

CA - VTU DevOps Course Code : BCSL657D Syllabus	2
Prologue Commit To Achieve	4
CA - VTU DevOps FDP TOC With Program Outcomes	5
CA - FDP Pre-requisites and Software Requirements	8
CA - Introduction To SDLC & Need For Version Control & Build Tools	11
CA - Git & GitHub Notes & Documentation	17
CA - Git & GitHub Simplified For DevOps VTU Lab	31
CA - Maven Notes & Documentation	34
CA - Experiment 1 - Introduction To Maven & Gradle Build Tools	43
CA - Experiment 2 - Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins	46
CA - Gradle Notes & Documentation	66
CA - Experiment 3 Part 1 - Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation	73
CA - Experiment 3 Part 2 : GRADLE KOTLIN DSL WORKFLOW IntelliJ Idea	83
CA - Experiment 4 - Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle	87
CA - Jenkins Notes & Documentation	91
CA - Experiment 5 - Introduction to Jenkins: What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use	97
CA - Experiment 6 - Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests	101
CA - Experiment 7 - Configuration Management With Ansible	104
CA - Experiment 8 - Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins	105
CA - Experiment 9 - Introduction To Azure DevOps	115
CA - Experiment 10 - Creating Build Pipelines in Azure DevOps	122
CA - Creating QA-Web-App & PROD-Web-App Instructions For Exp 10 & 11	140
CA - Experiment 11 - Creating Release Pipelines - Azure App Services	164
CA - Experiment 12 - Final Practical Exercise and Wrap-Up	188
CA - Thank You Message From Coders Arcade	211



CA - VTU DevOps Course Code : BCSL657D Syllabus

DevOps Lab Manual – Table of Contents (VTU)

Sl. No	Experiment Title	Topics Covered
1	Introduction to Maven and Gradle	Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup
2	Working with Maven	Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins
3	Working with Gradle	Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation
4	Practical Exercise: Build and Run a Java Application	Build and Run a Java Application with Maven, Migrate the Same Application to Gradle
5	Introduction to Jenkins	What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use
6	Continuous Integration with Jenkins	Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests
7	Configuration Management with Ansible	Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook
8	Practical Exercise: CI/CD with Jenkins and Ansible	Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins
9	Introduction to Azure DevOps	Overview of Azure DevOps Services, Setting Up an Azure DevOps Account

		and Project
10	Creating Build Pipelines in Azure DevOps	Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports
11	Creating Release Pipelines in Azure DevOps	Deploying Applications to Azure App Services, Managing Secrets and Configurations with Azure Key Vault, Hands-On: Continuous Deployment with Azure Pipelines
12	Final Practical Exercise and Wrap-Up	Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A



CODERS ARCADE

COMMIT TO ACHIEVE

Prologue || Commit To Achieve

About Coders Arcade

 **Coders Arcade** is your go-to edtech platform for mastering programming, DevOps, automation, and cloud technologies. We offer structured, hands-on training designed for both **beginners and professionals**, with a special focus on **VTU subjects** to help students ace their curriculum.

 Our content covers **C, C++, Java, Python, SQL, Data Science, Machine Learning, AWS, DevOps tools (Maven, Gradle, Jenkins, Ansible, Azure DevOps), and more!** Whether you're preparing for placements or upskilling for your career, **Coders Arcade** provides the right guidance with **industry-oriented** courses, real-world projects, and expert mentorship.

Why Choose Us?

- Beginner-friendly & advanced tutorials
- Industry-relevant projects & hands-on exercises
- 100% practical approach for better learning
- Affordable & high-quality training
- Strong VTU syllabus alignment



About the Author

 With **14+ years of experience** in the software industry, I have worked with leading companies like **Zaloni, TCS, and Flipkart**, taking on roles from **Test Automation Engineer** to **Backend Developer**. Currently, I am a **Senior Program Manager at Coders Arcade**, dedicated to delivering high-quality, accessible technical education.

 I have provided **placement trainings & FDPs** in various esteemed institutions across **Karnataka**, helping students bridge the gap between **academics and industry requirements**. Passionate about software development and teaching, I believe in making **learning simple, engaging, and career-focused**.

 **Let's Learn. Let's Code. Let's Grow. Commit To Achieve....!!!**

 Connect with me:

 LinkedIn: [Saurav Sarkar](#)

 YouTube: [Coders Arcade](#)



CA - VTU DevOps FDP TOC With Program Outcomes

Table Of Contents

Sl. No	Experiment Title	Topics Covered	Program Outcome	Key Takeaways
1	Introduction to Maven and Gradle	Overview of Build Automation Tools, Key Differences Between Maven and Gradle, Installation and Setup	Understand the importance of build automation tools and their role in DevOps workflows	Learn to install and configure Maven and Gradle for project builds
2	Working with Maven	Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins	Gain hands-on experience in managing dependencies and plugins using Maven	Learn to structure a Maven project and configure the <code>pom.xml</code> file
3	Working with Gradle	Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation	Learn to automate builds using Gradle and understand its scripting capabilities	Understand the difference between Groovy and Kotlin DSL for build automation
4	Practical Exercise: Build and Run a Java Application	Build and Run a Java Application with Maven, Migrate the Same Application to Gradle	Gain practical experience in building and running Java applications with both tools	Learn how to migrate an existing Maven-based project to Gradle
5	Introduction to Jenkins	What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use	Understand the fundamentals of Jenkins and its role in Continuous Integration	Learn to install, configure, and access Jenkins for project automation
6	Continuous Integration with Jenkins	Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle,	Learn to automate the software build and testing process	Implement CI/CD pipelines using Jenkins for better development efficiency

		Running Automated Builds and Tests		
7	Configuration Management with Ansible	Basics of Ansible: Inventory, Playbooks, and Modules, Automating Server Configurations with Playbooks, Hands-On: Writing and Running a Basic Playbook	Understand configuration management and automation with Ansible	Learn to write and execute Ansible playbooks for server configurations
8	Practical Exercise: CI/CD with Jenkins and Ansible	Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins	Apply Jenkins and Ansible together for an automated deployment pipeline	Understand the integration of Jenkins CI/CD with Ansible automation
9	Introduction to Azure DevOps	Overview of Azure DevOps Services, Setting Up an Azure DevOps Account and Project	Learn about cloud-based DevOps tools and their benefits	Gain hands-on experience in setting up and navigating Azure DevOps
10	Creating Build Pipelines in Azure DevOps	Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports	Implement automated build pipelines in a cloud environment	Learn to connect repositories and run tests in Azure Pipelines
11	Creating Release Pipelines in Azure DevOps	Deploying Applications to Azure App Services, Managing Secrets and Configurations with Azure Key Vault, Hands-On: Continuous Deployment with Azure Pipelines	Understand the release management process in a DevOps environment	Deploy applications securely using Azure Key Vault and Azure Pipelines
12	Final Practical Exercise and Wrap-Up	Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A	Gain end-to-end experience in setting up a DevOps pipeline	Understand best practices in DevOps and real-world applications

Overall Program Benefits For Participants:

- Master DevOps Workflows & Tools – Gain a comprehensive understanding of DevOps methodologies, including automation, CI/CD, and infrastructure management.**

- Hands-On Experience with Industry Tools – Work with Git (Version Control), Jenkins & Azure DevOps (CI/CD), and Ansible (Automation) to build real-world expertise.
 - Enhance Your Practical Skills – Apply your knowledge through live demonstrations, hands-on labs, and project-based learning to solidify your understanding.
 - Stay Aligned with Industry Best Practices – Learn cutting-edge DevOps techniques used by top organizations and apply them in real-world scenarios.
- Upgrade your skill set. Gain practical expertise. Accelerate your career in DevOps! 



CODERS ARCADE

COMMIT TO ACHIEVE



CA - FDP Pre-requisites and Software Requirements

Pre-requisites and Software Requirements

Pre-requisites

1. Java Programming Knowledge:

- Basic understanding of Java syntax, OOP concepts, and familiarity with building Java applications.

2. Operating System:

- Windows, macOS, or Linux (64-bit recommended).

3. Internet Connectivity:

- Reliable internet connection for downloading software, accessing GitHub, and Azure resources.

4. GitHub Account:

- All participants must have a fully functional **GitHub account** for version control and source code management.

Software Requirements

Software/Tool	Version	Purpose	Installation Notes
Java JDK	17.0 or above (preferred)	Required for building and running Java-based applications.	Download from Oracle or OpenJDK distributions.
IntelliJ IDEA	2022.2 or above	Integrated Development Environment (IDE) for Java development and project management.	Download from JetBrains .
Eclipse For Developers	2022-09 or above	Integrated Development Environment (IDE) for Java development and project management.	Download from Eclipse Packages .
Apache Tomcat Server	10 or above	Local Server For Basic WebSite Deployment.	Download from Apache Tomcat .

Git	Latest stable version	Version control system for managing source code and collaborating via GitHub.	Download from Git .
Maven	Latest stable version	Build automation tool for Java projects.	Download from Apache Maven .
Gradle	Latest stable version	Alternative build automation tool for managing dependencies and tasks.	Download from Gradle .
Jenkins	Latest stable version	CI/CD tool for automating builds, tests, and deployments.	Download from Jenkins .
Oracle VM VirtualBox	Latest stable version	Virtualization tool for creating and managing virtual machines.	Download from VirtualBox .
Vagrant	Latest stable version	Tool for managing and provisioning virtual machine environments.	Download from Vagrant .
Ansible	Latest stable version	Configuration management and automation tool.	Install via Ansible documentation .
Microsoft Azure	Free Tier Account	Needed for DevOps CI-CD Pipelines & Deployments.	Install via Microsoft Azure SignUp .

Important Installation Video Links :

Java : [YouTube](#) Installing Java and its Dependencies (VTU Syllabus)

Eclipse : [YouTube](#) Installing Eclipse IDE and Running Your First Java Program.

Apache Tomcat : [YouTube](#) Apache Tomcat Installation - Tomcat 10 Server on Windows 10/11 (Coders Arcade)

Git : [YouTube](#) Git Tutorials for Beginners - Installing Git || Step-by-Step Guide

Maven : [YouTube](#) The Best Way to Install Maven - Coders Arcade - Maven Installation: What is Maven? || Coders Arcade

Gradle : [GeeksforGeeks](#) How to Install Gradle on Windows? - GeeksforGeeks

Jenkins : [YouTube](#) Jenkins Installation - Step by Step Guide

Oracle Virtual Box : [YouTube](#) Installing Oracle Virtual Box On Windows 11 || Step By Step Guide

Ubuntu VM On Windows : [YouTube](#) Installing Ubuntu 23 On Windows 11 Using Oracle Virtual Box || Step By Step Guide | Coders Arcade

Important Note for Azure DevOps Setup

- An active Azure account is required for the Azure DevOps sessions.
- Azure accounts should only be created one week prior to the commencement of Azure DevOps topics in college laboratories.
- This ensures compliance with Microsoft's 30-day free tier policy and avoids restricted access to parallelism.

Resources for Practice:

- Additional learning materials and resources are available for participants to practice Azure DevOps concepts here → DevOps VTU Resources [Click Here](#).
-



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Introduction To SDLC & Need For Version Control & Build Tools

Introduction to DevOps & the Need for Build Tools

1. Understanding the Software Development Lifecycle (SDLC)

Before diving into build tools like **Maven** and **Gradle**, it's essential to understand how software has traditionally been developed. The **Software Development Lifecycle (SDLC)** provides a structured approach to building software efficiently.

One of the earliest models of SDLC is the **Waterfall Model**, which follows a **linear and sequential** process where each phase must be **fully completed** before moving to the next. However, this model has several limitations, which led to the adoption of **Agile** and **DevOps** methodologies.

2. Waterfall Model: A Step-by-Step Approach:

The **Waterfall Model** consists of six distinct phases:

1 Requirement Analysis

- 👉 The first interaction always happens between the **Business Analyst (BA)** and the **Client**.
- 👉 The BA prepares the **CRS (Client Requirement Specification)** before even preparing the **SRS (Software Requirement Specification)**.
- 👉 The final **SRS document** defines all system functionalities before development begins.

2 Design

- 👉 The **architecture of the software** is planned based on the requirements.
- 👉 High-Level Design (**HLD**) and Low-Level Design (**LLD**) are created.
- 👉 Technologies, frameworks, and databases are chosen.

3 Coding (Implementation)

- 👉 Developers **write the code** based on the design.
- 👉 Code is divided into modules and integrated later.
- 👉 Best practices like version control (e.g., **Git**) are followed.

4 Testing

- 👉 Testers verify if the software meets the **SRS specifications**.
- 👉 Types of testing include **unit testing, integration testing, and system testing**.
- 👉 Bugs are reported to developers for fixing.

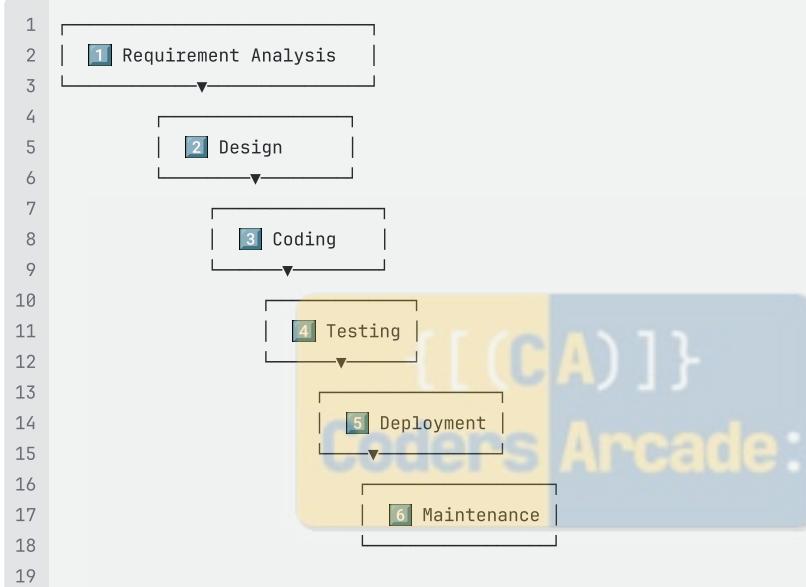
5 Deployment

- 👉 The software is **released** to production.
- 👉 Deployment can be **on-premises or cloud-based**.
- 👉 The software becomes available for real-world users.

6 Maintenance

- 📌 Post-deployment, updates, bug fixes, and improvements are continuously made.
- 📌 Performance monitoring and security patches are applied.

3. Waterfall Model - Block-Based Diagram



4. Problems with the Waterfall Model

- 🔴 **Rigid & Inflexible** – Changes cannot be made once a phase is completed.
- 🔴 **Late Bug Detection** – Testing happens after coding, making bug fixes costly.
- 🔴 **Slow Development Process** – Each phase must be completed before the next starts.
- 🔴 **Not Suitable for Modern Agile Environments** – Does not support rapid iteration.

5. Introduction to Agile Methodology

To overcome Waterfall's limitations, the **Agile Model** was introduced. Agile promotes **iterative and incremental development**, allowing teams to deliver software faster with continuous feedback.

Key Agile Concepts

- ✅ **Sprints** – Development happens in short cycles, usually lasting **7-14 days**.
- ✅ **Scrum Master** – Facilitates Agile processes and removes blockers for the team.
- ✅ **Daily Stand-up Meetings** – Short meetings where team members discuss:
 - 1 **What they did yesterday?**
 - 2 **What they plan to do today?**
 - 3 **Any blockers?**
- ✅ **Retrospective Meetings** – At the end of each sprint, the team reflects on what worked well and what can be improved.

Sprint Retrospective: The 4Ls Framework

The **4Ls retrospective** is a common technique used in tech interviews:

- **Loved** – What went well in the sprint?
- **Longed for** – What do we wish had happened?
- **Loathed** – What went wrong or was frustrating?
- **Learned** – What new knowledge or skills were gained?

