

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики



**РОЗРОБКА УНІФІКОВАНОГО CI/CD PIPELINE ДЛЯ  
ПІДТРИМКИ, РОЗРОБКИ ТА СУПРОВОДУ ВЕБ ЗАСТОСУВАНЬ**

**Текстова частина**

**магістерської роботи**

**за спеціальністю „Інженерія програмного забезпечення” 121**

**Керівник магістерської роботи  
професор, д.н М. М. Глибовець**

\_\_\_\_\_  
(підпис)

**“ \_\_\_\_ ” \_\_\_\_\_ 2021 р.**

**Виконав студент Т.В. Торба**

**“ \_\_\_\_ ” \_\_\_\_\_ 2021 р.**

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики, к.ф.-м.н.

\_\_\_\_\_ С. С. Гороховський  
(підпис)

„\_\_\_\_\_” \_\_\_\_\_ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ  
на магістерську роботу

студенту 2 р.н. магістерської програми Інженерія програмного забезпечення Торбі  
Тетяні Вікторівні

Розробити Розробка уніфікованого CI/CD pipeline для підтримки, розробки та  
супроводу веб застосувань"

Зміст текстової частини до магістерської роботи:

Зміст

Анотація

Вступ

1 Дослідження галузі та аналіз впливу веб застосунків на CI/CD

2 Проектування CI/CD pipeline для веб застосунків

3 Реалізація CI/CD pipeline для веб застосунків

4 Огляд результатів

Висновки

Список літератури

Додатки

Дата видачі „\_\_\_\_\_” \_\_\_\_\_ 2020 р.

Керівник

М.М. Глибовець, професор, д.н

\_\_\_\_\_  
(підпис)

Завдання отримав

Т.В. Торба

\_\_\_\_\_  
(підпис)

Work schedule:

№	Name of the phase	Deadline	Notes
1	Obtaining the theme of the course work.	25.10.2020	
2	Thematic materials researching and collection.	10.11.2020	
3	Work structure discussion.	01.12.2020	
4	Introduction.	24.12.2020	
5	Theoretical part	15.01.2021	
6	The practical part implementation.	15.03.2021	
7	Analysis and description of the practical part	06.04.2021	
8	Conclusion.	08.05.2021	
9	Work revision.	10.05.2021	
10	Corrections according to the mentor's remarks.	13.05.2021	
11	Presentation.	17.06.2021	

Керівник  
М.М. Глибовець, професор, д.н

\_\_\_\_\_  
(підпис)

Студент  
Т.В. Торба

\_\_\_\_\_  
(підпис)

„\_\_\_” \_\_\_\_\_ 2021 р.

## Table of Contents

Table of Contents	4
ABSTRACT	5
INTRODUCTION	6
1. Industry research and impact analysis CI / CD for web applications	9
1.1 Collaboration and planning	10
1.2 Server configuration and infrastructure as code	12
1.2.1 Infrastructure as Code	12
1.2.2 Cloud computing	12
1.3 Continuous integration and continuous delivery/deployment	13
1.3.1 Baseline development lifecycle	13
1.3.2 Elements of a CI/CD pipeline for web application	14
1.3.3 CI/CD as a single pipeline for Web application	16
1.4 Containerization and orchestration	17
1.5 Monitoring and alerting	18
1.6 Security and DevSecOps	21
1.6.2 Building Security In Maturity Model ( BSIMM)	21
1.6.3 Web Application Security Risks (OWASP)	22
1.6.4 Red and Blue Team Approach	23
2 CI\CD pipeline implementation for web applications	25
2.1 MVP: Web applications requirements	25
2.1.1 Functional requirements	25
2.1.2 Nonfunctional requirements	26
2.2 CI\CD pipeline	27
2.2.1 Source Stage	27
2.2.2 Build Stage	31
2.2.3 Test Stage	39
2.2.4 Deploy Stage	41
2.2.5 Monitoring	46
CONCLUSION	47
REFERENCES	49
ABBREVIATIONS AND TERMS	50
APPENDIX 1	51
APPENDIX 2	54

## **ABSTRACT**

In this work the main approaches for building CI\CD are reviewed, also the problems are described that might arise during CI\CD construction. At the same time in work modern approaches are analyzed and characterized with their pros and cons.

After architecture analysis is done. During the design and development stages we outlined a set of requirements that CI\CD should meet, worked out a system diagram and set of necessary tools.

In the end, several deployments are done in order to test CI\CD.

Keywords: CI\CD, Jenkins, Jira, Web application, Agile, Security.

# INTRODUCTION

## **The relevance:**

If we turn to history, we note that in the 80s and 90s the dominant development methodologies followed the waterfall process. The waterfall process dictated the rules and approaches, which were used in software development processes. Development followed a strictly sequential structure resulting in delays in time to market and a mismatch between product and business expectations at the time of launch. Taking into consideration business needs, the movement toward a more light approach started.

In 2001, 17 pioneering software developers met in Snowbird, Utah. Together they published the Agile Manifesto, which set out to find ways to build software that would be more collaborative, responsive and deliver more frequent updates. Later, in 2009, another group created the Scrum Alliance. Agile adoption had been growing slowly but steadily and by 2008 it had reached 13% uptake. By the next year, this had jumped to 17% and by 2010 it was 37%. Now based on research more than 90% of software development companies use Agile methodology [4].

Once Agile had gained momentum the necessity of a new process block was appeared - Developers + Operators (DevOps). DevOps was a new interpretation of what collaborative software development should be like. While agile brought development and testing together, DevOps allowed the development team to expand with non-business stuff and created an environment where new releases could be delivered faster and more frequently.

In order to meet the requirement of fast and frequent deployment a new pipeline appeared with three distinct stages: Continuous integration, Continuous delivery and Continuous deployment. At the moment, none of the projects is being developed without using the above concepts.

## **The aim of study:**

Develop a modern CI \ CD pipeline architecture and create a CI \ CD mechanism based on the developed architecture.

**The goal:**

Analyze existing approaches for the development of CI\CD pipeline architectures. Examine and explore software products that can be used as components in the CI\CD pipeline architecture. Choose the components that are best suited to solve the main goal. Choose approaches and tools for the practical implementation of the developed architecture.

**The object of study:**

Continuous Integration and Continuous Delivery pipeline for web application, specialized solutions for CI\CD development, software products for building and supporting existing architecture.

**The subject:**

Pipeline components, their capabilities for integration, API, application development with integration and organization of interaction of the studied components.

**Sources of research:**

Electronic versions of printed literature, software documentation, reference books to API links, electronic resources, including specialized forums and virtual conferences, source codes for programs and libraries, video instructions.

**Scientific novelty of the obtained results:**

Research is to create a modern CI\CD architecture, the introduction of new approaches to solving the problem of code delivery to the end user with minimal errors and timely.

**The practical significance of the obtained results:**

Due to the use of advanced practices in the development of architecture, the cost of resources for the development of software developed on the basis of the developed architecture is reduced. Optimization occurs through the use of reliable components that

are easily configurable and adaptable, the architecture is designed with the possibility of further independent expansion.



## **1. Industry research and impact analysis CI / CD for web applications**

Continuous integration (CI) is the software development practice of regularly integrating code changes into a shared code repository. Each commit triggers a build during which tests are run that help to identify if anything was broken by the changes. It aims at building, testing, and releasing software with greater speed and frequency broken by the changes.

Benefits of Continuous Integration for Businesses:

- Reliable, high-performing releases
- Reduce costs from fewer outages
- Software is delivered to market faster

Benefits of Continuous Integration for Development:

- Smaller code changes
- Test reliability
- Risk mitigation

Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, doing so manually. [5]

Benefits of Continuous delivery for Businesses:

- Reduce the costs of delivery to keep deliverable in a release-ready state
- Accelerate time-to-value
- Support data-driven decision making
- Increase quality
- Allow experimentation and innovation

Benefits of Continuous delivery for Development

- Streamline workflows
- Enhance teamwork
- Stable and highly available environments
- Test every edit and enhancement
- Better end-to-end visibility to trace the changes and error codes

Continuous deployment (CD) is a software release process that uses automated testing to validate if changes to a codebase are correct and stable for immediate autonomous deployment to a production environment [6].

### Benefits of Continuous deployment for Businesses:

- Improve customer satisfaction
- Gather metrics about application performance
- Streamline communication
- Collecte rapid feedback
- Decrease time-to-market of new features
- Scale from a single application to an Enterprise IT portfolio.

### Benefits of Continuous deployment for Development

- End-user involvement and feedback
- The backlog of non-critical defects is lower
- Mean time to resolution (MTTR) is shorter
- Containerization
- Single method of deployment
- Release cycles are shorter with targeted releases
- Test-driven development

Regardless of what is being developed, DevOps practices must be in development. The processes become increasingly programmable and dynamic, using a DevOps approach. The general idea can be explained with the CAMS acronym:

- Culture of collaboration and shared responsibilities between the team members,
- Automation of the manual processes,
- Monitoring and feedback gathering, and
- Sharing of responsibilities and information.

## **1.1 Collaboration and planning**

One of the keys of DevOps approach is collaboration within the team and supplier. Nowadays the team consists of not only developers, there can be devs and ops, testers, and designers, release managers, product and project managers etc. The members of the team should collaborate closely, share common responsibilities and should be involved in each stage of development.

The purpose of requirements management tools are to ensure product development goals are successfully met. It is a set of techniques for documenting, analyzing, prioritizing, and agreeing on requirements so that engineering teams always have current and approved requirements.

All participants of CI/CD need to be notified of various CI/CD events depending on the release workflow of the particular organization. If there are any manual steps or approvals needed in the deployment pipeline, the team needs to take action quickly so as not to hold up the release. All communication should be targeted and strictly limited in order to avoid spam and miscommunication within team members.

The common tools for usage are Trello, Jira, Clarizen, Bitbucket Server, Asana. These applications are project management tools, which support standard features for task management, time tracking, planning, invoicing, chatting, etc.

The second step towards efficiency of the developing process is IDEs and Editors for the project team.

A common IDE includes these main features: Text Editor, Compiler or Interpreter, Build and Debugger, Syntax Highlighter, Graphical User Interface (GUI)

According to GitHub data Top 3 IDEs are Visual Studio, Eclipse, Visual Studio Code (Figure 1.1). [7]

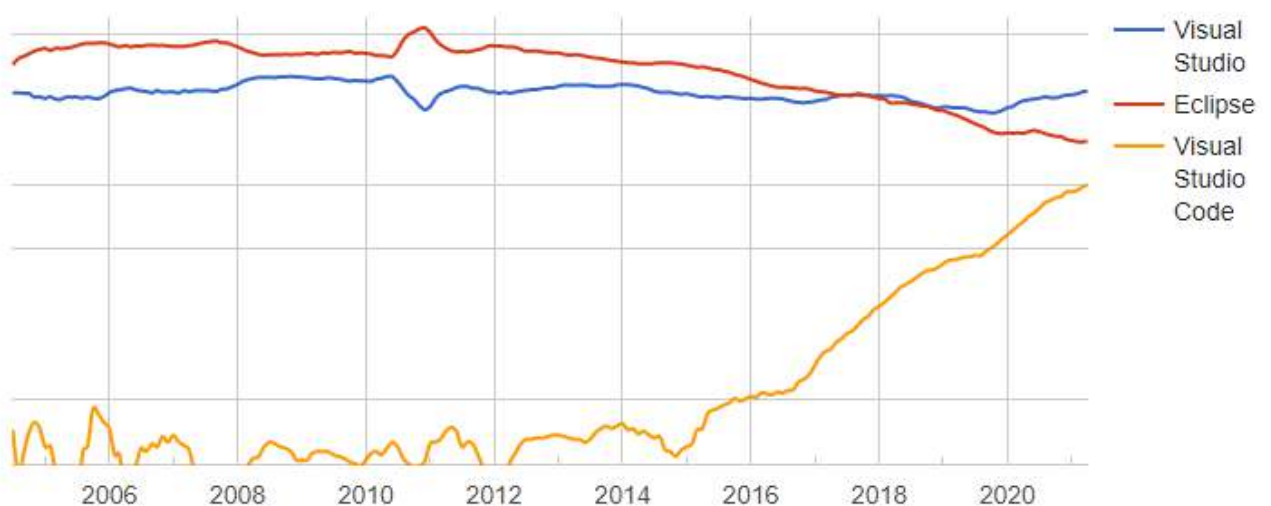


Figure 1.1 Worldwide trend for TOP3 IDEs for Web Application

## **1.2 Server configuration and infrastructure as code**

### **1.2.1 Infrastructure as Code**

Infrastructure as code is DevOps practices like CI\CD and Cloud Technologies and brings more value from DevOps adoption within an organization. Infrastructure as Code is the way of describing all software IT infrastructure, such as virtual machines, security rules, network settings in a simple code way that can be stored and controlled in your Version Control System (VCS) and versioned on demand. These repositories of records are called manifests. They are utilized by DevOps tools like Terraform, Kubernetes and other proprietary services from Cloud Providers, such as AWS CloudFormation, to automatically provide and configure new servers and their infrastructure for any kind of environment.

IaC ensures continuity, as all the environments are provisioned and configured automatically, with no ability for human error, which incredibly accelerates and streamlines the product advancement and foundation tasks. The IaC approach provides traceability for infrastructure changes and the easy-to-restore way of infrastructure deployment. That empowers overall IT infrastructure stability and availability. There are also multiple benefits to conducting operations based on the IaC principle.

Well-known apps which allow for automating and bringing infrastructure as code into the system are Puppet, Chef, Ansible, Terraform, SaltStack, AWS Cloudformation.

