

Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"

Network Technologies Department of the Faculty of Informatics



**Development of CI / CD platform deployment automation module for
group software development**

**Text part of master work
in specialty “Computer Science and Information Technologies” 112**

Coursework supervisor
Cherkasov D.I. senior teacher, Ph.D.

(signature)

“ ____ ” _____ 2021 yr.

Made by student
Ivanov O.A

“ ____ ” _____ 2021 yr.

Kyiv 2021

Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"
Network Technologies Department of the Faculty of Informatics

APPROVED

Head of Department of Computer Science
associate professor of Computer Science

S.S. Gorokhovsky

(signature)

“ ____ ” _____ 2020 yr.

INDIVIDUAL TASK

for master work

For the student of the Faculty of Informatics of 1 course of master program
THEME Development of CI / CD platform deployment automation module
for group software development

Output data:

Text part content of coursework:

An individual task

Calendar plan

Abstract

Introduction

Part 1: Problem statement. Existing approaches

Part 2: Review of existing solutions and key principles

Part 3: Solution development

Summary

References

Issue date “ ____ ” _____ 20 yr. Supervisor _____

(signature)

Task received _____

(signature)

Theme: Development of CI / CD platform deployment automation module for group software development

Calendar plan of master work execution:

No	Stage name	Deadline	Note
1.	Getting of master topic	20.03.2021	
2.	Searching of appropriate literature	25.03.2021	
3.	Part 1: Problem statement. Existing approaches	01.04.2021	
4.	Part 2: Review of existing solutions and key principles	01.04.2021	
5.	Part 3: Solution development	01.04.2021	
6.	Documentation forming	25.04.2021	
14.	Writing master work summary	01.05.2020	
15.	Master work analysis with the supervisor	10.05.2021	
16.	Master work changing according to the supervisor's remarks	20.05.2021	
17.	Creating of the presentation	05.06.2021	
18.	Defending of the master work	17.06.2021	

Student Ivanov O.A.

Supervisor Cherkasov D.I.

“ ”

ABSTRACT	5
INTRODUCTION.....	6
1. PROBLEM STATEMENT. EXISTING APPROACHES	8
1.1 Delivery Pipeline.....	8
1.2 Azure DevOps	9
1.3 AWS CodeCommit CodeBuild CodeDeploy	11
1.3.1 AWS CodeCommit	11
1.3.2 AWS CodeBuild	12
1.3.3 AWS CodeDeploy.....	12
1.4 GCP Cloud Build.....	13
2. REVIEW OF EXISTING SOLUTIONS AND KEY PRINCIPLES	15
2.1 Continuous Integration.....	15
2.2 The deployment pipelines	17
2.3 Tools and approaches for CI/CD	19
3. SOLUTION DEVELOPMENT	22
3.1 Creating Kubernetes.....	22
3.1.1 Kubernetes on Azure.....	22
3.1.2 Kubernetes on GCP.....	26
3.1.3 Kubernetes on AWS.....	31
3.2 Jenkins installation into Kubernetes	39
3.3 Requirement's installation and validation	43
SUMMARY	50
REFERENCES.....	50
ABBREVIATIONS AND TERMS.....	53
APPENDIX	54

ABSTRACT

In the presented work we reviewed the main CI/CD principles and delivery workflow. We provided definition and benefits for each of part of the CI/CD.

Latter we covered definition of CI and CD. Provided tools analysis and narrowed audience for expected module. We have chosen the platform base and picked up clouds for developing solution.

After that we developed modules for automatics deployment into cloud as easiest for user as possible and created all necessary scripts with ability to integration in bigger system or launching out of the box.

In the end test runs were done for every script and compared result and difficult of managing. Based on achieved results we provided the feedback.

INTRODUCTION

Each day billions of ideas appears in human minds. Some of them dies right after born. Others stick to us and moving us forward, like new technologies, approaches, solutions and tools. This process helps mankind evolve. The ideas in IT sphere are different from other. Inside IT you should move faster than your competitors to bring your idea in life and delivery it to customer. A lot of brilliant ideas are died on the start, the reasons different, however most of them related to speed of development.

Modern ideas are complicated and needs many resources and qualified workers. Especially high skilled developers, appropriate infrastructure, and environment for development. To became product every idea should pass SDLC through the delivery pipeline. And the main problem hided here. The development team might be super skilled, but due to poor integration process, they fail. Modern market proposes solutions for small teams and even to individuals, but they have limitation.

The building effective CI\CD system from the very beginning of the project, or for idea evaluation, will help teams to be success faster, save time and costs. In real life even huge enterprise solution creates CI\CD for months. The startups and small team have no time for so long term.

Another challenge that appears is infrastructure. Startups and small team, as usual, has limitation in money, so they must move to the cloud, this is cheaper, fast, and every cloud provider has a free plan, and some of the resources, even after ending of trial, remains free forever.

Bringing automation into the process of creating CI\CD into the cloud will significantly reduce the time of preparation, will allow a small team to start faster, and increase chances for success in the future.

What makes this paper relevant, as were stated before, is the fact of speedup development process, creating CI\CD from the very beginning, is crucial for startups. Moreover, if we bring this option as cloud agnostic solution it will allow teams to migrate faster and it is also increase chances for success.

The objective of this paper is to analyze existing approaches to the problem solution, characterize, explore and solve main problems that might arise when automatically deploy CI\CD system into the cloud. As a result of the research, a automation module for deployment CI\CD into the cloud should be developed and tested.

The object of study in this paper is automatic creation of CI\CD in the cloud environment.

The research methods include analyzing of available articles, papers, best practices, and development of a working module for automatic deploy CI/CD platform in the cloud.

1. PROBLEM STATEMENT. EXISTING APPROACHES

1.1 Delivery Pipeline

Every time that you start project you should know the release date (the point in time then end user will be able to use your product). A lot of estimation, planning and other activities could be done before, but this date will be delayed due to bugs or undetected defects in software.

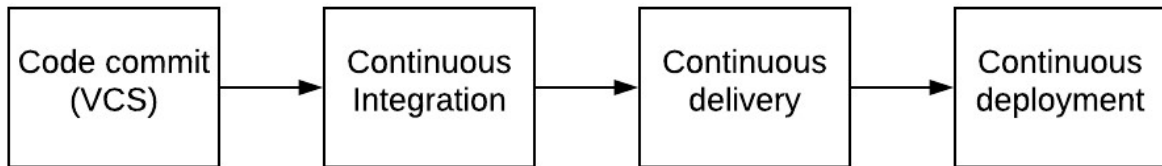


Figure 1.1 The delivery pipeline

On the Figure 1.1 shown general approach of Software delivery. Code, that was written by developer, should pass through all this stages. On the other hands: such structure is not implemented at every project. Common situations that code deployed manually, or test coverage is low and as a result CI system are not informative.

Continuous Integration, Delivery, and Deployment are relatively new development practices that have gained a lot of popularity in the past few years. Continuous Integration is all about validating software as soon as it's checked in to source control, more or less guaranteeing that software works and continues to work after new code has been written. Continuous Delivery succeeds Continuous Integration and makes software just a click away from deployment. Continuous Deployment then succeeds Continuous Delivery and automates the entire process of deploying software to your customers (or your own servers).[2]

If Continuous Integration, Delivery, and Deployment could be summarized with one word, it would be Automation. All three practices are about automating the process of testing and deploying, minimizing (or eliminating) the need for human intervention, minimizing the risk of errors, and making building and deploying software easier up to

the point where every developer in the team can do it (so you can still release your software when that one developer is on vacation or crashes into a tree).[2]

And automation should be done with appropriate tools and approaches. The speed and quality of implementation CI\CD system into your project will make a significant impact on your software quality, also it will decrease time to market for your product.

However, you should have qualified engineers in your team to make CI/CD system up and running in a short time with high quality. If you're working in a large company with good competency, it won't be hard, but for small teams, startups, and for single enthusiasts, CI/CD launch from scratch could become a problem. The main idea of this work is to make their life easier.

1.2 Azure DevOps

Azure DevOps is a cloud solution from Microsoft that helps developers build software faster and better. The composition includes the following services:

1. Azure Pipelines - CI service support for any language, connection to GitHub and any Git repository.
2. Azure Boards - a powerful workflow control tool: kanban boards, job logs, dashboards and custom reports.
3. Azure Artifacts - Maven, npm and NuGet channels.
4. Azure Repos - Closed cloud repositories Git unlimited storage for project files
Joint requests for inclusion, improved file management and more.
5. Azure Test Plans - a comprehensive solution for planning and random testing.

All Azure DevOps services are open and extensible. They are perfect for any type of application, regardless of environment and platform. They can be used together as a comprehensive DevOps solution or separately with other services [15].

Azure Boards - Plan, track, and discuss work across teams. Define and update issues, bugs, user stories, & other work with customizable Scrum, Kanban, and Agile tools. The Azure Boards app for Microsoft Teams enables users to perform the following tasks:

1. Set up and manage subscriptions for creating and updating work items
2. Manage other work item events
3. Receive and manage notifications for work item events in their Teams channel
4. Create work items from conversations in the channel
5. Search and share work items with other members in the channel using the messaging extension
6. View work item previews from their URLs to initiate discussions and keep the conversations contextual.

Azure Pipelines automatically builds and tests code projects to make them available to others. It works with just about any language or project type. Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target constantly and consistently. Implementing CI helps to catch bugs early in the development cycle, which makes them less expensive to fix. Automated tests execute as part of the CI process to ensure quality. Artifacts are produced from CI systems and fed to release processes to drive frequent deployments. The Build service in TFS helps you set up and manage CI for your applications. Deploying and testing in multiple environments drives quality. CI systems produce the deployable artifacts including infrastructure and apps. Automated release processes consume these artifacts to release new versions and fixes to existing systems. Monitoring and alerting systems run continually to drive visibility into the entire CD process. The Release service in TFS helps you set up and manage CD for your applications. Continuous Testing (CT) on-premises or in the cloud is the use of automated build-deploy-test workflows, with a choice of technologies and frameworks, that test your changes continuously in a fast, scalable, and efficient manner [15].

Azure Artifacts is a package management solution integrated into Azure DevOps that allows developers to create and share Maven, npm, and NuGet packages via feeds that can be both public and private to an organization with teams of any size. Azure Artifacts also allows include upstream feeds into configured package feeds to allow to cache packages that applications are dependent. This allows developers to continue to take

dependencies on packages that application is using even if they are no longer available from the original source feed [15].

Azure Artifacts additionally allows team to store other artifacts on feed in what are called universal packages that can be customized to meet developer's needs. One example of a universal package use case would be to host an internal PowerShell gallery where team could upload scripts and PowerShell modules that are used inside company [16].

Azure DevOps Services and TFS projects contain Git repositories, work items, builds, and releases. Azure DevOps and TFS provide rich and powerful tools everyone in the team can use to drive quality and collaboration throughout the development process. The easy-to-use, browser-based test management solution provides all the capabilities required for planned manual testing, user acceptance testing, exploratory testing, and gathering feedback from stakeholders [16].

1.3 AWS CodeCommit CodeBuild CodeDeploy

1.3.1 AWS CodeCommit

CodeCommit is a managed source control service that hosts private Git repositories. CodeCommit eliminates the need for managing your own source control system or scaling its infrastructure. CodeCommit could be used to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with existing Git-based tools. CodeCommit features:

1. Fully managed service hosted by AWS.
2. Encrypted repositories.
3. Pull requests support.
4. Scaling version control projects.
5. No limit on the size of repositories or files.
6. Integration with other AWS and third-party services.
7. Migration to CodeCommit from any Git-based repository.

CodeCommit provides a console for the easy creation of repositories and the listing of existing repositories and branches. In a few simple steps, users can find information about a repository and clone it to their computer, creating a local repo where they can make changes and then push them to the CodeCommit repository. Users can work from the command line on their local machines or use a GUI-based editor. [17]

1.3.2 AWS CodeBuild

AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale own build servers. It provides prepackaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more. You can also customize build environments in CodeBuild to use your own build tools. CodeBuild scales automatically to meet peak build requests. [18] CodeBuild benefits:

1. Fully managed (CodeBuild eliminates the need to set up, patch, update, and manage your own build servers).
2. On demand (CodeBuild scales on demand to meet your build needs. You pay only for the number of build minutes you consume).
3. Out of the box (CodeBuild provides preconfigured build environments for the most popular programming languages).

1.3.3 AWS CodeDeploy

AWS CodeDeploy is a deployment service that enables developers to automate the deployment of applications to instances and to update the applications as required. CodeDeploy can deploy application content that runs on a server and is stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories. CodeDeploy can also deploy

a serverless Lambda function. You do not need to make changes to your existing code before you can use CodeDeploy. CodeDeploy benefits:

1. Server, serverless, and container applications.
2. Automated deployments.
3. Maximizing application availability during update.
4. Automatic or manual rolling back.
5. Centralized control and tracking deployment status.
6. Concurrent deployments.

CodeDeploy provides two deployment type options:

In-place deployment: The application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated.

Blue/green deployment: The behavior of your deployment depends on which compute platform you use. For example, Blue/green on an AWS Lambda compute platform: Traffic is shifted from your current serverless environment to one with your updated Lambda function versions. You can specify Lambda functions that perform validation tests and choose the way in which the traffic shifting occurs. All AWS Lambda compute platform deployments are blue/green deployments. For this reason, you do not need to specify a deployment type. [19]

1.4 GCP Cloud Build

Cloud Build is a service that executes your builds on Google Cloud Platform's infrastructure. Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

Build config can be created by team to provide instructions to Cloud Build on what tasks to perform. Team can configure builds to fetch dependencies, run unit tests, static analyses, and integration tests, and create artifacts with build tools such as docker, gradle, maven, bazel, and gulp. Cloud Build executes build as a series of build steps, where each

build step is run in a Docker container. Executing build steps is analogous to executing commands in a script. Development team can either use the build steps provided by Cloud Build and the Cloud Build community, or write your own custom build steps:

1. Build steps provided by Cloud Build: Cloud Build has published a set of supported open-source build steps for common languages and tasks.
2. Community-contributed build steps: The Cloud Build user community has provided open-source build steps.
3. Custom build steps: You can create your own build steps for use in your builds.

