running almost anything through the containers. However, at the time of the writing, it does not offer any premade integrations. As such, to

utilize it to its full extend the team using GCP to run CI/CD Pipelines and wishing to get notifications delivered to Slack or Jira Cloud, should consider the workload of implementing the possible solutions.

Slack notifications should be possible to implement using the Slack webhook app. This could be accomplished by manually creating different steps to create HTTP requests to the specific Slack endpoint. This should also be implementable using the Google Cloud Pub/Sub with a specific topic to forward messages to specific service, a container or cloud function, and then processing them as wished. Jira Cloud integration should be achievable similarly, except that your communication should be from GCP to a Cloud Connect container that processes the messages, thus integrating the service into Jira Cloud.

5.4.5 Costs

Google Cloud Build was the cheapest cloud solution for this thesis' workflow. It is the most basic option considering the functionalities, but it can be customised to run pretty much anything within the containers. Table 5 contains a breakdown of the costs of this project.

Table 5 Pipeline cost breakdown in Google Cloud Platform environment

Component	Units	Price / Unit	Price	Free tier
Secret Manager Version replica storage	3.808 (month)	??	0.19 €	??
Secret access operations	13 (count)	??	0.00 €	??
Cloud Build – Build time	10.333 (minutes of build time)	??	0.00€	??

5.5 Azure Pipelines

New Azure DevOps account was created for this thesis. New organization and project were created on Azure DevOps. Billing information was provided and attached. This was done to understand the cost structure of the service. Azure DevOps subscription was bought for this thesis, as it would have required extra work to sign-up for the free benefits of public projects. Such benefits would have included build time among other things. Azure DevOps provides a search functionality to allow an easy way to navigate between different services available in the platform.

5.5.1 Integrating with SCM

Integrating with GitHub was completed as the first step of creating the pipeline. Initial connection was created using GitHub application connection. Application requires Read level permissions to access metadata, and read and write permissions to checks, code, commit statuses, issues and pull requests.

Write level permission to code, commit statuses are used by graphical development environment provided by Azure Web UI to modify the pipeline. This allows for easier implementation of different stages of pipeline as Azure provides snippets that can be used to implement functionalities and integrations to the pipeline.

5.5.2 Managing credentials

Credentials were managed on pipeline level. This was done using the Web editor and changing pipeline variables. Variables were defined for MONGODB_URI and SECRET environment variables, as GitHub and Docker Hub login credentials are handled by application integrations.

Docker Hub credentials were specified on project level settings as a new service connection. Registry type was set to Docker Hub and basic login information for Docker Hub was provided.

5.5.3 Pipeline

Pipeline was defined to trigger on changes to 'azure' branch. Pipeline was run on Microsoft hosted shared agent based on latest Ubuntu image. Code 8 describes the Pipelines first step, installing NPM version 12 on the build agent. Next step was to run npm install to provide all the required Nodejs dependencies and to test that it works as intended, one possible implementation of this can be seen in the Code 9 snippet. During this phase the variables earlier stored in Pipeline properties were used.

Code 8 Description of a task to install Node.js version 12.

```
steps:
- task: NodeTool@0
  inputs:
    versionSpec: '12.x'
    displayName: 'Install Node.js'
```

Code 9 Task to run npm install, npm test using the secret variables

```
- script: |
  npm install
  npm test
  displayName: 'npm install and test'
  env:
    TEST_MONGODB_URI: $(MONGODB_URI)
    SECRET: $(SECRET)
```

After succesfull test run the code was packaged in Docker Image and deployed to Docker registry. Azure provides a predefined task that can be used to accomplish this, as is described in the snippet Code 10.

Code 10 Task to build and push the container image to Docker Hub container registry

```
- task: Docker@2
  inputs:
    containerRegistry: 'Docker Hub'
    repository: 'vijoni/bloglist-backend-cicd'
    command: 'buildAndPush'
    Dockerfile: './Dockerfile'
    tags: 'azure'
```


