Issues in Information Systems
Volume 23, Issue 4, pp. 142-156, 2022

DOI: https://doi.org/10.48009/4_iis_2022_113

Using DevOps paradigm to deploy web applications

Abhijit Sen, *Kwantlen Polytechnic University, abhijit.sen@kpu.ca*
Sandro Falter, *OTH Regensburg*, *sarofalter@t-online.de*
Nicolas Mayer, *OTH Regensburg, nicolass.mayer@gmail.com*

Abstract

DevOps paradigm is widely used in industry to develop software faster, deploy high quality frequent releases of features by integrating and harmonizing the Development and IT Operations activities. Industries are taking strategic decisions to remove the barriers that existed between Development and Operational teams by encouraging collaborations among these teams throughout System Development Life Cycle (SDLC). These strategic decisions to implement DevOps paradigm resulted in the development and emergence of large arrays of tool chains to support, monitor, and automate activities of various SDLC stages. In this paper authors attempt to give practical insights on how the using of DevOps can speed up the management, development and deployment process of a simple web application. Widely used DevOps model consisting of eight stages is used to implement the example application. A toolchain consisting of state of arts tools is used at various DevOps stages. A detailed explanation of each tool, including details to their implementation and a short evaluation concludes the study. The results revealed that the usage of DevOps enables to accelerate the development process of web applications, as most steps during the build and testing process can be automated. Especially the outsourcing of operational overhead to an external cloud provider can lead to economic advantages, which will impact the future of software development.

Keywords: Continuous delivery, Continuous Deployment, Continuous integration, Continuous Monitoring, DevOps, SDLC, KPIs

Introduction

DevOps consists of a set of practices for software development that enable to reduce time between changes being committed to the software production environment thus speeding up the development process (Zhu, Bass, & Champlin-Scharff, 2016). The rise of this new Development approach went along with a rise in the development of web applications. Many software providers use web applications to implement their services, as they run inside a web browser which does not require any additional installation. Some of the most successful platforms are based on web applications and smartphone apps, like Facebook, Instagram, and Uber. Therefore, this study wants to give a practical insight in the DevOps process of a web application. After introducing the use case of the web application and some basic terminology the paper will proceed with the description of the underlying DevOps Model. The implementation of this DevOps pipeline using different tools will be described. The main focus of this project is to select a set of tools that will support and automate all activities of DevOps Life cycle.

Another motivation is to provide Computer Science students working in this project necessary exposure and skill sets to practice DevOps tool sets, popularly used in the industry, to manage and develop the software project using the principles of DevOps paradigms. The emphasis is not on complexity of

application, but on students getting practical experience in working with tools through all stages of DevOps life cycle. The main research question is whether students, with limited exposure to DevOps paradigm, would be able to select, learn, and apply appropriate tools for eight stages of DevOps to develop, manage, test, and monitor a web application within a short span of one University semester.

Literature Survey

The main objective of DevOps is to integrate software development(Dev) and information technology operations (Ops) through a pipeline enabling continuous integration and continuous deployment to reduce siloed group interaction within an an organization (Ebert, Gallardo, Hernantes, & Serrano, 2016) with the objectives of delivering high quality software product with rapid frequencies. Davies, Daniels (2016) describe the details of DevOps principles, their implementation and approaches to choose various tools to facilitate DevOps workflows. Many different models of DevOps life cycles can be chosen in practice. DevOps cycles define the stages that are necessary to implement a toolchain. Many of these models are very similar and only differ slightly. Sharma & Coyne (2015) discusses four stages in DevOps life cycle - Plan, Develop & Test, Deploy, and Operate, where Dev consists of {Plan, Develop & Test, and Deploy} segments, and Ops consist of {Operate} segments. The implementation of four stage was discussed by Sen, Baumgartner, Heiß, Wagner (2021). As shown in Figure 1, another DevOps model suggests eight stages - Plan, Code, Build and Test for the Development part (Dev) and Release, Deploy, Operate and Monitor to represent the operational part (Ops) of the DevOps cycle. For this project we have utilized the model that consists of eight stages as shown in Figure 1. Respective tools must be selected to support each stage of the lifecycle, which will be explained in detail in the subsequent section.