Each build step is run with its container attached to a local Docker network named cloudbuild. This allows build steps to communicate with each other and share data. Standard Docker Hub images could be used in Cloud Build, such as Ubuntu and Gradle.
[20]

The following steps describe, in general, the lifecycle of a Cloud Build build:

1. Prepare your application code and any needed assets.
2. Create a build config file in YAML or JSON format, which contains instructions for Cloud Build.
3. Submit the build to Cloud Build.
4. Cloud Build executes your build based on the build config you provided.
5. If applicable, any built images are pushed to Container Registry. Container Registry provides secure, private Docker image storage on Google Cloud.

2. REVIEW OF EXISTING SOLUTIONS AND KEY PRINCIPLES

2.1 Continuous Integration

Continuous integration was first written about in Kent Beck's book *Extreme Programming Explained* (first published in 1999). As with other Extreme Programming practices, the idea behind continuous integration was that, if regular integration of your codebase is good, why not do it all the time? In the context of integration, "all the time" means every single time somebody commits any change to the version control system. As one of our colleagues, Mike Roberts, says, "Continuously is more often than you think" [1].

Continuous integration (CI) is the process of integrating new code written by developers with a mainline or "master" branch frequently throughout the day. This contrasts with having developers working on independent feature branches for weeks or months at a time, merging their code back to the master branch only when it is completely finished. Long periods of time in between merges means that much more has been changed, increasing the likelihood of some of those changes being breaking ones. With bigger changesets, it is much more difficult to isolate and identify what caused something to break. With small, frequently merged changesets, finding the specific change that caused a regression is much easier. The goal is to avoid the kinds of integration problems that come from large, infrequent merges. [3]

In order to make sure that the integrations were successful, CI systems will usually run a series of tests automatically upon merging in new changes. When these changes are committed and merged, the tests automatically start running to avoid the overhead of people having to remember to run them—the more overhead an activity requires, the less likely it is that it will get done, especially when people are in a hurry.[1]

The outcome of these tests is often visualized, where "green" means the tests passed and the newly integrated build is considered clean and failing or "red" tests means the build is broken and needs to be fixed. With this kind of workflow, problems can be identified and fixed much more quickly.[3]

Continuous integration represents a paradigm shift. Without continuous integration, your software is broken until somebody proves it works, usually during a testing or integration stage. With continuous integration, your software is proven to work (assuming a sufficiently comprehensive set of automated tests) with every new change—and you know the moment it breaks and can fix it immediately. The teams that use continuous integration effectively can deliver software much faster, and with fewer bugs, than teams that do not. Bugs are caught much earlier in the delivery process when they are cheaper to fix, providing significant cost and time savings. Hence, CI an essential practice for professional teams, perhaps as important as using version control.[1]

Here are few benefits that have made continuous integration essential to any software development lifecycle.[4]

Early Bug Detection: If there is an error in the local version of the code that has not been checked previously, a build failure occurs at an early stage. Before proceeding further, the developer will be required to fix the error. This also benefits the QA team since they will mostly work on builds that are stable and error-free.

Reduces Bug Count: In any application development lifecycle, bugs are likely to occur. However, with Continuous Integration and Continuous Delivery being used, the number of bugs is reduced a lot. Although it depends on the effectiveness of the automated testing scripts. Overall, the risk is reduced a lot since bugs are now easier to detect and fix early.

Automating the Process: The Manual effort is reduced a lot since CI automates build, sanity, and a few other tests. This makes sure that the path is clear for a successful continuous delivery process.

The Process Becomes Transparent: A great level of transparency is brought in the overall quality analysis and development process. The team gets a clear idea when a test fails, what is causing the failure and whether there are any significant defects. This enables the team to make a real-time decision on where and how the efficiency can be improved.

Cost-Effective Process: Since the bug count is low, manual testing time is greatly reduced and the clarity increases on the overall system, it optimizes the budget of the project.

2.2 The deployment pipelines

Continuous integration is an enormous step forward in productivity and quality for most projects that adopt it. It ensures that teams working together to create large and complex systems can do so with a higher level of confidence and control than is achievable without it. CI ensures that the code that we create, as a team, works by providing us with rapid feedback on any problems that we may introduce with the changes we commit. It is primarily focused on asserting that the code compiles successfully and passes a body of unit and acceptance tests. However, CI is not enough.[2]

Continuous delivery is the next step of continuous integration in the software development cycle; it enables rapid and reliable development of software and delivery of product with the least amount of manual effort or overhead. In continuous integration, as we have seen, code is developed incorporating reviews, followed by automated building and testing. In continuous delivery, the product is moved to the preproduction (staging) environment in small frequent units to thoroughly test for user acceptance. The focus is on understanding the performance of the features and functionality related issues of the software. This enables issues related to business logic to be found early in the development cycle, ensuring that these issues are addressed before moving ahead to other phases such as deployment to the production environment or the addition of new features. Continuous delivery provides greater reliability and predictability on the usability of the intended features of the product for the developers. With continuous delivery, your software is always ready to release and the final deployment into production is a manual step as per timings based on a business decision.[5]

The benefits of the continuous delivery process are as follows:

- Developed code is continuously delivered

- Code is constantly and regularly reviewed
- High-quality software is deployed rapidly, reliably, and repeatedly
- Maximum automation and minimal manual overhead

The tools that perform continuous integration do the job of continuous delivery as well.

Continuous deployment is the fully matured and complete process cycle of code change, passing through every phase of the software life cycle to be deployed to production environments. Continuous deployment requires the entire process to be automated--also termed as automated application release--through all stages, such as the packaging of the application, ensuring the dependencies are integrated, deployment testing, and the production of adequate documentation for compliance.[3]

The benefits of continuous deployment and automated application release are as follows:

- Frequent product releases deliver software as fast as possible
- Automated and accelerated product releases with the code change
- Code changes qualify for production both from a technical and quality viewpoint
- The most current version of the product is ready in shippable format
- Deployment modeling reduces errors, resulting in better product quality
- Consolidated access to all tools, process and resource data leads to quicker troubleshooting and time to market
- Effective collaboration between dev, QA, and operation teams leads to higher output and better customer satisfaction
- Facilitates lower audit efforts owing to a centralized view of all phase activities

At an abstract level, a deployment pipeline is an automated manifestation of your process for getting software from version control into the hands of your users. Every change to your software goes through a complex process on its way to being released. That process involves building the software, followed by the progress of these builds through multiple stages of testing and deployment. This, in turn, requires collaboration between many individuals, and perhaps several teams. The deployment pipeline models

this process, and its incarnation in a continuous integration and release management tool is what allows you to see and control the progress of each change as it moves from version control through various sets of tests and deployments to release to users.[1]

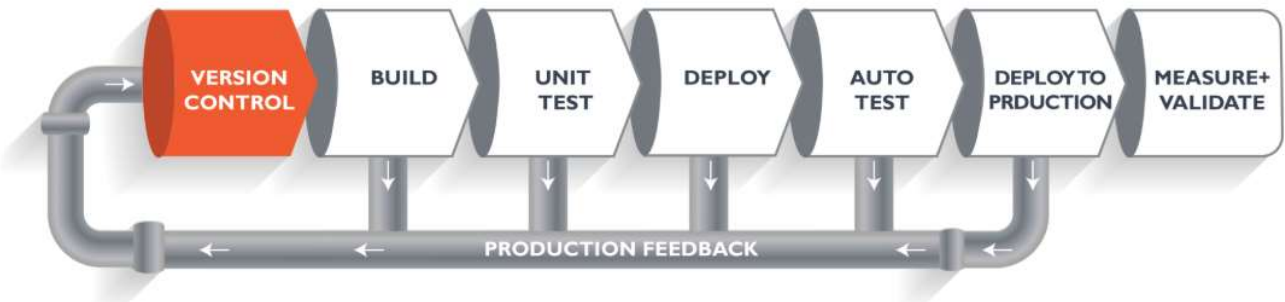


Figure 2.1 Basic deployment pipeline

The on figure 2.1 is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production.

2.3 Tools and approaches for CI/CD

As was discussed in previous sections the main target for us is to make life easier for small teams or even individuals. So, we focused on free reliable tools that could be managed without specific knowledge and experience. Also, we should keep in mind that after some time the small team could grow and even a startup with 2 persons can grow into a huge company. So scaling is also important. We will start from most popular tool Jenkins.

Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software. Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.[6]

Jenkins key features:

- Easy installation and upgrade to various operating systems
- Simple and easy to use interface
- Extensible with a huge community-based plugin resource
- Easy configuration of the environment in the user interface
- Supports distributed master-slave architecture builds
- Build schedules based on phrases
- Supports execution of Windows shells and commands in pre-build steps
- Supports notification of build status

Another one is TeamCity. [7] TeamCity is the build management and continuous integration server for JetBrains. TeamCity is a continuous integration tool which helps to develop and deploy various types of projects. TeamCity runs in a Java environment and integrates Visual Studio and IDEs. The tool can be installed on both Windows and Linux servers, and supports projects like .NET and open stack.

TeamCity 2020.1 provides conditional construction steps, allows build agents to be launched into a Kubernetes cluster, and integrates with Azure DevOps and Jira Software Cloud. It introduces more functionality in a multi-node system to secondary servers, comes with a new Slack notifier, and has several major enhancements to the experimental UI.

TeamCity key features:

- Provides several ways for the subproject to reuse parent project settings and configurations
- The parallel runs jobs on various environments simultaneously
- Allows to build history, view test history reports, pin, tag and add favorites
- Easy to customize, interact and server extension
- Keeps the CI Server stable and functional

- Flexible user management, assignment of user roles, grouping of users, different user authentication methods and a log with all user actions to ensure transparency of all server activities.

Also, a lot of other tools and solution present on market for example: Bamboo, CircleCI, GitLab, Travis CI, etc. [8]

Let's make a choice. Each of mentioned tools has both advantages and disadvantages. However, we should keep in mind that the solution will be used by users with low experience in CI/CD and most likely with limitations in money. Based on it we will develop module for automatic installation CI/CD systems into cloud environment. Each of cloud provider has trial plan that could be used for some time, and always free resources. Using this approach, we will save team from the responsibility of managing dedicated resources and reduce cost at the start.

The platform for CI/CD will be Kubernetes. The key benefits – portability. Each of AWS GCP Azure has Kubernetes as a Platform. So migration will be easy. And it could be run on local machine using Minikube. We rejected pure Docker as it has no power for auto-recovery and scaling as Kubernetes has.

3. SOLUTION DEVELOPMENT

3.1 Creating Kubernetes

3.1.1 Kubernetes on Azure

The base requirement for us is creating Kubernetes in a cloud provider. There are two options for how Kubernetes could be run – full managed cluster (IaaS approach. We should create necessary resources and install Kubernetes on them.) Or use existing service for Kubernetes (PaaS approach. We will get Kubernetes but underlying resources are not our responsibility).

First one will be Azure cloud. Solution will be created in PowerShell and bash and script will be same, as all commands will be used Azure CLI.

The assumption for successful script execution is – all necessary software were installed on client machine. They are: azure cli (az module) and PowerShell or bash. The check for this software will be executed in validation module. In this way the error handling principle will be implemented.

The next part of code responsible for user login into cloud account. It will clear all previous session for current user in Windows for Azure and initiate clean login.

```
Write-Host "All necessary software installed. "  
#login into cloud.  
#Azure login will prompt user for login into cloud env;  
Clear-AzContext -Scope CurrentUser -Force  
az login
```

After execution of this part PowerShell will open default browser with login prompt figure 3.1.

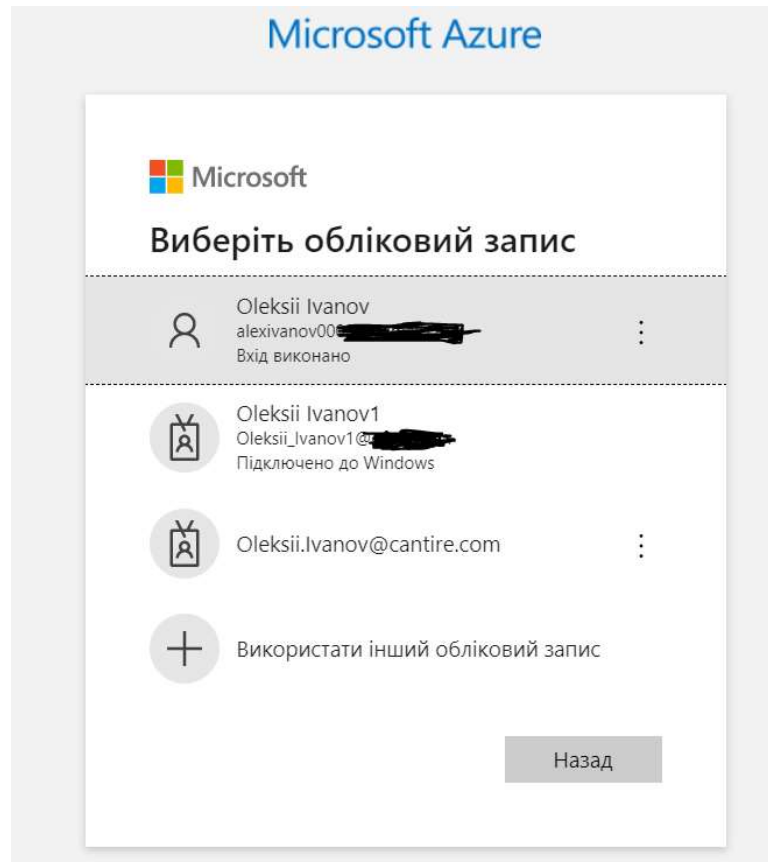


Figure 3.1 Login prompt for Azure

This will be the one interaction in script, all other steps are automated. And will be executed without user intervention.

Before we start cluster creation, we should create resource group. This is logical separation in Azure environment, and it helps in resource organization and management. [9]

```
#creating infrastructure I'll use the cheapest region in USA  
az group create --name $AzureResourceGroupName --location eastus
```

In this part of script the resource group will be created. However the name for it is configurable and set in variable `$AzureResourceGroupName`. It was done for automation and integration, and this name will be reused later in the script.