5.5.4 Integrations

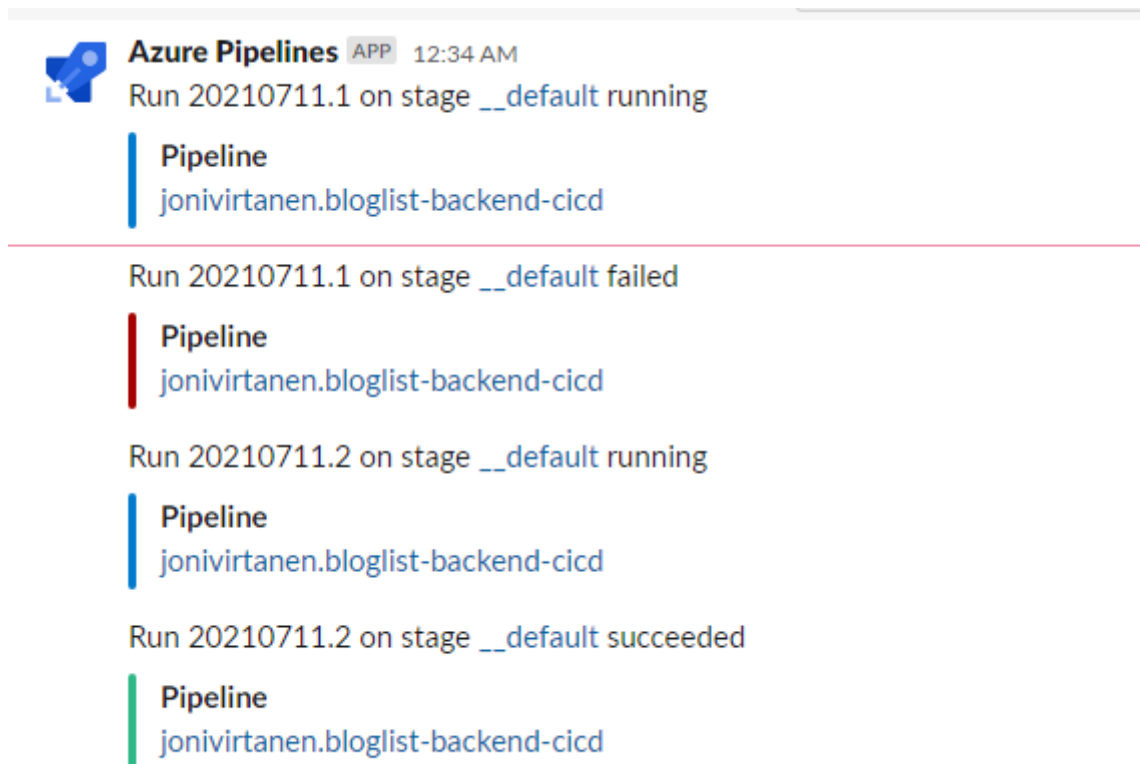
Azure Pipelines for Jira integration was easily configurable and could be started from Jira Cloud instances Web UI. Application was installed from the Atlassian Marketplace and the bidirectional communication between Jira and Azure DevOps was configured easily. However, Azure Pipelines for Jira integration was not too useful for this thesis' use case, as it only tracks deployment statuses on Pipelines. In order to achieve this kind of visibility from Azure Pipelines to Jira Cloud, one would need to configure Atlassian Connect and create custom logic for the functionality.

Azure Pipelines app for Slack integration can be installed using the Slack. Simply adding a new application to the workspace, commanding the application via `/azpipelines subscribe <pipeline url>` provides an interactive way to configure the application, as can be seen from Figure 16. Configuration starts by signing into the Azure and providing necessary information for the service. For this thesis' workflow we set the integration to send notifications on different stages of the build, as can be seen from the Figure 17.

Figure 16 Azure Pipelines Slack Integration installation

The screenshot shows a Slack channel conversation. At the top, a 'Today' separator is visible. The first message is from user 'vijoni91' at 12:10 AM, stating 'joined #blocklist-backend.'. The second message is from 'vijoni91' at 12:18 AM, stating 'added an integration to this channel: Azure Pipelines'. A red 'New' indicator is next to this message. The third message is from the 'Azure Pipelines' app at 12:18 AM, with a blue robot icon. The text says: 'Hi! Let's start monitoring your pipelines. Subscribe to one or more pipelines or all pipelines in a project with: `/azpipelines subscribe [pipeline url]/project url]`. To see what else you can do, type `/azpipelines help`. To know more see [documentation](#).' Below this is a greyed-out section with an eye icon and the text 'Only visible to you'. The fourth message is from the 'Azure Pipelines' app at 12:20 AM, with the text: 'Looks like you are not signed in to Azure Pipelines. Please sign in before making any request.' Below this is a 'Sign in' button. The text continues: 'Complete sign-in by entering the verification code presented to you post authentication.' Below this is an 'Enter code' button. The final message is from the 'Azure Pipelines' app at 12:22 AM, with the text: 'Successfully signed in to Azure Pipelines as vijoni91@gmail.com.'

Figure 17 Azure Pipelines notifications about the status changes of the pipeline.



5.5.5 Costs

Azure Pipelines offers a free tier for open-source projects. However, to claim such a benefit users must apply for it. During the implementation of Azure Pipeline for this thesis, the appliance process for the free tier was not clearly documented or easily found. As a result, it seemed like a more reasonable approach to purchase a parallel job subscription for the time being. And as things usually go, when one is finished with the project it is easily forgotten about and the subscription is left valid.

Table 5 Pipeline cost breakdown on Azure environment

Component	Units	Price / Unit	Price	Free tier
Azure Pipelines - Microsoft-hosted CI/CD	0,9677	33.7320 €	32,64	-

Concurrent Job				
----------------	--	--	--	--

5.6 GitHub Actions

My personal GitHub account, jonivirtanen, was used for this thesis. Free plan was deemed sufficient for the use case of this project, and as such no billing information was provided. GitHub did not require extra steps to get started with Actions, except for the process described in chapter 5.6.1.