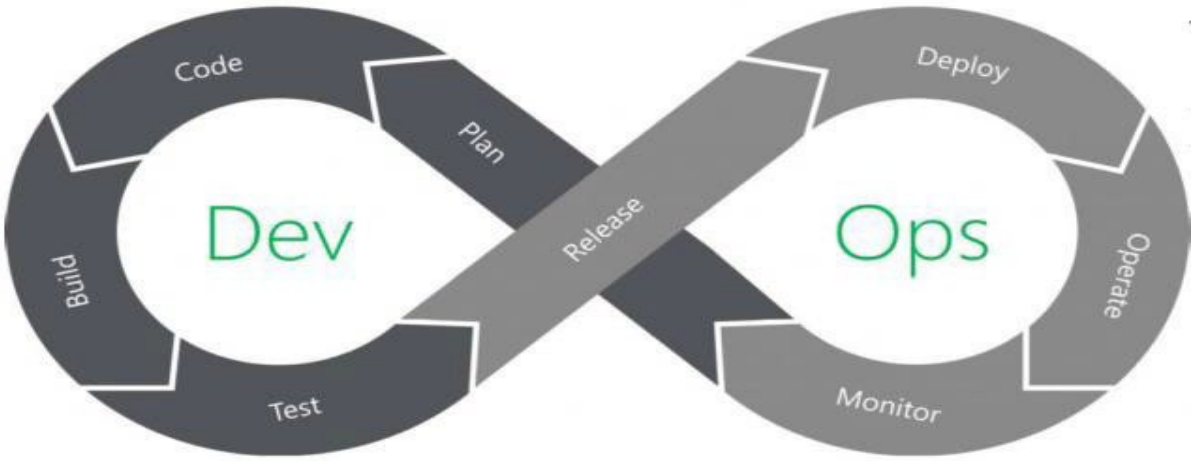


Figure 1 The DevOps Lifecyle("DevOps Solutions | DevOps-as-a-Service | ETG Global Services, Inc", 2020)

However, in implementing DevOps cycles one may encounter many challenges as described in Leite et.al (2019) particularly in selection of tools because of proliferations of tools that are available from variety of vendors for various stages of DevOps cycle as detailed in Armlin (2021), Dennis (2020), and Sen (2021). In this paper attempt is made to integrate DevOps principles in our example application using some popular tool sets. To keep the paper concise and to emphasize salient features of DevOps pedagogy, many details of the implementation are purposely left out in this paper.

The rest of the paper is organized as per the following manners.

The actual Project developed is discussed in section "Anatomy of the Project". Section titled "Tools Used for DevOps Project" discusses selected tools used for application development and deployment. The output of important DevOps stages is shown in section titled "Result" followed by conclusion in section titled "Conclusions". Figures for implementation are provided in the **Appendix** for ease of referring.

Anatomy of the Project

The project is to develop a simple Web application for Hotel Management to support a hotel manager. The basic functions that are supported are a guestbook, where customer can leave comments about their stay and a calculator for basic calculations, like the total billing amount. (Figure 2).

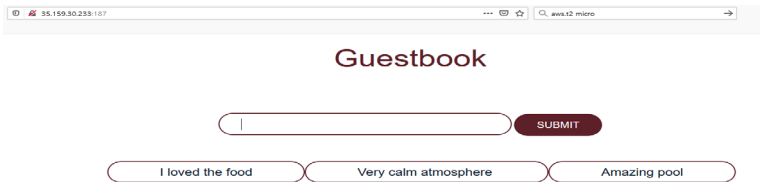


Figure 2. The guestbook displayed in a web browser

Each function is embedded inside a website which interacts with an Express Server. Amazon Elastic Compute Cloud (EC2) is used as a computing environment in the Amazon Web Services (AWS) Cloud to develop and deploy applications faster (Figure 3). The Express Server runs in a Docker container which is deployed on an AWS EC2 instance. The project is executed using DevOps principles by two team members.