### **1.2.2 Cloud computing**

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet.[8]

Cloud computing makes the adaptation of DevOps easier by engaging each progression of the software development lifecycle. Through the cloud, applications can be built and tested on different conditions with different environments, OS versions and devices emulated. Excluding the physical server, the test requirement ensures time savings and cost reduction due to the on-demand nature of the cloud. Cloud computing brings new possibilities in CI\CD and automation processes. The cloud offers to transfer

risks and improve overall IT infrastructure reliability through availability zones and the geolocation model used by most popular cloud service providers.

The most usable cloud infrastructure providers are Amazon Web Services, Google Cloud Platform, Microsoft Azure, IBM Cloud and Oracle Cloud Platform.

Five main Cloud Computing characteristics include:

- Self-service and on-demand
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

Three models (IaaS, PaaS, SaaS) introduce a kind of abstraction level. It is a level you don't think about. It's a level of generalization. If you simplify the picture given below, you can say that the upper level is customer responsibilities, and the lower level is the responsibilities of cloud providers or IT operation.

### 1.3 Continuous integration and continuous delivery/deployment

#### 1.3.1 Baseline development lifecycle

The core phases of the development life cycle (Figure 1.2) typically include a planning stage, followed by coding, building, integration, testing, release, and then deployment to the production environment.



Figure 1.2 Development life cycle

Within Agile methodology, which takes the iterative approach to software development, by adding layer upon layer, building, or rather evolving, the application one sprint at a time (Figure 1.3).



Figure 1.3 Development life cycle vs Development Environment

In terms of DevOps typically this methodology can overlap with Agile (Figure 1.4)

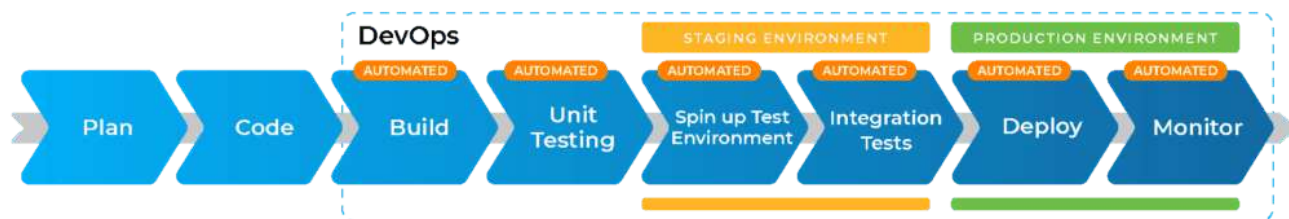


Figure 1.4 Development life cycle vs DevOps methodology

But, there are a lot of people mentioning DevOps and CI\CD together. It is very important to understand that these two concepts complement each other but don't replace each other.

While DevOps propose a culture/philosophy of team collaboration and work transparency, CI/CD are the pillars that enable DevOps. CI/CD focuses on automation of building, testing and deployment by properly setting up a CI/CD pipeline with appropriate CI/CD tools.

### 1.3.2 Elements of a CI/CD pipeline for web application

Most web application releases go through a couple of typical stages:

**Source stage:** The common approach is triggers, which are customized depending on needs. In some cases a pipeline run is triggered by a source code repository or can be automatically scheduled or supported user-initiated workflow. Also it can be integrated with additional tools like JIRA or Confluence in order to organize a full cycle for documents management. In this stage several steps need to be performed in order to improve quality code and speed of the market. Basically the operations performed in the Local Development Environment. The steps can be followed: compilation, unit testing, static code analysis and manual verification. After the feature is ready or on a daily basis changes will push to the remote feature branch and move to the build stage.

**Build stage :** In this stage we combine the source code and its dependencies to build a runnable instance of our product that we can potentially ship to end users. Depending on languages, programs need to be compiled (for example Java, C/C++, Go) or not (for example Ruby, Python and JavaScript). If concentrating on cloud-native software, it is typically deployed with Docker and in this stage the application builds the

Docker containers. In remote feature branches static code analysis, compilation and unit testing will run in order to check on the server's side. If automatic tests are passed, changes will be pushed to the feature environment where next steps will be performed - compilation, Auto API integration, testing, new feature testing and deployment. If code is approved by the Product Owner or responsible person, merge requests created and sent to code review.

Test stage: Here runs automated tests to validate code's correctness and the behavior of a web application. The main purpose of the stage is to prevent easily reproducible bugs from reaching the end-users. After code review code pushes to the Release branch for integration testing and unit testing. Taking into consideration that testing is the one of the important parts of development and enhancement of quality, testing in a UAT environment is important. In a UAT environment smoking and system testing are performed and should be approved by business users.

Deploy stages: There are usually multiple deploy environments: Development, QA, Staging and Production. Teams that have embraced the Agile model of development: guided by tests and real-time monitoring: usually deploy work-in-progress manually to a staging environment for additional manual testing and review, and automatically deploy approved changes from the master branch to production.

Performance testing is the process during which a product's quality or its ability to function in the required environment is evaluated. As a non-functional testing technique, performance testing is conducted to evaluate a systems ability in terms of responsiveness and stability under workload. This process is conducted on three major attributes, which includes scalability, reliability and resource usage.

- A number of techniques can be used to check performance of a software or hardware in a performance testing environment setup. This includes:
- Load testing is the simplest form of testing and is basically conducted to understand the behaviour of the system under a specific load.
- Stress testing is conducted to check a system's ability to cope with the increased load, if any. It is performed to determine the maximum capacity of the existing system

- Soak testing, also known as endurance testing, is a type of testing done to verify a system's ability to perform in situations of continuous load.
- Spike testing is conducted by abruptly increasing the number of users of a system and determining how the system performs under such load.

There are a number of simple ways in which one can ensure accuracy and better results in the tests. [16]

### **1.3.3 CI/CD as a single pipeline for Web application**

Continuous integration and delivery/deployment (CI/CD) is performed via a single pipeline with high automation at every stage of the process.

For the purpose described above, the following apps can be used: Jenkins, Gradle, GitLab CI, Travis CI, Bamboo CI, Teamcity, GitHub workflows, Circle CI, and Azure. In terms of categories, applications are divided into several categories.

**Installable software:** The applications which can be installed and configured for initial purpose. It can be open source or proprietary

**SaaS:** Services with a web interface provided by 3rd party (e.g., CircleCI, Azure DevOps)) and can be self-hosted or in the cloud.

**Integrated software:** Most repositories (e.g. GitLab, GitHub, BitBucket) support their own CI/CD web services that are integrated into their source code control systems.

Which category is chosen depends on the specific application and the requirements under consideration.

The tools can be evaluated based on there are lot of criteria like hosting, free version, price and integration possibilities (Figure 1.5)




	 Jenkins	 circleci	 TeamCity	 Bamboo	 GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★ ★ ★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★	★ ★ ★ ★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build Agent License Pricing	Free	From \$39 per month	From \$299 one-off payment	From \$10 one-off payment	From \$4 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

Figure 1.5 Top 5 CI/CD tools in 2020

## 1.4 Containerization and orchestration

Containers are popular among DevOps, because they offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. Orchestrators are the tools used to monitor and configure all the existing containers. Most often they come built into CI/CD pipelines as default tools or can be plugged in as extensions. In web applications dozens of containers will be interconnected making up the app.

Widely known providers for containers are Docker, Kubernetes, OpenShift.

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc. [17]

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and

automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available. The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.[18]

OpenShift service built around Docker containers orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux. The family's other products provide this platform through different environments: OKD serves as the community-driven upstream (akin to the way that Fedora is upstream of Red Hat Enterprise Linux), OpenShift Online is the platform offered as software as a service, and OpenShift Dedicated is the platform offered as a managed service. The OpenShift Console has developer and administrator oriented views. Administrator views allow one to monitor container resources and container health, manage users, work with operators, etc. Developer views are oriented around working with application resources within a namespace. OpenShift also provides a CLI that supports a superset of the actions that the Kubernetes CLI provides. [19]

## **1.5 Monitoring and alerting**

If one of the stages of CI/CD was failed or downgraded, all involved participants should be aware regarding the problem. For this purpose exist a range of tools and methods to application monitoring via a dedicated interface.

Performance monitoring gets more and more evolved as a necessary part of business processes. The availability and reliability of the system have a direct impact on a company's bottom line. Business wants to know how transactions are processed and what's going on at certain application components.

Complex systems and applications face a lot of issues and technical problems while working in production environments. The time to identify the problem may vary from 30 seconds to 30 hours or even more. But there are ways to decrease this time and improve the problem-solving of an IT System team. Monitoring is applied to handle and tackle this kind of problem.

Monitoring is the systematic process of collecting, analyzing, and using the information to track a program's progress toward reaching its objectives and guide management decisions [14]

Businesses need to utilize monitoring techniques across their networks, physical and virtual servers, and services. Gaining more visibility with monitoring, it is easier for System engineering teams to pinpoint, assess, and fix a problem within an application. Monitoring is also widely used in Security and Application performance management. Businesses derive a lot of value from monitoring, especially to:

- Decrease the costs spent on problem identification.
- Reduce the amount of time to solve an issue.
- Minimize the risk of a data leak/breach.
- Reduce the amount of time to solve an issue.
- Increase the revenue because the application will be available more time, and downtime will be reduced.

Four pillars (aka sources) of application monitoring:

- Business KPIs, Service-Level Agreement, web-environment data
- Infrastructure, technical equipment
- Databases, transaction, user requests
- Network data, user-behavior

The monitoring process consists of multiple steps. In order to obtain the value from it, each of these steps should be implemented in the following order:

Gather information from data producers.

- Ingest it.
- Transform the data.
- Search and analyze.
- Perform data visualization.

Data for monitoring can be collected from multiple sources. Monitoring tools enable appliances to configure many kinds of data inputs, including those specific to particular application needs. Some products provide the user with an option to configure any arbitrary data input types.

The most common data sources in Monitoring are:

- Directories and files
- Network events.
- Windows sources.
- Metadata.

Monitoring requires collecting and making the system observable. For that purpose, various metrics are used. Monitoring is used by Developers, IT Operations team, Security team, Technical Support, Business leaders. Monitoring can be augmented with various tools and methods. One of the most popular now is AIOps. It uses artificial intelligence to simplify IT operations management and accelerate and automate problem resolution in complex modern IT environments.

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes.[15].

Prometheus supports the collection of four main metric types. Prometheus architecture is modular and has some already available modules called exporters, which capture metrics from a software. Prometheus architecture is modular and has some already available modules called exporters, which capture metrics from a software.

As a communication channel following tools can be used:

Slack is a channel-based messaging platform which supports bots. A bot is a nifty way to run code and automate tasks. The bot can be used for posting messages in channels and reacting to members' activity. In terms of CI/CD slack can send messages to audiences who have subscriptions to the bot and notify regarding CI/CD processes and release results.

Telegram is a freeware, cross-platform, cloud-based instant messaging (IM) software and application service. The CI/CD can send messages directly to person or use the bot, which notify about processes or errors which occur.

Email is a standard and common way of communication which the digital world gets used to. Emails can potentially cause information overload. Some messages may be dismissed or left unread, especially if there are a lot coming in.

For monitoring health of system next application with dedicate interface can be used: Prometheus, Sensu GO, Nagios, Elastic Stack

## 1.6 Security and DevSecOps

### 1.6.1 Security development lifecycle

With the advances of cloud computing and automated pipelines, more security vulnerabilities start to appear - what the developers are working on are the flaws and vulnerabilities of the application itself. The direction SDL or SDLC - Security development lifecycle - was developed by Microsoft (Figure 1.6).

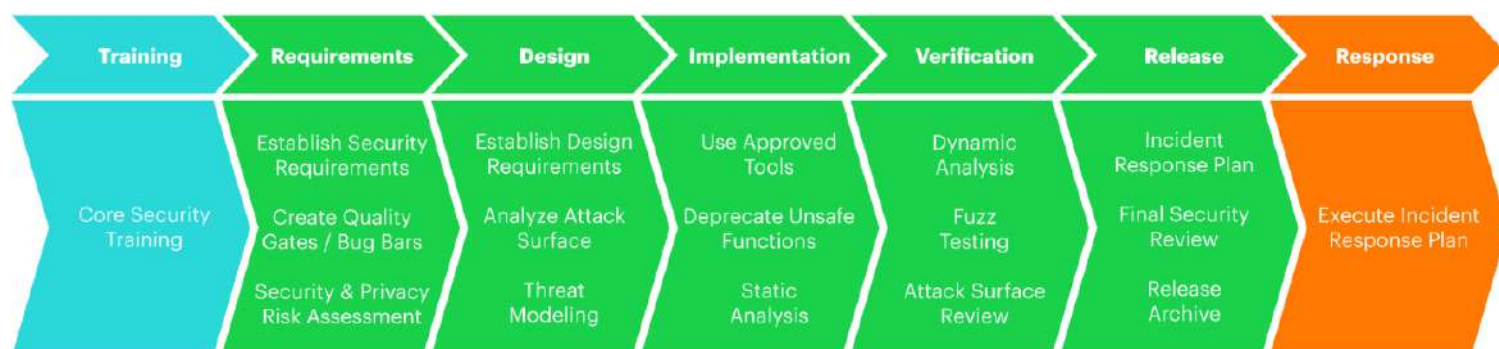


Figure 1.6 Application Security

Application security and SDLC are not about detecting vulnerabilities, but about preventing them from occurring. SDLC is detailed in various methodologies - OpenSAMM, BSIMM, OWASP

### 1.6.2 Building Security In Maturity Model ( BSIMM)

The methodology is based on dividing the Application Security process into 4 domains: Governance, Intelligence, SSDL Touchpoints, and Deployment. Each domain has 12 practices with 112 activities. This methodology can be used as a framework for security implementation (Figure 1.7)

## Building Security In Maturity Model



Figure 1.7 Development life cycle vs Development Environment

DevSecOps is an approach to discover the security of infrastructure and CI/CD pipeline. Codacy, SonarQube and Logz.io are commonly used for checking.