5.6.1 SCM

Integrating with GitHub Actions was achieved easily, as the repository is located within the GitHub. To activate actions “.github/workflows” directory structure was created within a branch. Inside of the workflows directory “actions.yml” file was created.

File actions.yml was then used to describe the whole workflow. GitHub’s web editor, that allows for easy usage of community made actions, was utilized during this project.

5.6.2 Managing credentials

Credentials were managed in GitHub by using repository specific variables and environment specific variables, as defined in the Table 6. This allows for easily understandable access control for the variables, whilst also making them more secure.

Table 6 Description of secrets used in GitHub

Secret name	Secret type
MONGODB_URI	Environment Secret

SECRET	Environment Secret
DOCKER_PASSWORD	Repository Secret
DOCKER_USERNAME	Repository Secret

5.6.3 Pipeline

Description of the pipeline trigger action can be seen in the snippet Code 11. Pipeline definition starts with a snippet defining the name of the pipeline, a trigger option (in this case triggered on push or pull request on branch “actions”. “workflow_dispatch:” attribute was added to grant end user the ability to run the pipeline manually when needed.

Code 11 GitHub Actions trigger configuration

```
name: bloglist-backend-CI

on:
  push:
    branches: [ actions ]

  workflow_dispatch:
```

Jobs are defined using the formula described in the following snippet, Code 12. Within a singular job different steps were created to checkout the source code, setup node.js environment, to run the tests written for the software. Docker Image building and pushing to Docker Hub was handled in a separate job.

Code 12 Basic structure of GitHub Actions workflow

```
jobs:
  #Defines the start of Jobs section
  job1:
    #Specify an identifier for the job
    environment: #Optional, specify environment variables
      name: bloglist-backend
    runs-on: ubuntu-latest #Specify the environment to run the jobs
    steps:
      - uses: actions/checkout@v2
```

5.6.4 Integrations

GitHub for Jira integration is easily configurable through Atlassian's Marketplace. Process started by installing the application to Jira, which will then redirect the user to GitHub to create an application link between the two applications. During the installation the necessary information was provided and jonivirtanen/bloglist-backend-cicd repo was selected to be used with the application. After this step has been completed Jira will start syncing configured projects with the repo and parsing information from repo.

GitHub + Slack integration is easily installable via Slack as it is distributed as an app. GitHub integration was added via Slack's app directory. After the application was added to workspace, users can subscribe to repositories using the command `/github subscribe <repository name>`. This will then prompt for the necessary configuration in GitHub for the correct privileges to be granted for the application, after which a message will be sent to inform about successful subscription to a repository (Figure 18). However, this integration does not seem to be able to send notifications about the current state of the workflow, unless it is pending an action from a user.

Figure 18 Slack message notifying about the successful subscription to repository.



5.6.5 Costs

Costs for this thesis project were non-existent. GitHub bundles GitHub actions minutes by different subscription plans. Thesis writer is using "Free" subscription in GitHub and thus granted 2 000 minutes of runtime per month for GitHub actions, this applies only to public repositories.

5.7 GitLab CI/CD

New user account and organization were created for GitLab. Billing information was not provided to GitLab as this project's repository, that was cloned from GitHub as described in chapter 5.7.1., and was eligible to 400 CI/CD minutes on SaaS runners.

5.7.1 SCM

GitHub repository was cloned to GitLab to enable CI/CD functionality in GitLab. This was easily doable via GitLab web user interface. To start the process of cloning the repository new project was created by importing from existing source.

As the source repository is publicly available in GitHub the cloning process is simple and requires one to specify source repository and the name of the new GitLab project. After submitting the form GitLab takes care of cloning the repository and creating a new remote repository.

5.7.2 Managing credentials

Credentials were managed by using the projects variables, that can be configured from projects CI/CD settings. Variables were created as described in Table 7. As the build in this thesis' case is run on GitLab shared runners, Docker Hub password is masked and protected. These credentials are used within the pipeline by stages that run the unit tests and push the build image to Docker Hub.

Table 7 Description of variables in GitLab

Variable name	Protected	Masked
DOCKER_PASSWORD	YES	YES
DOCKER_USER	NO	NO

MONGODB_URI	NO	NO
SECRET	NO	NO

5.7.3 Pipeline

Project's CI/CD workflow was configured to use GitLab shared runners. Due to this reason the workflow was also run inside containers, as running on a shared virtual machine could compromise environment variables such as Docker Hub credentials. Setting up the workflow is as simple as creating a file called ".gitlab-ci.yml" to the projects root directory. This file is editable through GitLab Web IDE.

As the project is run in multiple jobs and each job runs on different container instance "node_modules" directory was specified to be cached (Code 13), so the directory would persist between different jobs at project level, thus eliminating the need to run "npm install" on every job.