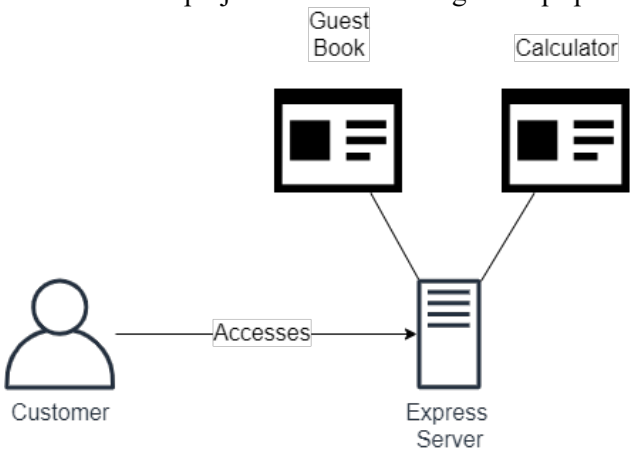


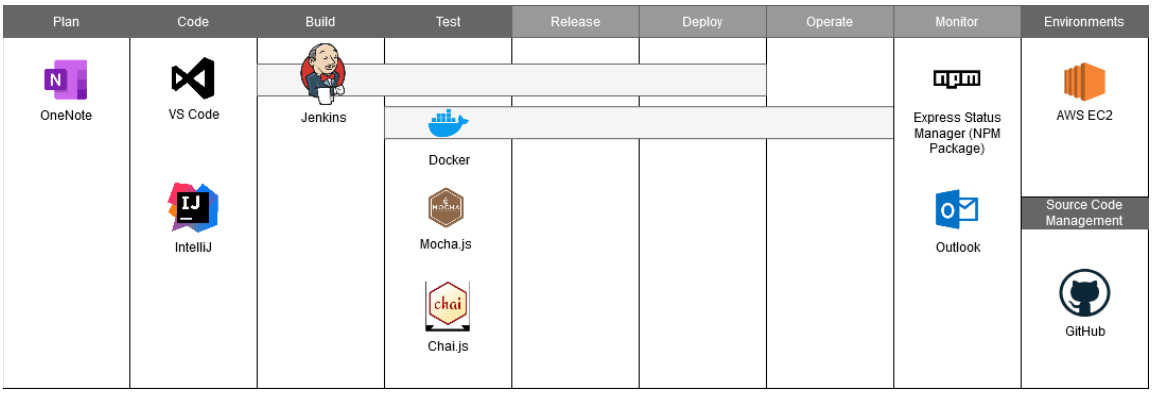
Figure 3. Architecture of the Web Application

Tools Used for DevOps Phases

Different well established industry standard tools are used for implementing the various stages of DevOps life cycles. Table 2 list the tools used for different DevOps stages for this application. For details of these tools one can refer to appropriate websites specified in the table. For comprehensive lists of DevOps tools one may refer Sen (2021), and Periodic Table of DevOps Tools in https://digital.ai/periodic-table-of-devops-tools (See Table 2 and associated diagram below).

Table 2. Selected Tools Chain Used at various DevOps stages

Name	Website	Version	DevOps Stage
OneNote	https://www.microsoft.com/en/microsoft-365/onenote/digital-note-taking-app?rtc=1	OneNote 2016	Plan
VS Code	https://code.visualstudio.com/	1.47 (13.07.2020)	Code
IntelliJ IDEA	https://www.jetbrains.com/idea/	2020.1.3 (8. July 2020)	Code
Docker	https://www.docker.com/	19.03 (1. June 2020)	Test, Release, Deploy, and Operate
Jenkins	www.jenkins.io	Jenkins 2.444	Build and Deployment
Mocha.js	https://mochajs.org/	7.1.1 (18 March 2020)	Test
Chai.js	https://www.chaijs.com/	4.2.0 (25 September 2018)	Test
Express Status Monitor	https://www.npmjs.com/package/express-status-monitor	1.3.3 (May 2020)	Monitor
AWS EC2	https://aws.amazon.com/ec2/		Environment
GitHub	https://github.com/	2.14.2 (5. March 2020)	Source Code Management



Results

Plan & Code:

OneNote is used to create conceptual planning page and to organize the team. To track the product backlog and the progress of the project, Kanban board is set up in OneNote. For this study, a simple Kanban board covering four states (backlog, next, in progress, done), was created, by using tables and some simple forms (Figure 4). One Note provides additional features to share notebooks with other team members. The tasks are assigned through color coding to the respective member of the team.

Figure 5 shows the initial state of the website. It implements a simple Guestbook, where guests can comment on their stay in a hotel or suggest improvements. This example assumes, that the hotel wants to change the color of their website to red, because this color matches the interior design of the entrance hall. Without a DevOps pipeline, the developer would have to manually follow a complex procedure that includes many steps, like testing or restarting and redeploying the docker containers. As a simple color update could not justify such a long downtime of the productive environment, the change would likely be postponed to a later date.

As the development team uses multiple platforms, like MacOS and Windows, a centralized source code management system is needed. For this project, git and GitHub are used. This enables the developers to use their favored code editors like VS Code and IntelliJ. Git and Github are very popular tools and these tools are not discussed in details in this paper.

The code editors can be extended by installing extensions like the Git extension. This enables the integration of Git into the editor. As a result of that, changes are displayed in the editors and different Git commands can be executed by using the graphical git interface of the code editor.
GitHub allows the implementation of webhook, which are used to push an event to a web URL. After a successful commit, the web hook will send a push event to Jenkins, which will trigger the execution of the Jenkins pipeline, described later.

Using DevOps, the developer can accelerate this process and conduct the changes on his own. To implement the design improvements, he will change the .css file of the project on his local development environment. The developer in this example uses Visual Studio Code. Figure 6 shows highlighted item is changed to new hexadecimal color code: #5e2028 which is a dark shade of pink-red.

After the developer has completed the changes to the code files of the project, he will commit them to the centralized source code management repository (Figure 7), which is hosted on GitHub and will trigger the execution of the Continuous Integration and Continuous Deployment process, which is controlled via Jenkins (Figure 8).