6. The Transition from Agile to DevOps

While Agile improved development, **deployment and operations remained slow**. This led to the birth of **DevOps**, which introduced:

- **Continuous Integration & Deployment (CI/CD)**
- **Automation of Builds, Testing, and Deployment**
- **Faster & More Reliable Software Releases**

To make DevOps efficient, we need **Build Tools** like **Maven & Gradle**, which simplify project management.

7. The Clumsy Manual Approach to Project Setup:

Before introducing **Maven & Gradle**, let's look at how we manually created a **Selenium Automation Test Project**.

Manual Setup Process

- 1 **Created a Simple Java Project** in IntelliJ IDEA (**Selenium Automation Test**).
- 2 **Manually Downloaded the Selenium JAR** from:
 <https://www.selenium.dev>
- 3 **Created a lib folder** and placed `selenium-server-4.28.1.jar` inside it.
- 4 **Manually Added the Dependency** in IntelliJ IDEA:
 - File → Project Structure → Libraries → Add Selenium JAR → Apply → OK
- 5 **Wrote a Selenium Test in Java** (`LoginTest.java`) to validate login at <https://www.saucedemo.com>.
- 6 **Ran the test manually** in IntelliJ IDEA.

Java Selenium Code for Login Automation

```

1 import org.openqa.selenium.By;
2 import org.openqa.selenium.WebDriver;
3 import org.openqa.selenium.chrome.ChromeDriver;
4
5 public class LoginTest {
6     public static void main(String[] args) throws InterruptedException {
7         WebDriver driver = new ChromeDriver();
8         driver.get("https://www.saucedemo.com/");
9         driver.manage().window().maximize();
10        Thread.sleep(2000);
11        driver.findElement(By.id("user-name")).sendKeys("standard_user");
12        Thread.sleep(2000);
13        driver.findElement(By.id("password")).sendKeys("secret_sauce");
14        Thread.sleep(2000);
15        driver.findElement(By.id("login-button")).click();
16        Thread.sleep(2000);
17        driver.quit();
18    }
  
```

```
19 }  
20
```

1 Explanation of the Selenium Code (LoginTest.java)

This Java program automates the login process for **SauceDemo** using **Selenium WebDriver**. Below is a detailed explanation of each part of the code:

1 Importing Required Selenium Libraries

```
1 import org.openqa.selenium.By;  
2 import org.openqa.selenium.WebDriver;  
3 import org.openqa.selenium.chrome.ChromeDriver;  
4
```

- `By` – Helps locate web elements on the page (e.g., text fields, buttons).
- `WebDriver` – Interface for automating browsers.
- `ChromeDriver` – A class that implements WebDriver to control Google Chrome.

2 Main Method Execution

```
1 public class LoginTest {  
2     public static void main(String[] args) throws InterruptedException {  
3
```

- The program starts execution from `main()`.
- The `throws InterruptedException` handles the `Thread.sleep()` method, which pauses execution temporarily.

3 Launching Chrome Browser

```
1 WebDriver driver = new ChromeDriver();  
2
```

- Creates an instance of `ChromeDriver`, which opens a new Chrome browser window.
- Selenium versions after 4.11 do not require separate browser drivers.

4 Navigating to the Website

```
1 driver.get("https://www.saucedemo.com/");  
2 driver.manage().window().maximize();  
3 Thread.sleep(2000);  
4
```

- `driver.get(URL)` – Opens the given website (`https://www.saucedemo.com/`).
- `manage().window().maximize()` – Maximizes the browser window.
- `Thread.sleep(2000)` – Waits for **2 seconds** to allow elements to load.

5 Entering Username & Password

```

1 driver.findElement(By.id("user-name")).sendKeys("standard_user");
2 Thread.sleep(2000);
3 driver.findElement(By.id("password")).sendKeys("secret_sauce");
4 Thread.sleep(2000);
5

```

- Locating elements by their `id` attribute and entering values:
 - Username: "standard_user"
 - Password: "secret_sauce"
- `sendKeys(value)` – Types the provided text into the input field.

6 Clicking the Login Button

```

1 driver.findElement(By.id("login-button")).click();
2 Thread.sleep(2000);
3

```

- Finds the login button using `id="login-button"` and clicks it.
- Waits for 2 seconds to observe the login action.



7 Closing the Browser

```

1 driver.quit();
2

```

- Closes the browser after execution to free up system resources.

CODERS ARCADE

- Opens Google Chrome
- Navigates to SauceDemo Login Page
- Enters Username and Password
- Clicks Login Button
- Closes the browser

COMMIT TO ACHIEVE

This simple example shows the power of Selenium for web automation, highlighting the disadvantages of manually managing dependencies. This is why build tools like **Maven & Gradle** are needed, which we will explore after Version Control Systems (Git & GitHub).

8. Problems with the Manual Approach

- Time-Consuming** – Downloading and adding dependencies manually is inefficient.
- Error-Prone** – Missing JAR files or incorrect configurations can break the project.
- Difficult to Manage** – Dependencies are not automatically updated.
- Not Scalable** – Every team member must manually configure their setup.

9. Why Do We Need Build Tools?

To overcome these inefficiencies, we use **Maven & Gradle**, which:

- Automatically manage dependencies** – No need to download JARs manually.
- Simplify project configuration** – A single configuration file (`pom.xml` for Maven, `build.gradle.kts` for Gradle) handles everything.
- Enable easy build & testing** – Run tests and package applications using simple commands.
- Ensure consistency** – The same project setup works on different machines.

10. What's Next? Understanding Version Control

Before moving to Maven & Gradle, we must first understand Version Control Systems (VCS) like Git. Version control plays a crucial role in DevOps, enabling:

- Efficient Code Management** – Track changes, revert to previous versions, and collaborate seamlessly.
- Team Collaboration** – Multiple developers can work on the same project without conflicts.
- Integration with CI/CD Pipelines** – Automates builds, testing, and deployment.

In the next section, we will explore Git, covering:

- What is Version Control?**
- Introduction to Git & GitHub**
- Basic Git Commands & Repository Setup**
- Branching, Merging & Collaboration**

Once we have a strong grasp of *Version Control*, we will then move to **Maven** & **Gradle** for efficient build automation. 

CODERS ARCADE

COMMIT TO ACHIEVE



CA - Git & GitHub Notes & Documentation

- Introduction
- Version Control
 - About Version Control
 - Local Version Control Systems
 - Centralized Version Control Systems
 - Distributed Version Control Systems
- Setting Up Git on Local System
 - Step 1: Check if git is already installed
 - Step 2 : If not installed download Git installer from <https://git-scm.com/>
 - Step 3 : Run installer and install git
 - Step 4 : Check if git is installed
- Create GitHub Account
 - Step 1: Go to <https://github.com/> and sign up for a new account
 - Step 2 : Login to GitHub
 - Step 3 : Create a new Repository (Private or Public)
- Basic Git Commands
 - Step 1: Create a new folder and open Git Bash/Cmd and go to the folder location
 - Step 2 : Run these commands for git configuration
 - Step 3 : Initialize git using this command
 - Step 4 : Add some sample files in the folder by using this command
 - Step 5 : Run these commands to check status, add files and commit your changes/updates
- Git Branches
 - Steps to follow while working with branches in Git
 - Step 1: Create branch `git branch "branch_name"`
 - Step 2 : Checkout branch `git checkout "branch_name"`
 - Step 3 : Make some changes to your project
 - Step 4 : On local repo checkout to master branch `"git checkout master"`
 - Step 5 : Merge new branch in master branch `"git merge branch_name"`
 - Step 6 : Push all your changes `"git push -u origin master"`
 - Step 7 : Delete a particular branch
- Git Tags
 - Step 1: Checkout the branch where you want to create the tag
 - Step 2 : Create a tag with some name
 - Step 3 : Display or Show tags
 - Step 3 : Push tags to remote repository `git push origin v1.0`
 - Step 4 : Delete tags (Only if required)
- Checking out with tags
- Creating tags from past commits

- Git Merge Conflicts
 - Understanding merge conflicts
 - Types of merge conflicts
 - Git fails to start the merge
 - Git fails during the merge
 - How to identify merge conflicts
 - How to Resolve Merge Conflicts in Git?
 - Git Commands to Resolve Conflicts
 - 1. git log --merge
 - 2. git diff
 - 3. git checkout
 - 4. git reset --mixed
 - 5. git merge --abort
 - 6. git reset



Git & GitHub

Introduction

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Version Control

Watch This Video To Understand More About Version Control : [What Is Version Control? | Git Version Control |](#)
[Version Control In Software Engineering](#)

About Version Control

What is “version control”, and why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

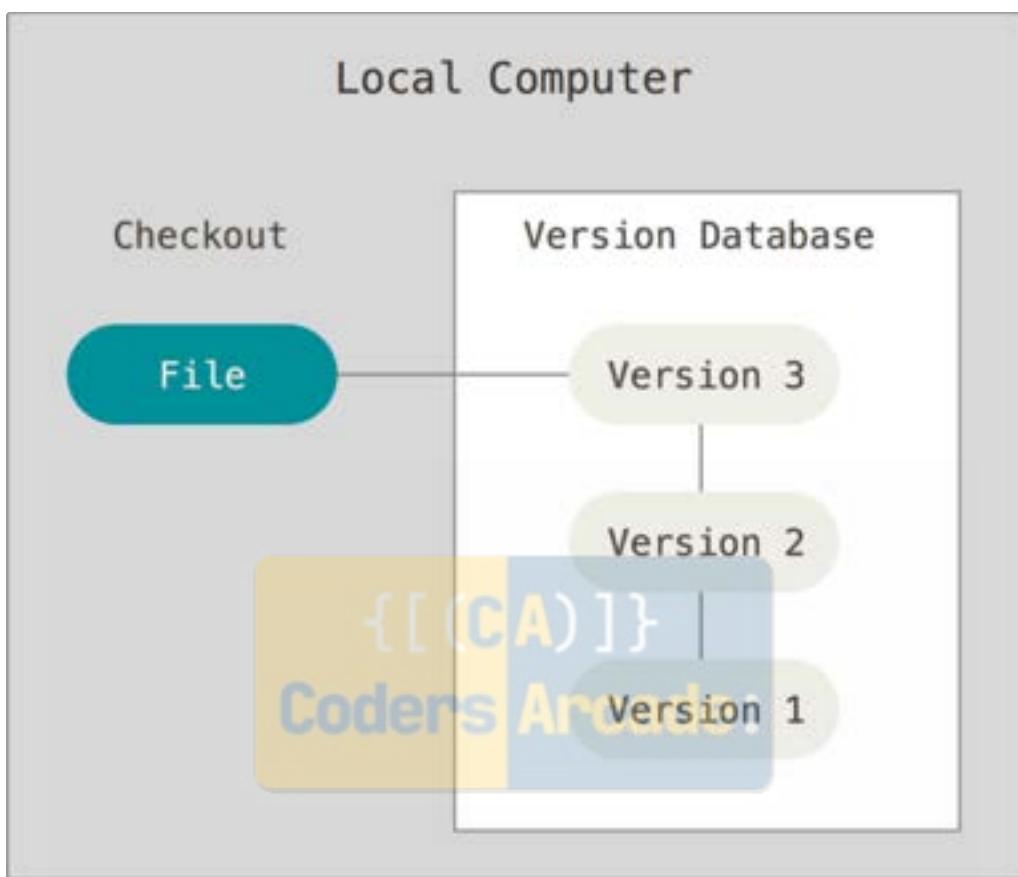
Local Version Control Systems

Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you’re in and accidentally write to the wrong file or copy over files you don’t mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

CODERS ARCADE

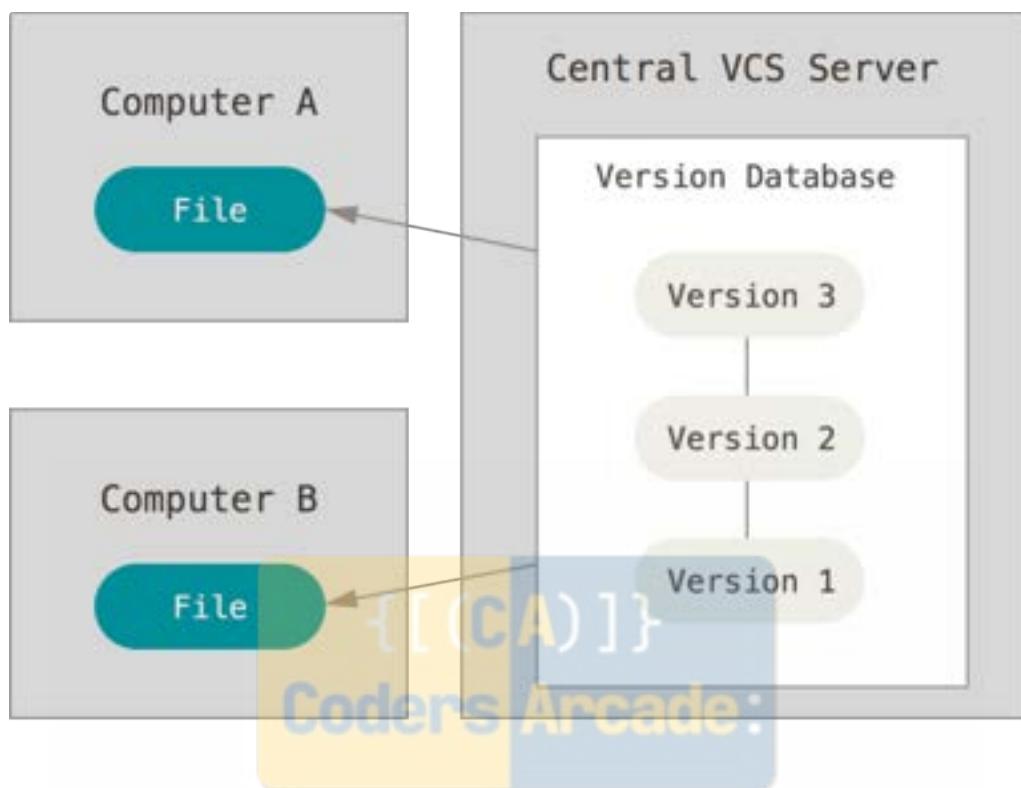
COMMIT TO ACHIEVE



Centralized Version Control Systems

The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.