Code 13 Configuration to cache node_modules directory between different containers.

```
cache:
  paths:
    - node_modules/
```

GitLab CI/CD uses the following format, as demonstrated in Code 14, to define a job. Job starts by defining a name for the job (e.g. "run_tests"), image to be used and stage this job is to be run on. Additional runtime environment variables can be set providing additional information as variables. When parameter can be used to define when the job should be run, in this case when the previous step, installation of dependencies to provide node modules, was run successfully.

Code 14 Job structure for GitLab CI/CD

```
run_tests:
  image: node:12.22-buster-slim
  stage: test
  variables:
    TEST_MONGODB_URI: $MONGODB_URI
```

```

SECRET: $SECRET
script:
  - npm test
when: on_success

```

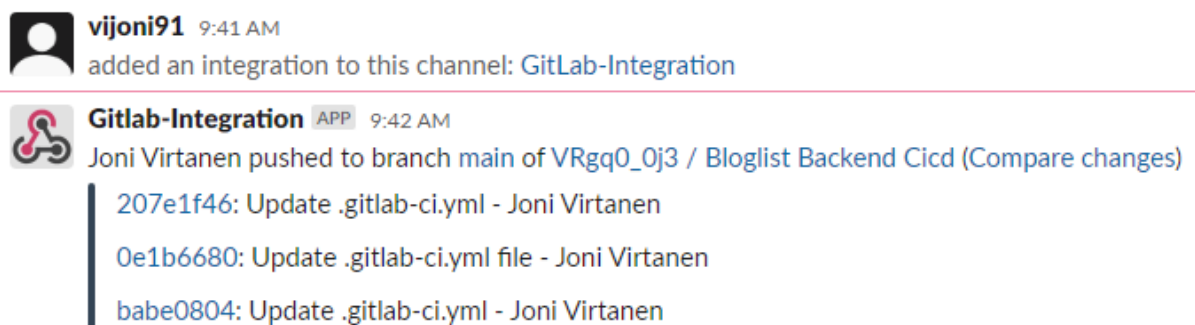
5.7.4 Integrations

GitLab for Jira integration is installable through Atlassian Marketplace. After the installation was done, configuration was done by adding namespace to follow changes on a specific GitLab repository. Installation process was simpler than expected, as having session open in GitLab seemed to set all the GitLab side configuration automatically. However, this integration does not seem to track build status or CI/CD runs.

GitLab Slack application offers ChatOps integration for the GitLab project. It does not offer the possibility to send notifications on certain events to different channels, but this can be solved using the Slack notifications integration found in GitLab integrations list.

Configuration of Slack notifications integration starts with selecting the actions that one wants to be notified about. In this thesis' use case, pipeline events are chosen and configured to notify people on #gitlab channel. Configuration also requires the specification of a webhook URL and a username to post to the channel as. After configuration of these variables, the test run was successful, as seen on Figure 19.

Figure 19 GitLab Integration test post using Incoming Webhook Slack app.



The screenshot shows a Slack channel interface. At the top, a user named 'vijoni91' (9:41 AM) has added an integration to the channel named 'GitLab-Integration'. Below this, a notification from the 'Gitlab-Integration' app (9:42 AM) is displayed. The notification states: 'Joni Virtanen pushed to branch main of VRgq0_0j3 / Bloglist Backend Cidc (Compare changes)'. A vertical bar on the left side of the notification lists three commit hashes: '207e1f46: Update .gitlab-ci.yml - Joni Virtanen', '0e1b6680: Update .gitlab-ci.yml file - Joni Virtanen', and 'babe0804: Update .gitlab-ci.yml - Joni Virtanen'.

5.7.5 Costs

No costs were accumulated during this thesis. Hence, no bill was sent to the author. The user account that was used for this thesis has a “Free” plan subscription on GitLab and is eligible for 400 minutes of CI/CD runner time on shared runners. The limits were never exceeded.

5.8 BitBucket Pipelines

Authors personal Atlassian Cloud account was used to sign up for the free subscription of Bitbucket Cloud. Subscription is limited to less than 5 users, and less than 50 minutes of build time. No billing information was provided for this subscription.

5.8.1 SCM

Bitbucket Pipeline requires the importing of the repository from GitHub. This was created to be simple, and the only mandatory variables were source repository address, project name for the new project in Bitbucket and remote repository name.

After the import had completed a new “bitbucket-pipelines.yml” -file needed to be created to the root of the repository.

5.8.2 Managing credentials

Credentials were managed by repository variable settings. As these variables are used within shared runner instances, sensitive variables were secured. Table 8 describes the credentials that were created in order for the Pipeline to function as required.

Table 8 Description of Variables for Bitbucket Pipelines

Variable name	Secured
<code>docker_pass</code>	YES
<code>TEST_MONGODB_URI</code>	YES
<code>docker_user</code>	NO
<code>SECRET</code>	YES

5.8.3 Pipeline

Setting up Bitbucket Pipeline started by importing the repository from GitHub. This was fairly simple, and the only mandatory variables were remote repository URL, project name for the new project in Bitbucket and repository name. From there, a new “bitbucket-pipelines.yml” file needed to be created to the root of the repository.