Next step is deploying Kubernetes cluster itself. The Azure has clear and full documentation so let's use it – QuickStart: Deploy an Azure Kubernetes Service cluster using the Azure CLI [10]

```
#enabling monitoring recording to Azure recommendation https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.OperationalInsights

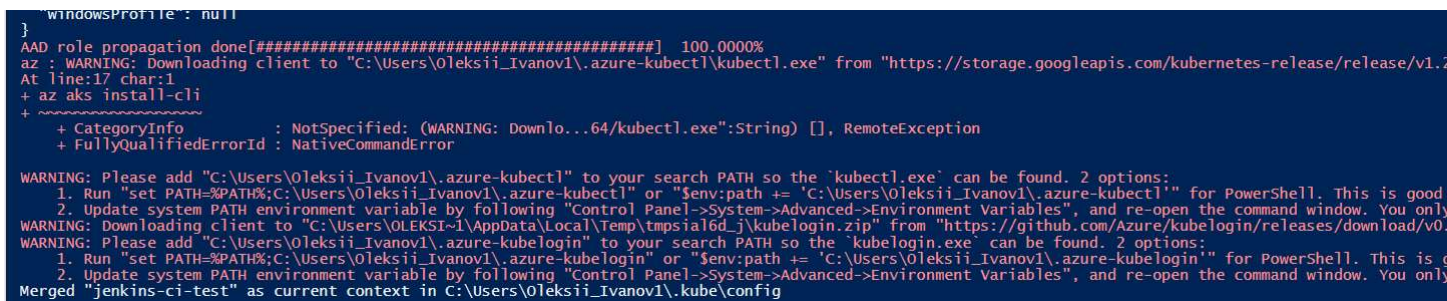
#creating AKS cluster
az aks create --resource-group $AzureResourceGroupName --name $AzureClusterName --node-count 1 --enable-addons monitoring --generate-ssh-keys

#Installation of kubectl
az aks install-cli

#Configure kubectl to work with newly created cluster
az aks get-credentials --resource-group $AzureResourceGroupName --name $AzureClusterName --overwrite-existing
```

This code section is responsible for Azure Kubernetes Service installation. According to best practices and Azure recommendation we will enable monitoring, then we will create cluster with one node below the param *--node-count* responsible for this. Also, cluster should have name, and again for automation and integration purpose we are using variable *\$AzureClusterName*.

After execution of script next warning appeared figure 3.2



```
WindowsProfile : null
}
AAD role propagation done[#####] 100.0000%
az : WARNING: Downloading client to "C:\Users\Oleksii_Ivanov1\.azure-kubectl\kubectl.exe" from "https://storage.googleapis.com/kubernetes-release/release/v1.20.0/bin/windows/amd64/kubectl.exe"
At line:17 char:1
+ az aks install-cli
+ ~~~~~
+ CategoryInfo          : NotSpecified: (WARNING: Downlo...64/kubectl.exe:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

WARNING: Please add "C:\Users\Oleksii_Ivanov1\.azure-kubectl" to your search PATH so the 'kubectl.exe' can be found. 2 options:
1. Run "set PATH=%PATH%;C:\Users\Oleksii_Ivanov1\.azure-kubectl" or "setx path += 'C:\Users\Oleksii_Ivanov1\.azure-kubectl'" for PowerShell. This is good.
2. Update system PATH environment variable by following "Control Panel->System->Advanced->Environment Variables", and re-open the command window. You only need to do this once.
WARNING: Downloading client to "C:\Users\OLEKSI~1\AppData\Local\Temp\tmp5a16d_j\kubelogin.zip" from "https://github.com/Azure/kubelogin/releases/download/v0.0.15/kubelogin.zip"
WARNING: Please add "C:\Users\Oleksii_Ivanov1\.azure-kubelogin" to your search PATH so the 'kubelogin.exe' can be found. 2 options:
1. Run "set PATH=%PATH%;C:\Users\Oleksii_Ivanov1\.azure-kubelogin" or "setx path += 'C:\Users\Oleksii_Ivanov1\.azure-kubelogin'" for PowerShell. This is good.
2. Update system PATH environment variable by following "Control Panel->System->Advanced->Environment Variables", and re-open the command window. You only need to do this once.
Merged "jenkins-ci-test" as current context in C:\Users\Oleksii_Ivanov1\.kube\config
```

Figure 3.2 Warning after first run

From the first perspective it is only warning. However, it has significant impact of the script execution and helm charts implementation. Without this values in PATH next part of script will not work as expected.

For the fix, this next function was introduced:

```
#Function for PATH modification
function Set-PathVariable {
    param (
        [string] $AddPath,
        [string] $RemovePath
    )
    $regexPaths = @()
    if ($PSBoundParameters.Keys -contains 'AddPath'){
        $regexPaths += [regex]::Escape($AddPath)
    }
    if ($PSBoundParameters.Keys -contains 'RemovePath'){
        $regexPaths += [regex]::Escape($RemovePath)
    }
}
```



```

$arrPath = $env:Path -split ';'
foreach ($path in $regexPaths) {
    $arrPath = $arrPath | where-object {$_. -notMatch "$path\?"}
}
$env:Path = ($arrPath + $addPath) -join ';'
}

```

This function will allow modification of PATH value and add new one if the not exist there. The call for function will have next order:

```

#Section will contain necessary patch updates.
#Adding patches

$addPathKubect1=$env:HOMEDRIVE+$env:HOMEPATH+'.azure-kubect1'
$addPathKubeLogin=$env:HOMEDRIVE+$env:HOMEPATH+'.azure-kubeLogin'

Set-PathVariable -AddPath $addPathKubect1;
Set-PathVariable -AddPath $addPathKubeLogin;

```

For automation and integration, we have used system variables to HomeDrive and HomePath. After script restart and cleanup, we can see created resources in Azure portal figure 3.3

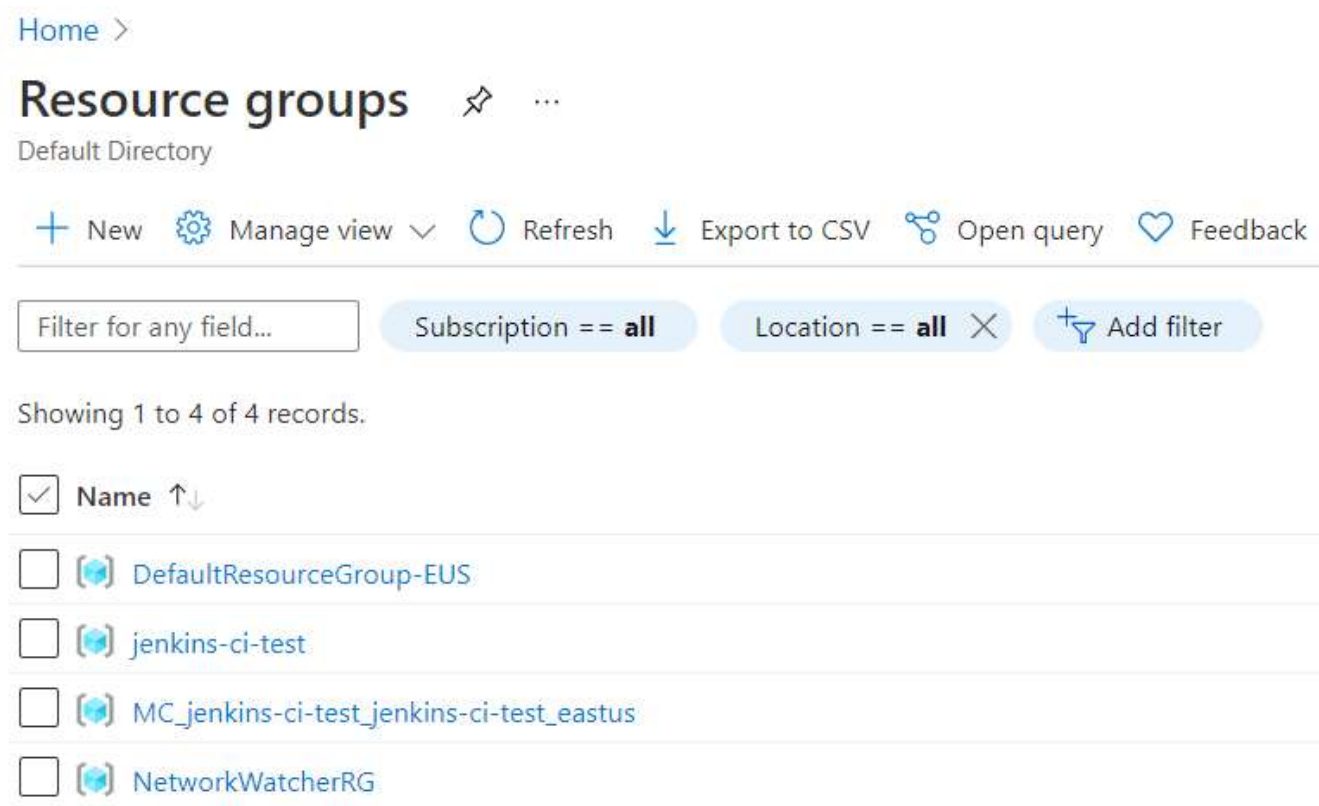


Figure 3.3 Created resource in Azure

According to documentation two resource group were created for AKS they are Jenkins-ci-test and MC_jenkins-ci-test_jenkins-ci-test_eastus. Two others responsible for monitoring. Let's connect to the created cluster and check if it's operable.

```
PS C:\WINDOWS\system32> kubectl cluster-info
Kubernetes master is running at https://jenkins-ci-jenkins-ci-test-8903b4-
f23210f8.hcp.eastus.azmk8s.io:443
healthmodel-replicaset-service is running at https://jenkins-ci-jenkins-ci-test-8903b4-
f23210f8.hcp.eastus.azmk8s.io:443/api/v1/namespaces/kube-system/services/healthmodel-
replicaset-service/proxy
CoreDNS is running at https://jenkins-ci-jenkins-ci-test-8903b4-
f23210f8.hcp.eastus.azmk8s.io:443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://jenkins-ci-jenkins-ci-test-8903b4-
f23210f8.hcp.eastus.azmk8s.io:443/api/v1/namespaces/kube-system/services/https:metrics-
server:/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\WINDOWS\system32>
```

Cluster up and running.

3.1.2 Kubernetes on GCP

For Kubernetes installation in GCP we should have installed GCP SDK and PowerShell or bash. Also, we should keep in mind that some resources and they activation in GCP could be done only manually. Depending on our choices we should install appropriate component in addition to SDK. We should login first.

```
#GCP clean login information
gcloud auth revoke --all
#Login into GCP
gcloud auth login
```

Like in previous section we cleaned up before login. And then forced user to login. The result of this code is shown on figure 3.4

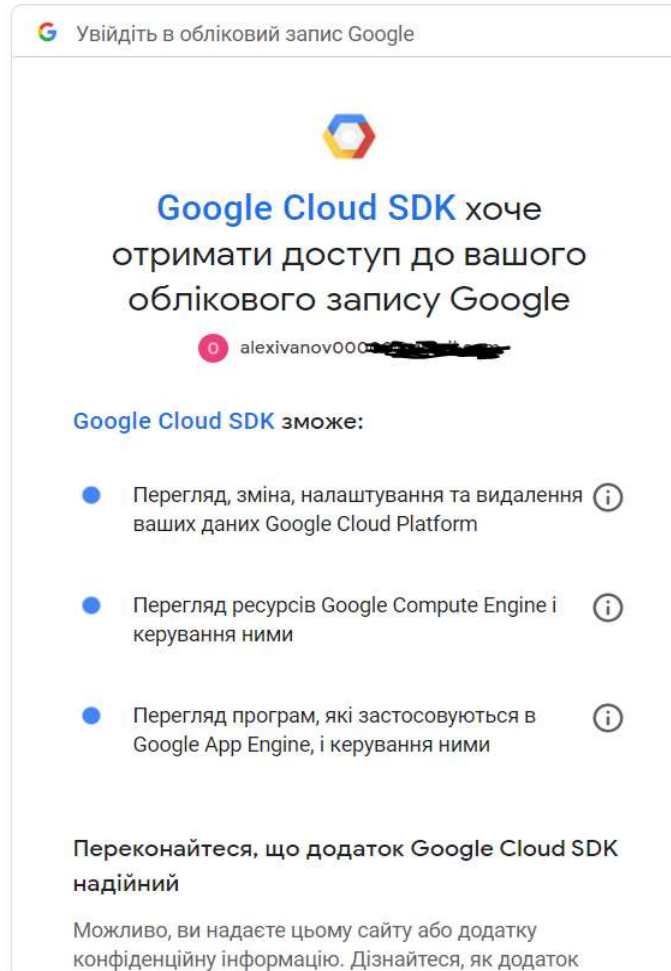


Figure 3.4 Login prompt for GCP

To proceed with working in clean GCP account we should create *project*. Google Cloud projects form the basis for creating, enabling, and using all Google Cloud services including managing APIs, enabling billing, adding and removing collaborators, and managing permissions for Google Cloud resources.[11]

For creating project, we should provide project id and project name. For this purpose we will use variables. Also, to make clear context we will set newly created project as a default for next operations. The code for this activity below.

#to operate in GCP Project should be created.