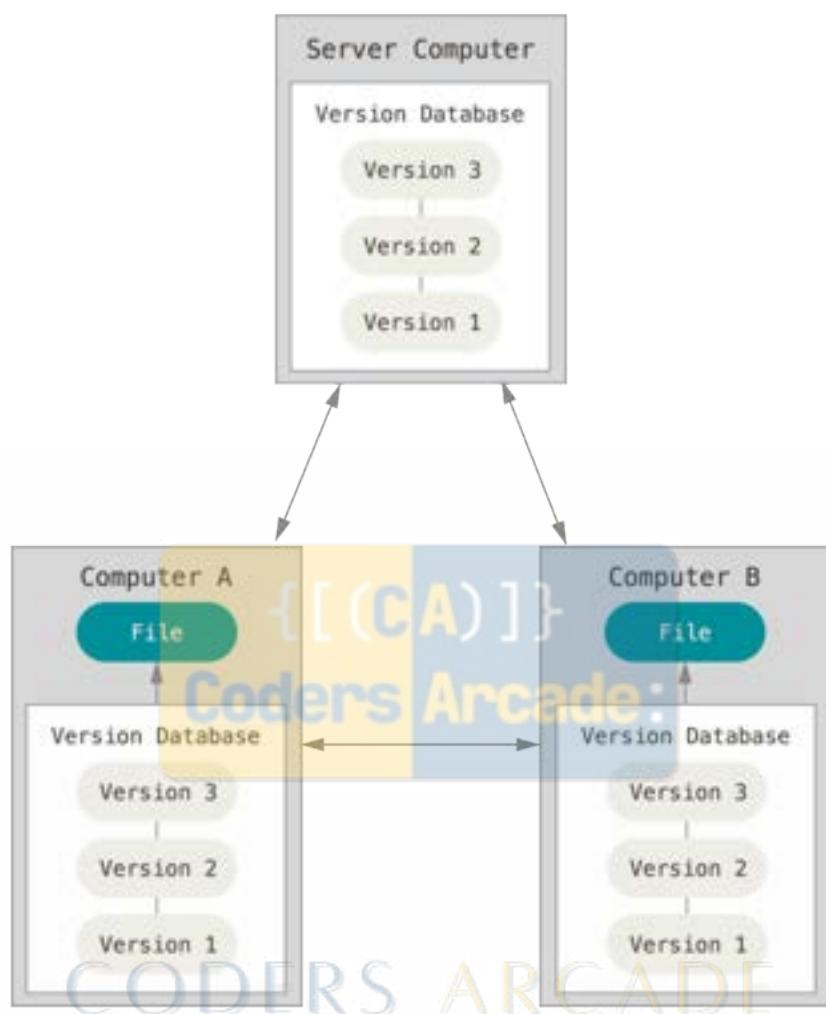
COMMIT TO ACHIEVE



Distributed Version Control Systems

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

COMMIT TO ACHIEVE



CVCS vs DVCS

Setting Up Git on Local System

Browse This **Playlist** To Learn **GIT & GITHUB** From Scratch : [Git Tutorial for Beginners](#)

Step 1: Check if git is already installed

```
git --version
```

Step 2 : If not installed download Git installer from <https://git-scm.com/>

Step 3 : Run installer and install git

Step 4 : Check if git is installed

```
git --version
```

Create GitHub Account

Step 1: Go to <https://github.com/> and sign up for a new account

Step 2 : Login to GitHub

Step 3 : Create a new Repository (Private or Public)

Basic Git Commands

Step 1: Create a new folder and open Git Bash/Cmd and go to the folder location

Step 2 : Run these commands for git configuration

```
git config --global user.email "sampleGitHub@email.com"
```

```
git config --global user.name "sampleGitHub_username"
```

Step 3 : Initialize git using this command

```
git init
```

Step 4 : Add some sample files in the folder by using this command

```
git touch <filename1.txt>
```

```
git touch <filename2.html>
```

```
git touch <filename3.py>
```

```
git touch <filename4.js>
```

Step 5 : Run these commands to check status, add files and commit your changes/updates

```
git status
```

[To check status]

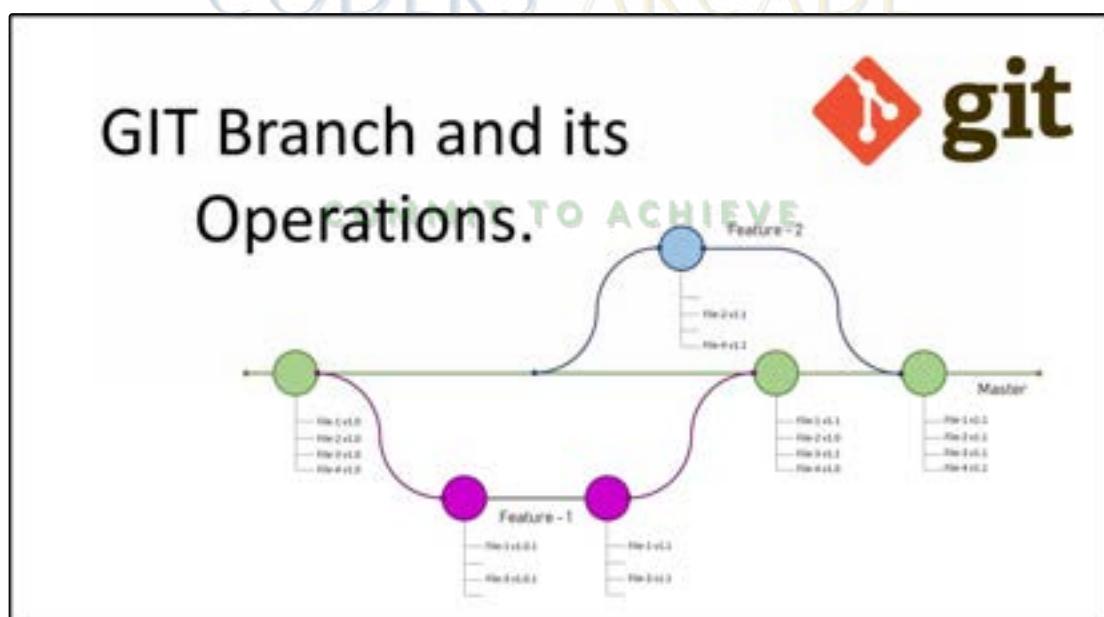
```
git add <filename>           [ To add a particular file ]
git add .                    [ To add all the files ]
git commit -m "Commit Message" [ To commit all your changes ]
git remote add origin "github-url" [ To add remote repo link to local repo]
git push -u origin master    [ To push your local changes to remote ]
Create a branch and add some files and make some changes and then add the branch
to remote by :
git push -u origin "branch-name" [ To push a particular branch to remote ]
git clone "github url"         [ To clone a GitHub Repo ]
```

Git Branches

💡 In this section you will be learning about :

- What are branches
- How to create a branch in git repository
- How to checkout to a particular branch in git repository
- How to merge a particular branch to the master branch in git repository
- How to delete a branch from local & remote repositories

- What are branches
 - Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository. You always create a branch from an existing branch. Typically, you might create a new branch from the default branch (**master**) of your repository.



- How to create a branch in git repository

- The “git branch” command can be used to create a new branch. When you want to start a new feature, you create a new branch off main using “`git branch new_branch`” .
- How to checkout to a particular branch in git repository
 - Once a branch is created you can then use “`git checkout new_branch`” to switch to that branch.
- How to merge a particular branch to the master branch in git repository
 - First we run `git checkout master` to change the active branch back to the master branch. Then we run the command `git merge new-branch` to merge the new feature into the master branch.

 **git merge merges the specified branch into the currently active branch. So we need to be on the branch that we are merging into.**

- How to delete a branch from local & remote repositories
 - To delete the local branch, just run the `git branch` command again, this time with the `-d (delete)` flag, followed by the name of the branch you want to delete.
 - You'll often need to delete a branch not only locally but also remotely. To do that, you use the following command: `git push <remote_name> --delete <branch_name>`. The branch still exists locally, if you haven't deleted it from your local repository.

Steps to follow while working with branches in Git

Step 1 : Create branch

`git branch "branch_name"`

Step 2 : Checkout branch

`git checkout "branch_name"`

Step 3 : Make some changes to your project

- Add files, commit, push `git push -u origin new_branch`
- Check if the branch is visible in GitHub repository

Step 4 : On local repo checkout to master branch `“git checkout master”`

Step 5 : Merge new branch in master branch `“git merge branch_name”`

Step 6 : Push all your changes

`“git push -u origin master”`

Step 7 : Delete a particular branch

- `git branch -d "branch_name"` // Will delete from local repository
- `git push origin --delete "branch_name"` // Will delete from remote repository

Git Tags

A Tag in Git is a **reference that points to a specific point in Git history**. Tagging is generally used to capture a point in history that is used for **marking a version release (for example : v1.0)**. A tag is like a branch that doesn't change. Unlike branches, tags, after being created, have no further history of commits.

Step 1 : Checkout the branch where you want to create the tag

- `git checkout "branch_name"` // Example : `git checkout master`

Step 2 : Create a tag with some name

- `git tag "tag_name"` // Example: `git tag v1.0`
- `git tag -a v1.1 -m "tag for release ver 1.1"` // Annotated Tag

Step 3 : Display or Show tags

- `git tag` // Show all the tags
- `git show v1.0` // Show a particular tag
- `git tag -l "v1.*"` // Show all tags starting with v1

Step 3 : Push tags to remote repository `git push origin v1.0`

- [push all tags to remote : `git push --tags`](#)

Step 4 : Delete tags (Only if required)

- To delete tags from local repository
 - `git tag -d v1.0`
 - `git tag --delete v1.0`
- To delete tags from remote repository
 - `git push origin -d v1.0`
 - `git push origin --delete v1.0`
 - `git push origin :v1.0`
- To delete multiple tags at one go
 - `git tag -d v1.0 v1.1` (local repository)
 - `git push origin -d v1.0 v1.1` (remote repository)

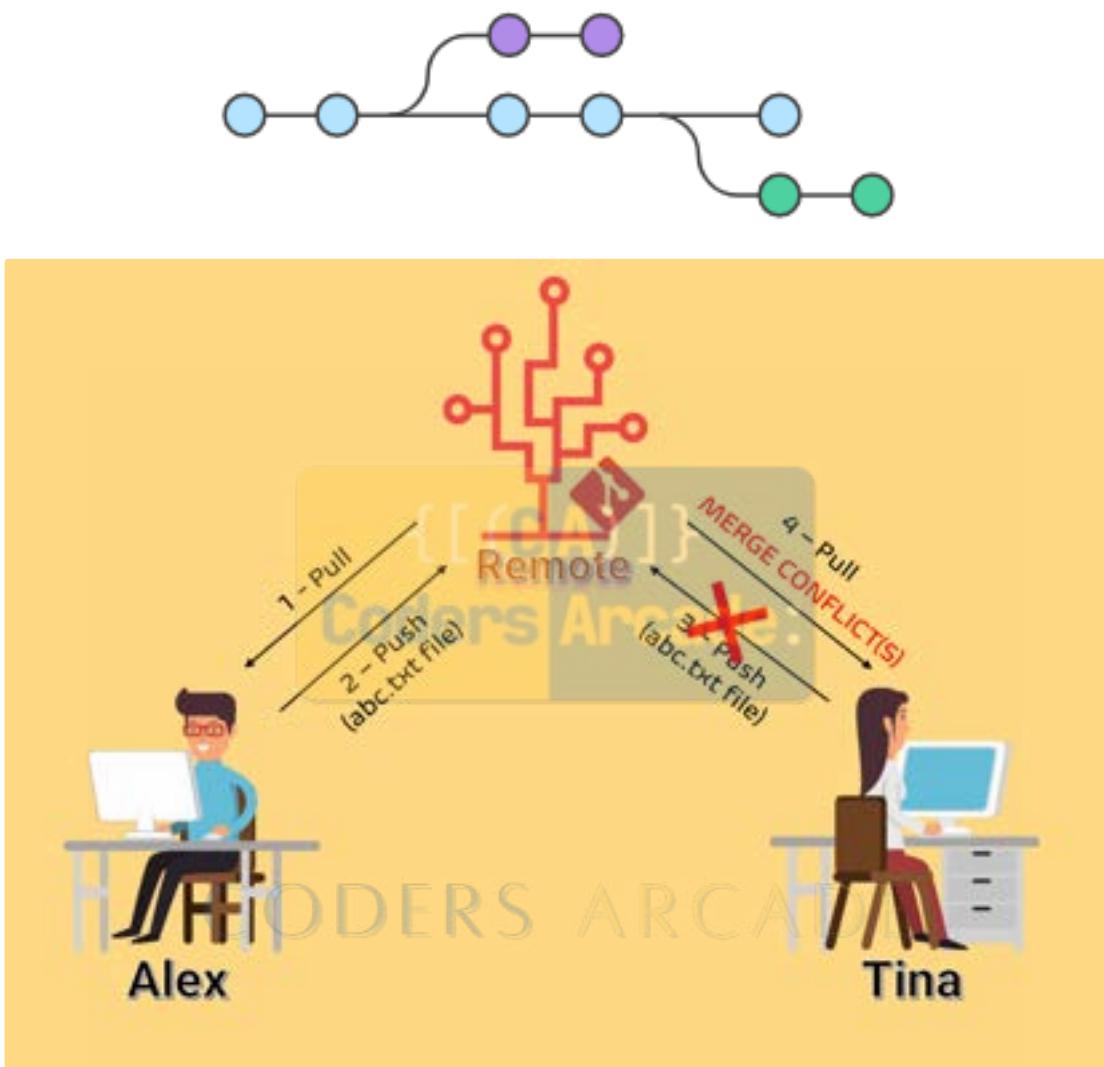
Checking out with tags

- We cannot checkout tags in Git
- We can create a branch from a tag and checkout the branch
 - `git checkout -b "branch_name" "tag_name"`
 - Example : `git checkout -b ReleaseVer1 v1.0`

Creating tags from past commits

- `git tag "tag_name" "reference of the past commit"`
- Example : `git tag v1.3 6gecf05`

Git Merge Conflicts



COMMIT TO ACHIEVE

- Version control systems are all about managing contributions between multiple distributed authors (usually developers).
- Sometimes multiple developers may try to edit the same content.
- If Developer A tries to edit code that Developer B is editing a conflict may occur.
- To alleviate the occurrence of conflicts developers will work in separate isolated branches.
- The `git merge` command's primary responsibility is to combine separate branches and resolve any conflicting edits.

Understanding merge conflicts

- Merging and conflicts are a common part of the Git experience.
- Conflicts in other version control tools like SVN can be costly and time-consuming.
- Git makes merging super easy.
- Most of the time, Git will figure out how to automatically integrate new changes.

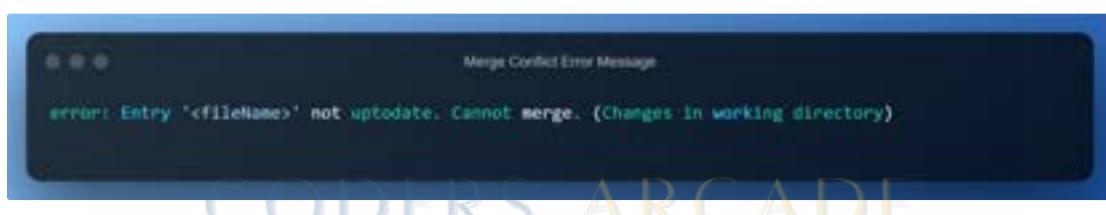
- Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it.
- In these cases, Git cannot automatically determine what is correct.
- Conflicts only affect the developer conducting the merge, the rest of the team is unaware of the conflict.
- Git will mark the file as being conflicted and halt the merging process.
- It is then the developers' responsibility to resolve the conflict.

Types of merge conflicts

A merge can enter a conflicted state at two separate points. When starting and during a merge process. The following is a discussion of how to address each of these conflict scenarios.

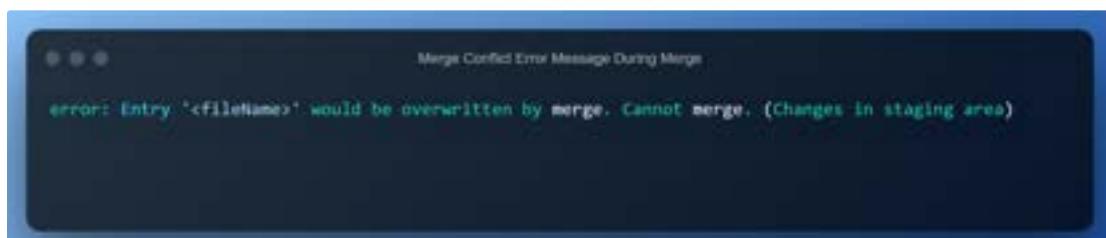
Git fails to start the merge

- A merge will fail to start when Git sees there are changes in either the working directory or staging area of the current project.
- Git fails to start the merge because these pending changes could be written over by the commits that are being merged in.
- When this happens, it is not because of conflicts with other developer's, but conflicts with pending local changes.
- The local state will need to be stabilized using `git stash`, `git checkout`, `git commit` or `git reset`. A merge failure on start will output the following error message:



Git fails during the merge

- A failure DURING a merge indicates a conflict between the current local branch and the branch being merged.
- This indicates a conflict with another developer's code.
- Git will do its best to merge the files but will leave things for you to resolve manually in the conflicted files.
- A mid-merge failure will output the following error message:



How to identify merge conflicts

- As we have experienced from the proceeding example, Git will produce some descriptive output letting us know that a **CONFLICT** has occurred.
- We can gain further insight by running the `git status` command as shown below :

```
Identifying A Merge Conflict

$ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified: merge.txt
```

- The output from `git status` indicates that there are unmerged paths due to a conflict.
- The `merge.txt` file now appears in a modified state.
- Below is a sample merge conflict example :

```
Merge Conflict

$ cat merge.txt
COMMIT TO ACHIEVE
<<<<< HEAD
this is some content to mess with
content to append
=====
totally different content to merge later
>>>>> new_branch_to_merge_later
```

How to Resolve Merge Conflicts in Git?