Code 15 describes the basic workflow for creating of a pipeline. Pipeline was defined to use `node:12.22` container image as base image to run pipeline steps. The keyword “default” describes that the following steps on this pipeline are to be run on every push event that happens within the repository. Should there be more than one branch this could be substituted to use “branches” section to define that the pipeline should only be run on push events happening on certain branches. First step is then used to install dependencies and to create cache using predefined keyword “node” (translates to `node_modules` directory inside the containers working directory).

Code 15 YAML structure for Bitbucket Pipelines

```

image: node:12.22

pipelines:
  default:
    - step:
        name: Build
        caches:
          - node
        script:
          - npm install

```

After the dependencies have been installed, the tests are run using the cached directory.

When tests succeed a step for Docker image building and pushing is launched. As this build step is run within a container and we will need support for Docker commands it is mandatory to define that docker service should be used in this step, as described in snippet Code 16.

Code 16 Step to build and push docker image to Docker Hub container registry

```

- step:
    name: Build and Push Docker Image
    caches:
      - node
    script:
      - docker login -u $docker_user -p $docker_pass
      - docker build -t $docker_user/bloglist-backend-cicd:bitbucket .
      - docker push $docker_user/bloglist-backend-cicd:bitbucket
    services:
      - docker

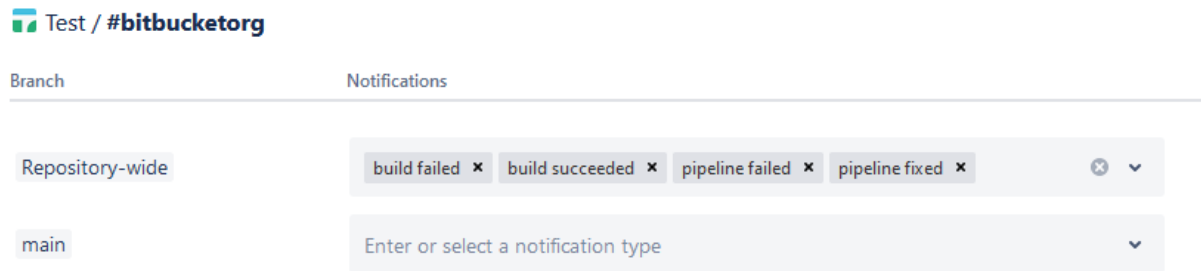
```

5.8.4 Integrations

Bitbucket Cloud was integrated with Slack by using the Atlassian provided Bitbucket app.

Application can be installed from the Bitbucket UI within the project settings, under the headline of Slack. After the application was installed, the necessary subscription was added to channel #bitbucketorg, with notifications configured to respond to build and pipeline events, as seen in Figure 20.

Figure 20 Description of Slack notification settings within a repository



Bitbucket Cloud was integrated with Jira Cloud. This was done through Bitbucket UI, by selecting the appropriate workspace and navigating to its settings, under the headline of Atlassian integrations and subitem Jira. As both of the instances are hosted in Atlassian Cloud, the Jira is automatically detected, and the applications can be connected with a click of a button. To track build statuses, the branch names must match issue keys.

5.8.5 Costs

No costs were accumulated during this thesis. And as such, no bill was sent to the author. The user account that was used for this thesis has a “Free” plan subscription on Bitbucket Cloud and is eligible for 50 minutes of build time. The limit was not exceeded.

6 Key findings

The aim of the thesis was to compare the different currently available CI/CD services by creating a simple CI/CD Pipeline. Pipeline was supposed to keep environment variables, mainly credentials, secret while running the pipeline. It was deemed essential to provide some easy method to communicate between the CI/CD service and Jira and Slack. The process of the thesis begun by exploring the documentations of different CI/CD services and getting the basic overview of the possible solutions and comparing the products.

Implementation phase was completed before the documentation or the writing of the theory sections of the thesis, as it seemed hopeless to get a complete picture of the needed services for the pipeline for every provider. Integration runs were run until they got successfully completed in each environment, sometimes requiring just a few runs and in other cases dozens of runs.

All of the services provided a way to read environment variables from either another service used to manage variables or from project or repository settings, except for the Google Cloud Platform. GCP required to think outside the box and the only way that was discovered to read secrets into environment variables was to create another container for the sole cause. They all offered a way to build the package in to a docker container image and upload it to a container registry. All of the different pipelines succeeded in this job with minimal customizations. Every service supports some form of the pipeline as a code definition.

During the process it was discovered that the creation of a CI/CD Pipeline is somewhat similar, and usually well instructed, in every service from the end user's perspective. From time consuming point of view, the workflows are well defined and similar, thus a single developer working on a specific project should be equally effective if they work within a single CI/CD service. From the point of perspective of project costs the services seemed to be similarly priced in the scope of this thesis, except for the Azure Pipelines. Azure Pipelines subscription that was bought for this thesis did include a Visual Studio license that needs to be accounted for when comparing products. During the comparison of the costs, it should be noted that Jenkins and Bamboo also generate costs from the infrastructure and management of the services. On the other hand, Azure, GCP and AWS can generate substantial costs from the services, storage and data transfer that are needed for a complete

implementation of CI/CD Pipeline. All the PaaS or SaaS services also bill by build time therefore it is possible to generate additional costs by launching an integration run that gets stuck in a job or some broken logic within the integration run that keeps transferring or storing same data over again.