Continuous Integration (Build Stage, Testing):

In this stage, the newly developed code is built, then verified through tests and later be forwarded to the next stages. The tool chosen for this stage is Jenkins, as it enables to set up flexible workflows that can be triggered through a commit to the central repository. After that, Jenkins manages the build process by executing a pre-configured pipeline which firstly executes several tests (unit tests, integration tests) on a test environment running inside a Docker container. After the deployment of the application, the acceptance tests are conducted to secure the flawless execution of the web application. The tests are based on Mocha.js framework and Chai.js assertion library. The combination of Mocha framework with Chai assertion library is used to test the endpoints of the web application as well as its functionality. While Mocha framework

provides the basic structure to write the tests, allowing the declaration of multiple test sets each with a variety of individual tests, Chai is the assertion library used in the individual tests, providing a very natural and therefore easy to use syntax. As Mocha writes its extensive logs directly into the console, they can be easily accessed in the Jenkins web interface by clicking on the test stage. Docker is used to build a container, based on the production container, exclusively for testing the endpoints and the functionality with a debugging server. Jenkins and the Docker environment run on an AWS EC2 instance. Jenkins generates reports about the success of the testing, which provides feedback for the developers to improve their code. (Arvind, 2016). In the test file (Figure 9) there are two describe blocks, one for the endpoint testing, the other one for the testing of the calculation function of the application. For endpoint testing, Mocha and Chai simply check if each endpoint returns the status code 200, which stands for "OK". For the calculation testing, each operand is tested with an example calculation and it is checked if the right result is being returned from the function.

The acceptance tests are written like the unit tests with the only difference being that instead of using an emulated server, the real public API endpoints are used. For sending the HTTP requests, the package "XMLHttpRequest" is used (Figure 12).

Continuous Deployment (Release, Deploy and Operate):

After tests were successful, the new code can be deployed to the production environment. Docker container running on AWS EC2 is deployed using Jenkins after a successful integration phase. For the sake of brevity, installation steps to interact AWS EC2 is not included in this paper. AWS EC2 Free Tier includes 750 hours of Linux and Windows EC2 Micro instances each month for one year (https://aws.amazon.com/ec2/pricing). Free tier instances offer enough resources to run small DevOps projects. The instance used for this project did not cost any money, as it operated within the free tier limits. For hosting an AWS EC2 instance, an AWS account must be initially created. After setting up the account and navigating to the "AWS Management console" a new instance can be easily hosted by following the single setup steps after pressing the "Launch instance" button. (Figure 14). After the successful set up, the user can see his running instances in the management console (Figure 14). The AWS management console provides important information about the server, like the Instance ID, or the public IPv4 address. This information is later important to connect to the instance. The user can easily connect to the AWS EC2 instance using putty or another SSH client. To set up the connection, the public DNS and a public access key must be entered (Figure 10).
It is possible to configure the instance using the command line interface and install the necessary software for the DevOps pipeline, for example docker and Jenkins (Figure 11).

Jenkins takes the code hosted on the centralized Git Repository and builds a testing environment on a dedicated AWS EC2 instance. The testing environment consists of a docker container hosting the express server with the website. After a successful build phase, Jenkins will execute Unit Tests and Acceptance Tests and deploy the new built code to the production environment if these tests where successful. Any faults, bugs or delays are visualized to provide an overview over the current state of the pipeline for the developer (Figure 8). In addition, Jenkins will send notifications in form of emails to the developer in case of any problems. The email as configured in Figure 16 gives detailed information of the build result.

After the commit to the GitHub Repository for any changes made, every step from testing over deployment to monitoring was executed automatically by the DevOps pipeline, while keeping the developer in control due to regular updates and notifications in case of problems. The pipeline relieves the developer from arbitrary tasks, so the developers can focus on the development and logic of the project, thus improving innovation and accelerating the development process. Figure 13 shows the improved design after the pipeline was successfully executed. The color of the website on the productive environment is now red.

As shown in Figure 15, Jenkins creates a visual interface out of the Jenkinsfile shown in Figure 14. The left section shows date and time of each build, including the number of commits that were done since the last build. The right side of the grid shows the results of each stage, their durations and offers the option to directly view the logs of the respective stage. For example, item# 80 shows Acceptance test is flagged as failure, which was corrected on next build as shown in item#81 in Figure 15.

After the Pipeline is configured, every new commit to the source code management triggers its execution. That means the bash commands configured beforehand will be executed directly on the agent (machine) and all logging information will be stored in the Jenkins interface, making it a central point of access for documentation and bug fixing information. Each finished Pipeline execution is followed by a notification the production environment of this web application is based on a Docker container running on AWS EC2, which is deployed after a successful integration phase using Jenkins. To generate a custom Docker image a Dockerfile is needed. This file contains commands for configuring the basic image and the bash command that should be run on each start of the container. A sample docker file is shown in Figure 17.