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

1. The easiest way to resolve a conflicted file is to open it and make any necessary changes

2. After editing the file, we can use the git add a command to stage the new merged content
3. The final step is to create a new commit with the help of the git commit command
4. Git will create a new merge commit to finalize the merge

Let us now look into the Git commands that may play a significant role in resolving conflicts.

Git Commands to Resolve Conflicts

1. git log --merge

The git log --merge command helps to produce the list of commits that are causing the conflict

2. git diff

The git diff command helps to identify the differences between the states repositories or files

3. git checkout

The git checkout command is used to undo the changes made to the file, or for changing branches

4. git reset --mixed

The git reset --mixed command is used to undo changes to the working directory and staging area

5. git merge --abort

The git merge --abort command helps in exiting the merge process and returning back to the state before the merging began

6. git reset

The git reset command is used at the time of merge conflict to reset the conflicted files to their original state

 Note : We can also use the [Git Merge Tool](#) for resolving Merge Conflicts. Merge Conflicts can be resolved in different ways based on the user's convenience. But, prior to that, one should be familiar with the [git commands](#) required to resolve [Merge Conflicts](#).

COMMIT TO ACHIEVE

CA - Git & GitHub Simplified For DevOps VTU Lab

Git & GitHub – A Beginner's Guide

Introduction

Git is a **distributed version control system (DVCS)** that helps in tracking changes in files and enables collaboration among developers.

Why Use Git?

- ✓ Keeps track of changes in files
- ✓ Allows you to revert to previous versions
- ✓ Enables collaboration in teams
- ✓ Helps in recovering lost files

◆ Understanding Git Workflow

A project in Git moves through **four key areas**:

- 1 Working Directory** – Where you edit files
- 2 Staging Area** – Where you mark files for commit
- 3 Local Repository** – Where Git stores committed versions
- 4 Remote Repository** – Where you push code to share with others

◆ Git Workflow Diagram



◆ Setting Up Git on Your System

Step 1: Check if Git is Installed

```

1 git --version
2

```

If not installed, download from git-scm.com and install it.

Step 2: Configure Git (One-Time Setup)

```
1 git config --global user.name "Your Name"
2 git config --global user.email "your-email@example.com"
3
```

◆ Basic Git Commands

1 Initialize a New Git Repository

```
1 git init
2
```

This creates a **hidden .git folder** in your project directory.

2 Check the Status of Your Files

```
1 git status
2
```

This shows which files are **untracked, modified, or staged**.

3 Add Files to the Staging Area

```
1 git add index.html      # Add a specific file
2 git add .                # Add all files
3
```

4 Commit Your Changes

```
1 git commit -m "Initial commit with project files"
2
```

Commits are **snapshots** of your project at different stages.

5 Connect to a Remote Repository (GitHub)

```
1 git remote add origin https://github.com/your-username/your-repo.git
2
```

6 Push Code to GitHub

```
1 git push -u origin master
2
```

COMMIT TO ACHIEVE

◆ Cloning a Repository (Downloading a Project)

To download a repository from GitHub:

```
1 git clone https://github.com/your-username/your-repo.git
2
```

◆ Summary of Common Git Commands

Command	Description
---------	-------------

<code>git init</code>	Initialize a new Git repository
<code>git status</code>	Check the status of your files
<code>git add <file></code>	Stage a specific file
<code>git add .</code>	Stage all files
<code>git commit -m "message"</code>	Save changes with a commit message
<code>git remote add origin <url></code>	Link local repo to GitHub
<code>git push -u origin master</code>	Upload changes to GitHub
<code>git clone <url></code>	Download a GitHub repository

⌚ Key Takeaways

- ✓ Git tracks your project changes efficiently
- ✓ Use `git add` to stage, `git commit` to save, and `git push` to upload
- ✓ GitHub allows easy collaboration and version control

🚀 Moving Forward: Build Tools (Maven & Gradle)

Now that we understand **Version Control with Git & GitHub**, we can move on to **build tools like Maven & Gradle** to manage dependencies and automate the build process efficiently.

CODERS ARCADE

COMMIT TO ACHIEVE



CA - Maven Notes & Documentation

Introduction to Maven

Maven is a popular open-source build tool that the **Apache** Group developed for building, publishing, and deploying several projects. Maven is written in Java and is used to create projects written in C#, Scala, Ruby, and so on. The tool is used to build and manage any Java-based project. It simplifies the day-to-day work of Java developers and helps them with various tasks.

▼ General Info from Apache Maven Site

i Maven, a [Yiddish word](#) meaning *accumulator of knowledge*, began as an attempt to simplify the build processes in the Jakarta Turbine project. There were several projects, each with their own Ant build files, that were all slightly different. JARs were checked into CVS. We wanted a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information, and a way to share JARs across several projects.

Maven is a powerful **build and project management tool** that is based on POM (project object model). It is used for projects build, dependency and documentation.

i What is a build tool?

A build tool is essential for the process of building, as these are the tools that automate the process of creating applications from the source code. The build tool compiles and packages the code into an executable form.

A build tool does the following tasks:

- Generates source code
- Generates documentation from the source code
- Compiles source code
- Packages the compiled codes into JAR files
- Installs the packaged code in the local repository, server, or central repository

💡 Why We Use Maven?

Let's us understand the problem we face without maven:

- Adding set of Jars in the Java Project : When creating any Java project, we need to manually download multiple jar files and their dependency jars and configure manually to build paths. Sharing and storing these projects are also heavy and time consuming.
- Creating right project structure : Deciding and creating right project structure is very important. Sometime because of wrong project structure, the project won't get executed.

- Building and Deploying the project: We have to manually build and deploy the project for it to work.

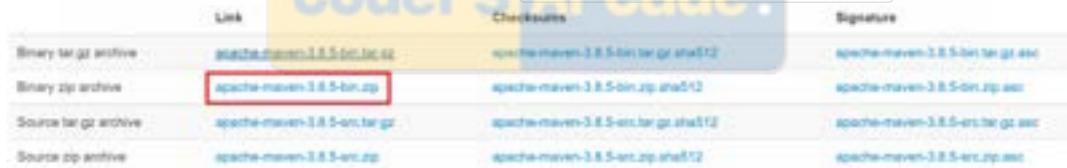
To solve all the above problems and automate the above process we use Maven and it performs following activities:

- Repository to get the dependencies.
- Having a similar folder structure across the organization.
- Integration with Continuous Integration tools like Jenkins.
- Plugins for test execution.
- It provides information on how the software/ project is being developed.
- The build process is made simpler and consistent.
- Provides guidelines for the best practices to be followed in the project.
- Enhances project performance.
- Easy to move to new attributes of Maven.
- Integration with version control tools like Git.

🛠️ Install Maven and Environment Setup

📘 Instructions to Install Maven and Setup

- Download latest maven “Binary zip archive” file from [Download Apache Maven – Maven](#)



Link	Checksums	Signature
Binary tar.gz archive	maven-maven-3.8.5-bin.tar.gz	maven-maven-3.8.5-bin.tar.gz.asc
Binary zip archive	apache-maven-3.8.5-bin.zip	apache-maven-3.8.5-bin.zip.asc
Source tar.gz archive	apache-maven-3.8.5-bin.tar.gz	apache-maven-3.8.5-bin.tar.gz.asc
Source zip archive	apache-maven-3.8.5-bin.zip	apache-maven-3.8.5-bin.zip.asc

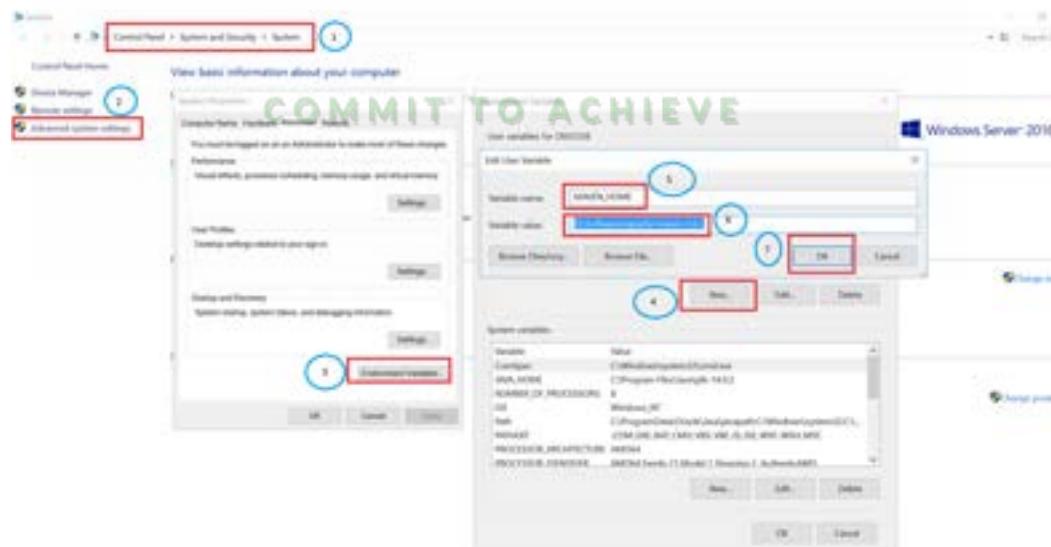
Maven Binary Zip File

- Unzip the content of the downloaded file and place it any desired location in your drive (either C or D).

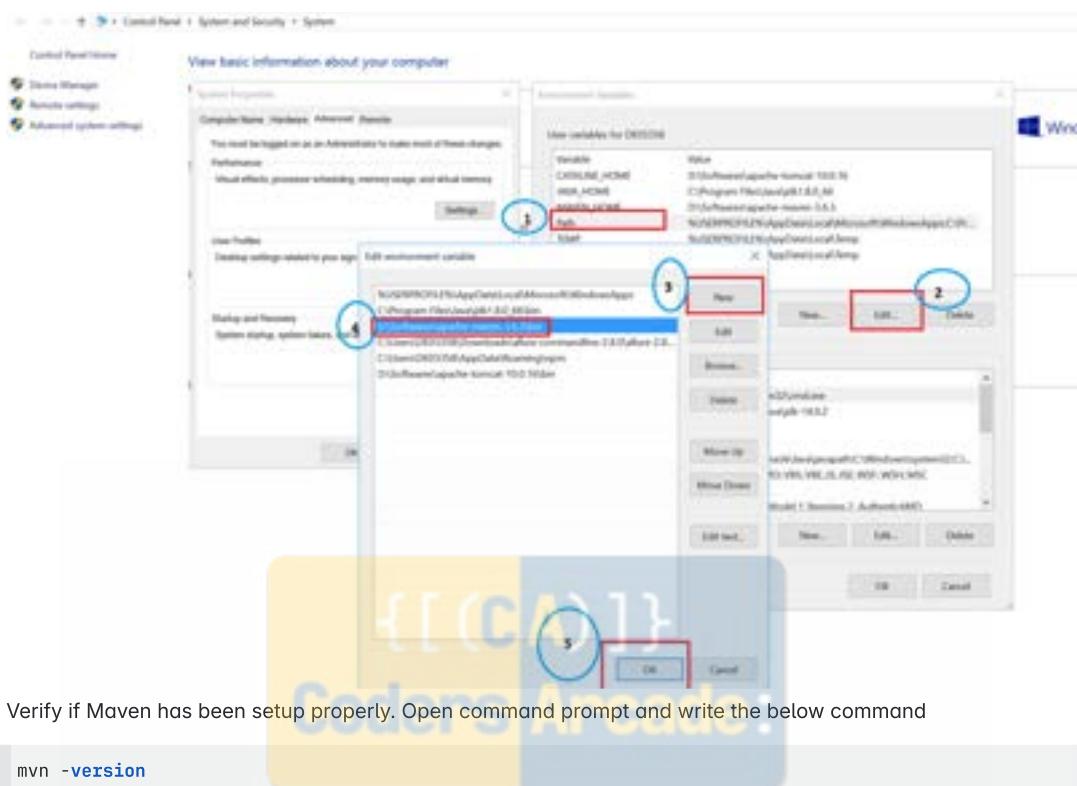
- Navigate inside the folder up to bin and copy the full path.

- Setting up Environment variables

- M2_HOME and MAVEN_HOME



- Editing “Path” environment variable and adding the maven directory path up until “bin” folder.



6. Verify if Maven has been setup properly. Open command prompt and write the below command

1 mvn -version

If Maven has been setup correctly then we will get following output of the above command:

```
C:\Users\... mvn -version
Apache Maven 3.6.3 (cecedad343002696d4abb59b32b541b8e6ba2883f)
maven home: D:\Softwares\apache-maven-3.6.3\bin\..
java version: 1.8.0_66, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_66\jre
default locale: en_AU, platform encoding: Cp1252
os name: "windows nt (unknown)", version: "10.0", arch: "amd64", family: "windows"
```

Maven Repository

A repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

- There are three types of maven repository
- Local
- Central
- Remote

Maven searches for dependency in following order



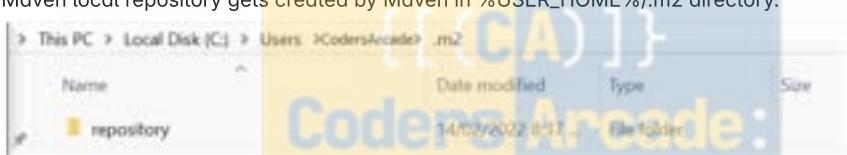
Dependency Search Order

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence –

- **Step 1** – Search dependency in local repository, if not found, move to step 2 else perform the further processing.
- **Step 2** – Search dependency in central repository, if not found and remote repository/repositories is/are mentioned then move to step 4. Else it is downloaded to local repository for future reference.
- **Step 3** – If a remote repository has not been mentioned, Maven simply stops the processing and throws error (Unable to find dependency).
- **Step 4** – Search dependency in remote repository or repositories, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

Local Repository

- Maven local repository is a folder location in developer's machine (local system). It gets created when we run any maven command for the first time.
- Maven local repository gets created by Maven in %USER_HOME%/.m2 directory.



- To change the default location, we can change the path in Maven settings.xml file available at %MAVEN_HOME%/conf directory. Enable the local repository and mention the correct path to your local repository.



- When the maven command is run for the first time, all the dependencies get downloaded from central and remote repository to the local repository. This helps to avoid references to dependencies stored on remote machine every time the project is build and saves time.

Central Repository

COMMIT TO ACHIEVE

- Maven central repository refers to the repository available on a web server provided by Maven community.
- It contains a large number of commonly used libraries.
- When maven doesn't find a dependency in our local repository, it starts searching in central repository using the URL: [Central Repository](#)
- Maven community has provided a website to search details of dependencies [Maven Central Repository Search](#)
- There is another website where we can search for all commonly used libraries <https://mvnrepository.com/>

- Maven central repository doesn't require any configuration
- Internet access is needed for searching the dependencies.

Remote Repository

Sometimes, there will be need of self developed or companies internal libraries which obviously won't be found in Maven central repository. In that case, maven stops the build process and throws the error message.