SaaS implementations of CI/CD Pipelines do offer the most straightforward way to get a working CI/CD pipeline, but they do require that the source code is stored within the service. This does raise some questions, about the data's physical location, who can access the data and the effects of GDPR, that were out of the scope of this thesis but should be accounted for when planning on implementation of such pipelines.

When comparing the integrations, it should be noted that the Jira instance used in this thesis was hosted in Atlassian Cloud, and as such server instances may require different configurations or plugins. It should also be noted that as the Jenkins and Bamboo instances used in this thesis were run on a virtual machine, hosted on my personal workstation, they do not resemble the exact production like environment that would otherwise be used. This mainly plays a part when thinking about integrations that would need to be able to communicate with for example, the Bamboo instance, as it was not possible due to network limitations. However, this was carefully considered, and integration plugins were chosen in a way that would still accomplish the job. Features of the plugins were sometimes really poorly listed, thus causing extra work in the form of validating if the plugin actually accomplished the job it was needed for.

Access controls within the services were well defined, documented and easily granularly configurable. Services provided a way to configure user access for variables, repositories (when applicable) and the pipelines. All the services provided either a way to configure these on user, group or role level.

Subjectively, I would say that the easiest service was either GitHub Actions or Bitbucket Online. As both of the services are SaaS products, there is zero overhead on management side, and the monthly bill is easily estimated. However, as both of the services are managed by a third party, there is always a certain amount of uncertainty when considering, for example the location of the data or service availability. Jenkins and Bamboo on the other hand provide an on-premise installation, thus enabling users to control their own data and

the environment. These products do come with the price tag of the maintenance of the services and the workforce.

As for the feedback received for the study, Ambientia has been satisfied with the content. The study has met the expectations that were set at the initial discussions about the thesis' scope. However, it is obvious that the different products are not easily compared against each other based on this study alone. It is also relatively hard to come up with a valuable measurement unit to compare the ease of use of a singular service or system. Even the pricing is hard to compare as there are multiple different components affecting the pricing of a service, that may change over the night. Even though the comparison of the costs is included in this thesis, I would argue that it is rather meaningless.

7 Summary

As software development projects become larger and the code base gets harder to grasp, the importance of a valid continuous integration automation becomes more apparent. This offers benefits, including tighter feedback loop, enabling developers to fix errors sooner and thus providing higher trust in the release process.

It became apparent that the research questions defined for the thesis were too broad, which resulted in far more complex approach to writing the thesis than I was originally estimating. This also resulted in the thesis' structure becoming repetitive. However, all things considered the thesis did, in my personal opinion, succeed in comparing the pipelines and possibly guiding Ambientia in the future. The pipeline designed for this thesis was uncomplicated, but it does demonstrate the key aspects of such an implementation.

This thesis taught me a lot about estimating the workload and working through repetitive tasks, while also giving me the opportunity to venture in to the multiple implementations of one of the key areas in my current work field. During the writing of this thesis I learned a lot about the CI/CD Pipelines and the differences between the implementations of different service providers. The familiarity with continuous integration and continuous delivery as concepts is now much more well defined, and the understanding of the underlying structure is more concise for me. Working on this thesis also provided a more concrete understanding of the cost management in different cloud environments.

References

- Amazon Web Services. (n.d.-a). *AWS CodeBuild features*. Retrieved June 1, 2021, from <https://aws.amazon.com/codebuild/features>
- Amazon Web Services. (n.d.-b). *AWS CodeBuild Pricing | Amazon Web Services*. Retrieved August 8, 2021, from <https://aws.amazon.com/codebuild/pricing/?nc=sn&loc=3>
- Amazon Web Services. (n.d.-c). *AWS CodePipeline features*. Retrieved June 1, 2021, from <https://aws.amazon.com/codepipeline/features/>
- Amazon Web Services. (n.d.-d). *AWS CodePipeline Pricing | Amazon Web Services*. Retrieved August 8, 2021, from <https://aws.amazon.com/codepipeline/pricing/>
- Amazon Web Services. (n.d.-e). *AWS Systems Manager Parameter Store*. Retrieved June 1, 2021, from <https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html>
- Amazon Web Services. (n.d.-f). *How AWS CodePipeline works with IAM*. Retrieved June 1, 2021, from https://docs.aws.amazon.com/codepipeline/latest/userguide/security_iam_service-with-iam.html
- Amazon Web Services. (n.d.-g). *What is Continuous Delivery?* Retrieved June 25, 2021, from <https://aws.amazon.com/devops/continuous-delivery/>
- Atlassian. (n.d.-a). *Atlassian Marketplace - Azure Pipelines for Jira*. Retrieved May 29, 2021, from <https://marketplace.atlassian.com/apps/1220515/azure-pipelines-for-jira?tab=overview&hosting=cloud>
- Atlassian. (n.d.-b). *Bamboo - Pricing | Atlassian*. Retrieved August 8, 2021, from <https://www.atlassian.com/software/bamboo/pricing>
- Atlassian. (n.d.-c). *Bamboo Specs | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/bamboo-specs-894743906.html>
- Atlassian. (n.d.-d). *Bamboo Specs encryption | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/bamboo-specs-encryption-970268127.html>
- Atlassian. (n.d.-e). *Bamboo Specs Reference*. Retrieved August 8, 2021, from <https://docs.atlassian.com/bamboo-specs-docs/8.0.0/specs.html?yaml#version-information>