### 1.6.3 Web Application Security Risks (OWASP)

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. The standard includes next:

1. Injection. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query.
2. Broken Authentication. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
3. Sensitive Data Exposure. Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII.
4. XML External Entities (XXE). Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

5. Broken Access Control. Restrictions on what authenticated users are allowed to do are often not properly enforced.
6. Security Misconfiguration. Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.
7. Cross-Site Scripting (XSS). XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript.
8. Insecure Deserialization. Insecure deserialization often leads to remote code execution.
9. Using Components with Known Vulnerabilities. Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application.
10. Insufficient Logging & Monitoring. Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. [9]

#### **1.6.4 Red and Blue Team Approach**

The Red and Blue Team came from the military and was one of the first approaches to test security. Where defenders or friendly forces were designated by the Blue Team and the attacker's forces by the Red Team. And despite its 'offensive' nature, the Red Team is an excellent defender. They enable organizations to better defend themselves against hacker attacks as they try to simulate them with precision. But there are not only 2 teams (Figure 1.8). In some situations, the efforts of both teams are required, and this is how the Green, Yellow, Purple Team came about:

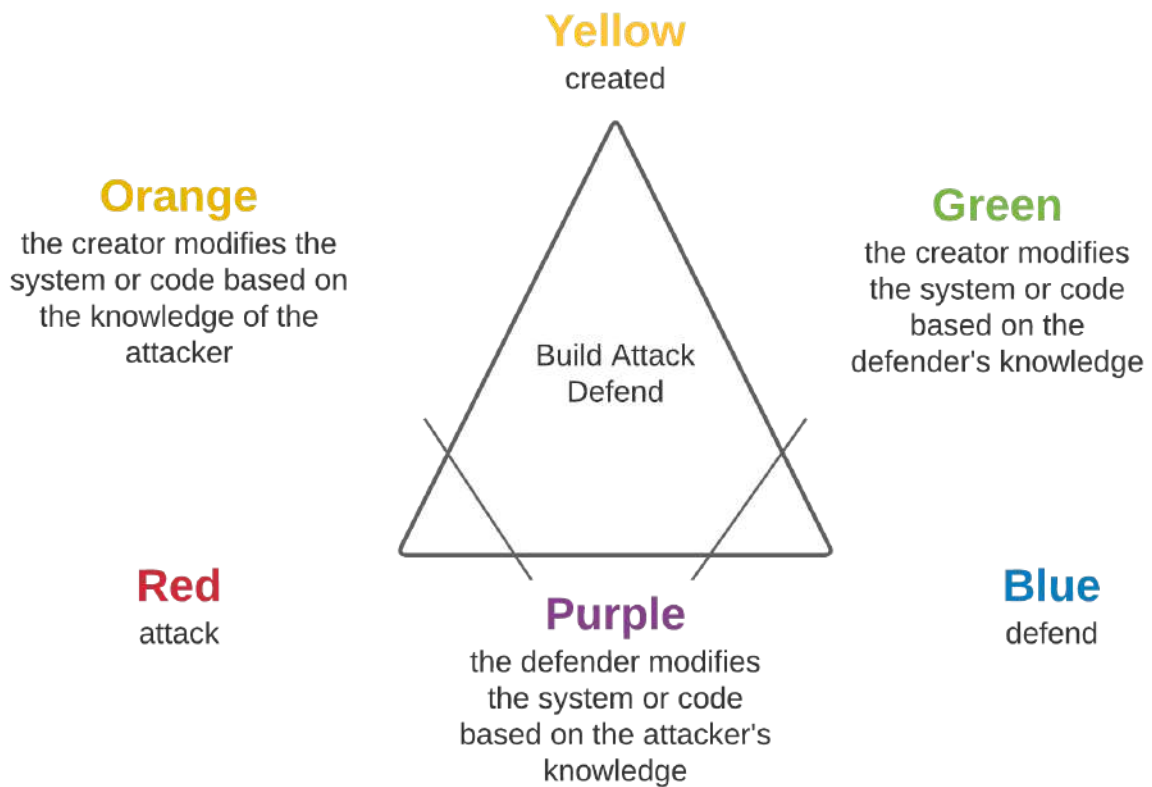


Figure 1.8 Attack Defend

If we are talking about purple Team - combining the skills of Red and Blue Team. Both teams work together to provide a complete audit. The red team provides detailed logs of all the operations performed, and the blue team fully documents all corrective actions that were taken to resolve the problems found during testing. The Purple Team has become commonplace in the security world over the past few years. The Purple Team can be a consulting group hired to conduct an audit, or company employees directly, but they do not focus solely on offense or defense. In terms of Orange and Green teams they are more related to soft developers.



## 2 CI\CD pipeline implementation for web applications

### 2.1 MVP: Web applications requirements

Overall Description : Web solutions provide the possibility to buy the defined set of products to customers without going to the physical shop. From a seller perspective it allows the seller to sell the product.

#### 2.1.1 Functional requirements

##### EPICs:

1. Minimum steps to make a purchase.

**Fact:** More than 74% of online shoppers abscond before completing an online transaction due to complex check-out process [1].

**Feature:** The streamlined shopping card with clear check - out process to support single one page flow.

2. Robust Return Policy

**Fact:** 42% of online shoppers have returned an item they bought online and also 63% of online shoppers said that they would not purchase if they couldn't find the return policy. [1].

**Feature:** The separate page with return policy which easy to find and understand

3. Mobile-friendliness.

**Fact:** Mobile commerce sales comprised 63.5% of the total sales in ecommerce in 2018.[1]

**Feature:** Mobile - friendly website with simple, defused and responsive design including customizable themes (RWD) [2]

4. Content management system

**Fact:** Nearly 55% of marketers' top priority will be content creation. 46% of consumers want product comparisons and 42% of customers want more testimonials. Along those same lines, 69% of online shoppers want more reviews. [1]

**Feature:** The tool allows CRUD [3] operation with site content including product reviews.

5. Social media integration

**Fact:** 30% of online shoppers say they would be likely to purchase from a social media network like Facebook, Pinterest, Instagram, Twitter or Snapchat.

**Feature:** Integrate social media to support cost effective way to promote and buy the product

#### 6. Customer Support and ChatBots

**Fact:** “When it comes to making a purchase, 64% of customers find customer experience more important than price.” [1]

**Feature:** Provide an state-of-the-art interface to customers to get immediate support when needed.

### 2.1.2 Nonfunctional requirements

**Performance:** The website’s load time should not be more than one second for users.

**Reliability:** Users can access their basket 98% of the time without failure.

**Availability:** In the case of unplanned system downtime, all features will be available again after one working day.

**Maintainability:** If the automated email services become unavailable, they can be under maintenance for approximately three hours.

**Recoverability:** If a major incident happens on the website, the business must take measures to go back to being fully operational within one days.

**Capacity:** Up to 5000 buyers can request for the one item. Up to 1,000,000 items can be stored.

**Serviceability:** The user's automated emails can be edited and replaced by uploading an XML file; there’s no need to recompile any code.

**Security:** Only the users with the role “site admin” can view the users personal data.

**Manageability:** When editing the code for applicants’ profile pages, the rest of the site stays up and running.

**Data integrity:** The system shall maintain data integrity by keeping backups of all updates to the database for every record transaction.

**Interoperability:** The website must follow the service-oriented architecture.

**Usability:** The website's interface has to be user-friendly and easy to use.

## 2.2 CI\CD pipeline

### 2.2.1 Source Stage

Any development starts from business problems which need to be solved in order to achieve business goals. For requirements management and issue tracking Jira and Confluence can be used and should be integrated into our pipeline. In Figure 2.1 EPIC was created and added issues and bugs which need to be solved in current sprint

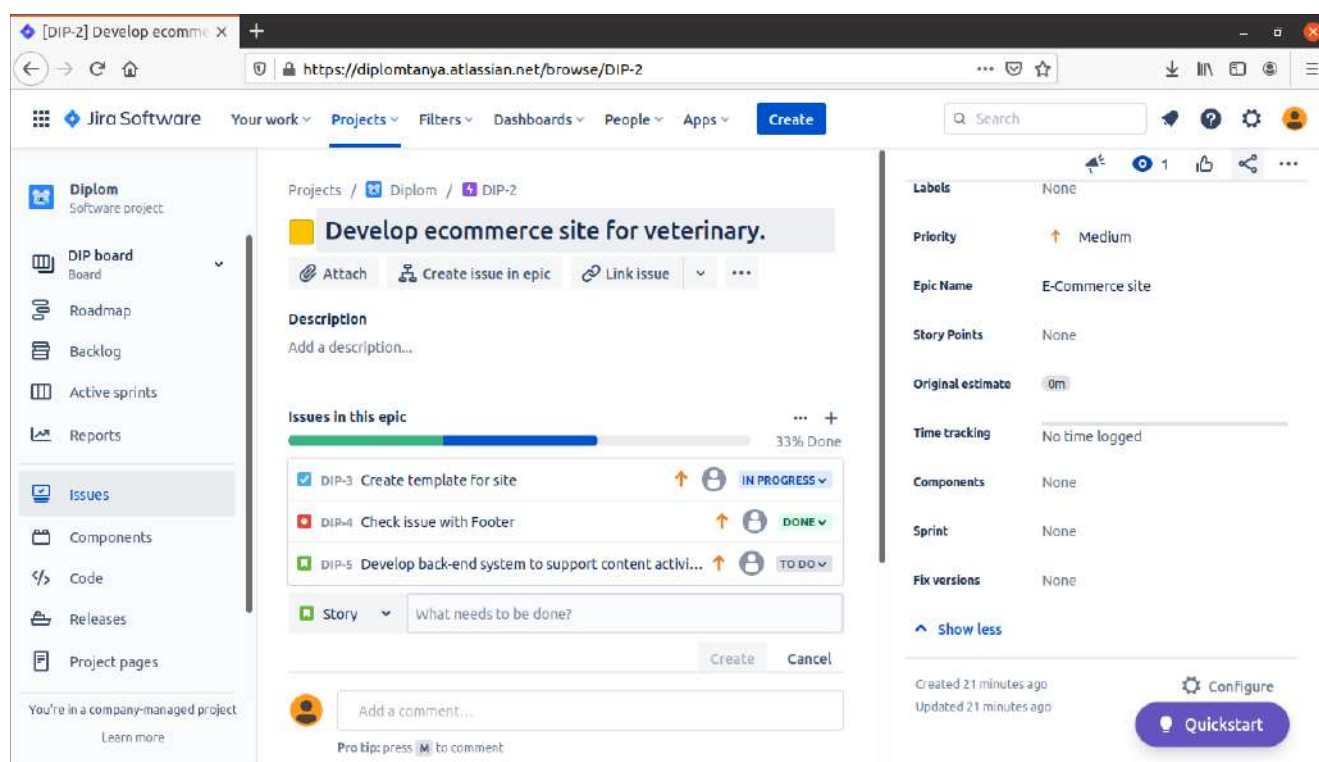


Figure 2.1 Jira Backlog view

Now the time to create a project and deploy it to the github repository to process the next stage for build and test. The main project was created in Atom IDE, where easyling manages projects and source code in it.

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in JavaScript, and embedded

Git Control, developed by GitHub. Atom is a desktop application built using web technologies. Most of the extending packages have free software licenses and are community-built and maintained. Atom is based on Electron (formerly known as Atom Shell), a framework that enables cross-platform desktop applications using Chromium and Node.js. Atom is written in CoffeeScript and Less, but much of it has been converted to JavaScript. Atom was released from beta, as version 1.0, on 25 June 2015. Its developers call it a "hackable text editor for the 21st Century". It is fully customizable in HTML, CSS, and JavaScript. [10]

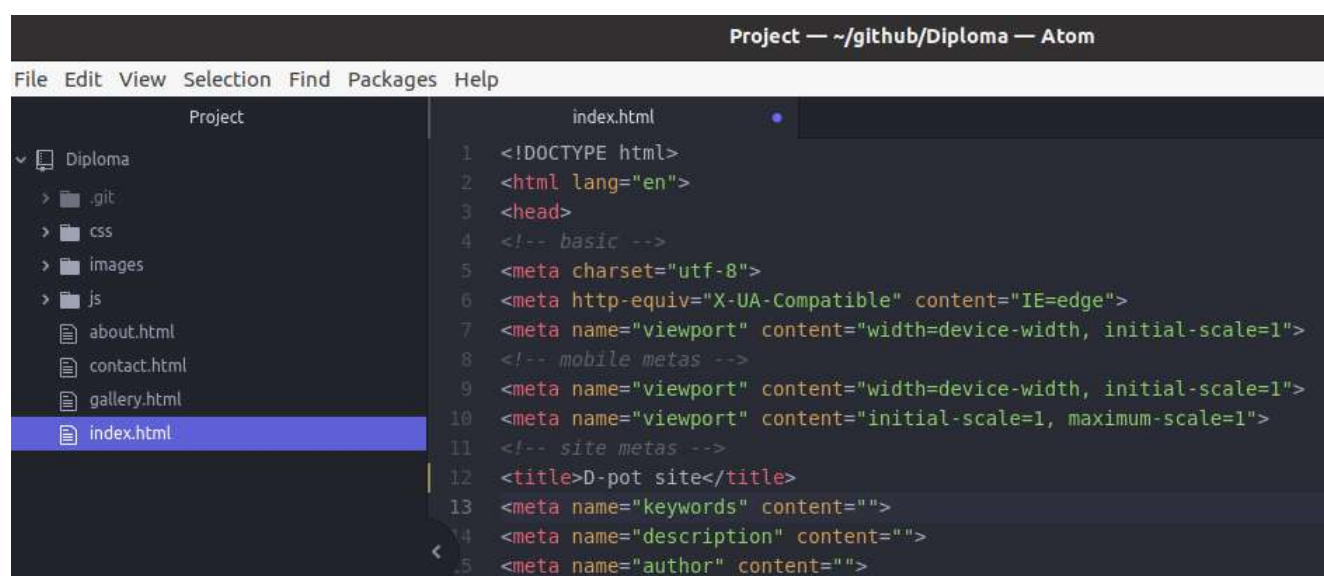


Figure 2.2 Atom project view

As a basic IDE, it can be connected to any source control for this case GitHub is used. GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project. Headquartered in California, it has been a subsidiary of Microsoft since 2018. [11]

In order to connect Atom with GitHub. There are two easiest ways on how to connect Atom and GitHub. First one is to create a repository within Atom itself and push the repository to the GitHub account, which was created beforehand. Second approach is easier and more appropriate, creating a repository in GitHub and clone it to Atom. I used the second method, because it is much easier and more common (Figure 2.2).