```
$GCPProjectID='jenkins-into-gke2'  
$GCPProjectName='Jenkins-ci'
```

```
gcloud projects create $GCPProjectID --name=$GCPProjectName --set-as-default
```

After executing the project will be created in GCP console figure 3.5

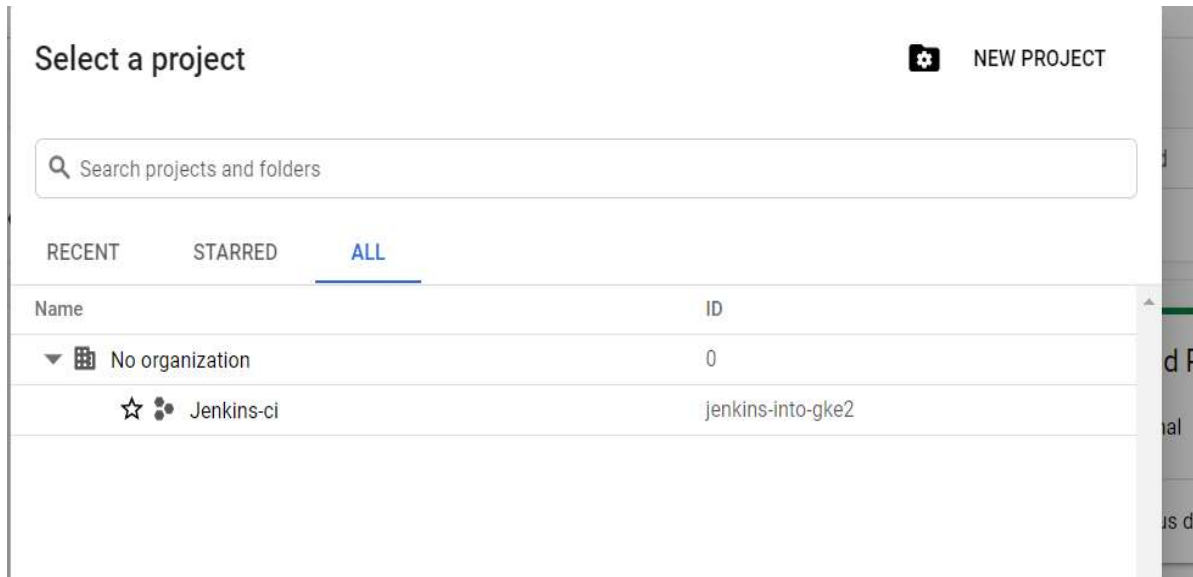


Figure 3.5 Project list in GCP cloud

Next step in organization infrastructure in GCP is choose the cluster type. We should review and pick up the most suitable Kubernetes cluster for us. Based on documentation One autopilot cluster or zonal cluster per billing account is free. [12] Autopilot is a new *mode of operation* in Google Kubernetes Engine (GKE) that is designed to reduce the operational cost of managing clusters, optimize your clusters for production, and yield higher workload availability. The *mode of operation* refers to the level of flexibility, responsibility, and control that you have over your cluster. In addition to the benefits of a fully managed control plane and node automations, GKE offers two modes of operation:

- **Autopilot:** GKE provisions and manages the cluster's underlying infrastructure, including nodes and node pools, giving you an optimized cluster with a hands-off experience.
- **Standard:** You manage the cluster's underlying infrastructure, giving you node configuration flexibility.

With Autopilot, you no longer have to monitor the health of your nodes or calculate the amount of compute capacity that your workloads require. Autopilot supports most

Kubernetes APIs, tools, and its rich ecosystem. You stay within GKE without having to interact with the Compute Engine APIs, CLIs, or UI, as the nodes are not accessible through Compute Engine, like they are in Standard mode. You pay only for the CPU, memory, and storage that your Pods request while they are running.[13]

Based on this information our choice is the autopilot cluster. Let's use information about cluster creation from SDK. From the very beginning we faced with limitation In GCP Kubernetes Engine API can be activated only in manual way. So, user should navigate to console and enabled API figure 3.6

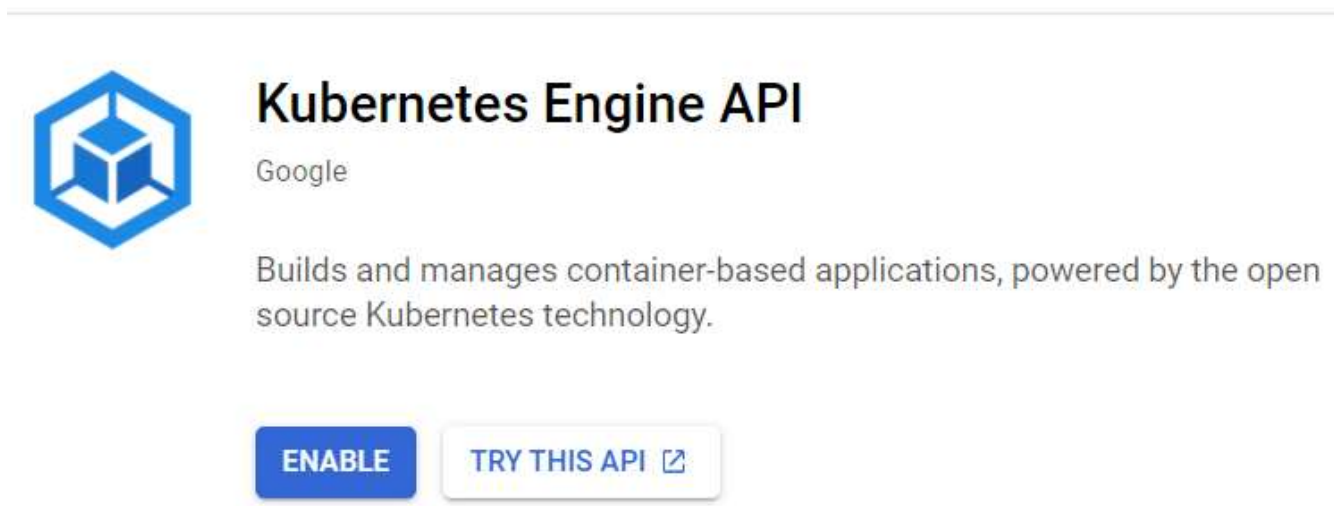


Figure 3.6 API activation at GCP

After activation we should additionally install necessary component for GCP SDK. In our case it is beta [14]. Creation for cluster will be done by next command

```
#Create autopilot cluster in us zones
gcloud container clusters create-auto $GCPclusterName --project $GCPProjectID --region "us-central1"
```

After cluster creation we should connect to it. The connection information will be grab in the command

```
#Get connection info for cluster
gcloud container clusters get-credentials $GCPclusterName --region us-central1 --project $GCPProjectID
```

The result of cluster installation on figure 3.7

```

Created [https://container.googleapis.com/v
To inspect the contents of your cluster, go
kubeconfig entry generated for autopilot-cl
NAME: autopilot-cluster-1
LOCATION: us-central1
MASTER_VERSION: 1.19.9-gke.1400
MASTER_IP: 34.72.168.131
MACHINE_TYPE: e2-medium
NODE_VERSION: 1.19.9-gke.1400
NUM_NODES: 3
STATUS: RUNNING

```

Figure 3.7 Cluster status in GCP

For the test purpose let's install Jenkins into the newly created cluster. We will use the default version and helm to make sure that cluster scaling allocates necessary resources. Code for test installation below:

```

#installing Jenkins into cluster

#create new name space for Jenkins
kubectl create namespace $GCPKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update

#run installation command
helm install jenkins-test jenkins/jenkins --namespace $GCPKubernetesNameSpace

```

However, after 30 minutes the deployment failed with *CrashLoopBackoff*, the output on figure 3.8

```

PS C:\Users\Oleksii_Ivanov1> kubectl get pods -n jenkins-ci-test
NAME          READY   STATUS              RESTARTS   AGE
jenkins-test-0 1/2     CrashLoopBackOff    7          39m

```

Figure 3.8 Failed deployment into autopilot cluster

Fast investigation uncovered problem with performance, the allocated resource in free tier is not enough for serving Jenkins itself. The output screen presented on figure 3.9

```

Normal      Pulled          39m          kubelet      Successfully pulled image "jenkins/jenkins:2.277.4-jdk11" in 10.675302464s
Normal      Created         39m          kubelet      Created container init
Normal      Started         39m          kubelet      Started container init
Normal      Pulled          31m          kubelet      Successfully pulled image "jenkins/jenkins:2.277.4-jdk11" in 82.006302ms
Normal      Pulling         31m          kubelet      Pulling image "kiwigrid/k8s-sidecar:0.1.275"
Normal      Pulled          31m          kubelet      Successfully pulled image "kiwigrid/k8s-sidecar:0.1.275" in 3.908816831s
Normal      Created         31m          kubelet      Created container config-reload
Normal      Started         31m          kubelet      Started container config-reload
Warning    Unhealthy       30m (x3 over 31m) kubelet      Startup probe failed: Get "http://10.8.0.194:8080/login": dial tcp 10.8.0.194:8080: connect: connection refused
Warning    Unhealthy       30m          kubelet      Startup probe failed: Get "http://10.8.0.194:8080/login": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
Normal      Pulling         28m (x2 over 31m) kubelet      Pulling image "jenkins/jenkins:2.277.4-jdk11"
Normal      Pulled          28m          kubelet      Successfully pulled image "jenkins/jenkins:2.277.4-jdk11" in 96.862355ms
Normal      Created         28m (x2 over 31m) kubelet      Created container jenkins
Warning    Unhealthy       19m (x34 over 30m) kubelet      Startup probe failed: HTTP probe failed with statuscode: 503
Normal      Started         14m (x7 over 31m) kubelet      Started container jenkins
Warning    Backoff         4m24s (x43 over 26m) kubelet      Back-off restarting failed container

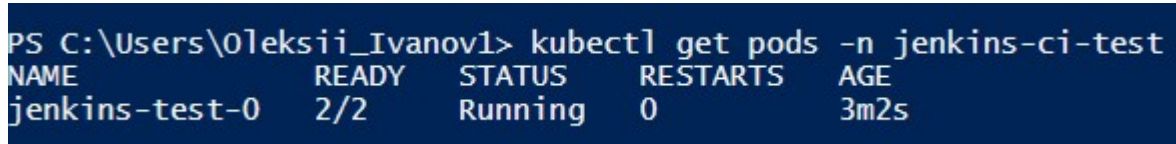
```

Figure 3.9 Error output from failed container.

Based on this information our new decision is regional cluster. It's also free per one billing account but will need additional input for resource management. Let's update cluster creation command.

```
#Create zonal cluster in us zones
gcloud container clusters create $GCPclusterName --project $GCPProjectID --region "us-central1" --num-nodes 1
```

After cluster creation, Jenkins installation was run again. The result better than on autopilot cluster. Jenkins managed to install and start in less than 4 minutes figure 3.10

A terminal window with a dark blue background and white text. The command 'kubectl get pods -n jenkins-ci-test' has been executed. The output is a table with five columns: NAME, READY, STATUS, RESTARTS, and AGE. There is one row of data for 'jenkins-test-0' which is in a 'Running' state.

NAME	READY	STATUS	RESTARTS	AGE
jenkins-test-0	2/2	Running	0	3m2s

Figure 3.10 Test Jenkins on zonal cluster in GCP

3.1.3 Kubernetes on AWS

Let's proceed with creation Kubernetes in EKS. According to documentation we should have created a lot of resources before starting EKS such as:

1. An existing VPC and a dedicated security group that meet the requirements for an Amazon EKS cluster.
2. An existing Amazon EKS cluster IAM role.
3. Installed aws cli.

Investigation of documentations bring us next result. Before starting, we need the aws access key and secret key for configuration. For humans we use username and password for authentication. But to authenticate any program we use an access key and secret key.

To get those things we have to go to AWS account from AWS Web Console. Then go to "IAM" service click on "Users" then click on "Add User" and create one user.

Then click on "Programmatic access" and if you read the description of this access you can see, it's giving us "access key" and "secret key" figure 3.11.

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name* test-user-master-work

+ Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type* ☒ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

Figure 3.11 Programmatic user creation

As you may see on figure 3.11, we should pass 5 stages and only after that we can use that user. It is much more difficult in comparison to Azure or GCP. Then we should configure our user. To do it we should execute command *aws configure* and provide keys generated in previous step and few other params such as default region and output format figure 3.12

```
Loading personal and system profiles took 624ms.
PS C:\Users\Oleksii_Ivanov1> aws configure
AWS Access Key ID [*****DQTC]: AKIAI7B8WU...XESWUOT
AWS Secret Access Key [*****LqWs]: HP1VNZtxWjUPfLC...RiaKzQQ8U6b+d
Default region name [eu-west-1]: us-west-2
Default output format [json]:
PS C:\Users\Oleksii_Ivanov1>
```

Figure 3.12 AWS profile configuration.

Next step is creating an Amazon VPC with public and private subnets that meets Amazon EKS requirements [21]. To Do so we should use documentation. And commands below

```
aws cloudformation create-stack \
  --region us-west-2 \
  --stack-name my-eks-vpc-stack \
```


`--template-url` <https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml>

After execution we will get created stack in AWS portal figure 3.13

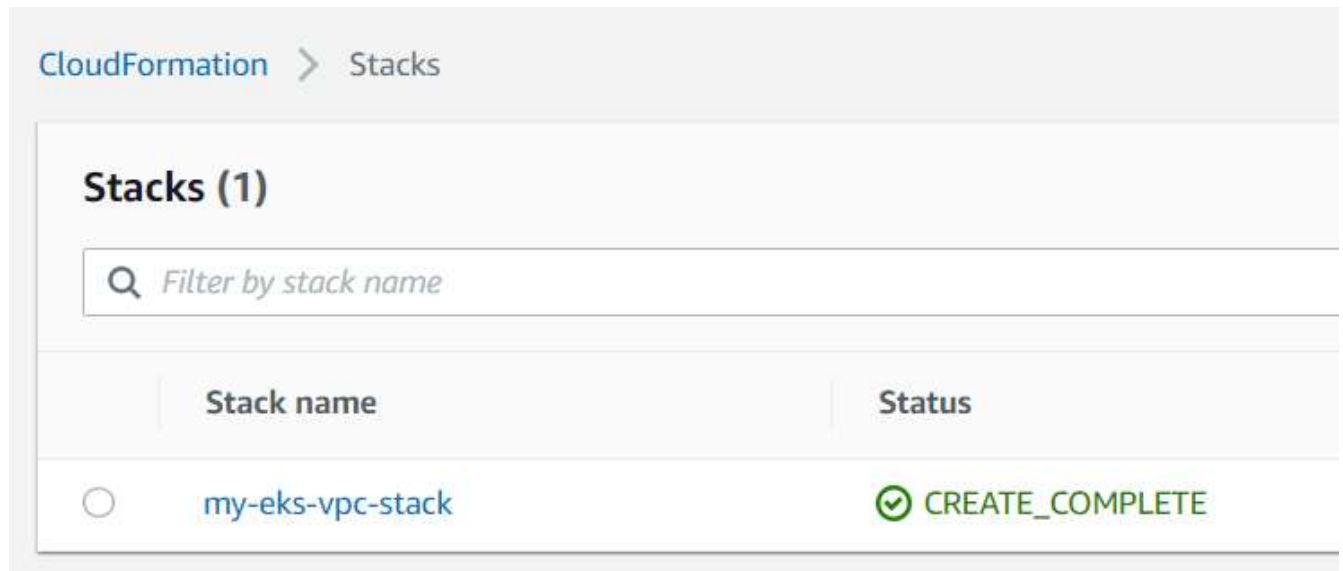


Figure 3.13 VPC stack in AWS

Next step is creating a cluster IAM role and attach the required Amazon EKS IAM managed policy to it. To do is we should create json file with next context.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The we should create roe by next command

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document file://cluster-role-trust-policy.json
```