- Atlassian. (n.d.-f). *Bitbucket - Pricing | Atlassian*. Retrieved June 30, 2021, from <https://www.atlassian.com/software/bitbucket/pricing>
- Atlassian. (n.d.-g). *Configure bitbucket-pipelines.yml | Bitbucket Cloud | Atlassian Support*. Retrieved June 30, 2021, from <https://support.atlassian.com/bitbucket-cloud/docs/configure-bitbucket-pipelinesyml/>
- Atlassian. (n.d.-h). *Configuring plans | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/configuring-plans-289276853.html>
- Atlassian. (n.d.-i). *Get started with Bitbucket Pipelines | Bitbucket Cloud | Atlassian Support*. Retrieved June 29, 2021, from <https://support.atlassian.com/bitbucket-cloud/docs/get-started-with-bitbucket-pipelines/>
- Atlassian. (n.d.-j). *GitHub - atlassian/github-for-jira: Connect your code with your project management in Jira*. Retrieved July 30, 2021, from <https://github.com/atlassian/github-for-jira#using-the-integration>
- Atlassian. (n.d.-k). *GitHub for Jira | Atlassian Marketplace*. Retrieved July 30, 2021, from <https://marketplace.atlassian.com/apps/1219592/github-for-jira?hosting=cloud&tab=overview>
- Atlassian. (n.d.-l). *Integrating Bamboo with JIRA applications | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/integrating-bamboo-with-jira-applications-289276945.html>
- Atlassian. (n.d.-m). *Slack Notifications for Bamboo | Atlassian Marketplace*. Retrieved August 8, 2021, from <https://marketplace.atlassian.com/apps/1213289/slack-notifications-for-bamboo?tab=overview&hosting=server>
- Atlassian. (n.d.-n). *Understanding the Bamboo CI Server | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html>
- Atlassian. (n.d.-o). *Using Bamboo | Bamboo Server 8.0 | Atlassian Documentation*. Retrieved August 8, 2021, from <https://confluence.atlassian.com/bamboo/using-bamboo-289276852.html>

- Atlassian. (n.d.-p). *Variables and secrets | Bitbucket Cloud | Atlassian Support*. Retrieved June 30, 2021, from <https://support.atlassian.com/bitbucket-cloud/docs/variables-and-secrets/>
- Bansal, S. (n.d.-a). *Azure Pipelines integration with Jira Software*. Retrieved May 30, 2021, from <https://devblogs.microsoft.com/devops/azure-pipelines-integration-with-jira-software/>
- Bansal, S. (n.d.-b). *Integrate with Jira Issue tracking*. Retrieved May 30, 2021, from <https://github.com/microsoft/azure-pipelines-jira/blob/master/tutorial.md#faqs>
- Duvall, P. M., Matyas, S., & Glover, A. (2007a). *Continuous Integration : improving software quality and reducing risk*. Addison-Wesley.
- Duvall, P. M., Matyas, S., & Glover, A. (2007b). *Continuous Integration : improving software quality and reducing risk*. Addison-Wesley.
- Fowler, M. (n.d.). *Continuous Integration*. Retrieved November 9, 2021, from <https://martinfowler.com/articles/continuousIntegration.html>
- GitHub. (n.d.-a). *About billing for GitHub Actions - GitHub Docs*. Retrieved July 30, 2021, from <https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions>
- GitHub. (n.d.-b). *Encrypted secrets - GitHub Docs*. Retrieved July 16, 2021, from <https://docs.github.com/en/actions/reference/encrypted-secrets>
- GitHub. (n.d.-c). *Introduction to GitHub Actions - GitHub Docs*. Retrieved July 16, 2021, from <https://docs.github.com/en/actions/learn-github-actions/introduction-to-github-actions>
- GitLab. (n.d.-a). *A surprising benefit of CI/CD: Changing development roles | GitLab*. Retrieved November 16, 2021, from <https://about.gitlab.com/blog/2020/07/16/ci-cd-changing-roles/>
- GitLab. (n.d.-b). *CI/CD pipelines | GitLab*. Retrieved August 1, 2021, from <https://docs.gitlab.com/ee/ci/pipelines/>
- GitLab. (n.d.-c). *Get started with GitLab CI/CD | GitLab*. Retrieved August 1, 2021, from https://docs.gitlab.com/ee/ci/quick_start/
- GitLab. (n.d.-d). *GitLab CI/CD variables | GitLab*. Retrieved August 1, 2021, from <https://docs.gitlab.com/ee/ci/variables/>
- GitLab. (n.d.-e). *GitLab Runner | GitLab*. Retrieved August 1, 2021, from <https://docs.gitlab.com/runner/>