To prevent these situations, Maven provides concepts of Remote Repository, which is developer's own custom repository which contains required Jar files.

There are two ways of doing this

1. A little dirty way of doing this is creating a directory in the project and providing the path of the jar in the pom as shown below:

```

1 <dependency>
2   <groupId>com.sample</groupId>
3   <artifactId>samplifact</artifactId>
4   <version>1.0</version>
5   <scope>system</scope>
6   <systemPath>C:\DEV\myfunnylib\yourJar.jar</systemPath>
7 </dependency>
```

2. The other (proper) way of achieving remote repository is to move your internal Jar file to an archive location (some where on internet).

```

1 <project xmlns = "http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
4   http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>com.companyname.projectgroup</groupId>
7   <artifactId>project</artifactId>
8   <version>1.0</version>
9   <dependencies>
10    <dependency>
11      <groupId>com.companyname.common-lib</groupId>
12      <artifactId>common-lib</artifactId>
13      <version>1.0.0</version>
14    </dependency>
15   <dependencies>
16   <repositories>
17     <repository>
18       <id>companyname.lib1</id>
19       <url>http://download.companyname.org/maven2/lib1</url>
20     </repository>
21     <repository>
22       <id>companyname.lib2</id>
23       <url>http://download.companyname.org/maven2/lib2</url>
24     </repository>
25   </repositories>
26 </project>
```

Maven – POM

- Project Object Model (POM) refers to the XML files with all the information regarding project and configuration details.
- It contains the project description, as well as details regarding the versioning and configuration management of the project.
- The XML file is in the project home directory. Maven searches for the POM in the current directory when any given task needs to be executed.

- Some of the configuration that can be specified in the POM are following –
 - project dependencies
 - plugins
 - goals
 - build profiles
 - project version
 - developers
 - mailing list
- POM Example:

```

1 <project xmlns = "http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
4   http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.companyname.project-group</groupId>
8   <artifactId>project</artifactId>
9   <version>1.0</version>
10  </project>
```

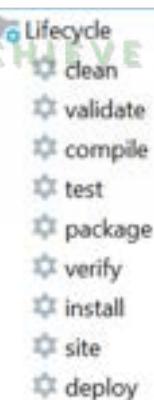
- i**
- groupId** : This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
 - artifactId** : This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.
 - version** : This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example –
 - com.company.bank:consumer-banking:1.0
 - com.company.bank:consumer-banking:1.1.

Maven Build Life Cycle

Maven has three standard life cycles

- clean
- default/build
- site

COMMIT TO ACHIEVE



Clean Life Cycle

The clean life cycle handles project cleaning. The clean phase consists of the following three steps:

- Pre-clean

- Clean
- Post-clean

The mvn post-clean command is used to invoke the clean lifestyle phases in Maven. When `mvn clean` command executes, Maven deletes the build directory.

Default/Build Life Cycle

Default/Build Life Cycle contains these most important life cycle.

- *validate* – checks the correctness of the project
- *compile* – compiles the provided source code into binary artifacts
- *test* – executes unit tests
- *package* – packages compiled code into an archive file
- *integration-test* – executes additional tests, which require the packaging
- *verify* – checks if the package is valid
- *install* – installs the package file into the local Maven repository
- *deploy* – deploys the package file to a remote server or repository



Site Life Cycle

The Maven site plugin handles the project site's documentation, which is used to create reports, deploy sites, etc.

The phase has four different stages:

- Pre-site
- Site
- Post-site
- Site-deploy

Ultimately, the site that is created contains the project report.

Manage Dependencies

One of the core features of Maven is Dependency Management. Managing dependencies is a difficult task once we've to deal with multi-module projects (consisting of hundreds of modules/sub-projects). Maven provides a high degree of control to manage such scenarios.

Transitive Dependencies Discovery

It is pretty often a case, when a library, say A, depends upon other library, say B. In case another project C wants to use A, then that project requires to use library B too.

Maven helps to avoid such requirements to discover all the libraries required. Maven does so by reading project files (pom.xml) of dependencies, figure out their dependencies and so on.

We only need to define direct dependency in each project pom. Maven handles the rest automatically.

Dependency Scope

Scope & Description	
compile	This scope indicates that dependency is available in classpath of project. It is default scope.
provided	This scope indicates that dependency is to be provided by JDK or web-Server/Container at runtime.
runtime	This scope indicates that dependency is not required for compilation, but is required during execution.
test	This scope indicates that the dependency is only available for the test compilation and execution phases.
system	This scope indicates that you have to provide the system path.
import	This scope is only used when dependency is of type pom. This scope indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section.

Maven Plugins

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to –

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

CODERS ARCADE

COMMIT TO ACHIEVE

Command to execute any plugin is

```
1 mvn [plugin-name]:[goal-name]
```

There are two types of maven plugins according to Apache Maven,

1. Build Plugins
2. Reporting Plugins

Build Plugins

These plugins are executed at the time of build. These plugins should be declared inside the <build> element.

Reporting Plugins

These plugins are executed at the time of site generation. These plugins should be declared inside the <reporting> element.

Maven Core Plugins

A list of maven core plugins are given below:

Plugin	Description
clean	clean up after build.
compiler	compiles java source code.
deploy	deploys the artifact to the remote repository.
failsafe	runs the JUnit integration tests in an isolated classloader.
install	installs the built artifact into the local repository.
resources	copies the resources to the output directory for including in the JAR.
site	generates a site for the current project.
surefire	runs the JUnit unit tests in an isolated classloader.
verifier	verifies the existence of certain conditions. It is useful for integration tests.

- To see the list of maven plugins, you may visit apache maven official website
<http://repo.maven.apache.org/maven2/org/apache/maven/plugins/>

CODERS ARCADE

COMMIT TO ACHIEVE

CA - Experiment 1 - Introduction To Maven & Gradle Build Tools

Experiment 1: Introduction to Maven and Gradle

Objective

To understand build automation tools, compare **Maven** and **Gradle**, and set up both tools for software development.

1. Overview of Build Automation Tools

Build automation tools simplify the process of **compiling, testing, packaging, and deploying** software projects. They manage dependencies, execute tasks, and integrate seamlessly with CI/CD pipelines.

Why Use Build Automation?

-  Ensures **consistency** across builds
-  Handles **dependency management** automatically
-  Reduces **manual errors** and increases efficiency
-  Integrates with tools like **Jenkins & Azure DevOps**

Popular Build Automation Tools

- **Apache Ant** → Script-based, manual dependency management
- **Apache Maven** → XML-based, convention-over-configuration model
- **Gradle** → Flexible, fast, and supports Groovy/Kotlin DSL

2. Key Differences Between Maven and Gradle

Feature	 Maven (XML)	 Gradle (Groovy/Kotlin)
Build Script	<code>pom.xml</code>	<code>build.gradle</code> / <code>build.gradle.kts</code>
Performance	Sequential, slower	Parallel execution, faster
Flexibility	Convention-based	Highly customizable
Dependency Mgmt.	Uses Maven Repository	Supports multiple repositories
Ease of Use	Simple XML structure	Slightly complex but powerful
Caching Support	No build caching	Supports incremental builds
Best For	Standard Java projects	Complex, high-performance builds

 **Maven** is great for structured, **enterprise-level** projects.

 **Gradle** is ideal for **scalable and performance-driven** applications.

◆ 3. Installation and Setup

● 3.1 Installing Maven

✓ Step 1: Install Java (JDK 17 Recommended)

Check if Java is installed:

```
1 java -version
2 javac -version
3
```

✓ Step 2: Download and Install Maven

[🔗 Download from: !\[\]\(2027f710942186bc2d02193053a49d40_img.jpg\) Download Apache Maven – Maven](#)

 Extract it to a folder (e.g., `C:\Maven`).

✓ Step 3: Configure Environment Variables

📌 Windows:

- Add `MAVEN_HOME` → `C:\Maven\apache-maven-<version>`
- Update `Path` → `%MAVEN_HOME%\bin`

📌 Linux/macOS:

Add the following to `.bashrc` or `.zshrc`:

```
1 export MAVEN_HOME=/opt/maven/apache-maven-<version>
2 export PATH=$MAVEN_HOME/bin:$PATH
3
```

✓ Step 4: Verify Installation

Run:

```
1 mvn -version
2
```

Expected Output:

```
1 Apache Maven 3.x.x
2 Maven home: C:\Maven\apache-maven-<version>
3 Java version: 17.0.4
4
```

● 3.2 Installing Gradle

✓ Step 1: Install Java (JDK 17 Recommended)

Same steps as Maven.

✓ Step 2: Download and Install Gradle

[🔗 Download from: !\[\]\(706ba7576b12931a568b824454f224e5_img.jpg\) Gradle | Releases](#)

 Extract it to a folder (e.g., `C:\Gradle`).

✓ Step 3: Configure Environment Variables

📌 Windows:

- Add `GRADLE_HOME` → `C:\Gradle\gradle-<version>`
- Update `Path` → `%GRADLE_HOME%\bin`

 **Linux/macOS:**

Add the following to `.bashrc` or `.zshrc`:

```
1 export GRADLE_HOME=/opt/gradle/gradle-<version>
2 export PATH=$GRADLE_HOME/bin:$PATH
3
```

 **Step 4: Verify Installation**

Run:

```
1 gradle -v
2
```

Expected Output:

```
1 Gradle 8.x
2 Build time: YYYY-MM-DD HH:MM:SS
3 Kotlin: X.Y
4 Groovy: X.Y
5
```

 **Assessment Questions**

- 1** What are the **key advantages** of using a build automation tool?
- 2** Mention **three** major differences between Maven and Gradle.
- 3** How does Gradle achieve **faster build times** compared to Maven?
- 4** What are the **necessary environment variables** for setting up Maven and Gradle?
- 5** How do you **verify** that Maven and Gradle are installed correctly?

CODERS ARCADE

COMMIT TO ACHIEVE



CA - Experiment 2 - Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins

Maven: Basic Introduction

Maven is a build automation tool primarily used for Java projects. It simplifies the build process, manages project dependencies, and supports plugins for different tasks. It uses XML files (called `pom.xml`) for project configuration and dependency management.

Key Benefits of Maven:

- Dependency management (automates downloading and including libraries).
- Build automation (compiles code, runs tests, creates artifacts).
- Consistent project structure (standardizes how Java projects are set up).
- Integration with CI tools (like Jenkins).

i Key Features of Maven:

- **Project Management:** Handles project dependencies, configurations, and builds.
- **Standard Directory Structure:** Encourages a consistent project layout across Java projects.
- **Build Automation:** Automates tasks such as compilation, testing, packaging, and deployment.
- **Dependency Management:** Downloads and manages libraries and dependencies from repositories (e.g., Maven Central).
- **Plugins:** Supports many plugins for various tasks like code analysis, packaging, and deploying.

Steps to Create a Maven Project in IntelliJ IDEA

1. Install Maven (if not already installed):

- Download Maven from the [official website](#).
- Set the `MAVEN_HOME` environment variable and update the system `PATH`.

2. Create a New Maven Project:

- Open IntelliJ IDEA.
- Go to `File > New > Project`.
- Select `Maven` from the project types.
- Choose `Create from Archetype` (optional) or proceed without.
- Set the project name and location, then click `Finish`.

3. Set Up the `pom.xml` File:

- The `pom.xml` file is where you define dependencies, plugins, and other configurations for your Maven project.
- Example of a basic `pom.xml`:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

3         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4 4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.example</groupId>
8     <artifactId>simple-project</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11    <dependencies>
12        <!-- Add your dependencies here -->
13    </dependencies>
14
15 </project>

```

4. Add Dependencies for Selenium and TestNG:

- In the `pom.xml`, add Selenium and TestNG dependencies under the `<dependencies>` section.

```

1 <dependencies>
2     <dependency>
3         <groupId>org.seleniumhq.selenium</groupId>
4         <artifactId>selenium-java</artifactId>
5         <version>3.141.59</version>
6     </dependency>
7     <dependency>
8         <groupId>org.testng</groupId>
9         <artifactId>testng</artifactId>
10        <version>7.4.0</version>
11        <scope>test</scope>
12    </dependency>
13 </dependencies>
14

```

5. Create a Simple Website (HTML, CSS, and Logo):

- In the `src/main/resources` folder, create an `index.html` file, a `style.css` file, and place the `logo.png` image.

Example of a simple `index.html`:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>My Simple Website</title>
7     <link rel="stylesheet" href="style.css">
8 </head>
9 <body>
10    <header>
11        
12    </header>
13    <h1>Welcome to My Simple Website</h1>
14 </body>
15 </html>
16

```

Example of a simple `style.css`:

```

1 body {
2     font-family: Arial, sans-serif;
3     background-color: #f4f4f4;
4     text-align: center;

```

```

5 }
6 header img {
7     width: 100px;
8 }
9

```

6. Upload the Website to GitHub:

- Initialize a Git repository in your project folder:

```

1 git init
2

```

- Add your files and commit them:

```

1 git add .
2 git commit -m "Initial commit"
3

```

- Create a GitHub repository and push the local project to GitHub:

```

1 git remote add origin <your-repository-url>
2 git push -u origin master
3

```

Deployment :

To deploy your Maven project to **GitHub Pages** using the `/docs` folder (** **having all files inside root folder/dir not recommended**), you can follow these simple steps. This method is easy and doesn't require switching branches—just use the `/docs` folder of your main branch.

Steps to Deploy to GitHub Pages Using `/docs` Folder

- 1. Modify Maven Configuration to Copy Static Files to `/docs` Folder:** First, you need to ensure that Maven places your static files (`index.html`, `style.css`, `logo.png`) into the `/docs` folder instead of the `target` directory.

To do this, configure the **Maven Resources Plugin** in your `pom.xml` to copy the files directly into `/docs`:

```

1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.apache.maven.plugins</groupId>
5       <artifactId>maven-resources-plugin</artifactId>
6       <version>3.2.0</version>
7       <executions>
8         <execution>
9           <phase>prepare-package</phase> <!-- Before packaging -->
10          <goals>
11            <goal>copy-resources</goal>
12          </goals>
13          <configuration>
14            <outputDirectory>${project.basedir}/docs</outputDirectory> <!-- Deploy to
15 /docs folder -->
16            <resources>
17              <resource>
18                <directory>src/main/resources</directory>
19                <includes>
20                  <include>**/*</include> <!-- Copy all files in
src/main/resources -->
21                </includes>
22

```

```

21          </resource>
22      </resources>
23  </configuration>
24  </execution>
25  </executions>
26  </plugin>
27  </plugins>
28</build>
29

```

In this configuration:

- The `maven-resources-plugin` copies all files from `src/main/resources` to the `/docs` folder in the root of your project (not the `target` directory).
- This is done during the `prepare-package` phase, just before Maven prepares the package.

2. Build the Project: Run the following Maven command to build your project and copy the resources to the `/docs` folder:

```

1 mvn clean install
2

```

After this, your `index.html`, `style.css`, and `logo.png` files should now be inside the `docs` folder in the root of your project.

3. Push Changes to GitHub: Now that the files are in the `/docs` folder, they are ready to be served by GitHub Pages.

Follow these steps:

- **Stage the changes** (the updated `/docs` folder):

```

1 git add docs/*
2 git commit -m "Deploy site to GitHub Pages"
3

```

- **Push to GitHub:**

```

1 git push origin master # Or the branch you are using, maybe 'main' in some cases
2

```

4. Enable GitHub Pages: After pushing to the main branch, follow these steps to enable GitHub Pages:

- Go to your GitHub repository.
- Navigate to **Settings > Pages** (on the left sidebar).
- Under the **Source** section, select the `main` branch and `/docs` folder as the source.
- Click **Save**.