Attach the required Amazon EKS managed IAM policy to the role.

```
aws iam attach-role-policy --policy-arn
arn:aws:iam::aws:policy/AmazonEKSClusterPolicy --role-name
myAmazonEKSClusterRole
```

The output of execution presented on figure 3.14



```
C:\Users\Oleksii_Ivanov1> aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document file://cluster-role-trust-policy.json
{
  "Role": {
    "Path": "/",
    "RoleName": "myAmazonEKSClusterRole",
    "RoleId": "AROAWBLRHLLGCULG4FRLY",
    "Arn": "arn:aws:iam::415238675148:role/myAmazonEKSClusterRole",
    "CreateDate": "2021-06-02T19:17:57+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "eks.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}

C:\Users\Oleksii_Ivanov1> aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy --role-name myAmazonEKSClusterRole
C:\Users\Oleksii_Ivanov1>
```

Figure 3.14 Role creation and attachments

Next step is creating cluster itself. To do so we should execute next code:

```
aws eks create-cluster --region us-west-2 --name Jenkins-cluster --kubernetes-version
1.20 --role-arn arn:aws:iam::415238675148:role/myAmazonEKSClusterRole --
resources-vpc-config subnetIds=subnet-0813b235fa2398ef1,subnet-
09eb728945497b023,securityGroupIds=sg-00b30ed6f8ba0e3c9
```

The output of creation is next:

```
"cluster": {
```

```
"name": "Jenkins-cluster",
"arn": "arn:aws:eks:us-west-2:415238675148:cluster/Jenkins-cluster",
"createdAt": "2021-06-02T22:31:26.046000+03:00",
"version": "1.20",
"roleArn": "arn:aws:iam::415238675148:role/myAmazonEKSClusterRole",
"resourcesVpcConfig": {
  "subnetIds": [
    "subnet-0813b235fa2398ef1",
    "subnet-09eb728945497b023"
  ],
  "securityGroupIds": [
    "sg-00b30ed6f8ba0e3c9"
  ],
  "vpcId": "vpc-0ee001417fc2a8224",
  "endpointPublicAccess": true,
  "endpointPrivateAccess": false,
  "publicAccessCidrs": [
    "0.0.0.0/0"
  ]
},
"kubernetesNetworkConfig": {
  "serviceIpv4Cidr": "10.100.0.0/16"
},
"logging": {
  "clusterLogging": [
    {
      "types": [
        "api",
        "audit",
```

```

        "authenticator",
        "controllerManager",
        "scheduler"
    ],
    "enabled": false
}
]
},
"status": "CREATING",
"certificateAuthority": {},
"platformVersion": "eks.1",
"tags": {}
}
}

```

Cluster created and operable figure 3.15

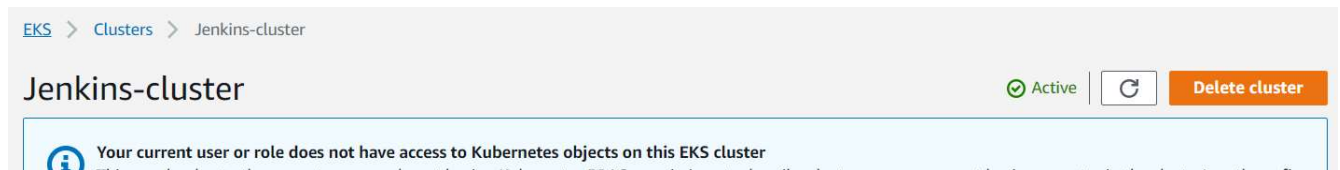


Figure 3.15 Created cluster in AWS

However proposed way is hard for automation, contains a lot of manual work and resources created. In this case, we should change the strategy to something more user-friendly. AWS provided `eksctl` A command line tool for working with EKS clusters that automates many individual tasks. Let's use it. The pre-requirements are: `chocolatey`, `kubectrl`, `eksctl`, `helm`.

For cluster creation we should use next command

```

eksctl create cluster --version=1.19 --name=jenkins-cluster --node-private-networking --
alb-ingress-access --region=us-west-2 --asg-access --without-nodegroup

```

The output:

```

PS C:\Users\Oleksii_Ivanov1> D:\User Files\NaukMA\Diploma\git\ci-instalation\Instalation_into_AWS.ps1
2021-06-02 23:07:05 eksctl version 0.52.0
2021-06-02 23:07:05 using region us-west-2
2021-06-02 23:07:06 setting availability zones to [us-west-2a us-west-2d us-west-2c]
2021-06-02 23:07:06 subnets for us-west-2a - public:192.168.0.0/19 private:192.168.96.0/19
2021-06-02 23:07:06 subnets for us-west-2d - public:192.168.32.0/19 private:192.168.128.0/19
2021-06-02 23:07:06 subnets for us-west-2c - public:192.168.64.0/19 private:192.168.160.0/19
2021-06-02 23:07:06 using Kubernetes version 1.19
2021-06-02 23:07:06 creating EKS cluster "jenkins-cluster" in "us-west-2" region with
2021-06-02 23:07:06 if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --
region=us-west-2 --cluster=jenkins-cluster'
2021-06-02 23:07:06 CloudWatch logging will not be enabled for cluster "jenkins-cluster" in "us-west-2"
2021-06-02 23:07:06 you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-
TYPES-HERE (e.g. all)} --region=us-west-2 --cluster=jenkins-cluster'
2021-06-02 23:07:06 Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for
cluster "jenkins-cluster" in "us-west-2"
2021-06-02 23:07:06 2 sequential tasks: { create cluster control plane "jenkins-cluster", 2 sequential sub-tasks: { wait for
control plane to become ready, 1 task: { create addons } } }
2021-06-02 23:07:06 building cluster stack "eksctl-jenkins-cluster-cluster"
2021-06-02 23:07:08 deploying stack "eksctl-jenkins-cluster-cluster"
2021-06-02 23:07:38 waiting for CloudFormation stack "eksctl-jenkins-cluster-cluster"
cluster-cluster"
2021-06-02 23:20:24 waiting for CloudFormation stack "eksctl-jenkins-cluster-cluster"
2021-06-02 23:20:30 waiting for the control plane availability...
2021-06-02 23:20:30 saved kubeconfig as "C:\\Users\\Oleksii_Ivanov1\\.kube/config"
2021-06-02 23:20:30 no tasks
2021-06-02 23:20:30 all EKS cluster resources for "jenkins-cluster" have been created
2021-06-02 23:20:36 kubectl command should work with "C:\\Users\\Oleksii_Ivanov1\\.kube/config", try 'kubectl get nodes'
2021-06-02 23:20:36 EKS cluster "jenkins-cluster" in "us-west-2" region is ready

```

After that we should install nodegroup

```

eksctl create nodegroup --cluster=jenkins-cluster --region=us-west-2 --name=jenkins-server-
ng --managed --nodes=2 --node-labels="lifecycle=OnDemand,intent=jenkins-server"

```

The output

```

PS C:\Users\Oleksii_Ivanov1> eksctl create nodegroup --cluster=jenkins-cluster --region=us-west-2 --name=jenkins-server-
ng --managed --nodes=2 --node-labels="lifecycle=OnDemand,intent=jenkins-server"
2021-06-02 23:25:33 eksctl version 0.52.0
2021-06-02 23:25:33 using region us-west-2
2021-06-02 23:25:34 will use version 1.19 for new nodegroup(s) based on control plane version

```

```

2021-06-02 23:25:41 1 nodegroup (jenkins-server-ng) was included (based on the include/exclude rules)
2021-06-02 23:25:41 will create a CloudFormation stack for each of 1 managed nodegroups in cluster "jenkins-cluster"
2021-06-02 23:25:41 2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "jenkins-
server-ng" } } }
2021-06-02 23:25:41 checking cluster stack for missing resources
2021-06-02 23:25:42 cluster stack has all required resources
2021-06-02 23:25:42 building managed nodegroup stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:25:42 deploying stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:25:42 waiting for CloudFormation stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:25:59 waiting for CloudFormation stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:29:42 waiting for CloudFormation stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:29:59 waiting for CloudFormation stack "eksctl-jenkins-cluster-nodegroup-jenkins-server-ng"
2021-06-02 23:30:00 no tasks
2021-06-02 23:30:00 created 0 nodegroup(s) in cluster "jenkins-cluster"
2021-06-02 23:30:01 nodegroup "jenkins-server-ng" has 2 node(s)
2021-06-02 23:30:01 node "ip-192-168-2-233.us-west-2.compute.internal" is ready
2021-06-02 23:30:01 node "ip-192-168-94-190.us-west-2.compute.internal" is ready
2021-06-02 23:30:01 waiting for at least 2 node(s) to become ready in "jenkins-server-ng"
2021-06-02 23:30:01 nodegroup "jenkins-server-ng" has 2 node(s)
2021-06-02 23:30:01 node "ip-192-168-2-233.us-west-2.compute.internal" is ready
2021-06-02 23:30:01 node "ip-192-168-94-190.us-west-2.compute.internal" is ready
2021-06-02 23:30:01 created 1 managed nodegroup(s) in cluster "jenkins-cluster"
2021-06-02 23:30:03 checking security group configuration for all nodegroups
2021-06-02 23:30:03 all nodegroups have up-to-date configuration

```

After the installation we should validate cluster configuration. The result on figure 3.16

```

2021-06-02 23:30:01 [B,] node ip-192-168-94-190.us-west-2.compute.internal is ready
2021-06-02 23:30:01 [B,] created 1 managed nodegroup(s) in cluster "jenkins-cluster"
2021-06-02 23:30:03 [B,] checking security group configuration for all nodegroups
2021-06-02 23:30:03 [B,] all nodegroups have up-to-date configuration

PS C:\Users\Oleksii_Ivanov1>
PS C:\Users\Oleksii_Ivanov1>
PS C:\Users\Oleksii_Ivanov1> kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
ip-192-168-2-233.us-west-2.compute.internal Ready    <none>    5m59s   v1.19.6-eks-49a6c0
ip-192-168-94-190.us-west-2.compute.internal Ready    <none>    5m58s   v1.19.6-eks-49a6c0

PS C:\Users\Oleksii_Ivanov1> |

```

Figure 3.16 AWS cluster verification

To check functionality let's install Jenkins

```

$AWSKubernetesNamespace='jenkins-ci-test'
#installing jenkins into cluster

```

```
#create new name space for Jenkins
kubectl create namespace $AWSKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update

#run instalation command
helm install jenkins-test jenkins/jenkins --namespace $AWSKubernetesNameSpace
```

The result of installation presented on figure 3.17

```
NOTE: Consider using a custom image with pre-installed plugins

PS C:\Users\Oleksii_Ivanov1> kubectl get pods -A
NAMESPACE          NAME                                READY   STATUS    RESTARTS   AGE
jenkins-ci-test     jenkins-test-0                    0/2     Init:0/1   0           72s
kube-system         aws-node-hxcmc                   1/1     Running   0           16m
kube-system         aws-node-pgkcx                   1/1     Running   0           16m
kube-system         coredns-6548845887-m5vnb        1/1     Running   0           28m
kube-system         coredns-6548845887-njphm        1/1     Running   0           28m
kube-system         kube-proxy-f5nml                 1/1     Running   0           16m
kube-system         kube-proxy-fgmxn                 1/1     Running   0           16m

PS C:\Users\Oleksii_Ivanov1>
```

Figure 3.17 Kubernetes pods in AWS

As we can see cluster works as expected. And we are ready to start deploying Jenkins.

3.2 Jenkins installation into Kubernetes

Based on previous section we have operable platform and now time to install Jenkins into Kubernetes cluster. To achieve the best performance and productivity we will use latest helm charts, that officially distributed. As we used the same platform (Kubernetes) across all cloud the helm chart could be simply reused in every scripts with small modification. For example, let's start from installation into Kubernetes, based in Azure cloud. The initial script will be next:

```
#installing Jenkins into cluster

#create new name space for Jenkins
kubectl create namespace $AzureKubernetesNameSpace
```



```
#adding latest repository:
helm repo add jenkins https://charts.jenkins.io
helm repo update

#run instalation command
helm install jenkins-test jenkins/jenkins --namespace $AzureKubernetesNameSpace
```

First of all, we should create a namespace in the cluster, for better resources organization and management. Then we should include official repository with Jenkins's helm chart. That repository is managed and supported by Jenkins team and huge community. Next step is updating repository to latest stable version. And final step is installation Jenkins via helm chart call. The output provided on figure 3.18

```
namespace/jenkins-ci-test created
"jenkins" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "otemocharts" chart repository
...Successfully got an update from the "jenkins" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. Happy Helming!
NAME: jenkins-test
LAST DEPLOYED: Wed Jun  2 21:13:27 2021
NAMESPACE: jenkins-ci-test
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:
   kubectl exec --namespace jenkins-ci-test -it svc/jenkins-test -c jenkins -- /bin/cat /run/secrets/chart-admin-password && echo
2. Get the Jenkins URL to visit by running these commands in the same shell:
   echo http://127.0.0.1:8080
   kubectl --namespace jenkins-ci-test port-forward svc/jenkins-test 8080:8080
3. Login with the password from step 1 and the username: admin
4. Configure security realm and authorization strategy
5. Use Jenkins Configuration as Code by specifying configScripts in your values.yaml file, see documentation: http://configuration-as-code/jenkins/master/demos
For more information on running Jenkins on Kubernetes, visit:
https://cloud.google.com/solutions/jenkins-on-container-engine
For more information about Jenkins Configuration as Code, visit:
https://jenkins.io/projects/jcasc/
NOTE: Consider using a custom image with pre-installed plugins
```

Figure 3.18 Jenkins installation output

As we can see on the output the namespace was created. All updates were downloaded from the repository, and installation was launched. After few minutes we can check the status of deployment using *kubectl* figure 3.19.

```
PS C:\Users\Oleksii_Ivanov1> kubectl get pods -n jenkins-ci-test
NAME                READY   STATUS    RESTARTS   AGE
jenkins-test-0      2/2     Running   0           5m8s
```

Figure 3.19 The deployment status check.