First GitHub account needs to be created, it is free. After that a new repository needs to be created. During creation, a new name should be assigned. It can be any name, in my case my repository name is Diplom. In a free account, the repository can be only public, it means that anyone can access and view the repository. After creating the repository's link will be available and short instructions on how to create a new repository on a common line will be available as well. Instruction describes how to push an existing repository from the command line and how to import code from another repository.

In order to connect Atom with GitHub I need to copy my link - <https://github.com/Tetiana1234567/Diploma.git>. Next step is to open Atom and press Ctrl + Shift + P at the same time in the popup window type Git clone command then paste the GitHub repository link. To finish cloning, the Clone button need to be pressed.

After GitHub repository will be cloned and new files will be marked as green and all changed need to be pushed in remote repository.

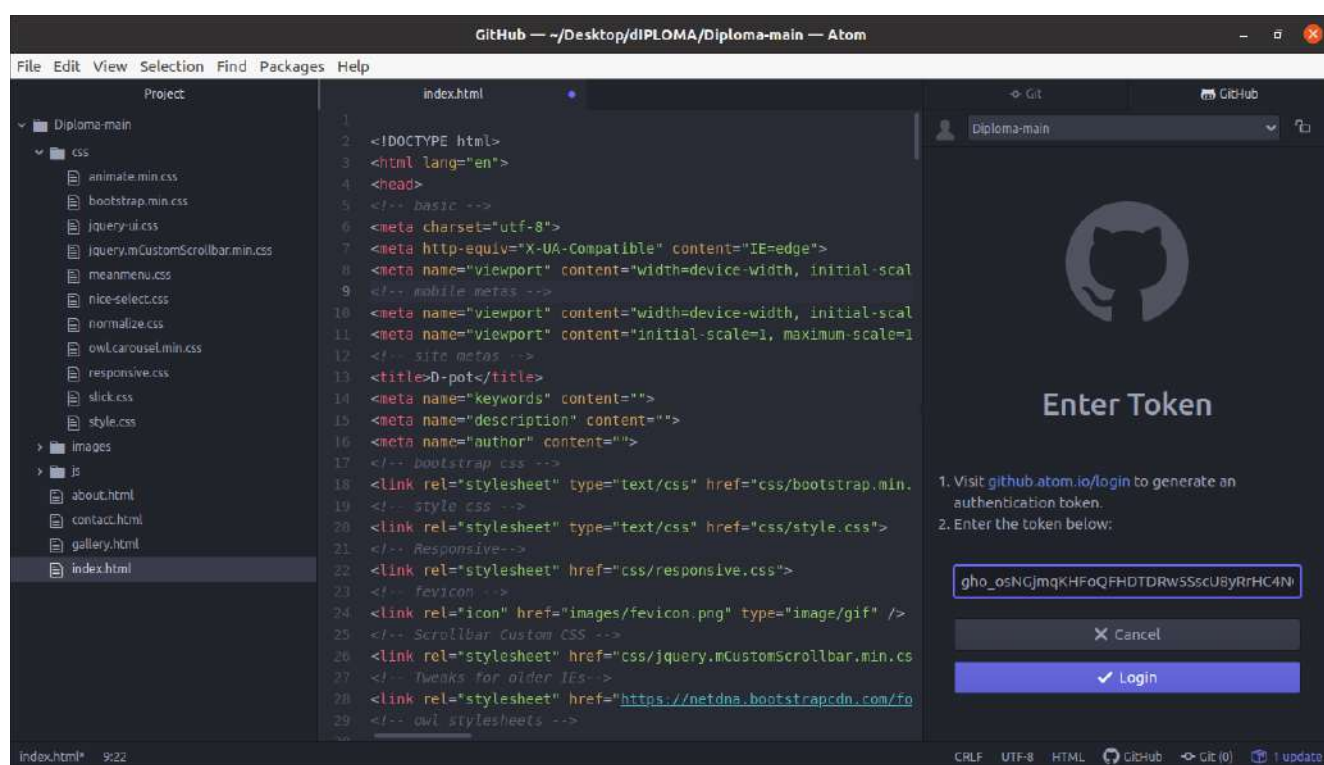


Figure 2.3 GitHub connection

Next step to authorize GitHub to Atom in Figure 3.2, click on <https://github.atom.io/login> to get a new authorization token for Atom after input GitHub credential. Once I have done it and copied a new token and go ahead and open

Atom. On the right corner I put my new token and press the login button After my GitHub account and Atom connect.

After making some changes and saving them locally, the files with changes have moved to the unstaged changes section. If new changes need to be investigated and compared, these changes can be reviewed in an unstaged section. Unstaging files is very beneficial : it can be used to separate files in different commits, or to do work on some other modifications. From command line to get files in unstaged section following command can be used:

```
git reset <commit> -- <path>
```

In this unstaged area new changes are located and are up to commits. Basically to move files on stage area only one button should be pressed - Stage file. Alternative way to right click and select from the popup menu Stage item.

Basically those files are queued and are ready for commnet and commit. In the first commit I added initial comments and clin to create and deached comments.

Next step is branch creation. Git offers easier branch implementation compared to other version control systems. Instead of copying files from directory to directory, Git stores a branch as a commit link. It turns out that the branch is the top of a series of commits, not a container for commits. The history of the branch spreads through hierarchical relationships with other commits.

In Git a branch is simply a lightweight movable pointer to one of commits. If you make changes and commit them, the next commit stores a pointer to a commit that came before. The default branch name is master. When you make commits, you are given a master branch that points to the latest commit made.

In order to create a new branch, I need to click on a master and then click on a new branch and put some name which can be appropriate in this case. After publish button should be pressed, eventually GitHub credentials will be promoted and selected changes will go away from the stage area and initial commit was done.

To check the state of changes we can use the common line or just go directly to GitHub public repository (Figure 2.4 )

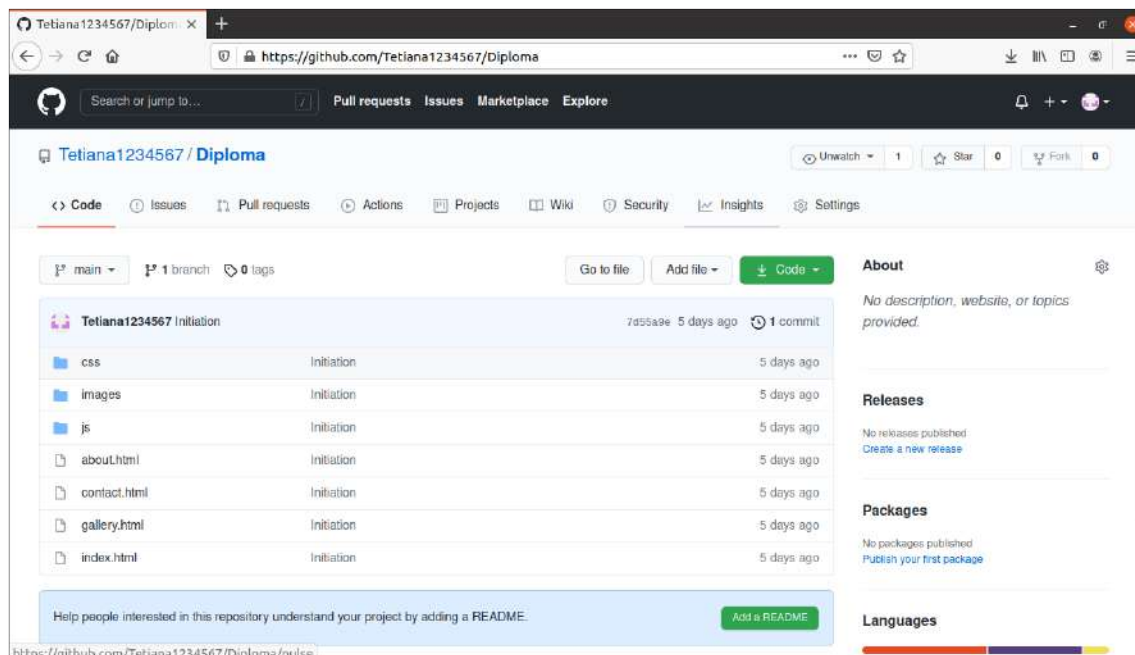


Figure 2.4 GitHub project

The other way to integrate changes from one branch into another is to use rebase command. When I was working with git add not so carefully, I added files I don't want to commit. git rm command moves this file from the staging area and from the file system, which may be not what I want. In this situation, make sure you remove only the staged version and add the file to your .gitignore to avoid making the same mistake again.

Other troubles which can be faced during GitHub usage: Sometimes faulty commits make it into the repository. To avoid these situations, git revert can be used. This command allows to revert single to multiple commits. On the other hand, unfortunately, typos can happen. In the case of commit messages, it is possible to fix them in an easy manner. To fix it, a git -- amend command can be used. git -- amend command allows to add and amend the previous comment. Take into account that -- amend creates a new commit to replace the previous one, so try not to use it to modify commits that were already pushed to the repository. It is possible not to use this rule only if no other developer has already checked out the previous version and based his own work on it, in which case a forced push (git push --force) can be ok. The --force option is necessary here as the tree's history was locally modified. It means that the push is rejected by the remote server since no fast-forward merge is possible.

## 2.2.2 Build Stage



This occurs when a source code and its associated dependencies are compiled to build a runnable instance of the application product. To build this stage Jenkins pipeline can be used.

Before using Jenkins, the application needs to be set up and configured. Terraform is an open-source infrastructure as code software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON. Terraform manages external resources (such as public cloud infrastructure, private cloud infrastructure, network appliances, software as a service, and platform as a service) with "providers". [12]

Deploy Jenkins to AWS instances several steps need to be performed. Provider configurations belong in the root module of a Terraform configuration.

A provider configuration is created using a provider block:

```
provider "aws" {  
  profile  = "default"  
  region   = "us-east-2"  
  access_key = "AKIXXXXWEZ37VXXC4R4R"  
  secret_key = "isaTYvCaLialt+k1I42XXuk37hgXsq97IYXXXIEA" }
```

The name is the local name of the provider to configure. This provider should already be included in a required\_providers block.

The body of the block (between { and }) contains configuration arguments for the provider.

Next part of the configuration is the resource section. Provides an EC2 instance resource. This allows instances to be created, updated, and deleted. Instances also support provisioning.

```
resource "aws_instance" "Ubuntu" {  
  count          = 1  
  ami           = "ami-01e7ca2ef94a0ae86"  
  instance_type = "t2.micro"  
  vpc_security_group_ids = ["sg-05118f212a2ddd293"]  
}
```



```

associate_public_ip_address = "true"
key_name                     = "newkey"
provisioner "remote-exec" {
  inline = [
    "wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key",
    "add -",
    "sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >",
    "/etc/apt/sources.list.d/jenkins.list'",
    "sudo apt update -qq",
    "sudo apt install -y default-jre",
    "sudo apt install -y jenkins",
    "sudo systemctl start jenkins",
    "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT",
    "--to-port 8080",
    "sudo sh -c \"iptables-save > /etc/iptables.rules\"",
    "echo iptables-persistent iptables-persistent/autosave_v4 boolean true | sudo",
    "debconf-set-selections",
    "echo iptables-persistent iptables-persistent/autosave_v6 boolean true | sudo",
    "debconf-set-selections",
    "sudo apt-get -y install iptables-persistent",
    "sudo ufw allow 8080",
  ]
}

```

The provisioner requires access to the remote resource via SSH, and expects a nested connection block with details about how to connect.

```

connection {
  type      = "ssh"
  host      = self.public_ip
  user      = "ubuntu"
  private_key = file("newkey.pem")
}

```

```
}
```

When there are many resources (for example, instances, VCNs, load balancers, and block volumes) across multiple compartments in tenancy, it can become difficult to track resources used for specific purposes, or to aggregate them, report on them, or take bulk actions on them. Tagging allows to define keys and values and associate them with resources. The tags can be used to help organize and list resources based on business needs.