5. Access Your Website: Your static website is now hosted on GitHub Pages! You can access it at:

```

1 https://<your-github-username>.github.io/<your-repository-name>/
2

```

Summary:

- **Maven Resources Plugin** is configured to copy static files (`index.html`, `style.css`, `logo.png`) into the `/docs` folder.
- **Build and push** the changes to your GitHub repository.
- Enable **GitHub Pages** using the `/docs` folder as the source.

By using the `/docs` folder in the main branch, you avoid the complexities of working with a separate `gh-pages` branch, and it's easier to maintain your website alongside your project.

Information about the /docs folder

The `/docs` folder does **not need to be manually created**. When you configure the Maven build to copy resources to the `/docs` folder, Maven will automatically create this folder when you run the `mvn clean install` command (if it doesn't already exist).

Here's a clearer breakdown:

1. The `/docs` Folder:

- This folder will be automatically created during the build process when Maven runs the `maven-resources-plugin`.
- Maven will copy the contents from `src/main/resources` into the `/docs` folder.

2. No Need for Manual Creation:

You don't need to manually create the `/docs` folder. Maven handles this automatically based on the configuration in your `pom.xml` file.

After the Build:

Once the build completes, check the root of your project directory, and you should see a `docs` folder with all the copied files inside it.

Summary:

The `/docs` folder is not something you need to create manually. Maven will generate it automatically and place the static files inside it based on your configuration in the `pom.xml` file. After building and pushing the changes to GitHub, you can enable GitHub Pages to serve the content from this folder.

Concise step-by-step guide to deploy your Maven project to GitHub Pages using the `/docs` folder, assuming you already have necessary files inside it.

Here's a concise step-by-step guide to deploy your Maven project to GitHub Pages using the `/docs` folder, assuming you already have your `index.html`, `style.css`, and other necessary files:

1. Prepare Your Maven Project

- Open your `pom.xml` file and configure the `maven-resources-plugin` to copy your static files (like `index.html`, `style.css`, etc.) into the `/docs` folder:

```

1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.apache.maven.plugins</groupId>
5       <artifactId>maven-resources-plugin</artifactId>
6       <version>3.2.0</version>
7       <executions>
8         <execution>
9           <phase>prepare-package</phase>
10          <goals>
11            <goal>copy-resources</goal>
12          </goals>
13          <configuration>
14            <outputDirectory>${project.basedir}/docs</outputDirectory>
15            <resources>
16              <resource>
17                <directory>src/main/resources</directory>
18                <includes>
19                  <include>**/*</include>
20                </includes>
21              </resource>

```

```

22         </resources>
23     </configuration>
24   </execution>
25 </executions>
26 </plugin>
27 </plugins>
28 </build>
29

```

This ensures that all your static files from `src/main/resources` are copied into the `docs` folder.

2. Build the Project

Run Maven to build the project:

```

1 mvn clean install
2

```

This will generate the `/docs` folder and place your static files there.

3. Push Changes to GitHub

- Stage and commit the changes (i.e., the new `/docs` folder with your static files):

```

1 git add docs/*
2 git commit -m "Deploy site to GitHub Pages"
3

```

- Push to your repository (assuming you're working with the `master` branch):

```

1 git push origin master
2

```

4. Enable GitHub Pages

- Go to your **GitHub repository**.
- Navigate to **Settings > Pages** (in the sidebar).
- Under the **Source** section, select:
 - Branch:** `main`
 - Folder:** `/docs`
- Click **Save**.



COMMIT TO ACHIEVE

5. Access Your Website

Your website will now be live at:

```

1 https://<your-github-username>.github.io/<your-repository-name>/
2

```

Summary:

- Configure Maven:** Set up `maven-resources-plugin` in `pom.xml` to copy files to `/docs`.
- Build the Project:** Run `mvn clean install` to generate the `/docs` folder.
- Push to GitHub:** Stage and commit the `/docs` folder, then push to the `main` branch.
- Enable GitHub Pages:** Configure GitHub Pages to use the `/docs` folder.
- Access:** Your site will be hosted on GitHub Pages at `https://<your-username>.github.io/<repo-name>/`.

7. Write a Simple Selenium Test with TestNG:

- Create a new Java class `WebPageTest.java` in the `src/test/java` directory.

Example of a simple TestNG test using Selenium:

```

1 package org.test;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.chrome.ChromeDriver;
5 import org.testng.Assert;
6 import org.testng.annotations.AfterTest;
7 import org.testng.annotations.BeforeTest;
8 import org.testng.annotations.Test;
9
10 import static org.testng.Assert.assertTrue;
11
12 public class WebpageTest {
13     private static WebDriver driver;
14
15     @BeforeTest
16     public void openBrowser() throws InterruptedException {
17         driver = new ChromeDriver();
18         driver.manage().window().maximize();
19         Thread.sleep(2000);
20         driver.get("https://sauravssarkar-codersarcade.github.io/CA-MVN/"); // "Note: You should
use your GITHUB-URL here...!!!"
21     }
22
23     @Test
24     public void titleValidationTest(){
25         String actualTitle = driver.getTitle();
26         String expectedTitle = "Tripillar Solutions";
27         Assert.assertEquals(actualTitle, expectedTitle);
28         assertTrue(true, "Title should contain 'Tripillar'");
29     }
30
31     @AfterTest
32     public void closeBrowser() throws InterruptedException {
33         Thread.sleep(1000);
34         driver.quit();
35     }
36 }
```

8. Run the Test:

- In IntelliJ, right-click the `WebPageTest` class and select `Run 'WebPageTest'`.
- The test will launch Chrome, open the webpage, and validate the title.

Summary of Steps:

- Set up Maven project and configure `pom.xml`.
- Create a simple website with HTML, CSS, and a logo image.
- Upload the project to GitHub.
- Write and run a Selenium test with TestNG to validate the webpage title.

 **Testing** the title of your website using **Selenium**, **Java**, and **TestNG**:

To test the title of your website using **Selenium**, **Java**, and **TestNG**, follow these steps. This will include the installation of necessary dependencies, creating test scripts, and running tests.

1. Set Up Selenium and TestNG Dependencies

In your Maven project, add the necessary dependencies for **Selenium WebDriver** and **TestNG** to the `pom.xml` file: (i
Skip if already added..!!)

```

1 <dependencies>
2     <!-- Selenium WebDriver dependency -->
3     <dependency>
4         <groupId>org.seleniumhq.selenium</groupId>
5         <artifactId>selenium-java</artifactId>
6         <version>4.8.0</version> <!-- Ensure this is the latest version -->
7     </dependency>
8
9     <!-- TestNG dependency -->
10    <dependency>
11        <groupId>org.testng</groupId>
12        <artifactId>testng</artifactId>
13        <version>7.7.0</version> <!-- Ensure this is the latest version -->
14        <scope>test</scope>
15    </dependency>
16 </dependencies>
17

```

- **Selenium WebDriver:** This is used for browser automation.
- **TestNG:** This is a testing framework used to run Selenium tests.

2. Create Selenium Test Class Using TestNG

Next, create a test class in your `src/test/java` directory. You can name it `WebsiteTitleTest.java`.

Sample Code for Testing Website Title:

```

1 package org.test;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.chrome.ChromeDriver;
5 import org.testng.Assert;
6 import org.testng.annotations.AfterTest;
7 import org.testng.annotations.BeforeTest;
8 import org.testng.annotations.Test;
9
10 import static org.testng.Assert.assertTrue;
11
12 public class WebpageTest {
13     private static WebDriver driver;
14
15     @BeforeTest
16     public void openBrowser() throws InterruptedException {
17         driver = new ChromeDriver();
18         driver.manage().window().maximize();
19         Thread.sleep(2000);
20         driver.get("https://sauravsharkar-codersarcade.github.io/CA-MVN/"); // "Note: You should use
your GITHUB-URL here...!!!"
21     }
22
23     @Test
24     public void titleValidationTest(){
25         String actualTitle = driver.getTitle();

```

```

26     String expectedTitle = "Tripillar Solutions"; // "Replace with Your HTML WebPage Title"
27     Assert.assertEquals(actualTitle, expectedTitle);
28 }
29
30 @AfterTest
31 public void closeBrowser() throws InterruptedException {
32     Thread.sleep(1000);
33     driver.quit();
34 }
35
36 }
37

```

Code Explanation (Step-by-Step)

Package Declaration

- package org.test;
 - Defines the package name as `org.test` (helps organize code).

Imports Required Libraries

- `import org.openqa.selenium.WebDriver;` → Selenium WebDriver interface.
- `import org.openqa.selenium.chrome.ChromeDriver;` → Controls Google Chrome.
- `import org.testng.Assert;` → Provides assertion methods for validation.
- `import org.testng.annotations.*;` → TestNG annotations for test execution.
- `import static org.testng.Assert.assertTrue;` → Allows direct usage of `assertTrue()`.

Class Declaration

- `public class WebpageTest {}` → Defines a **test class** for webpage testing.

WebDriver Declaration

- `private static WebDriver driver;`
 - Declares a static WebDriver instance for browser control.

1 @BeforeTest - Setup Method

```

1 @BeforeTest
2 public void openBrowser() throws InterruptedException {
3

```

- Runs **before any test case** in this class.
- Initializes `ChromeDriver()` (opens Chrome).
- Maximizes the browser window.
- Waits for **2 seconds** (`Thread.sleep(2000)`).
- Navigates to "<https://sauravssarkar-codersarcade.github.io/CA-MVN/>".

2 @Test - Title Validation Test

```

1 @Test
2 public void titleValidationTest() {
3

```

- Retrieves the **actual title** of the web page using `driver.getTitle()`.
- Defines the **expected title** as "Tripillar Solutions".

- Uses `Assert.assertEquals(actualTitle, expectedTitle);`
 - **Passes** if the title matches "Tripillar Solutions".
 - **Fails** if the title is different.

3 @AfterTest - Cleanup Method

```
1 @AfterTest
2 public void closeBrowser() throws InterruptedException {
3
```

- Runs **after all test cases** in this class.
- Waits for **1 second** (`Thread.sleep(1000)`).
- Closes the browser using `driver.quit();`.

Key Takeaways

- Selenium WebDriver** automates browser interaction.
- TestNG Annotations** manage test setup (`@BeforeTest`), execution (`@Test`), and teardown (`@AfterTest`).
- Assertions** (`Assert.assertEquals()`) validate webpage content.

3. Run the Test Using TestNG

Option 1: Run TestNG from IntelliJ IDEA

1. Right-click the `WebpageTest.java` file.
2. Select **Run WebpageTest**.
 - IntelliJ IDEA will execute the TestNG test and show the results in the output console.

Option 2: Run TestNG via Command Line

If you want to run the tests from the command line, use the following Maven command:

```
1 mvn test
2
```

This will trigger the TestNG tests and execute all the tests in the `src/test/java` directory.

4. Verify the Test Results

After running the test, TestNG will output the results:

- If the website title matches the expected title, the test will pass.
- If the title is incorrect, the test will fail, and you will get an assertion error with a message indicating that the title doesn't match.

Summary of Steps:

1. Add Dependencies:

- Add `selenium-java`, `testng`, and `webdrivermanager` dependencies in `pom.xml`.

2. Write Test Class:

- Create a class `WebpageTest.java` with methods to launch the browser, navigate to the website, and verify the title.

3. Run the Tests:

- Use IntelliJ IDEA or Maven to run the tests and check the output for validation.

4. Verify Results:

- Ensure that the actual title matches the expected title. If they match, the test passes; if not, it fails.

This approach tests whether the website is correctly deployed and whether the title is as expected on your GitHub Pages site.

 Use of **maven-jar-plugin** to **package** the project into a **JAR file**. | | Showing how to **run a main class** and show simple **output**:

Great! You're on the right track. Here's a quick summary of what we have covered and additional steps you can take to demonstrate the use of **maven-jar-plugin** and how to package the project into a JAR file, including running a **main class** and showing simple output.

Summary of What We've Done So Far:

1. Website Deployment:

- You've deployed your simple HTML, CSS, and assets (like logo.png) to **GitHub Pages** using Maven.
- You've created a **Selenium Test** to validate the website title using **TestNG** and ran the test to ensure the website is functioning as expected.

2. Next Steps:

- We can show how to use the **maven-jar-plugin** to create a runnable JAR file.
- Demonstrate running a **main class** inside this JAR to produce a simple output.

Steps to Package the Project as a JAR and Run a Main Class

1. Add **maven-jar-plugin** to **pom.xml**:

To package your Maven project as a JAR file and specify the **main class**, we need to configure the **maven-jar-plugin** in the **pom.xml**.

Add the following configuration to your **pom.xml**:

```

1 <build>
2   <plugins>
3     <!-- Maven JAR Plugin -->
4     <plugin>
5       <groupId>org.apache.maven.plugins</groupId>
6       <artifactId>maven-jar-plugin</artifactId>
7       <version>3.2.0</version>
8       <configuration>
9         <!-- Specify the main class to be executed -->
10        <archive>
11          <manifestEntries>
12            <Main-Class>com.example.MainClass</Main-Class> <!-- Replace with your main
13            class path -->
14          </manifestEntries>
15        </archive>
16      </configuration>
17    </plugin>
18  </plugins>
19 </build>

```

This will tell Maven to include the **Main-Class** in the JAR manifest and specify the main class that should be executed when the JAR is run.

2. Create a Main Class:

In your `src/main/java` directory, create a class with a `main` method. For example, create a `MainClass.java` under `com.example`:

```
1 package com.example;
2
3 public class MainClass {
4     public static void main(String[] args) {
5         System.out.println("Hello, this is a simple output from the main class!");
6     }
7 }
8
```

This class contains a simple `main` method that prints output when run.

3. Package the Project into a JAR:

After configuring the plugin and creating the `MainClass`, run the following Maven command to build the project and package it into a JAR file:

```
1 mvn clean package
2
```

This will clean any previous builds, compile the source code, and package it into a JAR file located in the `target` directory (e.g., `target/your-project-name.jar`).

4. Run the JAR File:

Once the JAR is created, you can run it with the following command:

```
1 java -jar target/your-project-name.jar
2
```

This will execute the `main` method from your `MainClass` and print the message:

```
1 Hello, this is a simple output from the main class!
2
```

Summary:

- Add maven-jar-plugin:** Configure the plugin in `pom.xml` to specify the `main class`.
- Create Main Class:** Write a simple `MainClass` with a `main` method that outputs a message.
- Package with Maven:** Run `mvn clean package` to package the project into a JAR file.
- Run the JAR:** Use `java -jar target/your-project-name.jar` to run the packaged JAR and print the output.

Final Thoughts:

- By showing this process, you demonstrate how Maven is used to automate the build, test, deployment, and packaging of projects.
- The `maven-jar-plugin` allows you to easily create runnable JAR files, and running the `main` class from the JAR file is a common way to run Java applications.

This provides a good conclusion to the topic, demonstrating not only the deployment and testing of the website but also the power of Maven for packaging and managing Java projects.

Important Note :

The **maven-resources-plugin** that we used earlier for copying files (like HTML, CSS, images) to the `/docs` folder doesn't get replaced or removed when you configure the **maven-jar-plugin** for packaging the project. These two plugins serve different purposes and can coexist in your `pom.xml`.

How They Work Together:

1. **maven-resources-plugin**: This plugin is responsible for copying your resources (like HTML, CSS, images, etc.) to the appropriate location in the target directory (e.g., `/docs` or `/target/classes`). We used it to ensure that all static assets were copied to the correct folder for GitHub Pages deployment.
2. **maven-jar-plugin**: This plugin handles the packaging of your Java code into a JAR file. It creates the JAR with the specified resources, including the class files and any configured files (like your `MainClass`), but it doesn't interfere with the static files used for GitHub Pages.

Key Points:

- **maven-resources-plugin** will still be used to copy your static resources to the `/docs` folder as configured.
- **maven-jar-plugin** will only package your Java classes and any other necessary resources for the JAR file.
- Both plugins can run independently during the build process, so there's no conflict between them.