From the deployment status, we can see that Jenkins ready in approximately 5 minutes. Last check we should be able to see the Jenkins Admin console to check the functionality of Jenkins itself. To achieve this we should use commands that provided on figure 3.13 First of all we should obtain Jenkins admin password (Good practice is change it immediately after installation.) *kubectl exec --namespace jenkins-ci-test -it svc/jenkins-test -c jenkins -- /bin/cat /run/secrets/chart-admin-password* Execution and password present on Figure 3.20

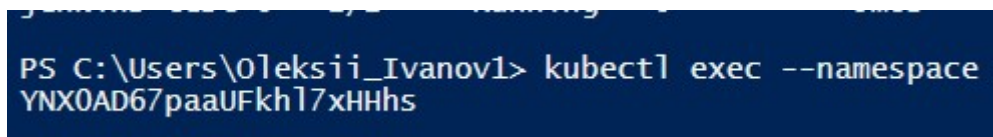
A terminal window with a dark blue background and white text. The prompt is 'PS C:\Users\Oleksii_Ivanov1>'. The command entered is 'kubectl exec --namespace jenkins-ci-test -it svc/jenkins-test -c jenkins -- /bin/cat /run/secrets/chart-admin-password'. The output of the command is 'YNX0AD67paaUFkh17xHHhs'.

Figure 3.20 Jenkins password

The password stored inside the container so only user with access to cluster could get it and this matches security practices. The next command will allow us to see the Jenkins admin panel on local host. We will lunch port forwarding from cluster in cloud to local machine *kubectl --namespace jenkins-ci-test port-forward svc/jenkins-test 8080:8080* after executing we should be able to se Jenkins on 127.0.0.1:8080 with possibility to login with admin password figure 3.21

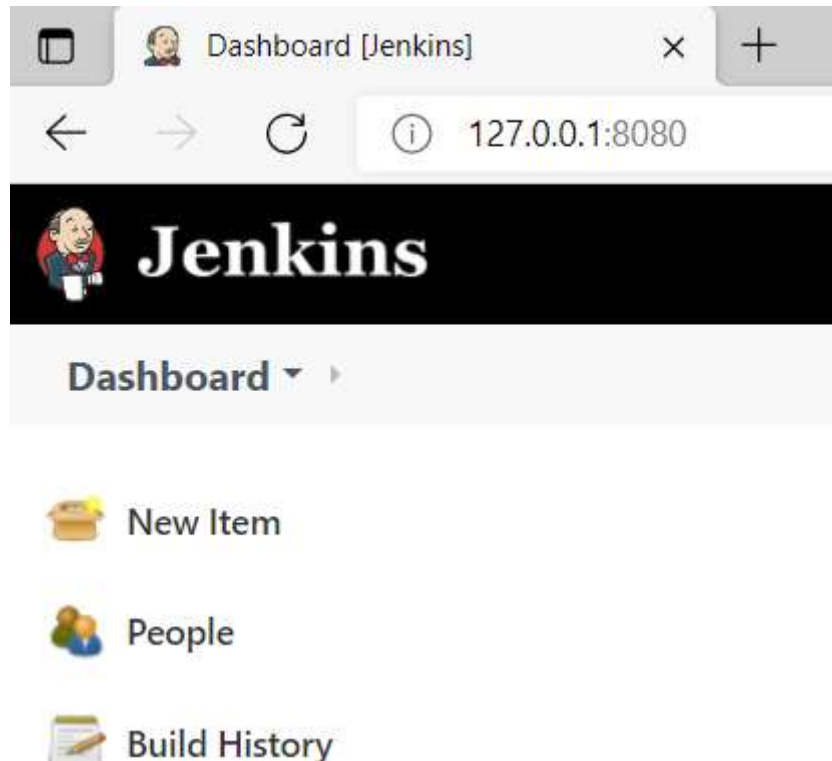


Figure 3.21 Jenkins home page on local host

To make it more usable for user and teams we will expose service to open world.

```
kubect1 expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer --  
namespace $AzureKubernetesNameSpace
```

After the execution Jenkins will be available on external public IP that allocated to cluster figure 3.22

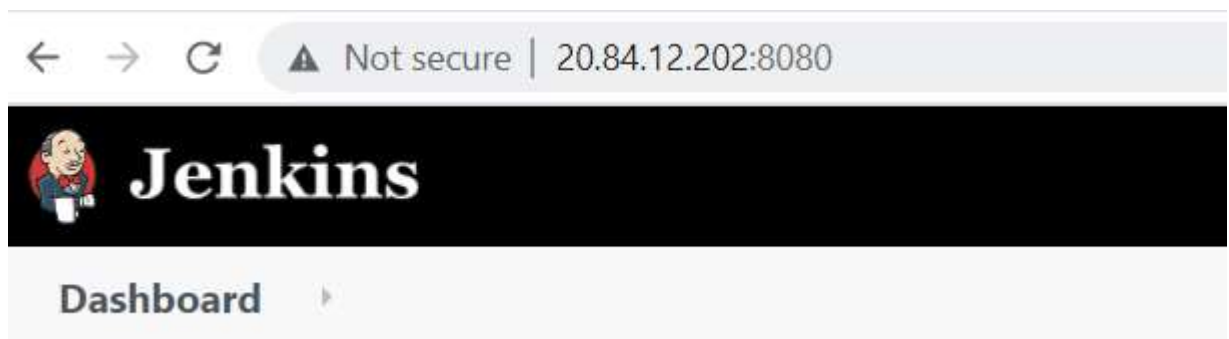


Figure 3.22 Jenkins with public IP

Based on project needs additional customization is possible for this installation. The main recommendation is to install backup plugin in addition to default plugins list.

3.3 Requirement's installation and validation

To make scripts automation successful. Necessary software should be installed. Let's start from the Azure cloud and configure PowerShell script to install az, kubectl, helm. To do so we will use chocolatey – it is package manager for windows. [22]

The code provided below.

```
#section will contain instalation for prerequired software helm cli for clouds, etc.
#Install the package manager for windows chocolately
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

#install kubectl for cluster communication
choco install kubernetes-cli
#Install helm for chart operating
choco install kubernetes-helm

#install az for communication with Azure
Install-Module -Name Az -Scope CurrentUser -Repository PSGallery -Force
```

For testing purpose, we will create clear VM in Azure. Let's download script and start execution. The output provided on figure 3.23.

```

Creating Chocolatey folders if they do not already exist.

WARNING: You can safely ignore errors related to missing log files when
  upgrading from a version of Chocolatey less than 0.9.9.
  'Batch file could not be found' is also safe to ignore.
  'The system cannot find the file specified' - also safe.
chocolatey.nupkg file not installed in lib.
  Attempting to locate it from bootstrapper.
PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...
WARNING: Not setting tab completion: Profile file does not exist at
'C:\Users\oleksii\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.
Chocolatey (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
  first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
Chocolatey v0.10.15
Installing the following packages:
kubernetes-cli
By installing you accept licenses for the packages.
Progress: Downloading kubernetes-cli 1.21.1... 100%

kubernetes-cli v1.21.1 [Approved]
kubernetes-cli package files install completed. Performing other installation steps.
The package kubernetes-cli wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): A

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows

```

Figure 3.23 Automation installation on “clear” VM

However, after the installation of az module we faced with error, provided on figure 3.24

```

'C:\Users\oleksii\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install
and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
All necessary software installed.

az : The term 'az' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the
spelling of the name, or if a path was included, verify that the path is correct and try again.
At D:\Installation_into_Azure.ps1:63 char:1
+ az login
+ ~~~
+ CategoryInfo          : ObjectNotFound: (az:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

```

Figure 3.24 Error after installation of az module

That error mean that module az were installed but it is not accessible in current PowerShell run. So we should separate modules installation from the main script and put it into additional file, that should be execute separately. During this procedure let's provide small update to minimize interaction with user with adding -y flag to chocolately calls.

To make test result more reliable we redeploy VM in Azure to het clear installation. And after downloading from git start installation. We already got waring from Windows system figure 3.25

```
PS C:\Users\oleksii> cd D:\msdp-main\msdp-main
PS D:\msdp-main\msdp-main> .\Requirements_installation_Windows.ps1

Security warning
Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your
computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning
message. Do you want to run D:\msdp-main\msdp-main\Requirements_installation_Windows.ps1?
[D] Do not run [R] Run once [S] Suspend [?] Help (default is "D"):
```

Figure 3.25 Security waring from Windows system

Next interruption is related to additional modules that should be added to PowerShell figure 3.26

```
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet
provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or
C:\Users\oleksii\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install
and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"):
```

Figure 3.25 Additional package installation for Windows system

After the confirmation, all necessary software were installed without any error. We should keep in mind, that PowerShell should be reopened to make az module available [23].

The final code version for Azure cloud provided below

```
#section will contain instalation for prerequired software helm cli for clouds, etc.
#Install the package manager for windows chocolately
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

#Install puthon3
choco install python3

#install kubectl for cluster communication
choco install kubernetes-cli -y
#Install helm for chart operating
choco install kubernetes-helm -y

#install az for communication with Azure
choco install azure-cli -y
```

As we have chosen the Kubernetes platform the tools set will have only few differences from cloud to cloud. For example, for AWS cloud we should install corresponding cli.

```
#Install aws cli and eksctl
choco install awscli -y
choco install eksctl -y
```

The same situation for GCP

```
#Install gcp sdk
choco install gcloudsdk -y
```

Let's test full set of cli for every cloud on clear Windows machine. Execution time approximately 5 minutes. Full logs outputs could be found in appendix A.

Next step is translation from PowerShell to shell, that could be executed on Linux machines. To do it we will create separate file *Requirements_instalation_Linux.sh*

After the launch on clear Linux machine (Ubuntu 20.04 were used). We can see that all necessary software were installed without any errors. Logs output below:

```
oleksii@linux:~$ helm version
version.BuildInfo{Version:"v3.6.0", GitCommit:"7f2df6467771a75f5646b7f12afb408590ed1755",
GitTreeState:"clean", GoVersion:"go1.16.3"}
oleksii@linux:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.1",
GitCommit:"5e58841cce77d4bc13713ad2b91fa0d961e69192", GitTreeState:"clean", BuildDate:"2021-05-13T02:40:46Z",
GoVersion:"go1.16.3", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify the right host or port?
oleksii@linux:~$ az --version
azure-cli          2.24.2
core               2.24.2
```



```

telemetry                  1.0.6
Python location '/opt/az/bin/python3'
Extensions directory '/home/oleksii.azure/cliextensions'
Python (Linux) 3.6.10 (default, Jun  2 2021, 06:55:51)
[GCC 9.3.0]
Legal docs and information: aka.ms/AzureCliLegal
Your CLI is up-to-date.

Please let us know how we are doing: https://aka.ms/azureclihats
and let us know if you're interested in trying out our newest features: https://aka.ms/CLIUXstudy
oleksii@linux:~$ aws --version
aws-cli/1.18.69 Python/3.8.5 Linux/5.4.0-1047-azure botocore/1.16.19
oleksii@linux:~$ eksctl version
0.52.0
oleksii@linux:~$ gcloud --version
Google Cloud SDK 343.0.0
alpha 2021.05.27
beta 2021.05.27
bq 2.0.69
core 2021.05.27
gsutil 4.62
minikube 1.20.0
skaffold 1.25.0
oleksii@linux:~$

```

Making this thing will allow us to launch any installation to any cloud by simple copying context of file. Let's do it for Azure and launch on fresh created machine. The result provided on figures 3.26-3.27.