There are two types of tags:

First one is defined tags are set up in tenancy by an administrator. Another one is that free-form tags can be applied by any user with permissions on the resource.

```
tags = {  
  "Name"    = "Jenkins_Server"  
  "Terraform" = "true"  
}
```

After installation is completed, Jenkins is available on the local host address with default port 8080 like <http://jenkins.diplom:8080>. In the first run, the Unlock Jenkins window appears, which will show the location of the initial password. To view the key cat command used to display password: `$$ sudo cat initial Admin Password path`. In the same time the Jenkins console log indicates the location (in the Jenkins home directory) where this password can also be obtained. Pipeline is a set of plugins, which supports implementing and integrating continuous delivery. In the simplest and most common use case, you can now make one job run only if several other, parallel jobs have completed successfully. Fundamentally, a step tells Jenkins what to do at a particular point in time for my case, to execute the shell command and make use of the build number'. When a plugin extends the Pipeline DSL, that typically means the plugin has implemented a new step.

Jenkins suggests installing some plugins in Figure 2.5. There are folders plugins, build tools plugins such as Ant and Gradle. Plugins for work with Git and GitHub and some plugins for pipeline. There are pipeline plugins, pipeline stages view, pipeline GitHub Groove Libraries and plugins supporting SSH.

All appliances have started, and getting started form will appear for creating a power user. In form name, password and other data need to be filled in. After Jenkins URL needs to be provided and after the last button with "Start using Jenkins" ' ' can tap.

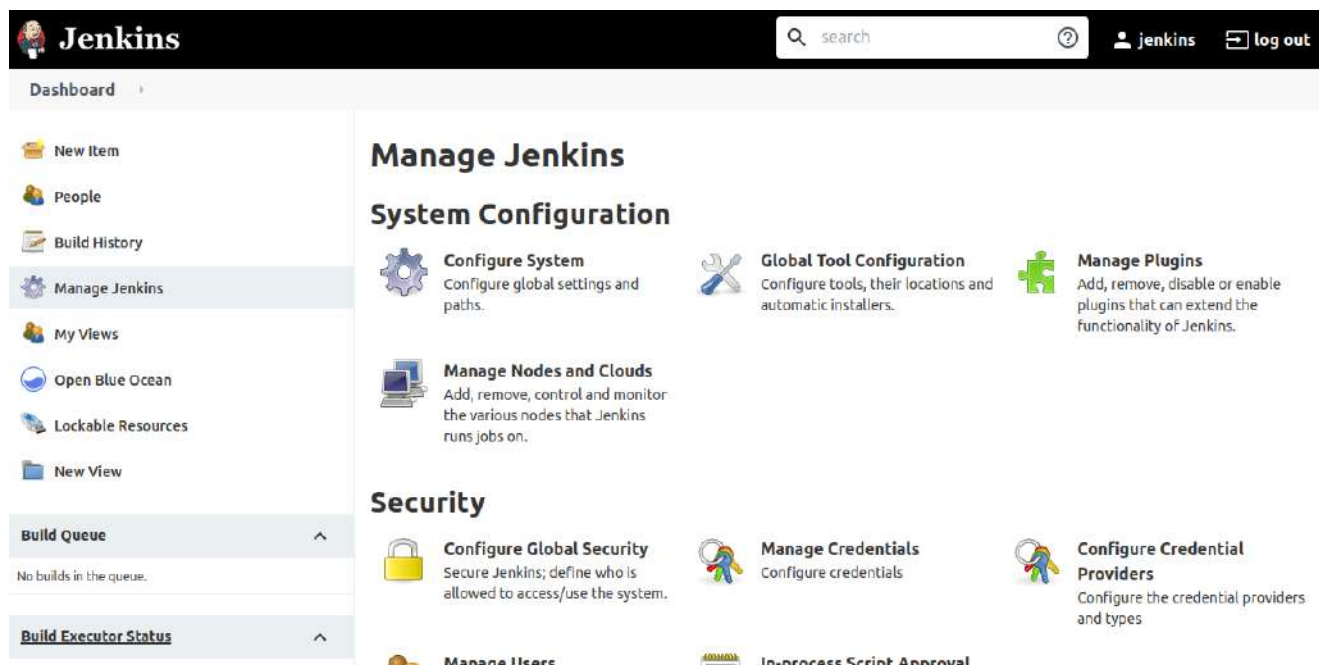


Figure 2.5 Jenkins main page

In order to configure the pipeline it needs to perform several steps. The configuration consists of credential setup and plugin configuration including basic properties.

Plugins are the primary means of enhancing the functionality of a Jenkins environment to suit organization- or user-specific needs. There are over a thousand different plugins that can be installed on a Jenkins master and to integrate various build tools, cloud providers, analysis tools, and much more. Plugins can be automatically downloaded, with their dependencies, from the Update Center. The Update Center is a service operated by the Jenkins project, which provides an inventory of open-source plugins developed and maintained by various members of the Jenkins community. The plugins are packaged as self-contained .hpi files, which have all the necessary code, images, and other resources which the plugin needs to operate successfully. [13]

After installation via web or command line, each plugin needs to be configured separately, depending on what and how this plugin will be used for the pipeline.

Next step is to create freestyle jobs to automate pipelines from scratch. First one Control Version needs to be added as a source of code like <https://github.com/Tetiana1234567/Diploma>. Second step build triggers need to be configured depending on necessity and technical requirements Figure 2.6.

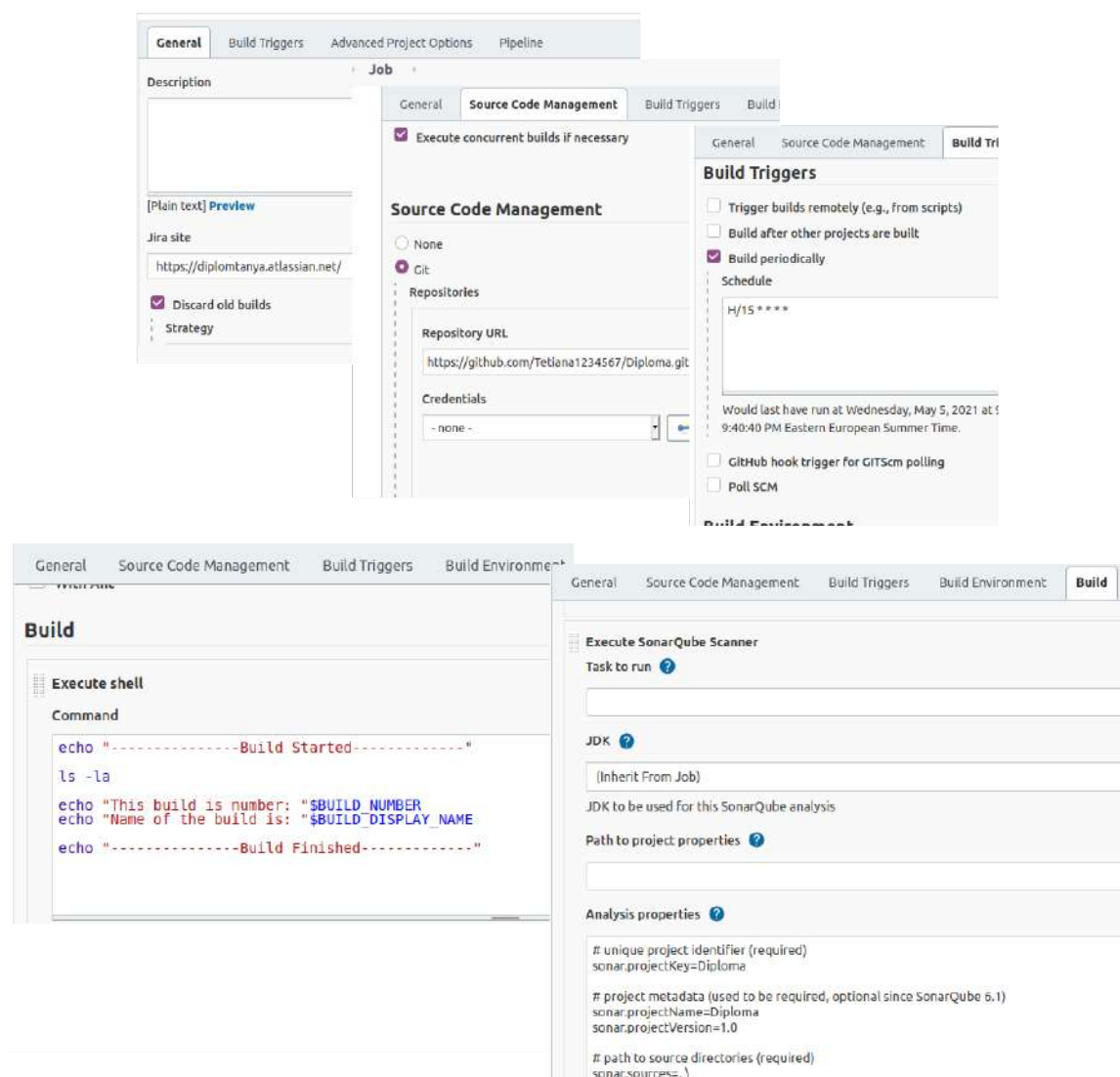


Figure 2.6 Jenkins main page

As a result each jobs have logs:

Jobs logs:

Started by timer

Running as SYSTEM

Building in workspace /var/lib/jenkins/workspace/Job

The recommended git tool is: NONE

No credentials specified

> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Job/.git # timeout=10

Fetching changes from the remote Git repository

```
> git config remote.origin.url https://github.com/Tetiana1234567/Diploma.git #  
timeout=10
```

Fetching upstream changes from https://github.com/Tetiana1234567/Diploma.git

```
> git --version # timeout=10  
> git --version # 'git version 2.25.1'  
> git fetch --tags --force --progress -- https://github.com/Tetiana1234567/Diploma.git  
+refs/heads/*:refs/remotes/origin/* # timeout=10  
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
```

Checking out Revision eea08f02f2b0b3423d2180e5492af97b57a895e0

(refs/remotes/origin/main)

```
> git config core.sparsecheckout # timeout=10  
> git checkout -f eea08f02f2b0b3423d2180e5492af97b57a895e0 # timeout=10
```

Commit message: "Update README.md"

```
> git rev-list --no-walk 62a886567f6f3bea58fd835fc487e90667223cd7 # timeout=10
```

[Job] \$ /bin/sh -xe /tmp/jenkins14857247871819482941.sh

```
+ echo -----Build Started-----
```

```
-----Build Started-----
```

```
+ ls -la
```

```
total 84
```

```
drwxr-xr-x 7 jenkins jenkins 4096 tpa 27 00:25 .
```

```
drwxr-xr-x 4 jenkins jenkins 4096 tpa  5 20:35 ..
```

```
-rw-r--r-- 1 jenkins jenkins 8244 tpa 27 00:25 about.html
```

```
-rw-r--r-- 1 jenkins jenkins 9936 кВі 29 22:29 contact.html
```

```
drwxr-xr-x 2 jenkins jenkins 4096 tpa 27 00:25 css
```

```
-rw-r--r-- 1 jenkins jenkins 11264 кВі 29 22:29 gallery.html
```

```
drwxr-xr-x 8 jenkins jenkins 4096 tpa 27 00:25 .git
```

```
drwxr-xr-x 2 jenkins jenkins 4096 кВі 29 22:29 images
```

```
-rw-r--r-- 1 jenkins jenkins 14575 кВі 29 22:29 index.html
```

```
drwxr-xr-x 2 jenkins jenkins 4096 кВі 29 22:29 js
```

```
-rw-r--r-- 1 jenkins jenkins  440 tpa 27 00:25 README.md
```

```
drwxr-xr-x 3 jenkins jenkins 4096 tpa  7 01:55 .scannerwork
```

Git Build Data

Revision: eea08f02f2b0b3423d2180e5492af97b57a895e0

refs/remotes/origin/main

Changes:

## Summary

[Add Codacy badge \(details\)](#)

[Commit \(details\)](#)

[Commit \(details\)](#)

[Update README.md \(details\)](#)

Commit 4968e9c044789734d34aa288b9c1dd623a4f0d6e by badger

[Add Codacy badge](#)

The file was modified [README.md \(diff\)](#)

Commit a41718ea4dc76352d04f88eadc162da64ab92d11 by skapustasertey

[Commit](#)

The file was modified [about.html \(diff\)](#)

Commit 2c94face3a04f6296dab6d030c0ec9c6508598a2 by tanyasoft

[Commit](#)

The file was modified [css/nice-select.css \(diff\)](#)

The file was modified [about.html \(diff\)](#)

Commit eea08f02f2b0b3423d2180e5492af97b57a895e0 by noreply

[Update README.md](#)

The file was modified [README.md \(diff\)](#)

Build result:

Build #175 (27 May 2021, 00:25:00)

[add description](#)

[Changes](#)

[Add Codacy badge \(details / githubweb\)](#)

[Commit \(details / githubweb\)](#)

[Commit \(details / githubweb\)](#)

[Update README.md \(details / githubweb\)](#)

Started by timer

Revision: eea08f02f2b0b3423d2180e5492af97b57a895e0

refs/remotes/origin/main

Built Branches

refs/remotes/origin/main: Build #175 of Revision  
eea08f02f2b0b3423d2180e5492af97b57a895e0 (refs/remotes/origin/main)

Complex systems and applications face a lot of issues and technical problems while working in production environments. The time to identify the problem may vary from 30 seconds to 30 hours or even more. But there are ways to decrease this time and improve the problem-solving of an IT System team.

### 2.2.3 Test Stage

This is one of the most important aspects of the CI/CD pipeline; using automated tests to validate the code's correctness and viability. For this purpose there are a lot of tools that can be used for. For code review, code quality analysis and security code analysis codacy and coverity can be used.