Continuous Monitoring (Monitor):

To prevent downtime of the web application, the production environment must be monitored all the time. The Express Status Monitor provides several metrics and information of the current state of web application. The provided data includes memory, heap and CPU usage, response time, requests per second and a diagram about the sent status codes (Figure 18). If an error occurs, an email is sent to the responsible administrator, as discussed earlier, who will be able to react accordingly to fix the underlying problem to prevent downtime. This email includes the Jenkins job's name, the build number, the result, and a log file. This step is particularly important, as the build is not only triggered in the Jenkins web interface, but also starts after every push to the SCM, which would lead to developers not getting immediate feedback after the build, as they do not necessarily have the Jenkins web page available.

Discussions & Conclusions

The goal of this study is to show how a DevOps pipeline can be used to accelerate the development process of web applications and to answer the research question whether students with limited prior exposure to DevOps paradigm can learn and implement DevOps principles and practices within short span of one semester. From the result it is evident students with only a team of two members accomplished their tasks admirably well, although enormous amount of times was spent in setting up the tool chain and then ensuring each tool accomplished it stated tasks. One Note and GitHub enable flexible planning and coding, while providing collaborating platforms for the developers. Continuous Integration and Deployment are the most important aspects that bridge the gap between developers and operational personnel. The implementation of web application is made possible by using few widely used tools, such as Jenkins, Docker and Mocha.js. The feedback on the build and integration process is provided automatically, by Jenkins pipeline or mails, to the corresponding developer in case of errors providing transparency. Furthermore, costs can be reduced, by using a cloud provider like AWS EC2, which enables the outsourcing of huge part of the operational overhead. Small teams can profit from this implementation, as their resource consumptions usually stays within the free tier limit of AWS EC2, without incurring any additional cost.

Despite the huge benefits a DevOps toolchain offers, there are still some points to consider before using DevOps practices. DevOps represents a fundamental shift in terms of software delivery mechanism it is yet to be acknowledged by the majority of businesses. Therefore, establishing a DevOps approach in a rather traditional enterprise might be fairly hard, as it also requires changes in the fundamental structure of

software. Another thing to keep in mind is the quite large amount of time needed to initially familiarize with the DevOps tools and to setup the toolchain for the first time, making it not exactly suitable for quick proof of concepts. However, as innovation accelerates in today fast paced digital businesses, short development cycles and a reduced time to market becomes more and more important. Web applications and their development through the proficient use of DevOps paradigms will be an integral component to gain competitive advantage on the market and will shape the future of software development.

References

Arvind. (2016). What is DevOps? DevOps Methodology, Principles & Stages Explained, Retrieved February 22, 20232 https://www.edureka.co/blog/what-is-devops/

Armlin, D. (2020). 9 Essential DevOps Tools for 2021, Retrieved June 4, 2022 from https://tinyurl.com/e2xy9x42

Bluestone, D, Drew, G. Electrify Web Developments with DevOps, Retrieved June 4, 2022 from https://www.cyber-duck.co.uk/insights/resources/electrify-web-development-with-devops

CD Foundation (2020). Jenkins, Retrieved June 4, 2022 1 from https://www.jenkins.io/

Davies, J, Daniels, K (2016). Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale, O'Reilly Media; 1st edition

Dennis, D. (2020). DevOps Tools Every Engineer Should Know, Retrieved July 9, 2021 https://dev.to/daviddennis02/devops-tools-every-engineer-should-know-hf2

DevOps Solutions | DevOps-as-a-Service | ETG Global Services, Inc (2020). Retrieved June 4, 2022, from https://www.etggs.com/services/managed-services/devops/.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. IEEE Software, 33(3), 94–100.

Lette, L, Rocha, C, Fabio, K, Dejan, M, Paulo, M (2019), A Survey of DevOps Concepts and Challenges, Retrieved June 4, 2022 from https://arxiv.org/pdf/1909.05409.pdf

Sen, A. (2021). DevOps, DevSecOps, AIOPS- Paradigms to IT Operations. Evolving Technologies for Computing, Communication and Smart World, Pages 211-221, Springer

Sen, A , Baumgartner, L, Heiß. K**,** Wagner, **C** (2022). DevOps paradigm -a pedagogical approach to manage and implement IT project, Issues in Information Systems Volume 22, Issue 4, pp.117-133, 2021, Retrieved June 4, 2022 from https://iacis.org/iis/2021/4_iis_2021_117-133.pdf

Sharma, S. & Coyne, B. (2015). DevOps for Dummies. Hoboken, NJ: John Wiley & Sons, Inc.