```

oleksii@linux:~$ wget https://raw.githubusercontent.com/AlexIvan0v/msdp/main/Installation_into_Azure.sh
--2021-06-05 15:18:15-- https://raw.githubusercontent.com/AlexIvan0v/msdp/main/Installation_into_Azure.sh
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1563 (1.5K) [text/plain]
Saving to: 'Installation_into_Azure.sh'

Installation_into_Azure.sh      100%[=====>] 1.53K --.-KB/s  in 0s
2021-06-05 15:18:15 (13.3 MB/s) - 'Installation_into_Azure.sh' saved [1563/1563]

oleksii@linux:~$ bash Installation_into_Azure.sh
Installation_into_Azure.sh: line 1: #before: command not found
There are no active accounts.
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code F2ZEF345C to authenticate.
https://microsoft.com/devicelogin
F2ZEF345C[
{
  "cloudName": "AzureCloud",
  "homeTenantId": "27cb58f2-9de2-4cc4-aa23-7b48cbbc7989",
  "id": "8903b493-345c-4a6d-b2e3-a7eal0e3de27",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Pay-As-You-Go",

```

Figure 3.26 Download and launch script

```

Merged "jenkins-ci-test-1" as current context in /home/oleksii/.kube/config
namespace/jenkins-ci-test-1 created
"jenkins" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "jenkins" chart repository
Update Complete. ☺Happy Helming!☺
NAME: jenkins-test
LAST DEPLOYED: Sat Jun  5 15:23:00 2021
NAMESPACE: jenkins-ci-test-1
STATUS: deployed
REVISION: 1
NOTES:
1. Get your 'admin' user password by running:
   kubectl exec --namespace jenkins-ci-test-1 -it svc/jenkins-test -c jenkins -- /bin/cat /run/secrets/chart-admin-password && echo
2. Get the Jenkins URL to visit by running these commands in the same shell:
   echo http://127.0.0.1:8080
   kubectl --namespace jenkins-ci-test-1 port-forward svc/jenkins-test 8080:8080
3. Login with the password from step 1 and the username: admin
4. Configure security realm and authorization strategy
5. Use Jenkins Configuration as Code by specifying configScripts in your values.yaml file, see documentation: http://configuration-as-code-plugin/tree/master/demos
For more information on running Jenkins on Kubernetes, visit:
https://cloud.google.com/solutions/jenkins-on-container-engine
For more information about Jenkins Configuration as Code, visit:
https://jenkins.io/projects/jcasc/
NOTE: Consider using a custom image with pre-installed plugins
service/jenkins-out exposed
oleksii@linux:~$ https://microsoft.com/devicelogin

```

Figure 3.27 Finished execution of installation script

As we can see from figure 3.26, we have downloaded script from GitHub, and launch on prepared machine without any modification in this script. Next step is checking the cloud resources and make sure that we have Jenkins installed in newly created Kubernetes cluster. The result provided on figure 3.28.

Home > Resource groups > jenkins-ci-test-1 > jenkins-ci-test-1

jenkins-ci-test-1 | Services and ingresses

Kubernetes service

Search (Ctrl+/)

+ Add - Delete Refresh Show labels

Services Ingresses

Filter by service name: Filter by namespace:

<input type="checkbox"/>	Name	Namespace	Status	Type	Cluster IP	External IP	Ports	Age ↓
<input type="checkbox"/>	kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP	24 minutes
<input type="checkbox"/>	kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,53/TCP	24 minutes
<input type="checkbox"/>	healthmodel-replicaset-service	kube-system	Ok	ClusterIP	10.0.188.155		25227/TCP	24 minutes
<input type="checkbox"/>	metrics-server	kube-system	Ok	ClusterIP	10.0.34.136		443/TCP	24 minutes
<input type="checkbox"/>	jenkins-test	jenkins-ci-test-1	Ok	ClusterIP	10.0.227.129		8080/TCP	22 minutes
<input type="checkbox"/>	jenkins-test-agent	jenkins-ci-test-1	Ok	ClusterIP	10.0.194.140		50000/TCP	22 minutes
<input type="checkbox"/>	jenkins-out	jenkins-ci-test-1	Ok	LoadBalancer	10.0.186.14	20.81.91.198	8080:31626/TCP	22 minutes

Figure 3.28 Created cluster and installed Jenkins

The same configuration changes will be created for each cloud GCP and AWS. And placed in corresponding files.

The repository available via <https://github.com/AlexIvan0v/msdp> . The last stage is update README.md file. As we have faced with limitation from GCP and AWS that can not be automated we should add them into description.

SUMMARY

In this work we introduced and described the most important requirements that CI/CD system must meet in order to satisfy developers needs. To accomplish this task we explored existing approaches, reviewed best practices, main CI/CD principles and delivery workflow. Deep analysis was performed on top cloud providers (AWS GCP Azure) to uncover native CI/CD tools such as Azure DevOps, AWS CodeBuild CodeDeploy, CodeCommit and GCP Cloud Build. Also we have reviewed Jenkins and TeamCity as a CI/CD systems. Based on modern trends and business needs the platform for CI/CD system installation was chosen (Kubernetes).

In the next part of work was designed and created scripts for automation infrastructure provision for every cloud provider (AWS, GCP, Azure), and scripts for CI system installation on the created infrastructure. Each part of work was done in two scripts language PowerShell and shell. The test result was presented in the text and in the appendixes. Created scripts could be used as independent parts or could be included into some automation jobs for bigger solution.

Based on the development result we can tell that the most easiest solution was created for Azure than GCP and the hardest one for AWS. This could be explained by the numbers of parts that could not be automated such as API and Billing activation for GCP and IAM user creation in AWS.

The work results were published in public GitHub repository and they are open for everyone.

REFERENCES

1. “Continuous delivery: reliable software releases through build, test, and deployment automation” / Jez Humble, David Farley/RR Donnelley in Crawfordsville, Indiana./ August 2010
2. “Continuous Integration, Delivery, and Deployment” / by Sander Rossel / Packt Publishing / 2018
3. “Effective DevOps” / by Jennifer Davis, Ryn Daniels / Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472./ May 2018
- 4.
5. “DevOps: Continuous Delivery, Integration, and Deployment with DevOps” / by Sricharan Vadapalli / Packt Publishing / March 2018
6. <https://www.jenkins.io/>
7. <https://www.jetbrains.com/teamcity/>
8. <https://www.zfort.com/blog/Devops-as-a-Service-Cloud-Computing-AWS-GCP-Azure-AliBaba-DigitalOcean>
9. <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal>
10. <https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough>
11. <https://cloud.google.com/resource-manager/docs/creating-managing-projects>
12. <https://cloud.google.com/kubernetes-engine>
13. <https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview>
14. <https://cloud.google.com/sdk/docs/components>
15. <https://azure.microsoft.com/en-au/services/devops/>
16. <https://newsignature.com/articles/all-about-azure-artifacts/>
17. <https://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html>
18. <https://docs.aws.amazon.com/codebuild/latest/userguide/welcome.html>

19. <https://docs.aws.amazon.com/govcloud-us/latest/UserGuide/govcloud-codedeploy.html>
20. <https://cloud.google.com/build/docs/overview>
21. <https://docs.aws.amazon.com/eks/latest/userguide/getting-started-console.html>
22. <https://docs.chocolatey.org/en-us/choco/setup#non-administrative-install>
23. <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-powershell>

ABBREVIATIONS AND TERMS

CI (Continuous Integration) – practice of merging all developers' working copies to a shared mainline several times a day.

CD (Continuous delivery) – is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

CD (Continuous deployment) – is a practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

SDLC (software development lifecycle) - is a framework that development teams use to produce high-quality software in a systematic and cost-effective way.

APPENDIX

Appendix A

Log outputs for Windows machine for “Requirements_instalation_Windows.ps1” (Informative)

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\oleksii> d:

PS D:\> cd .\msdp-main\

PS D:\msdp-main> ls

Directory: D:\msdp-main

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a----	6/4/2021 3:35 PM	876	Instalation_into_AWS.ps1
-a----	6/4/2021 3:35 PM	2365	Instalation_into_Azure.ps1
-a----	6/4/2021 3:35 PM	1288	Instalation_into_GCP.ps1
-a----	6/4/2021 3:35 PM	66	README.md
-a----	6/4/2021 3:35 PM	781	Requirements_instalation_Windows.ps1

PS D:\msdp-main> .\Requirements_instalation_Windows.ps1

Security warning

Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning message. Do you want to run D:\msdp-main\Requirements_instalation_Windows.ps1?

[D] Do not run [R] Run once [S] Suspend [?] Help (default is "D"): r

Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)

Getting latest version of the Chocolatey package for download.

Not using proxy.

Getting Chocolatey from <https://community.chocolatey.org/api/v2/package/chocolatey/0.10.15>.

Downloading <https://community.chocolatey.org/api/v2/package/chocolatey/0.10.15> to

C:\Users\oleksii\AppData\Local\Temp\chocolatey\chocolInstall\chocolatey.zip

Not using proxy.

Extracting C:\Users\oleksii\AppData\Local\Temp\chocolatey\chocolInstall\chocolatey.zip to

C:\Users\oleksii\AppData\Local\Temp\chocolatey\chocolInstall

Installing Chocolatey on the local machine

Creating ChocolateyInstall as an environment variable (targeting 'Machine')

Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'

WARNING: It's very likely you will need to close and reopen your shell
before you can use choco.

Restricting write permissions to Administrators

We are setting up the Chocolatey package repository.

The packages themselves go to 'C:\ProgramData\chocolatey\lib'

(i.e. C:\ProgramData\chocolatey\lib\yourPackageName).

A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

Creating Chocolatey folders if they do not already exist.

WARNING: You can safely ignore errors related to missing log files when
upgrading from a version of Chocolatey less than 0.9.9.

'Batch file could not be found' is also safe to ignore.

'The system cannot find the file specified' - also safe.

chocolatey.nupkg file not installed in lib.

Attempting to locate it from bootstrapper.

PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...

WARNING: Not setting tab completion: Profile file does not exist at

'C:\Users\oleksii\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1'.

Chocolatey (choco.exe) is now ready.

You can call choco from anywhere, command line or powershell by typing choco.

Run choco /? for a list of functions.

You may need to shut down and restart powershell and/or consoles
first prior to using choco.

Ensuring Chocolatey commands are on the path

Ensuring chocolatey.nupkg is in the lib folder

Chocolatey v0.10.15

Installing the following packages:

python3

By installing you accept licenses for the packages.

Progress: Downloading vcredist2015 14.0.24215.20170201... 100%

Progress: Downloading vcredist140 14.29.30037... 100%

Progress: Downloading chocolatey-core.extension 1.3.5.1... 100%

Progress: Downloading KB3033929 1.0.5... 100%

Progress: Downloading chocolatey-windowsupdate.extension 1.0.4... 100%

Progress: Downloading KB3035131 1.0.3... 100%

Progress: Downloading KB2919355 1.0.20160915... 100%

Progress: Downloading KB2919442 1.0.20160915... 100%

Progress: Downloading KB2999226 1.0.20181019... 100%

Progress: Downloading python3 3.9.5... 100%

chocolatey-core.extension v1.3.5.1 [Approved]

chocolatey-core.extension package files install completed. Performing other installation steps.

Installed/updated chocolatey-core extensions.

The install of chocolatey-core.extension was successful.

Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-core'

chocolatey-windowsupdate.extension v1.0.4 [Approved]

chocolatey-windowsupdate.extension package files install completed. Performing other installation steps.

Installed/updated chocolatey-windowsupdate extensions.

The install of chocolatey-windowsupdate.extension was successful.

Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-windowsupdate'

KB3035131 v1.0.3 [Approved]

kb3035131 package files install completed. Performing other installation steps.

Skipping installation because update KB3035131 does not apply to this operating system (Microsoft Windows 10 Pro).

The install of kb3035131 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

KB3033929 v1.0.5 [Approved]

kb3033929 package files install completed. Performing other installation steps.

Skipping installation because update KB3033929 does not apply to this operating system (Microsoft Windows 10 Pro).

The install of kb3033929 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

KB2919442 v1.0.20160915 [Approved]

kb2919442 package files install completed. Performing other installation steps.

Skipping installation because this hotfix only applies to Windows 8.1 and Windows Server 2012 R2.

The install of kb2919442 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

KB2919355 v1.0.20160915 [Approved]

kb2919355 package files install completed. Performing other installation steps.

Skipping installation because this hotfix only applies to Windows 8.1 and Windows Server 2012 R2.

The install of kb2919355 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

KB2999226 v1.0.20181019 [Approved] - Possibly broken

kb2999226 package files install completed. Performing other installation steps.

Skipping installation because update KB2999226 does not apply to this operating system (Microsoft Windows 10 Pro).

The install of kb2999226 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

vc_redist140 v14.29.30037 [Approved]

vc_redist140 package files install completed. Performing other installation steps.

Downloading vc_redist140-x86

from 'https://download.visualstudio.microsoft.com/download/pr/76a91598-ca94-410b-b874-c7fa26e400da/91C21C93A88DD82E8AE429534DACBC7A4885198361EAE18D82920C714E328CF9/VC_redist.x86.exe'

Progress: 100% - Completed download of

C:\Users\oleksii\AppData\Local\Temp\chocolatey\vc_redist140\14.29.30037\VC_redist.x86.exe (13.14 MB).

Download of VC_redist.x86.exe (13.14 MB) completed.

Hashes match.

Installing vc_redist140-x86...

vc_redist140-x86 has been installed.

Downloading vc_redist140-x64 64 bit

from 'https://download.visualstudio.microsoft.com/download/pr/f1998402-3cc0-466f-bd67-d9fb6cd2379b/A1592D3DA2B27230C087A3B069409C1E82C2664B0D4C3B511701624702B2E2A3/VC_redist.x64.exe'

Progress: 100% - Completed download of

C:\Users\oleksii\AppData\Local\Temp\chocolatey\vc_redist140\14.29.30037\VC_redist.x64.exe (24 MB).

Download of VC_redist.x64.exe (24 MB) completed.

Hashes match.

Installing vc_redist140-x64...

vc_redist140-x64 has been installed.

vc_redist140 may be able to be automatically uninstalled.

The install of vc_redist140 was successful.

Software installed as 'exe', install location is likely default.

vc_redist2015 v14.0.24215.20170201 [Approved]

vc_redist2015 package files install completed. Performing other installation steps.

The install of vcredist2015 was successful.

Software install location not explicitly set, could be in package or default install location if installer.

python3 v3.9.5 [Approved]

python3 package files install completed. Performing other installation steps.

Installing 64-bit python3...

python3 has been installed.

Python installed to: 'C:\Python39'

Restricting write permissions to Administrators

python3 can be automatically uninstalled.

Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).

The install of python3 was successful.

Software installed as 'exe', install location is likely default.

Chocolatey installed 10/10 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Installed:

- kb2919355 v1.0.20160915
- kb3033929 v1.0.5
- chocolatey-core.extension v1.3.5.1
- kb2999226 v1.0.20181019
- python3 v3.9.5
- vcredist2015 v14.0.24215.20170201
- kb2919442 v1.0.20160915
- vcredist140 v14.29.30037
- kb3035131 v1.0.3
- chocolatey-windowsupdate.extension v1.0.4

Chocolatey v0.10.15

Installing the following packages:

kubernetes-cli

By installing you accept licenses for the packages.

Progress: Downloading kubernetes-cli 1.21.1... 100%

kubernetes-cli v1.21.1 [Approved]

kubernetes-cli package files install completed. Performing other installation steps.

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar.gz to C:\ProgramData\chocolatey\lib\kubernetes-cli\tools...

C:\ProgramData\chocolatey\lib\kubernetes-cli\tools

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar to
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
ShimGen has successfully created a shim for kubectrl-conver.exe
ShimGen has successfully created a shim for kubectrl.exe
The install of kubernetes-cli was successful.
Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Chocolatey v0.10.15

Installing the following packages:

kubernetes-helm

By installing you accept licenses for the packages.

Progress: Downloading kubernetes-helm 3.6.0... 100%

kubernetes-helm v3.6.0 [Approved]

kubernetes-helm package files install completed. Performing other installation steps.

Downloading kubernetes-helm 64 bit

from 'https://get.helm.sh/helm-v3.6.0-windows-amd64.zip'

Progress: 100% - Completed download of C:\Users\oleksii\AppData\Local\Temp\chocolatey\kubernetes-helm\3.6.0\helm-v3.6.0-windows-amd64.zip (13.68 MB).

Download of helm-v3.6.0-windows-amd64.zip (13.68 MB) completed.

Hashes match.

Extracting C:\Users\oleksii\AppData\Local\Temp\chocolatey\kubernetes-helm\3.6.0\helm-v3.6.0-windows-amd64.zip to C:\ProgramData\chocolatey\lib\kubernetes-helm\tools...

C:\ProgramData\chocolatey\lib\kubernetes-helm\tools

ShimGen has successfully created a shim for helm.exe

The install of kubernetes-helm was successful.

Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-helm\tools'

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Chocolatey v0.10.15

Installing the following packages:

azure-cli

By installing you accept licenses for the packages.

Progress: Downloading azure-cli 2.24.2... 100%

azure-cli v2.24.2 [Approved]

azure-cli package files install completed. Performing other installation steps.

Downloading azure-cli

from 'https://azcliproduct.blob.core.windows.net/msi/azure-cli-2.24.2.msi'

Progress: 100% - Completed download of C:\Users\oleksii\AppData\Local\Temp\chocolatey\azure-cli\2.24.2\azure-cli-2.24.2.msi (47.98 MB).

Download of azure-cli-2.24.2.msi (47.98 MB) completed.

Hashes match.

Installing azure-cli...

azure-cli has been installed.

azure-cli may be able to be automatically uninstalled.

Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).

The install of azure-cli was successful.

Software installed to 'C:\Program Files (x86)\Microsoft\Edge\Application'

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Chocolatey v0.10.15

Installing the following packages:

awscli

By installing you accept licenses for the packages.

Progress: Downloading awscli 2.2.9... 100%

awscli v2.2.9 [Approved]

awscli package files install completed. Performing other installation steps.

Downloading awscli 64 bit

from 'https://awscli.amazonaws.com/AWSCLIV2-2.2.9.msi'

Progress: 100% - Completed download of

C:\Users\oleksii\AppData\Local\Temp\chocolatey\awscli\2.2.9\AWSCLIV2-2.2.9.msi (26.48 MB).

Download of AWSCLIV2-2.2.9.msi (26.48 MB) completed.

Hashes match.

Installing awscli...

awscli has been installed.

awscli may be able to be automatically uninstalled.

Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).

The install of awscli was successful.

Software installed as 'MSI', install location is likely default.

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Chocolatey v0.10.15

Installing the following packages:

eksctl

By installing you accept licenses for the packages.

Progress: Downloading eksctl 0.52.0... 100%

eksctl v0.52.0 [Approved]

eksctl package files install completed. Performing other installation steps.

eksctl is going to be installed in 'C:\ProgramData\chocolatey\lib\eksctl\tools'

Downloading eksctl 64 bit

from 'https://github.com/weaveworks/eksctl/releases/download/0.52.0/eksctl_Windows_amd64.zip'

Progress: 100% - Completed download of

C:\Users\oleksii\AppData\Local\Temp\chocolatey\eksctl\0.52.0\eksctl_Windows_amd64.zip (17.72 MB).

Download of eksctl_Windows_amd64.zip (17.72 MB) completed.

Hashes match.

Extracting C:\Users\oleksii\AppData\Local\Temp\chocolatey\eksctl\0.52.0\eksctl_Windows_amd64.zip to

C:\ProgramData\chocolatey\lib\eksctl\tools...

C:\ProgramData\chocolatey\lib\eksctl\tools

ShimGen has successfully created a shim for eksctl.exe

The install of eksctl was successful.

Software installed to 'C:\ProgramData\chocolatey\lib\eksctl\tools'

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Chocolatey v0.10.15

Installing the following packages:

gcloudsdk

By installing you accept licenses for the packages.

Progress: Downloading gcloudsdk 0.0.0.20210318... 100%

gcloudsdk v0.0.0.20210318 [Approved]

gcloudsdk package files install completed. Performing other installation steps.

Downloading gcloudsdk

from 'https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe'

Progress: 100% - Completed download of

C:\Users\oleksii\AppData\Local\Temp\chocolatey\gcloudsdk\0.0.0.20210318\GoogleCloudSDKInstaller.exe (147.4 KB).

Download of GoogleCloudSDKInstaller.exe (147.4 KB) completed.

Hashes match.

Installing gcloudsdk...

gcloudsdk has been installed.

gcloudsdk can be automatically uninstalled.

Environment Vars (like PATH) have changed. Close/reopen your shell to see the changes (or in powershell/cmd.exe just type `refreshenv`).

The install of gcloudsdk was successful.

Software installed to "C:\Program Files (x86)\Google\Cloud SDK"

Chocolatey installed 1/1 packages.

See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

PS D:\msdp-main>

Appendix B

“Requirements_instalation_Windows.ps1”

(Informative)