In order to integrate these tools on-premises and local installation can be proposed to discover. To check code, the codacy.com site can help to analyse the project data. Registration comes first, the basic project property needs to be configured and set up. My Github repository was added to the codacy - <https://github.com/Tetiana1234567/Diploma>. The initial setup consists of several steps including cloning repository, detecting languages and other properties, running code patterns and the last step is finalizing analysis with detailed reposts.

Codacy triggers analysis for every commit of the repository. It allows us to review the evolution and progress of the setup. To apply two ways integration webhooks should be configured with some private key like <https://app.codacy.com/events/github/3391b874f54043429740a0eadbe840db> (Figure 2.7)

## Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓ <https://app.codacy.com/events/g...> (pull\_request, push, and r...)

Edit

Delete

✓ <https://app.codacy.com/events/g...> (all events)

Edit

Delete

Figure 2.7 Web hooks setup

After reviewing the badge can be assigned to the project and can be updated automatically based on review results in Figure 2.8

In email following notification will be sent:

You can view, comment on, or merge this pull request online at:

<https://github.com/Tetiana1234567/Diploma/pull/2>

### Commit Summary

- Add Codacy badge

### File Changes

- M [README.md](#) (1)

### Patch Links:

- <https://github.com/Tetiana1234567/Diploma/pull/2.patch>
- <https://github.com/Tetiana1234567/Diploma/pull/2.diff>

---

You are receiving this because you are subscribed to this thread.

Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).



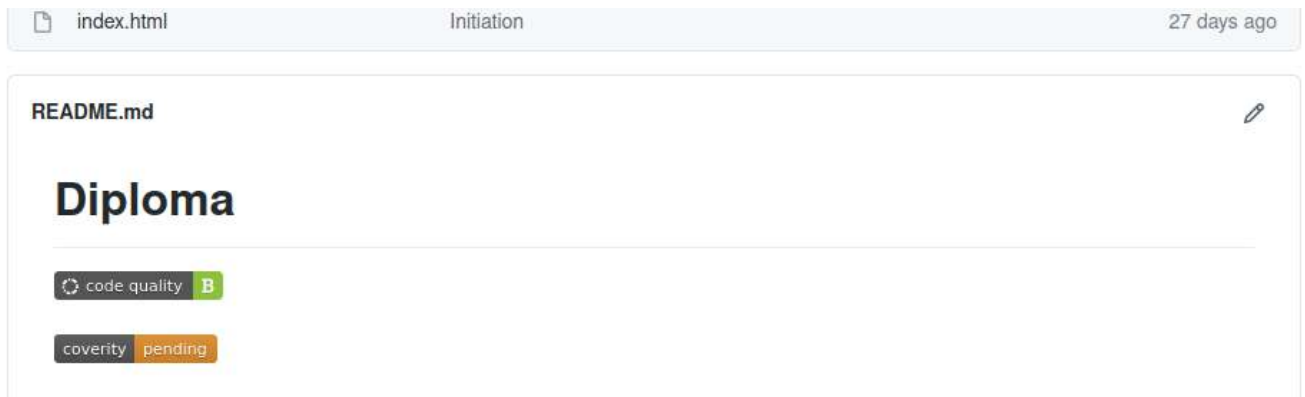


Figure 2.8 Code quality badge

For notification slack (Figure 2.9) can be used and configured. Incoming webhooks can be added to the basic configuration with Webhook key like <https://hooks.slack.com/services/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>

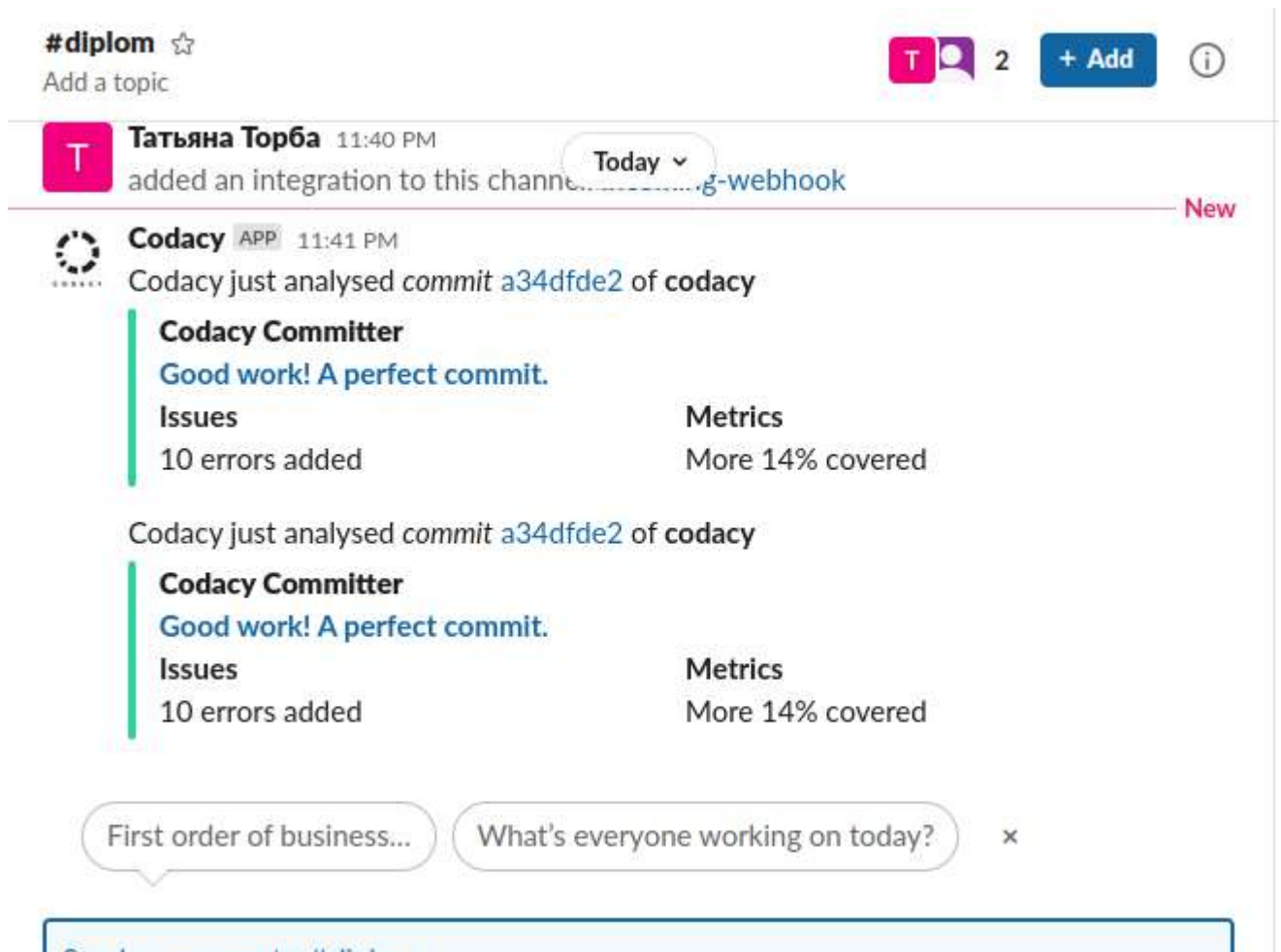


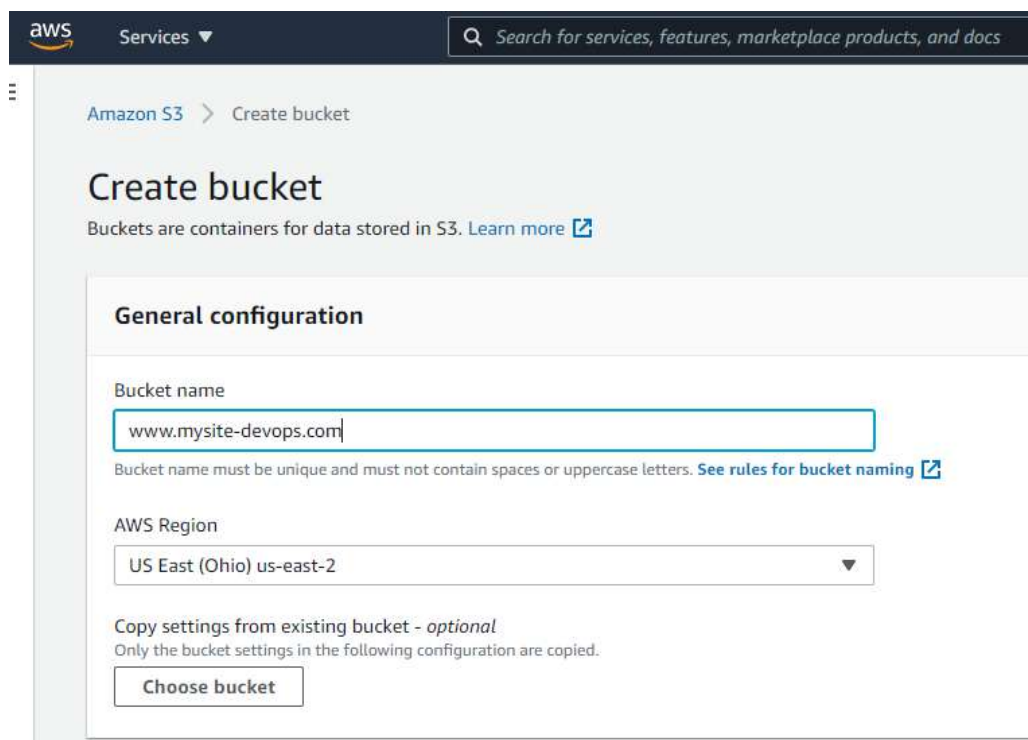
Figure 2.9 Slack notification

## 2.2.4 Deploy Stage

After the built runnable instance has passed all its tests, then the next phase in the pipeline is deployment. Deploy stage is the main phase in CI\CD pipeline, because this last stage delivers the main business value to the business and to the customers. For web hosting I used AWS S3. In order to create the S3 stage, several steps need to be done beforehand:

- Create an S3 bucket in the AWS console (Figure 2.10)
- Configure Jenkins to deploy artifacts to the static local storage
- Upload assets into the bucket and configure it for static hosting.

In order to create static web hosting, logging to AWS console as root user. After in the main menu S3 items need to be chosen. When Amazon S3 successfully creates the chosen bucket, the console displays an empty bucket in the Buckets pane.



The screenshot shows the AWS Management Console interface for creating a new S3 bucket. At the top, the AWS logo and 'Services' menu are visible. Below the search bar, the breadcrumb 'Amazon S3 > Create bucket' is shown. The main heading is 'Create bucket', followed by a subtext: 'Buckets are containers for data stored in S3. [Learn more](#)'. The 'General configuration' section contains a 'Bucket name' text input field with the value 'www.mysite-devops.com'. Below this field is a warning: 'Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)'. The 'AWS Region' is selected as 'US East (Ohio) us-east-2' from a dropdown menu. At the bottom of the configuration section, there is a note: 'Copy settings from existing bucket - optional' and 'Only the bucket settings in the following configuration are copied.' Below this note is a button labeled 'Choose bucket'.

Figure 2.10 AWS S3 configuration

The basic static website hosting starts from the configuration part. Several options need to be configured during the bucket setup, some of them can be changed during exploitation. To enable web hosting in S3 bucket hosting needs to be selected and enabled (Figure 2.11)

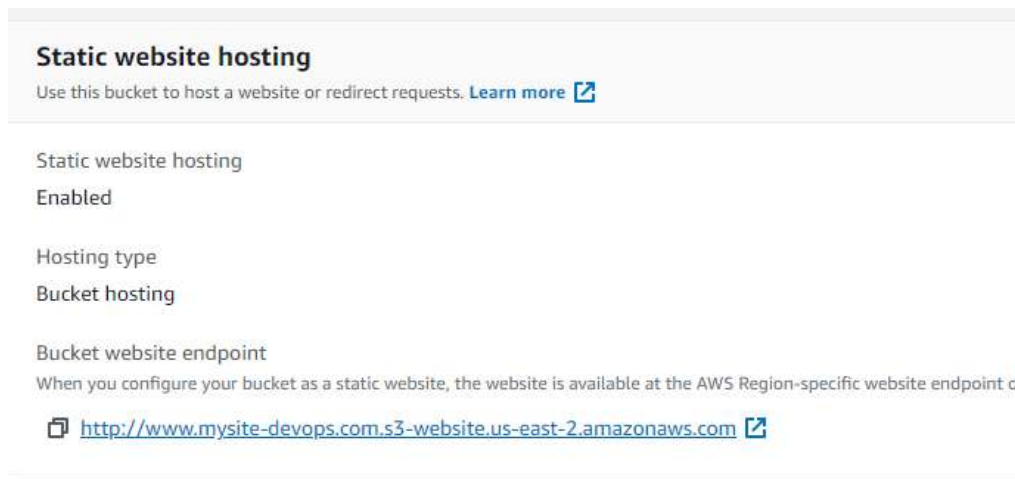


Figure 2.11 Hosting setup in S3

The next step is configuring static main hosting namely to allow public access to the storage. By default, Amazon S3 blocks public access static web hosting and needs to be granted additionally (Figure 2.12).