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). DevOps and Its Practices. IEEE Software, 33(3), 32–34

Appendix: Screen Shots

Figure 4. Planning of the project using OneNote

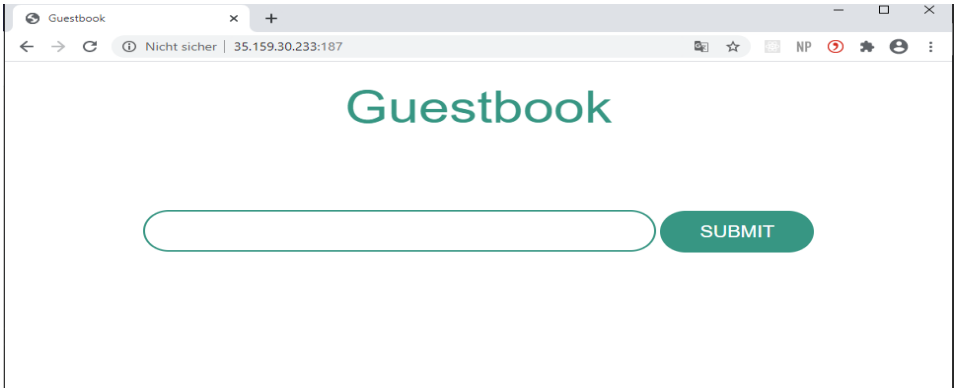


Figure 5. Initial state of the website (color: green)

```
# style.css
public > CSS > # style.css > input
form {
  margin: 0 auto;
  max-width: 50em;
  text-align: center;
}

input {
  border: 0;
  border-radius: 1000px;
  box-shadow: inset 0 0 0 2px #5E2028;
  display: inline;
  font-size: 1.5em;
  margin-bottom: 1em;
  outline: none;
  padding: .5em 5%;
  width: 55%;
}
li{
  border: 0;
  border-radius: 1000px;
  box-shadow: inset 0 0 0 2px #5E2028;
  display: inline;
  font-size: 1.5em;
  margin-bottom: 1em;
  outline: none;
  padding: .5em 5%;
  width: 55%;
}

form a {
  background: #5E2028;
  border: 0;
  border-radius: 1000px;
  color: #FFF;
  font-size: 1.25em;
  font-weight: 400;
  padding: .75em 2em;
  text-decoration: none;
  text-transform: uppercase;
```

Figure 6. Changes in the CSS File (highlighted item is the changed color code))

```
C:\Users\sarof\VSCode_Projekte\DevOpsFinalProjekt\DevOpsFinalProject>git add public/css/style.css

C:\Users\sarof\VSCode_Projekte\DevOpsFinalProjekt\DevOpsFinalProject>git commit -m "DevOpsTest: Color Change for the website to red"
```

```
C:\Users\sarof\VSCode_Projekte\DevOpsFinalProjekt\DevOpsFinalProject>git push
```

Figure 7. Commit and push of the local changes to the centralized GitHub Repository

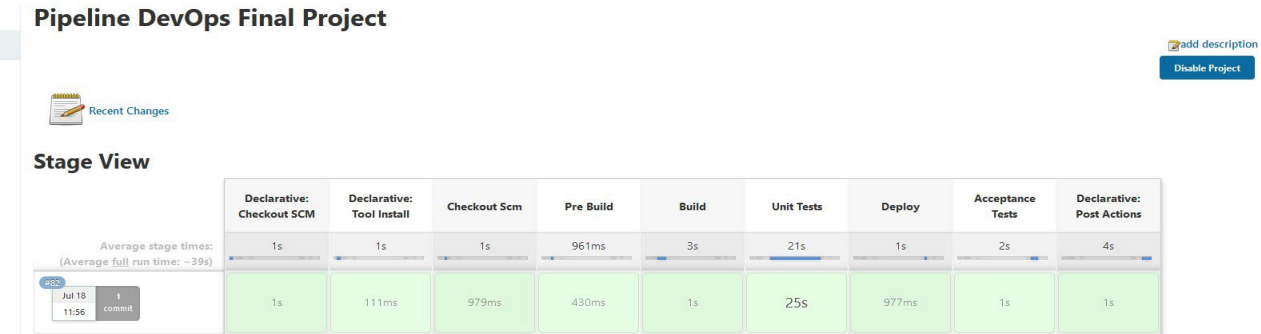


Figure 8. CI and CD pipeline in Jenkins

```
const server = require('../server');
let request = require('supertest');
let assert = require('assert');
let expect = require('chai').expect;
//Testing of the endpoints
//  /
// /hellothere
// /calculate
// /test


describe("Endpoints Reachable", function(){


    it("Test of the Enpoint: /", function testSlash(done){
        request(server)
            .get('/')
            .expect(200,done);
    });

    it("Test of the Enpoint: /hellothere", function testHellothere(done){
        request(server)
            .get('/hellothere')
            .expect(200,done);
    });

    it("Test of the Enpoint: /calculate", function testCalculcate(done){
        request(server)
            .get('/calculate')
            .expect(200,done);
    });

    it("Test of the Enpoint: /test", function testTest(done){
        request(server)
            .get('/test')
            .expect(200,done);

    });
});

describe("Test Calculations", function(){

    it("Test addition:", function testadd(done){
        let result = server.calculate( num1: 12,  operand: "+",  num2: 14);
        expect(result).to.equal(26);
        console.log(result);
        done();
    });

    it("Test subtraction:", function testsub(done){
        let result = server.calculate( num1: 45,  operand: "-",  num2: 14);
        expect(result).to.equal(31);
        console.log(result);

        done();
    })
```