```
#section will contain installation for prewired software helm cli for clouds, etc.
#Install the package manager for windows chocolatey
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

#Install puthon3
choco install python3 -y

#install kubectl for cluster communication
choco install kubernetes-cli -y
#Install helm for chart operating
choco install kubernetes-helm -y

#install az for communication with Azure
choco install azure-cli -y

#Install aws cli and eksctl
choco install awscli -y
choco install eksctl -y

#Install gcp sdk
choco install gcloudsdk -y
```

Appendix C

“Requirements_instalation_Linux.sh”

(Informative)

```
#section will contain installation for prewired software helm cli for clouds, etc.
#Install the package manager for Linux snap
sudo apt update
sudo apt install snapd -y
sudo apt-get install python3 -y
sudo snap install helm --classic
sudo snap install kubectl --classic

#Azure cli installation
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
az --version

#Install AWS cli and eksctl

curl --silent --
location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname
-s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
sudo apt install awscli -y

#installing gcloud
sudo snap install google-cloud-sdk --classic
```


Appendix D

“Instalation_into_AWS.ps1”

(Informative)

```
#before use this script you should perform aws configure

eksctl create cluster --version=1.19 --name=jenkins-cluster --node-private-networking --
alb-ingress-access --region=us-west-2 --asg-access --without-nodegroup

eksctl create nodegroup --cluster=jenkins-cluster --region=us-west-2 --name=jenkins-server-
ng --managed --nodes=2 --node-labels="lifecycle=OnDemand,intent=jenkins-server"

$AWSKubernetesNameSpace='jenkins-ci-test'

#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $AWSKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update

#run installation command
helm install jenkins-test jenkins/jenkins --namespace $AWSKubernetesNameSpace --set
controller.ingress.enabled=true

#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer --
namespace $AWSKubernetesNameSpace
```

Appendix E

“Instalation_into_AWS.sh”

(Informative)

```
#before use this script you should perform aws configure

eksctl create cluster --version=1.19 --name=jenkins-cluster --node-private-
networking --alb-ingress-access --region=us-west-2 --asg-access --without-nodegroup

eksctl create nodegroup --cluster=jenkins-cluster --region=us-west-2 --name=jenkins-
server-ng --managed --nodes=2 --node-labels="lifecycle=OnDemand,intent=jenkins-server"

AWSKubernetesNameSpace='jenkins-ci-test'

#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $AWSKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update

#run installation command
helm install jenkins-test jenkins/jenkins --namespace $AWSKubernetesNameSpace --
set controller.ingress.enabled=true

#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer -
--namespace $AWSKubernetesNameSpace
```

Appendix F

“Instalation_into_Azure.ps1”

(Informative)

```
#Function for PATH modification
#Before use this script please provide necessary values to variables below.

function Set-PathVariable {
    param (
        [string]$AddPath,
        [string]$RemovePath
    )
    $regexPaths = @()
    if ($PSBoundParameters.Keys -contains 'AddPath'){
        $regexPaths += [regex]::Escape($AddPath)
    }

    if ($PSBoundParameters.Keys -contains 'RemovePath'){
        $regexPaths += [regex]::Escape($RemovePath)
    }

    $arrPath = $env:Path -split ';'
    foreach ($path in $regexPaths) {
        $arrPath = $arrPath | Where-Object {$_ -notMatch "^$path\\?"}
    }
    $env:Path = ($arrPath + $AddPath) -join ';'
}

#Section will contain necessary patch updates.
#Adding patches
$addPathKubectl=$env:HOMEDRIVE+$env:HOMEPATH+'\.azure-kubectl'
$addPathKubelogin=$env:HOMEDRIVE+$env:HOMEPATH+'\.azure-kubelogin'
Set-PathVariable -AddPath $addPathKubectl;
Set-PathVariable -AddPath $addPathKubelogin;
$AzureResourceGroupName='jenkins-ci-test-1'
$AzureClusterName='jenkins-ci-test-1'
$AzureKubernetesNameSpace='jenkins-ci-test-1'

#Infrastructure creation
#Azure login will prompt user for login into cloud env;
az logout
az login
#creating infrastructure I'll use the cheapest region
az group create --name $AzureResourceGroupName --location eastus

#enabling monitoring recording to Azure recommendation https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.OperationalInsights

#creating AKS cluster
az aks create --resource-group $AzureResourceGroupName --name $AzureClusterName --node-count 1 --enable-addons monitoring --generate-ssh-keys

#Instalation of kubectl
az aks install-cli

#Configure kubectl to work with newly created cluster
az aks get-credentials --resource-group $AzureResourceGroupName --name $AzureClusterName --
overwrite-existing
#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $AzureKubernetesNameSpace
#adding latest repository:
helm repo add jenkins https://charts.jenkins.io
helm repo update
#run installation command
helm install jenkins-test jenkins/jenkins --namespace $AzureKubernetesNameSpace --set
controller.ingress.enabled=true
#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer --
namespace $AzureKubernetesNameSpace
```

Appendix J

“Instalation_into_Azure.sh”

(Informative)

```
#Before use this script please provide necessary values to variables below.

AzureResourceGroupName='jenkins-ci-test-1'
AzureClusterName='jenkins-ci-test-1'
AzureKubernetesNameSpace='jenkins-ci-test-1'


#Infrastructure creation
#Azure login will prompt user for login into cloud env;
az logout
az login
#creating infrastructure I'll use the cheapest region
az group create --name $AzureResourceGroupName --location eastus
#enabling monitoring recording to Azure recomendation https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough
az provider register --namespace Microsoft.OperationsManagement
az provider register --namespace Microsoft.OperationalInsights
#creating AKS cluster
az aks create --resource-group $AzureResourceGroupName --name $AzureClusterName --
node-count 1 --enable-addons monitoring --generate-ssh-keys
#Inastalation of kubectl
az aks install-cli
#Configure kubectl to work with newly created cluster
az aks get-credentials --resource-group $AzureResourceGroupName --
name $AzureClusterName --overwrite-existing
#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $AzureKubernetesNameSpace
#adding latest repository:
helm repo add jenkins https://charts.jenkins.io
helm repo update
#run installation command
helm install jenkins-test jenkins/jenkins --namespace $AzureKubernetesNameSpace --
set controller.ingress.enabled=true
#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer -
-namespace $AzureKubernetesNameSpace
```

Appendix H

“Instalation_into_GCP.ps1”

(Informative)

```
# before use this package you should register in GCP platfrom
# create project
# enable billing
# open Kubernetes Engine API for that project and provide information into variable below

$GCPProjectID='jenkins-into-gke2'
$GCPProjectName='Jenkins-ci'
$GCPclusterName='regional-cluster-1'
$GCPKubernetesNameSpace='jenkins-ci-test'

#Assumption all neceary software installed.

#GCP clean login information
gcloud auth revoke --all
#Login into GCP
gcloud auth login
#Set project by ID
gcloud config set project $GCPProjectID

#Create zonal cluster in us zones
gcloud container clusters create $GCPclusterName --project $GCPProjectID --region "us-central1" --num-nodes 1

#Get connection info for cluster
gcloud container clusters get-credentials $GCPclusterName --region us-central1 --project $GCPProjectID

#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $GCPKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update

#run installation command
helm install jenkins-test jenkins/jenkins --namespace $GCPKubernetesNameSpace

#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer --namespace $GCPKubernetesNameSpace
```

Appendix I

“Instalation_into_GCP.sh”

(Informative)

```
# before use this package you should register in GCP platfrom
# create project
# enable billing
# open Kubernetes Engine API for that project and provide information into variable below
GCPProjectID='jenkins-into-gke2'
GCPProjectName='Jenkins-ci'
GCPclusterName='regional-cluster-1'
GCPKubernetesNameSpace='jenkins-ci-test'
#Assumption all necessary software installed.
#GCP clean login information
gcloud auth revoke --all
#Login into GCP
gcloud auth login
#Set project by ID
gcloud config set project $GCPProjectID
#Create autopilot cluster in us zones
#UPDATE unfortunately auto is not the good choice.
#gcloud container clusters create-auto $GCPclusterName --project $GCPProjectID --region "us-central1"
#Create zonal cluster in us zones
gcloud container clusters create $GCPclusterName --project $GCPProjectID --region "us-central1" --num-nodes 1

#Get connection info for cluster
gcloud container clusters get-credentials $GCPclusterName --region us-central1 --project $GCPProjectID

#installing jenkins into cluster
#create new namespace for Jenkins
kubectl create namespace $GCPKubernetesNameSpace

#adding latest repository:

helm repo add jenkins https://charts.jenkins.io
helm repo update
#run installation command
helm install jenkins-test jenkins/jenkins --namespace $GCPKubernetesNameSpace --set controller.ingress.enabled=true
#opens our pod to the internet
kubectl expose pod jenkins-test-0 --port=8080 --name=jenkins-out --type=LoadBalancer --namespace $GCPKubernetesNameSpace
```