- GitLab. (n.d.-f). *GitLab Slack application | GitLab*. Retrieved August 7, 2021, from https://docs.gitlab.com/ee/user/project/integrations/gitlab_slack_application.html
- GitLab. (n.d.-g). *Jira integrations | GitLab*. Retrieved August 7, 2021, from <https://docs.gitlab.com/ee/integration/jira/>
- GitLab. (n.d.-h). *Slack notifications service | GitLab*. Retrieved August 7, 2021, from <https://docs.gitlab.com/ee/user/project/integrations/slack.html>
- Google. (n.d.-a). *Cloud Build pricing | Cloud Build Documentation | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/build/pricing>
- Google. (n.d.-b). *Overview | Cloud IAM Documentation | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/iam/docs/overview>
- Google. (n.d.-c). *Overview of Cloud Build | Cloud Build Documentation | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/build/docs/overview>
- Google. (n.d.-d). *Pricing | Secret Manager | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/secret-manager/pricing>
- Google. (n.d.-e). *Quickstart: Build | Cloud Build Documentation | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/build/docs/quickstart-build>
- Google. (n.d.-f). *Secret Manager conceptual overview | Secret Manager Documentation*. Retrieved May 29, 2021, from <https://cloud.google.com/secret-manager/docs/overview>
- Google. (n.d.-g). *Service accounts | Cloud IAM Documentation | Google Cloud*. Retrieved May 29, 2021, from <https://cloud.google.com/iam/docs/service-accounts>
- Jenkins.io. (n.d.-a). *Installing Jenkins* . Retrieved June 20, 2021, from <https://www.jenkins.io/doc/book/installing/>
- Jenkins.io. (n.d.-b). *Pipeline*. Retrieved June 15, 2021, from <https://www.jenkins.io/doc/book/pipeline/>
- Jenkins.io. (n.d.-c). *Jira | Jenkins plugin*. Retrieved August 8, 2021, from <https://plugins.jenkins.io/jira/>
- Jenkins.io. (n.d.-d). *Slack Notification | Jenkins plugin*. Retrieved August 8, 2021, from <https://plugins.jenkins.io/slack/>
- Jenkins.io. (n.d.-e). *Using a Jenkinsfile*. Retrieved June 15, 2021, from <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>
- Jenkins.io. (n.d.-f). *Using credentials*. Retrieved June 15, 2021, from <https://www.jenkins.io/doc/book/using/using-credentials/>

- JFrog. (n.d.). *Top 5 CI/CD Tools to Look Out for in 2021*. Retrieved November 16, 2021, from <https://jfrog.com/knowledge-base/top-5-ci-cd-tools-to-look-out-for-in-2021/>
- Microsoft. (n.d.-a). *Pricing for Azure DevOps*. Retrieved May 29, 2021, from <https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/>
- Microsoft. (n.d.-b). *Azure Pipelines agents*. Retrieved May 30, 2021, from <https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser>
- Microsoft. (n.d.-c). *Azure Pipelines with Slack*. Retrieved May 29, 2021, from <https://docs.microsoft.com/en-us/azure/devops/pipelines/integrations/slack?view=azure-devops>
- Microsoft. (n.d.-d). *Define variables*. Retrieved May 29, 2021, from <https://docs.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops&tabs=yaml%2Cbatch>
- Microsoft. (n.d.-e). *Key concepts for new Azure Pipelines users*. Retrieved May 30, 2021, from <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops>
- Microsoft. (n.d.-f). *What is Azure Pipelines?* Retrieved May 30, 2021, from <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops>
- Nikhil, P. (2017). *Learning Continuous Integration with Jenkins* (Second Edition). Packt.
- Red Hat. (n.d.). *What is a CI/CD pipeline?* Retrieved June 25, 2021, from <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>
- Slack. (n.d.). *GitHub for Slack | Slack*. Retrieved July 30, 2021, from <https://slack.com/intl/en-fi/help/articles/232289568-GitHub-for-Slack>

Liite 1: Aineistonhallintasuunnitelma

Tutkimuksellinen työ:

Työtä varten ei tehdä haastatteluja tai kyselyitä.

Työtä tehdään virtuaalikoneella, minkä tiedostojärjestelmä on varmuuskopioinnin piirissä. Tämän lisäksi työtä siirrellään työasemien välillä käyttäen Nextcloud palvelua, missä on tiedoston versiointi päällä.

Käytetyt kuvat on varmistettu käyttöoikeudeltaan hyväksyttäväksi.

Aineistoa ei ole tarpeen säilyttää, kaikki aineisto on julkisesti saatavilla lähdeluettelon mukaisesti.