Figure 9. Unit Tests written with Mocha and Chai

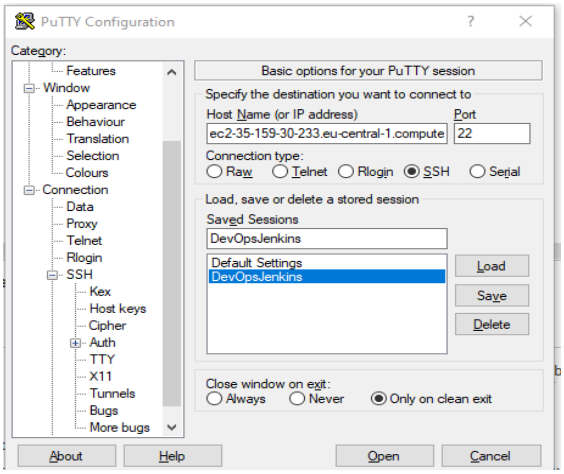


Figure 2. Putty configuration to access the AWS EC2 instance

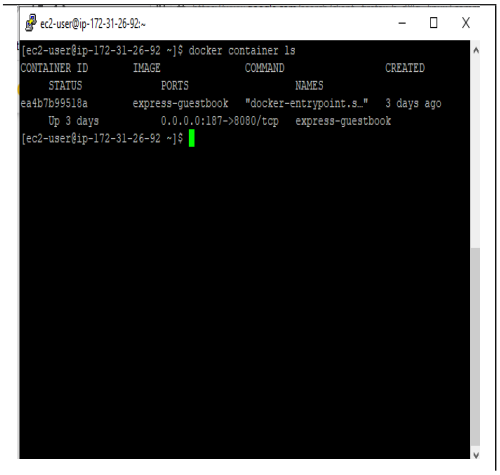


Figure 3. Command Line Interface of the EC2 instance

```
let XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
let expect = require('chai').expect;

describe("Testing API...", function(){

    it("Testing /test Endpoint:", function testTest(done){
        let result = httpGet( theUrl: "http://35.159.30.233:187/test");
        //console.log(result);
        expect(result.status).to.equal(200);
        expect(result.responseText).to.equal("<h2>Test was successful</h2>");
        done();

    });

    it("Testing calculator:", function testCalc(done){
        let result = httpGet( theUrl: "http://35.159.30.233:187/calculate/?num1=3&operand=*&num2=4");
        //console.log(result);
        expect(result.status).to.equal(200);
        expect(result.responseText).to.equal("<h2>Calculation:</h2><p>3 * 4</p><h2>Result:</h2><p>12</p>");
        done();

    })

    it("Testing /hellothere:", function testHello(done){
        let result = httpGet( theUrl: "http://35.159.30.233:187/hellothere");
        //console.log(result);
        expect(result.status).to.equal(200);
        expect(result.responseText).to.equal("<img src=\"https://i.ytimg.com/vi/frszEJb0aOo/maxresdefault.jpg\"><h2>General Kenobi</h2>");
        done();
    } )

    it("Testing /:", function testHome(done){
        let result = httpGet( theUrl: "http://35.159.30.233:187");
        //console.log(result);
        expect(result.status).to.equal(200);
        done();
    } )


})

function httpGet(theUrl)
{
    let xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", theUrl, false ); // false for synchronous request
    xmlHttp.send( null );
    return xmlHttp;
}
```