Example Setup for Both Plugins:

Here's how both plugins would coexist in your `pom.xml`:

```

1 <build>
2   <plugins>
3     <!-- maven-resources-plugin for copying static resources (HTML, CSS, images) -->
4     <plugin>
5       <groupId>org.apache.maven.plugins</groupId>
6       <artifactId>maven-resources-plugin</artifactId>
7       <version>3.2.0</version>
8       <executions>
9         <execution>
10          <phase>process-resources</phase>
11          <goals>
12            <goal>copy-resources</goal>
13          </goals>
14          <configuration>
15            <outputDirectory>${project.build.directory}/docs</outputDirectory>
16            <resources>
17              <resource>
18                <directory>src/main/resources</directory>
19                <includes>
20                  <include>**/*</include>
21                </includes>
22                </resource>
23              </resources>
24            </configuration>
25          </execution>
26        </executions>
27      </plugin>
28
29      <!-- maven-jar-plugin for packaging the project into a JAR -->
30      <plugin>
31        <groupId>org.apache.maven.plugins</groupId>
32        <artifactId>maven-jar-plugin</artifactId>
33        <version>3.2.0</version>
34        <configuration>
35          <archive>
36            <manifestEntries>
```

```

37         <Main-Class>com.example.MainClass</Main-Class> <!-- Replace with your main
38     class path -->
39         </manifestEntries>
40     </archive>
41     </configuration>
42   </plugin>
43 </plugins>
44 </build>

```

When You Run `mvn clean package`:

1. **maven-resources-plugin** will copy the static files from `src/main/resources` to the `/docs` folder.
2. **maven-jar-plugin** will package your project's Java classes into a JAR file and add the necessary **Main-Class** entry to the JAR's manifest.

Conclusion:

You don't need to worry about the **maven-resources-plugin** being overridden or replaced. Both plugins work independently, and you can use them together in the same build process to handle different aspects of your project (static resources for GitHub Pages and packaging the Java code into a JAR).

Maven Build Lifecycle Explained :

Here's a **clear, concise, and easy-to-understand** documentation on the **Maven Build Lifecycle**:

Introduction

Maven follows a structured build lifecycle, consisting of a sequence of predefined phases. These phases automate tasks like compiling code, running tests, packaging, and deploying the project.

Maven's Three Built-in Lifecycles

1. **Clean Lifecycle** – Cleans the project.
2. **Default (Build) Lifecycle** – Handles project compilation, testing, packaging, and deployment.
3. **Site Lifecycle** – Generates project documentation.

Each lifecycle has a sequence of phases, which execute in order.

1. Clean Lifecycle

Used to remove old build files before a new build.

Phases:

- `pre-clean` → Executes tasks before cleaning.
- `clean` → Deletes the `/target` directory (removes compiled files).
- `post-clean` → Executes tasks after cleaning.

Command to Run:

```

1 mvn clean
2

```

(This removes all compiled files and resets the build environment.)

2. Default (Build) Lifecycle

This is the **main lifecycle** that compiles, tests, and packages the project.

Phases & Explanation:

1. **validate** → Ensures project structure and configuration are correct.
2. **compile** → Compiles the source code.
3. **test** → Runs unit tests using **JUnit/TestNG** (does not require deployment).
4. **package** → Packages the compiled code into a deployable format (e.g., JAR/WAR).
5. **verify** → Runs integration tests to check if the package is valid.
6. **install** → Installs the package into the local repository (`~/.m2/repository`).
7. **deploy** → Uploads the package to a remote repository (e.g., Nexus, Artifactory).

Commands to Run Each Phase:

```

1 mvn validate      # Check project structure
2 mvn compile       # Compile the source code
3 mvn test          # Run unit tests
4 mvn package        # Create JAR/WAR file
5 mvn verify         # Run integration tests
6 mvn install        # Install JAR to local repository
7 mvn deploy         # Deploy to remote repository
8

```

Note: Running `mvn package` will also execute **validate, compile, and test** (because earlier phases are executed automatically).

3. Site Lifecycle

This lifecycle generates project documentation.

Phases:

- `pre-site` → Prepares documentation.
- `site` → Generates site documentation.
- `post-site` → Finalizes site generation.
- `site-deploy` → Deploys documentation to a web server.

Command to Run:

```

1 mvn site
2

```

(This generates a project documentation site inside `target/site`.)

Summary Table:

Lifecycle	Phase	Description
Clean	<code>clean</code>	Deletes previous build files.
Build	<code>validate</code>	Ensures project correctness.

	<code>compile</code>	Compiles Java code.
	<code>test</code>	Runs unit tests.
	<code>package</code>	Creates JAR/WAR.
	<code>verify</code>	Runs integration tests.
	<code>install</code>	Installs JAR to local repo.
	<code>deploy</code>	Deploys to remote repo.
Site	<code>site</code>	Generates documentation.
	<code>site-deploy</code>	Deploys documentation.

Conclusion

Maven's lifecycle ensures an automated, structured build process. Running any phase also executes all previous phases automatically, making builds efficient and repeatable.

For daily use, the most common commands are:

```
1 mvn clean package    # Clean & build the project
2 mvn clean install   # Clean, build & install in local repo
3 mvn deploy          # Deploy to a remote repository
4
```

Now, you have a **simple and complete** understanding of Maven's lifecycle! 🎉

📌 Maven site & deploy Commands - Documentation

🚀 1. `mvn site` Command

The `mvn site` command is used to **generate a project website** containing reports like dependencies, build details, test results, and more.

👉 Steps to Use `mvn site`

📝 Step 1: Add Site Plugin in `pom.xml`

Before running the `site` command, you need to add the **Maven Site Plugin** inside the `<build>` section of your `pom.xml`:

```
1 <build>
2   <plugins>
3     <plugin>
4       <groupId>org.apache.maven.plugins</groupId>
5       <artifactId>maven-site-plugin</artifactId>
6       <version>3.12.1</version> <!-- Use latest version -->
7     </plugin>
8   </plugins>
9 </build>
10
```

Step 2: Run the Site Command

Once the plugin is added, execute:

```
1 mvn site
2
```

What Happens?

- Maven scans your project for available reports.
- Generates an **HTML-based website** inside `target/site/`.
- Includes various reports like dependencies, plugin management, and test results.

Step 3: Open the Generated Site

After successful execution, open the following file in a browser:

```
1 D:\Idea Projects\CA-MVN\target\site\index.html
2
```

You'll See Reports Like:

- ✓ Project Summary
- ✓ Dependencies Report
- ✓ Plugin Management
- ✓ Unit Test Results (if configured)
- ✓ Code Coverage (if applicable)



2. mvn deploy Command

The `mvn deploy` command is used to **upload the built artifact (JAR, POM, etc.)** to a repository for distribution and sharing.

Steps to Use `mvn deploy`

Since you don't have a remote repository, we will configure a **local repository**.

Step 1: Create a Local Repository

```
1 mkdir D:\my-local-maven-repo
2
```

COMMIT TO ACHIEVE

Step 2: Configure `pom.xml` for Local Deployment

Add the following inside `<project>` in `pom.xml`:

```
1 <distributionManagement>
2   <repository>
3     <id>local-repo</id>
4     <url>file:///D:/my-local-maven-repo</url>
5   </repository>
6 </distributionManagement>
7
```

Step 3: Run the Deploy Command

```
1 mvn deploy
2
```

What Happens?

- Maven **builds** the project.
- Stores the artifact (JAR, POM, etc.) in `D:/my-local-maven-repo`.

Step 4: Verify Deployment

Navigate to `D:/my-local-maven-repo/` and check if the project is stored correctly:

```
1 D:/my-local-maven-repo/
2   └── org/example/CA-MVN/1.0-SNAPSHOT/
3     ├── CA-MVN-1.0-SNAPSHOT.jar
4     └── CA-MVN-1.0-SNAPSHOT.pom
5
```

Conclusion

- ✓ Use `mvn site` to generate a project website with reports.
- ✓ Use `mvn deploy` to store artifacts in a local or remote repository.
- ✓ Ensure you configure the **Maven Site Plugin** and **Distribution Management** in `pom.xml` before running these commands.

Now, you're all set to **document and deploy** your Maven project efficiently! 

 Optional : [Adding to Remote Repository](#) || [Out Of Syllabus](#) || [Just For Information](#)

You can deploy your Maven artifacts (JARs, POMs, etc.) to **GitHub Packages** as a remote repository. 

GitHub Packages acts as a **private Maven repository**, and you can deploy artifacts using **Maven Deploy Plugin** with authentication.



Steps to Deploy a Maven Project to GitHub Packages

1. Create a GitHub Repository

- Go to **GitHub** → **Create a new repository**
- Name it something like `maven-repo`
- **DO NOT initialize** with a `README`, `.gitignore`, or license.
- Copy your **GitHub Username** and **Personal Access Token (PAT)** (for authentication).

2. Modify `pom.xml` for GitHub Deployment

Add the **GitHub repository** under `<distributionManagement>` in your `pom.xml`:

```
1 <distributionManagement>
2   <repository>
```

```

3      <id>github</id>
4      <url>https://maven.pkg.github.com/YOUR_GITHUB_USERNAME/maven-repo</url>
5  </repository>
6 </distributionManagement>
7

```

 Replace `YOUR_GITHUB_USERNAME` with your actual GitHub username.

3. Configure Authentication (`settings.xml`)

You need to add **GitHub authentication credentials** inside Maven's `settings.xml` file (found in `C:\Users\YourUser\.m2\` on Windows or `~/.m2/` on Linux/macOS).

Add the following inside `<servers>`:

```

1 <server>
2   <id>github</id>
3   <username>YOUR_GITHUB_USERNAME</username>
4   <password>YOUR_GITHUB_PERSONAL_ACCESS_TOKEN</password>
5 </server>
6

```

 Replace `YOUR_GITHUB_PERSONAL_ACCESS_TOKEN` with a **GitHub Personal Access Token (PAT)** that has `read:packages` and `write:packages` permissions.

4. Run the Deploy Command

Now, deploy your Maven project using:

```

1 mvn deploy
2

```

What Happens?

- Maven builds the project.
- Uploads the artifact (`JAR` , `POM` , etc.) to **GitHub Packages**.
- You can now **use this package in other projects!** 

5. Use Your GitHub Package in Another Maven Project

To **use the deployed package in another project**, add this to the new project's `pom.xml`:

```

1 <repositories>
2   <repository>
3     <id>github</id>
4     <url>https://maven.pkg.github.com/YOUR_GITHUB_USERNAME/maven-repo</url>
5   </repository>
6 </repositories>
7
8 <dependencies>
9   <dependency>
10    <groupId>com.example</groupId>
11    <artifactId>YOUR_ARTIFACT_NAME</artifactId>
12    <version>1.0-SNAPSHOT</version>
13  </dependency>
14 </dependencies>
15

```

Now, your Maven project will **download the dependency from GitHub Packages** instead of Maven Central. 

Conclusion

- ✓ You **can deploy** Maven artifacts to **GitHub Packages**.
- ✓ Requires **GitHub authentication** (username + PAT).
- ✓ Can be **used in other projects** like a private Maven repository.



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Gradle Notes & Documentation

Introduction to Gradle

Gradle is a **powerful open-source build automation tool** used for developing, testing, deploying, and publishing software applications. It was created by **Hans Dockter, Szczepan Faber, Adam Murdoch, Luke Daley, Peter Niederwieser, Daz DeBoer, and Rene Gröschke** around **13 years ago** as an improvement over **Apache Ant** and **Apache Maven**. 

Unlike its predecessors, **Gradle supports multiple programming languages** including **Java, Kotlin, Groovy, Scala, and C++**, making it a **versatile and industry-standard build tool**. It is used for tasks ranging from **compilation and dependency management to packaging and deployment**.

History of Gradle

Gradle was first introduced in **late 2007** as a **more stable and flexible alternative to Apache Ant and Maven**. It addressed many of their limitations while **enhancing performance and scalability**.

- Its **first stable version** was released in **2019**.
- The latest version (as of the last update) is **Gradle 6.6**.

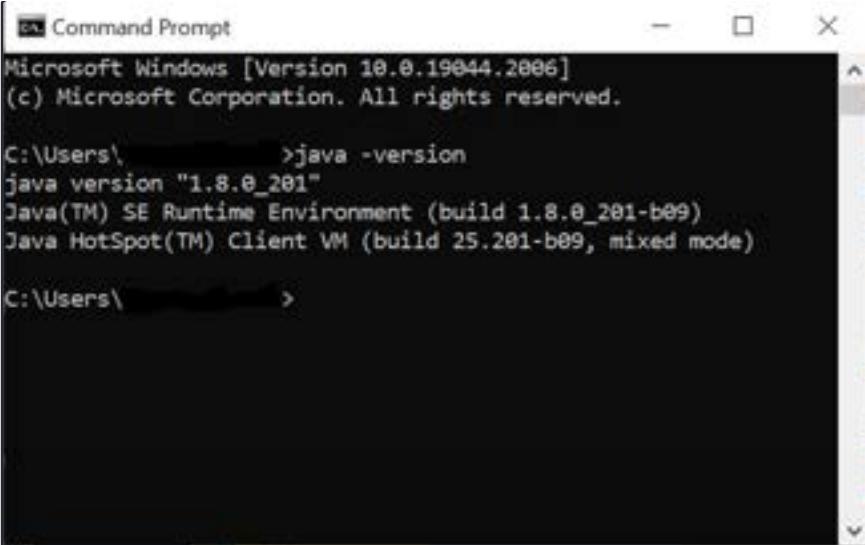
Gradle has now become a **popular choice for developers** due to its **speed, flexibility, and powerful dependency management system**.

How to Install Gradle on Windows?

Gradle is a flexible build automation tool to build software. It is open-source, and also capable of building almost any type of software. A build automation tool automates the build process of applications. The Software building process includes compiling, linking, and packaging the code, etc. Gradle makes the build process more consistent. Using Gradle we can build **Android, Java, Groovy, Kotlin JVM, Scala, C++, Swift** Libraries, and Applications.

Prerequisites to Install Gradle

To install Gradle, we have to make sure that we have Java JDK version 8 or higher to be installed on our operating system because Gradle runs on major operating systems only Java Development Kit version 8 or higher. To confirm that we have JDK installed on our system. The '**Java -version**' command is used to verify if you run it on a windows machine.



```
Microsoft Windows [Version 10.0.19044.2886]
(c) Microsoft Corporation. All rights reserved.

C:\Users\          >java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) Client VM (build 25.201-b09, mixed mode)

C:\Users\          >
```

You should see output like in the above image it confirms that you have JDK installed and the JAVA_HOME path set properly if you don't have JDK Install. then you should install it first before going to the next step. in this tutorial, we are not covering the installation of JDK. we consider that we have JDK already on the system.

Installing Gradle Manually

Gradle Enterprise provides the installation of Gradle with package manager [SDKMAN!](#) but here we are covering the manual installation of Gradle for Microsoft windows machines. before starting we confirmed we have JDK installed properly on the machine.