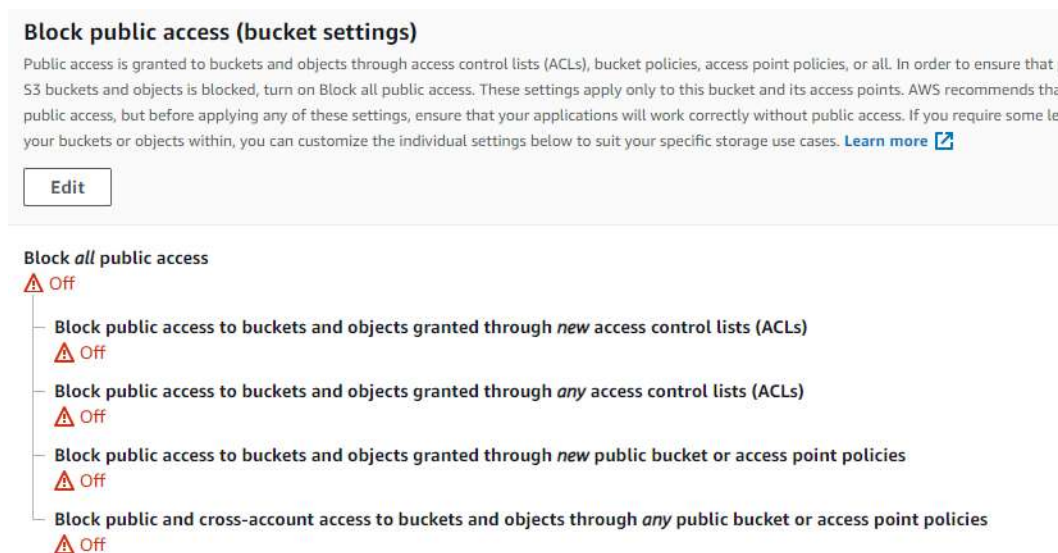


Figure 2.12 Access rights configuration

In order to provide access to the objects in the bucket, following code need to be added to the policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
```

```

        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::www.mysite-devops.com/*"
    }
]
}

```

From the other side, AWS accounts need to be added to the policy. That's why Access Control list (ACL) needs to be managed properly and CRUD operation needs to be managed and controlled from different users' side like Bucket owner, Public access, Authenticated users group from AWS side and S3 log delivery group.

In order to log web traffic from AWS side, separate buckets need to be created and be updated automatically every two hours (Figure ). The logs will record whole requests that are made to the static web hosting.

```

{
  "Version": "2020-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::logs.mysite.com/*"
    }
  ]
}

```

**Edit server access logging**

**Server access logging**  
Log requests for access to your bucket. [Learn more](#)

Server access logging

☐ Disable

☒ Enable

**Warning:** By enabling server access logging, S3 console will automatically update your bucket access control list (ACL) to include access to the S3 log delivery group.

Target bucket

s3://logs.mysite.com [Browse S3](#)

Format: s3://bucket/prefix

When your source bucket and target bucket are the same, additional logs are created for the logs that are

Figure 2.13 AWS Server access logging

After creating and setting all necessary settings from AWS side, deploy tool should be configured accordingly with additional plugin Figure 2.14



Figure 2.14 Jenkins S3 plugin

The log of execution can be following:

INFO: -----

Publish artifacts to S3 Bucket Build is still running

Publish artifacts to S3 Bucket Using S3 profile: S3Deploy

Publish artifacts to S3 Bucket bucket=www.mysite-devops.com, file=.sonar\_lock region=us-east-2, will be uploaded from slave=false managed=false , server encryption false

.....

Finished: SUCCESS

As a result, after publishing content into bucket, web site will be available via <http://www.mysite-devops.com.s3-website-us-east-2.amazonaws.com/> link and logs will be available in separate hosting in Figure 2.15.

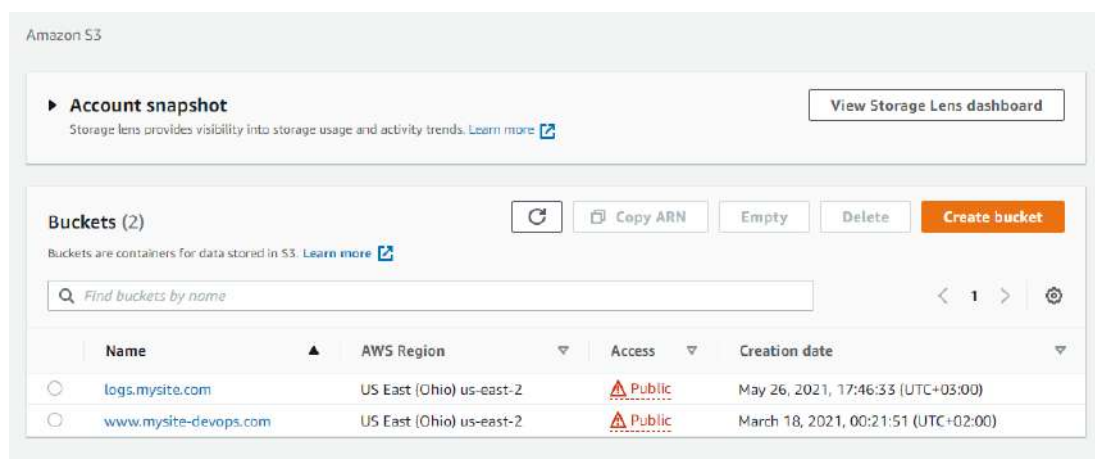


Figure 2.15 AWS S3

The example of code:

```
<Error>
<Code>AccessDenied</Code>
<Message>Access Denied</Message>
<RequestId>187MRE14TBFEBPNX</RequestId>
<HostId>
tNYlsF3mHvFvv+Afv2zqY8os1r1zXy/SyGDXq1aNMG8WFd9X3L8aglGKzLn1KCcc
4HL3Ty3R/+s=
</HostId>
</Error>
```

Click on the link below to access the main page of the site (Figure 2.16).

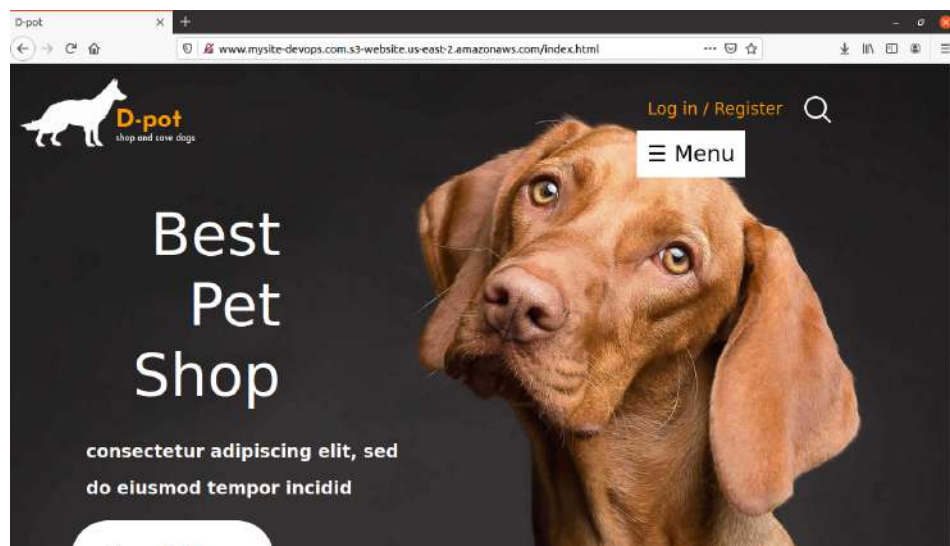


Figure 2.16 Web application hosted on S3 bucket

By further expanding the capabilities of the web site, this site can make requests to other systems to obtain additional data, as well as implement additional actions necessary for the operation of the system. For example geo location or additional SEO activities like advertisement or customer preferences.

### 2.2.5 Monitoring

Monitoring will be supported by Prometheus and Grafana. In order to integrate tools below additional add ons should be installed in Jenkins.

For monitoring I used docker containers to get servers up and running. After services run, export data from Jenkins passing it to Prometheus and then to Grafana.

As I mentioned, Prometheus docker is up and run after basic docker configuration. Default port for Prometheus is 9090. Installation was done by command

```
docker run -d --name diplom_prometheus -p 9090:9090 prom/diplom_prometheus.
```

An additional plugin was added to the pipeline. Plugin exposes an endpoint with metrics, where Prometheus Server can scrape logs. The output of plugin looks like:

```
jvm_info{version="11.0.11+9-Ubuntu-0ubuntu2.20.04",vendor="Ubuntu",runtime="OpenJDK Runtime Environment"}, 1.0
```

```
# HELP default_jenkins_builds_duration_milliseconds_summary Summary of Jenkins build times in milliseconds by Job
```

```
# TYPE default_jenkins_builds_duration_milliseconds_summary summary
default_jenkins_builds_duration_milliseconds_summary_count{jenkins_job="Diplom",repo="NA"},
```

```
1.0default_jenkins_builds_duration_milliseconds_summary_sum{jenkins_job="Diplom",repo="NA"},
```

```
5432.0default_jenkins_builds_duration_milliseconds_summary_count{jenkins_job="Job",repo="NA"},
```

```
2.0default_jenkins_builds_duration_milliseconds_summary_sum{jenkins_job="Job",repo="NA"}, 674973.0
```

```
# HELP default_jenkins_builds_success_build_count Successful build count
```

```
# TYPE default_jenkins_builds_success_build_count counter
default_jenkins_builds_success_build_count{jenkins_job="Diplom",repo="NA"}, 1.0default_jenkins_builds_success_build_count{jenkins_job="Job",repo="NA"}, 2.0
```

```
# HELP default_jenkins_builds_health_score Health score of a job
```

```
# TYPE default_jenkins_builds_health_score gauge
default_jenkins_builds_health_score{jenkins_job="Diplom",repo="NA"},
```

```
100.0default_jenkins_builds_health_score{jenkins_job="Job",repo="NA"}, 100.0
```

```
# HELP default_jenkins_builds_last_build_result ordinal Build status of a job.
```

```

#          TYPE          default_jenkins_builds_last_build_result_ordinal
gaugedefault_jenkins_builds_last_build_result_ordinal{jenkins_job="Diplom",repo="N
A",}
0.0default_jenkins_builds_last_build_result_ordinal{jenkins_job="Job",repo="NA",}
0.0

# HELP default_jenkins_builds_last_build_result Build status of a job as a
boolean (0 or 1)

#          TYPE          default_jenkins_builds_last_build_result
gaugedefault_jenkins_builds_last_build_result{jenkins_job="Diplom",repo="NA",}
1.0default_jenkins_builds_last_build_result{jenkins_job="Job",repo="NA",} 1.0

# HELP default_jenkins_builds_last_build_duration_milliseconds Build times in
milliseconds of last build

#          TYPE          default_jenkins_builds_last_build_duration_milliseconds
gaugedefault_jenkins_builds_last_build_duration_milliseconds{jenkins_job="Diplom",
repo="NA",}
5432.0default_jenkins_builds_last_build_duration_milliseconds{jenkins_job="Job",rep
o="NA",} 338198.0

# HELP default_jenkins_builds_last_build_start_time_milliseconds Last build
start timestamp in milliseconds

#          TYPE          default_jenkins_builds_last_build_start_time_milliseconds
gaugedefault_jenkins_builds_last_build_start_time_milliseconds{jenkins_job="Diplom
",repo="NA",}
1.620147476553E12default_jenkins_builds_last_build_start_time_milliseconds{jenkins
_job="Job",repo="NA",} 1.622409964268E12

# HELP default_jenkins_builds_last_stage_duration_milliseconds_summary
Summary of Jenkins build times by Job and Stage in the last build

# TYPE default_jenkins_builds_last_stage_duration_milliseconds_summary
summarydefault_jenkins_builds_last_stage_duration_milliseconds_summary_count{jen
kins_job="Diplom",repo="NA",stage="Hello",}
1.0default_jenkins_builds_last_stage_duration_milliseconds_summary_sum{jenkins_jo
b="Diplom",repo="NA",stage="Hello",} 219.0

```



To configure monitoring server, some yml file needs to be updated with next code:

```
- job_name: 'jenkins'
  metrics_path: /prometheus
  static_configs:
    - targets: ['http:\\jenkins\\prometheus:9090']
```

As a result a comprehensive dashboard can be created and will be available for everyone.

## CONCLUSION

In the first section industry research and impact analysis was discovered. The advantages and disadvantages of different approaches are highlighted, which makes it possible to take into account the results of the review in further development in order to develop and create an improved architecture taking into account these results. The second part of the first section is devoted to the analysis of approaches to the development of the architecture, investigating software products that can be used as components in the architecture, in particular their capabilities and methods of integration.

The second section describes the practical implementation of CI\CD pipeline, which is based on the developed architecture. Jenkins Server was used to support basic postulates, testing tools to install quality gates and AWS cloud to ensure product availability to a wide range of users. At the same time all these components integrated in one pipeline and implemented the ability to work with pipelines based on triggers and schedules. Behind monitoring system setup and configured in order to monitor and improve the problem solving time for business.

The resulting architecture can be further expanded with new components as needed. In particular, pipeline stage notification components may be added. In addition, significant potential for expansion exists in the testing phase, as this stage may include different types of testing and systems that support them. In particular, support for integration tests can be added. Depending on chosen programming languages different tools and approaches can be used. But the main focus should always be on business value and speed of market.

In general, the created CI\CD pipeline confirms the relevance and effectiveness of this architecture, and can serve as a prototype for further expansion and used for web application development. To expand the possible approaches external systems and addons can be used.

Information technology is one of the most dynamic and directions are constantly appearing, so opportunities for change and development exist for any software solution. Because of this, improving and adding new features to the developed architecture may

be part of future work, and may focus on implementing additional stages and phases to the pipeline.

For example Docker and Kubernetes can be one of the areas of improvement. That approach expands the CI\CD pipeline and brings new vision in the continuous integration sphere. On the other hand, investigation and enhancement can consider the expansion of communication channels. For example viber, telegram or any other corporate messengers can be added to the pipeline to send notification in an appropriate way to appropriate people.

## REFERENCES

1. 10 Essential Ecommerce Website Requirements and Best-in-class Features, 18 June 2020,  
<https://www.purchasecommerce.com/blog/10-essential-ecommerce-requirement-best-in-class-feature>, Accessed May 24 2021.
2. Jenna Erickson, “Adaptive Vs. Responsive Design” , 2021  
<https://usabilitygeek.com/adaptive-vs-responsive-design/> Accessed May 24 2021.
3. Wikipedia, [https://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete), 2021 Accessed May 24 2021.
4. Manish Mathuria, “The Road to CI/CD – A Short History of Agile Development”, 2016,  
<https://www.infostretch.com/blog/the-road-to-cicd-a-short-history-of-agile-development/> Accessed May 24 2021.
5. “Wikipedia”, [https://en.wikipedia.org/wiki/Continuous\\_delivery](https://en.wikipedia.org/wiki/Continuous_delivery) , 2021, Accessed May 24 2021.
6. Sten Pitten, “What is Continuous Deployment?”, 2020,  
[https://www.atlassian.com/continuous-delivery/continuous-deployment#:~:text=Continuous%20Deployment%20\(CD\)%20is%20a,cycle%20has%20evolved%20over%20time](https://www.atlassian.com/continuous-delivery/continuous-deployment#:~:text=Continuous%20Deployment%20(CD)%20is%20a,cycle%20has%20evolved%20over%20time), Accessed May 24 2021.
7. “Top IDE index”, <https://pypl.github.io/IDE.html>, April, 2020, Accessed May 24 2021.
8. “Wikipedia”, [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing), 2021, Accessed May 24 2021.
9. “OWASP Top Ten”, <https://owasp.org/www-project-top-ten/>, April, 2021, Accessed May 24 2021.
10. “Wikipedia”, [https://en.wikipedia.org/wiki/Atom\\_\(text\\_editor\)](https://en.wikipedia.org/wiki/Atom_(text_editor)), 2020, Accessed May 24 2021.
11. “Wikipedia”, <https://en.wikipedia.org/wiki/GitHub> , 2021, Accessed May 24 2021.
12. “Wikipedia”, [https://en.wikipedia.org/wiki/Terraform\\_\(software\)](https://en.wikipedia.org/wiki/Terraform_(software)), 2021, Accessed May 24 2021.

13. “Managing Plugins”, 2021, <https://www.jenkins.io/doc/book/managing/plugins/>, Accessed May 24 2021.
14. “Monitoring, Evaluation & Learning”, 2021, <https://www.ri.org/content/uploads/2019/08/relief-international-monitoring-evaluation-learning.pdf>, Accessed May 24 2021.
15. “What is Prometheus?”, 2021, <https://prometheus.io/docs/introduction/overview/>, Accessed May 24 2021.
16. “5 Tips to Setup a Better Performance Testing Environment”, 2017, <https://www.testbytes.net/blog/performance-testing-environment/>, Accessed May 24 2021.
17. “Wikipedia”, [https://en.wikipedia.org/wiki/Docker\\_\(software\)](https://en.wikipedia.org/wiki/Docker_(software)), 2021, Accessed May 24 2021.
18. “What is Kubernetes?” April 30, 2021, <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>, Accessed May 24 2021.
19. “Wikipedia”, <https://en.wikipedia.org/wiki/OpenShift>, 2020, Accessed May 24 2021.

## **ABBREVIATIONS AND TERMS**

CI - continuous integration

CD - continuous Delivery

CRUD - create read update delete

## APPENDIX 1

Started by user jenkins

Running as SYSTEM

Building in workspace /var/lib/jenkins/workspace/Job

The recommended git tool is: NONE

No credentials specified

```
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Job/.git # timeout=10
```

Fetching changes from the remote Git repository

```
> git config remote.origin.url https://github.com/Tetiana1234567/Diploma.git #
timeout=10
```

Fetching upstream changes from https://github.com/Tetiana1234567/Diploma.git

```
> git --version # timeout=10
```

```
> git --version # 'git version 2.25.1'
```

```
> git fetch --tags --force --progress -- https://github.com/Tetiana1234567/Diploma.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

```
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
```

Checking out Revision 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3  
(refs/remotes/origin/main)

```
> git config core.sparsecheckout # timeout=10
```

```
> git checkout -f 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3 # timeout=10
```

Commit message: "Initiation"

```
> git rev-list --no-walk 7d55a9e7f1e5c4024a2f5e590f5969462e9262c3 # timeout=10
```

[Job] \$ /bin/sh -xe /tmp/jenkins15679617849493207118.sh

```
+ echo -----Build Started-----
```

```
-----Build Started-----
```

```
+ ls -la
```

total 80

```
drwxr-xr-x 7 jenkins jenkins 4096 тpa 3 20:50 .
```

```
drwxr-xr-x 4 jenkins jenkins 4096 тpa 5 20:35 ..
```

```
-rw-r--r-- 1 jenkins jenkins 8292 кBi 29 22:29 about.html
```

```
-rw-r--r-- 1 jenkins jenkins 9936 кBi 29 22:29 contact.html
```

```
drwxr-xr-x 2 jenkins jenkins 4096 кBi 29 22:29 css
```

```
-rw-r--r-- 1 jenkins jenkins 11264 кBi 29 22:29 gallery.html
```

```
drwxr-xr-x 8 jenkins jenkins 4096 тpa 5 21:38 .git
```

```
drwxr-xr-x 2 jenkins jenkins 4096 кBi 29 22:29 images
```

```
-rw-r--r-- 1 jenkins jenkins 14575 кBi 29 22:29 index.html
```

```
drwxr-xr-x 2 jenkins jenkins 4096 кBi 29 22:29 js
```

```
drwxr-xr-x 3 jenkins jenkins 4096 тpa 5 21:25 .scannerwork
```

```
+ echo This build is number: 86
```

This build is number: 86

```
+ echo Name of the build is: #86
```

Name of the build is: #86

```
+ echo -----Build Finished-----
```

```
-----Build Finished-----
```

```

[Job]
/var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/bin/so
nar-scanner -Dsonar.host.url=http://localhost:9000 *****
-Dsonar.projectKey=Diploma -Dsonar.projectName=Diploma
-Dsonar.projectVersion=1.0 -Dsonar.sources=.
-Dsonar.projectBaseDir=/var/lib/jenkins/workspace/Job
INFO: Scanner configuration file:
/var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/conf/s
onar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarScanner 4.6.1.2450
INFO: Java 11.0.11 Ubuntu (64-bit)
INFO: Linux 5.8.0-50-generic amd64
INFO: User cache: /var/lib/jenkins/.sonar/cache
INFO: Scanner configuration file:
/var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/Sonarscanner/conf/s
onar-scanner.properties
INFO: Project root configuration file: NONE
INFO: Analyzing on SonarQube server 8.8.0
INFO: Default locale: "en_US", source code encoding: "UTF-8" (analysis is platform
dependent)
INFO: Load global settings
INFO: Load global settings (done) | time=72ms
INFO: Server id: BF41A1F2-AXkyEzeA7vbvpIlfIjj7
INFO: User cache: /var/lib/jenkins/.sonar/cache
INFO: Load/download plugins
INFO: Load plugins index
INFO: Load plugins index (done) | time=42ms
INFO: Load/download plugins (done) | time=127ms
INFO: Process project properties
INFO: Process project properties (done) | time=8ms
INFO: Execute project builders
INFO: Execute project builders (done) | time=1ms
INFO: Project key: Diploma
INFO: Base dir: /var/lib/jenkins/workspace/Job
INFO: Working dir: /var/lib/jenkins/workspace/Job/.scannerwork
INFO: Load project settings for component key: 'Diploma'
INFO: Load project settings for component key: 'Diploma' (done) | time=14ms
INFO: Auto-configuring with CI 'Jenkins'
INFO: Load quality profiles
INFO: Load quality profiles (done) | time=48ms
INFO: Auto-configuring with CI 'Jenkins'
INFO: Load active rules
INFO: Load active rules (done) | time=1390ms
INFO: Indexing files...
INFO: Project configuration:

```



INFO: Load project repositories  
INFO: Load project repositories (done) | time=14ms  
WARN: Invalid character encountered in file /var/lib/jenkins/workspace/Job/js/plugin.js at line 128 for encoding UTF-8. Please fix file content or configure the encoding to be used using property 'sonar.sourceEncoding'.  
INFO: 37 files indexed  
INFO: 0 files ignored because of scm ignore settings  
INFO: Quality profile for css: Sonar way  
INFO: Quality profile for js: Sonar way  
INFO: Quality profile for web: Sonar way  
INFO: ----- Run sensors on module Diploma  
INFO: Load metrics repository  
INFO: Load metrics repository (done) | time=31ms  
INFO: Sensor CSS Metrics [cssfamily]  
INFO: Sensor CSS Metrics [cssfamily] (done) | time=358ms  
INFO: Sensor CSS Rules [cssfamily]  
INFO: Sensor CSS Rules [cssfamily] (done) | time=661ms  
INFO: Sensor JaCoCo XML Report Importer [jacoco]  
INFO: 'sonar.coverage.jacoco.xmlReportPaths' is not defined. Using default locations: target/site/jacoco/jacoco.xml,target/site/jacoco-it/jacoco.xml,build/reports/jacoco/test/jacocoTestReport.xml  
INFO: No report imported, no coverage information will be imported by JaCoCo XML Report Importer  
INFO: Sensor JaCoCo XML Report Importer [jacoco] (done) | time=4ms  
INFO: Sensor JavaScript analysis [javascript]  
INFO: Sensor JavaScript analysis [javascript] (done) | time=1192ms  
INFO: Sensor C# Project Type Information [csharp]  
INFO: Sensor C# Project Type Information [csharp] (done) | time=2ms  
INFO: Sensor C# Properties [csharp]  
INFO: Sensor C# Properties [csharp] (done) | time=1ms  
INFO: Sensor JavaXmlSensor [java]  
INFO: Sensor JavaXmlSensor [java] (done) | time=3ms  
INFO: Sensor HTML [web]  
INFO: Sensor HTML [web] (done) | time=472ms  
INFO: Sensor VB.NET Project Type Information [vbnet]  
INFO: Sensor VB.NET Project Type Information [vbnet] (done) | time=1ms  
INFO: Sensor VB.NET Properties [vbnet]  
INFO: Sensor VB.NET Properties [vbnet] (done) | time=1ms  
INFO: ----- Run sensors on project  
INFO: Sensor Zero Coverage Sensor  
INFO: Sensor Zero Coverage Sensor (done) | time=1ms  
INFO: CPD Executor Calculating CPD for 4 files  
INFO: CPD Executor CPD calculation finished (done) | time=41ms  
INFO: Analysis report generated in 94ms, dir size=386 KB  
INFO: Analysis report compressed in 53ms, zip size=101 KB  
INFO: Analysis report uploaded in 72ms

INFO: ANALYSIS SUCCESSFUL, you can browse  
<http://localhost:9000/dashboard?id=Diploma>  
INFO: Note that you will be able to access the updated dashboard once the server has  
processed the submitted analysis report  
INFO: More about the report processing at  
<http://localhost:9000/api/ce/task?id=AXk90rc47vkvplIfioms>  
INFO: Analysis total time: 7.042 s  
INFO: -----  
INFO: EXECUTION SUCCESS  
INFO: -----  
INFO: Total time: 8.272s  
INFO: Final Memory: 7M/37M  
INFO: -----  
Publish artifacts to S3 Bucket Build is still running  
FINISHED: SUCCESS

## APPENDIX 2

```
provider "aws" {
  profile  = "default"
  region   = "us-east-2"
  access_key = "AKIAYQUWEZ37V2VC4R4R"
  secret_key = "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"
}

resource "aws_instance" "Ubuntu" {
  count          = 1
  ami            = "ami-01e7ca2ef94a0ae86"
  instance_type  = "t2.micro"
  vpc_security_group_ids = ["sg-05118f212a2ddd293"]
  associate_public_ip_address = "true"
  key_name       = "newkey"

  provisioner "remote-exec" {
    inline = [
      "wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add",
      "sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'",
      "sudo apt update -qq",
      "sudo apt install -y default-jre",
      "sudo apt install -y jenkins",
      "sudo systemctl start jenkins",
      "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080",
      "sudo sh -c \"iptables-save > /etc/iptables.rules\"",
      "echo iptables-persistent iptables-persistent/autosave_v4 boolean true | sudo debconf-set-selections",
      "echo iptables-persistent iptables-persistent/autosave_v6 boolean true | sudo debconf-set-selections",
      "sudo apt-get -y install iptables-persistent",
      "sudo ufw allow 8080",
    ]
  }
}
```

```

connection {
  type      = "ssh"
  host      = self.public_ip
  user      = "ubuntu"
  private_key = file("newkey.pem")
}

tags = {
  "Name"      = "Jenkins_Server"
  "Terraform" = "true"
}
}

resource "aws_instance" "Jira" {
  count          = 1
  ami            = "ami-01e7ca2ef94a0ae86"
  instance_type  = "t2.micro"
  vpc_security_group_ids = ["sg-05118f212a2ddd293"]
  associate_public_ip_address = "true"
  key_name       = "newkey"
}

```