Figure 12. Acceptance Tests using XMLHttpRequest

Figure 13. Result of the design improvement

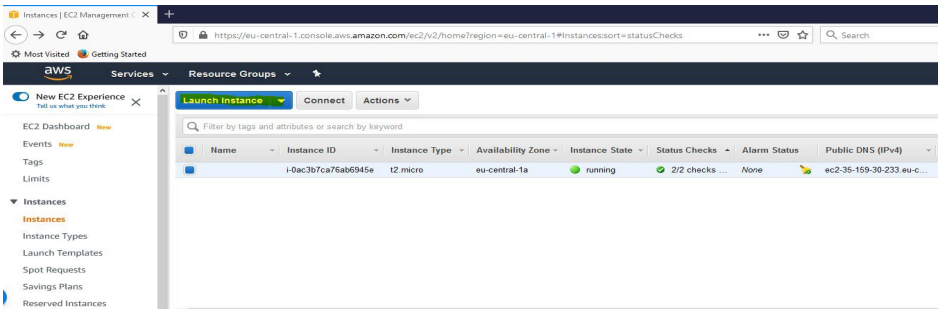


Figure 4. AWS Management console with Launch instance button

Figure 15. Jenkins Pipeline "Stage View"

```
pipeline {
  agent any
  tools {
  nodejs 'node'
  }
  stages {
    stage('Checkout Scm') {
      steps {
        git(url: 'https://github.com/LuckySan/DevOpsFinalProject', credentialsId: '806566a3-1079-499a-8312-649a95c6fcd5')
      }
    }
    stage('Pre Build')
    {
    steps{
    sh '''
    sudo service docker start
    sudo docker rename express-guestbook express-guestbook-old || sudo docker stop express-guestbook-old && sudo docker rm express-guestbook-old && sudo docker r
    '''
    }
    }
    stage('Build') {
      steps {
        sh '''sudo docker build -t express-guestbook:latest .'''
      }
    }
    stage ('Unit Tests'){

       steps{

       sh '''
              sudo docker stop express-guestbook-old || echo "no such container"
              sudo docker build -t express-guestbook-test -f Dockerfile.test .
              sudo docker run --rm express-guestbook-test'''
}
    }
    stage('Deploy') {
      steps {
       sh '''sudo docker run -p 187:8080 --name express-guestbook -d express-guestbook'''
      }
    }
    stage('Acceptance Tests'){
    steps{
       sh '''mocha  acceptanceTests/'''
    }
    }
  }
  post {
            failure{
             sh '''
             sudo docker stop express-guestbook || echo "no such container"
             sudo docker start express-guestbook-old || echo "no such container"'''
             }
             success{
             sh '''
             sudo docker stop express-guestbook-old || echo "no such container"
             sudo docker rm -f express-guestbook-old || echo "no such container"'''
             }
             always {
                    emailext attachLog: true, body: "${currentBuild.currentResult}: Job ${env.JOB_NAME} build ${env.BUILD_NUMBER}\n More info at: ${env.BUILD_U
                    recipientProviders: [[$class: 'DevelopersRecipientProvider'], [$class: 'RequesterRecipientProvider']],
                    subject: "Jenkins Build ${currentBuild.currentResult}: Job ${env.JOB_NAME}"

                | }
```

Figure 16. Jenkinsfile for creating the whole Pipeline

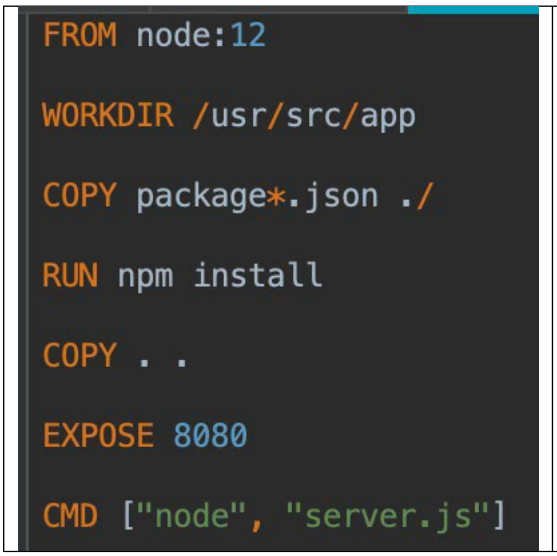


Figure 17. Dockerfile for production container Server

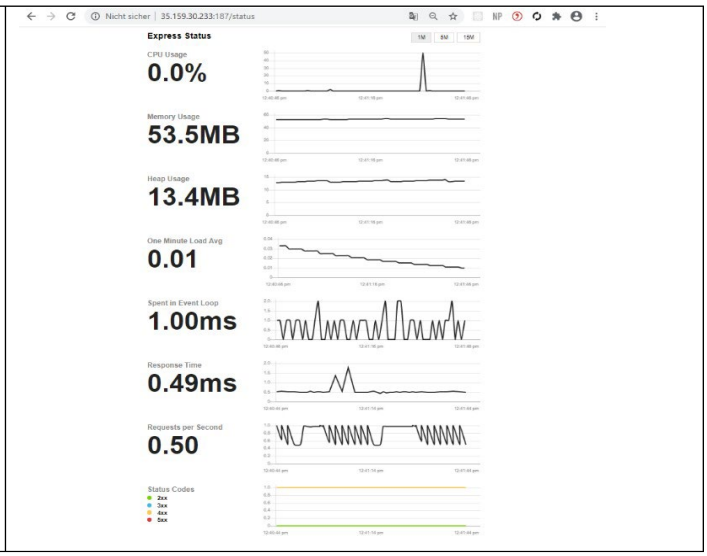


Figure 18. - Monitoring of the web application by Express