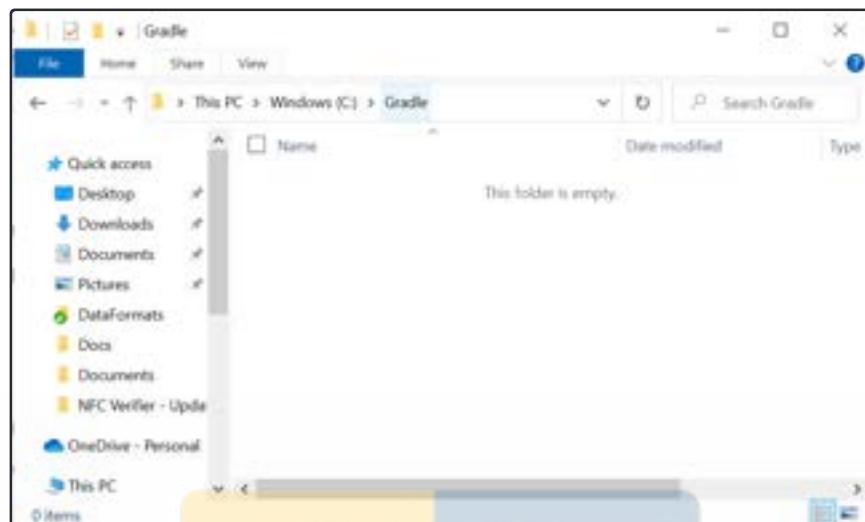
Step 1. Download the latest Gradle distribution

To install Gradle we need the Gradle distribution ZIP file, we can [download](#) the latest version of Gradle from the official website. there are other versions of Gradle available but we will strongly suggest using the official. The current release of Gradle is version v8.12.1, released on Jan 24, 2025. it might be another latest release available when you read this article. Always download the latest version. The Gradle distribution zip file is available in two flavors:

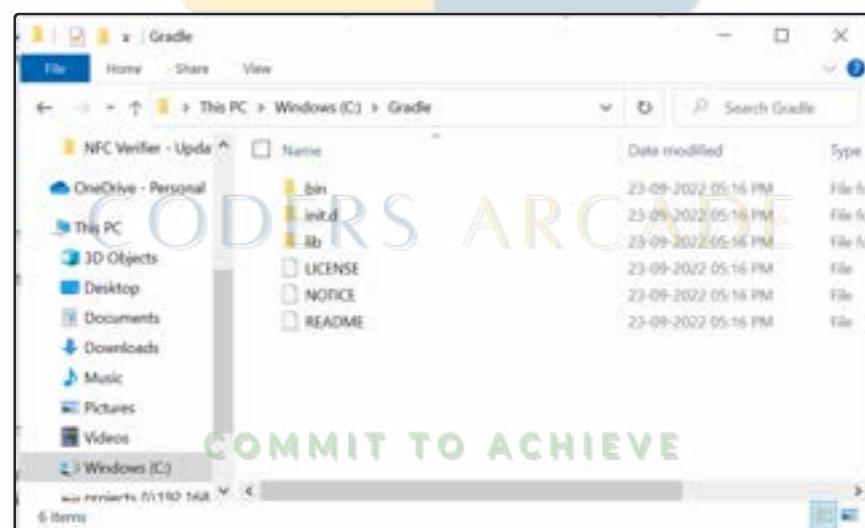
1. [Binary-only](#): Binary version contains only executable files no source code and documentation. You can [browse](#) docs, and [sources](#) online.
2. [Complete, with docs and sources](#): This package contains docs and sources offline. I recommend going with binary-only.

Step 2. Unpack the Gradle distribution zip file

After successfully downloading the Gradle distribution ZIP file, we need to unzip/unpack the compressed file. In File Explorer, create a new directory **C:\Gradle** (you can choose any path according to your choice) as shown In the image.



Now unpack the Gradle distribution ZIP into **C:\Gradle** using an archiver tool of your choice.



Step 3. Configure your system environment

We have successfully unzipped the Gradle distribution Zip file. Now set the System environment **PATH** variable to the bin directory of the unzipped distribution. We have extracted files in the following location. you can choose any location to unzip:

C:\Gradle\bin

To set the Environment variable, Right-click on the “**This PC /Computer**” icon, then select **Properties → Advanced System Settings → Environmental Variables**. The following screenshot may help you.

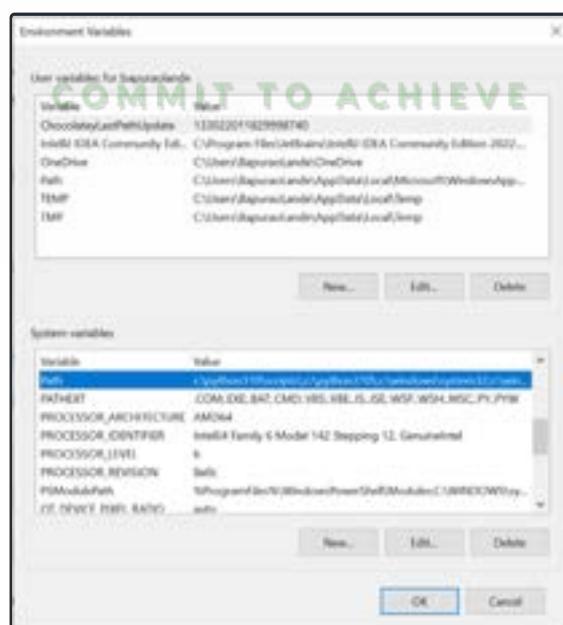
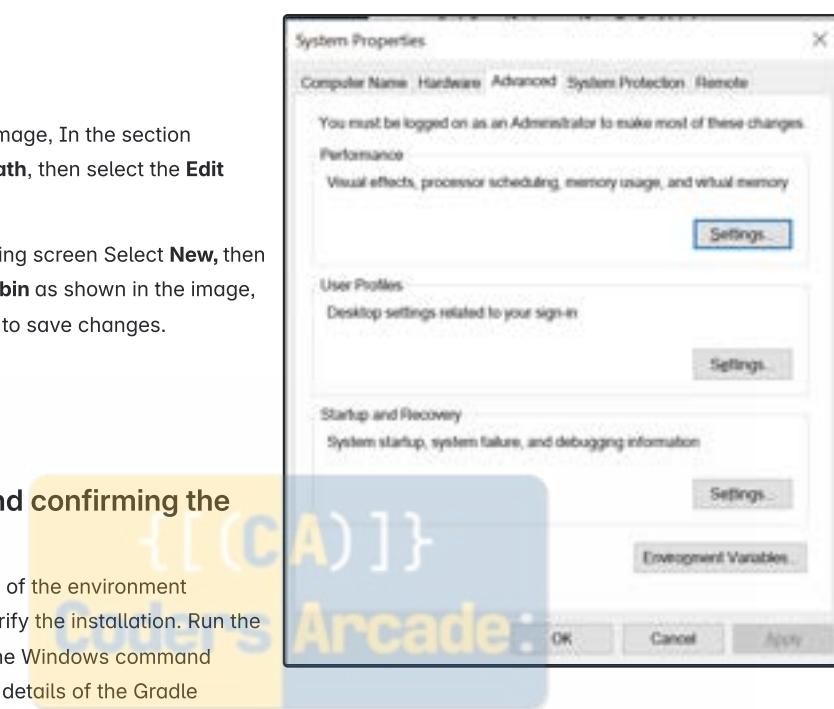
As shown in the following image, In the section **System Variables** select **Path**, then select the **Edit** option.

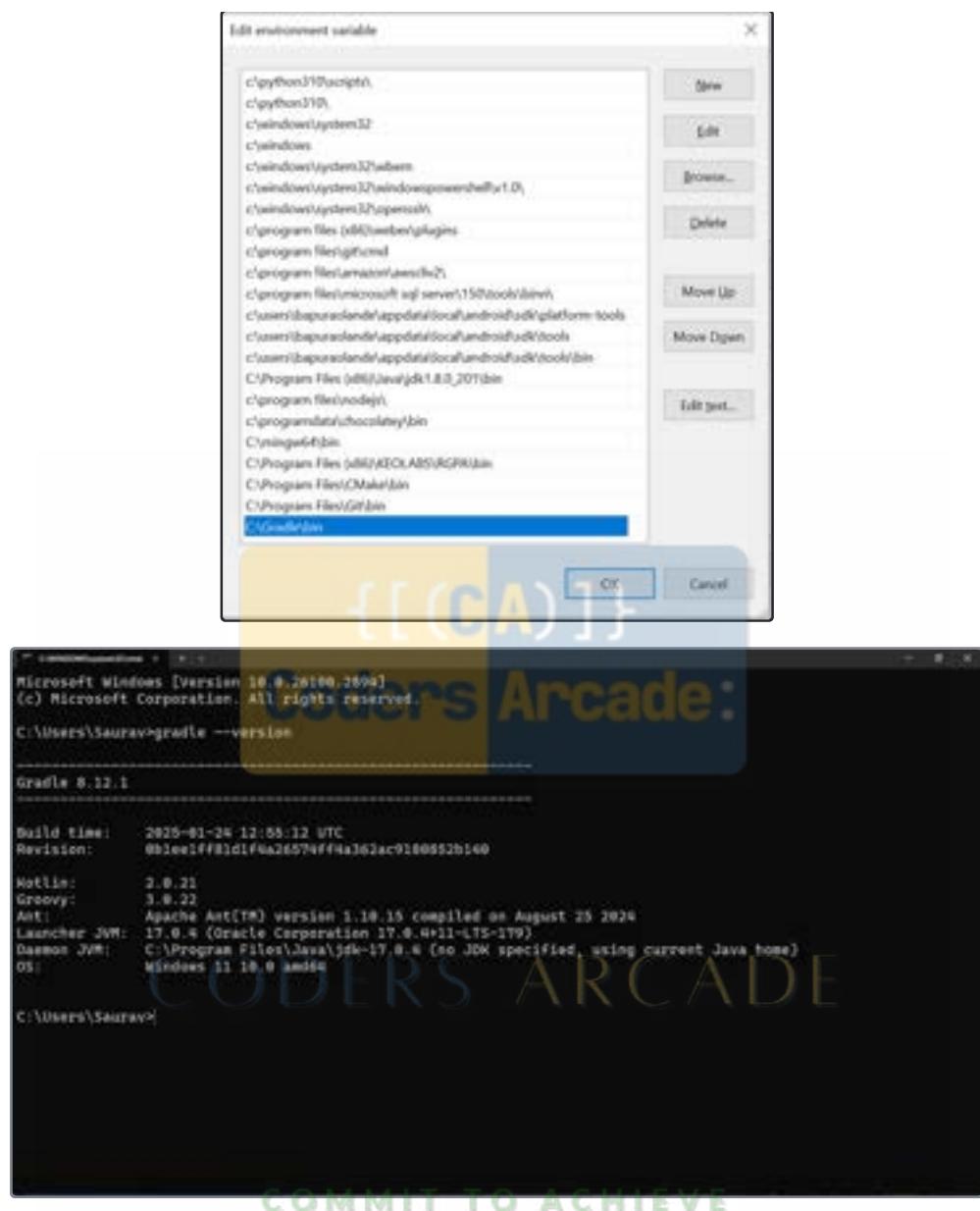
You will now see the following screen Select **New**, then add an entry for **C:\Gradle\bin** as shown in the image, now click on the **OK** button to save changes.

Step 4. Verifying and confirming the installation

After the successful setting of the environment variable, now it's time to verify the installation. Run the “**gradle -v**” command on the Windows command prompt, which displays the details of the Gradle version installed on the PC. If you get

similar output as in the image, Gradle has been installed successfully, and you can now use Gradle in your projects.





How Gradle Works?

A **Gradle project** consists of **one or more subprojects**, and each project is made up of **tasks**. Let's break it down:

Gradle Projects

- A **Gradle project** can be a **web application**, **JAR file**, or even a **collection of scripts**.
- Each project is made up of **tasks** that perform specific actions, such as **compiling**, **testing**, and **packaging**.
- Think of a **Gradle project** like a **wall**, where **each brick (task)** builds up the structure.

Gradle Tasks

Tasks in Gradle define **what needs to be executed** in a project.

There are **two main types of tasks**:

1 Default Tasks:

- These are **predefined by Gradle** and **execute automatically** when no specific task is mentioned.
- Example: `init` and `wrapper` tasks.

2 Custom Tasks:

- These are **user-defined tasks** that allow developers to **customize** the build process.
- Example: Let's create a simple **Gradle task** that prints a message:

```
1 // build.gradle
2 task hello {
3     doLast {
4         println 'Welcome to Gradle!'
5     }
6 }
7
```

💻 Running the task:

```
1 gradle -q hello
2
```

💻 Output:

```
1 Welcome to Gradle!
2
```



⭐ Features of Gradle

Gradle comes with a variety of **powerful features** that make it a preferred build tool for developers. 🚀

◆ IDE Support 🖥️

- Works with **IntelliJ IDEA, Eclipse, NetBeans, and Visual Studio**.

◆ Java-Friendly ☕

- Gradle projects run on **JVM (Java Virtual Machine)** and support **Java-based APIs**.

◆ Tasks & Repository Support 📦

- Supports **Maven** and **Ant repositories**.
- Allows easy **integration** with existing **Maven** and **Ant** projects.

◆ Incremental Builds ⚡

- **Only compiles changes** made after the last build, reducing build time significantly.

◆ Dependency Management 🔁

- Automatically **resolves and downloads dependencies**.
- Works with **Maven** and **Ivy repositories**.

◆ Build Caching 🔄

- Stores results from previous builds and **reuses them**, reducing build times.

◆ Plugin System

- Supports **plugins** for various languages (**Java, Kotlin, Scala, C++, Android, and more**).

◆ Extensibility

- Highly **customizable** with **user-defined tasks and configurations**.

◆ Continuous Integration Support

- Works seamlessly with **Jenkins, GitHub Actions, Travis CI, and Azure DevOps**.

◆ Multi-Project Support

- Allows **managing multiple projects within the same build**.

Pros & Cons of Using Gradle

Pros

- ✓ **Declarative Builds** – Uses **Groovy/Kotlin DSL** for configuration.
- ✓ **Highly Scalable** – Efficient for both **small** and **large projects**.
- ✓ **Performance Boost** – **Incremental builds** and **parallel execution** speed up the process.
- ✓ **Cross-Language Support** – Works with **Java, Groovy, Kotlin, Scala, C++, etc.**
- ✓ **Strong Dependency Management** – Handles **dependencies** efficiently.
- ✓ **Free & Open Source** – Available under **Apache License 2.0**.

Cons

- ! **Requires Technical Knowledge** – Needs prior understanding of **Groovy/Kotlin DSL**.
- ! **Complex Configurations** – Initial **setup can be tricky** for beginners.
- ! **Resource Consumption** – Uses **more memory** compared to **Maven**.
- ! **Migration Difficulty** – Moving from **Maven/Ant to Gradle** requires effort.

Conclusion

Gradle is a **powerful, flexible, and efficient** build automation tool. It has become a **developer-favorite** for projects of all sizes, from **small applications to large enterprise systems**. With **incremental builds, dependency management, and extensive plugin support**, Gradle makes software development **faster and smoother**. 



CA - Experiment 3 Part 1 - Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation

Here's the **full step-by-step guide** for working with **Gradle in IntelliJ IDEA**, including **setup, project creation, website deployment, Selenium testing, and JAR packaging**.

Gradle in IntelliJ IDEA: A Complete Guide

1 Introduction to Gradle

Gradle is a powerful build automation tool used for **Java, Kotlin, and Android projects**. It provides **fast builds, dependency management, and flexibility** using Groovy/Kotlin-based scripts.

Why Use Gradle?

- **Incremental builds** (faster execution)
- **Dependency management** (supports Maven/Ivy)
- **Customizable tasks**
- **Multi-project support**

2 Installing and Setting Up Gradle in IntelliJ IDEA (Already Covered In Experiment 1)

Step 1: Install Gradle

- **Windows** (using Chocolatey):

```
1 choco install gradle
2
```

- **macOS** (using Homebrew):

```
1 brew install gradle
2
```

- **Linux**:

```
1 sdk install gradle
2
```

Step 2: Verify Installation

```
1 gradle -v
2 gradle --version
```

This should display the Gradle version.

3 Creating a Gradle Project in IntelliJ IDEA

Step 1: Open IntelliJ IDEA and Create a New Project

1. Click on "New Project".
2. Select "**Gradle**" (under Java/Kotlin).
3. Choose **Groovy or Kotlin DSL (Domain Specific Language)** for the build script.
4. Set the **Group ID** (e.g., `com.example`).
5. Click **Finish**.

Step 2: Understanding Project Structure

```
1 my-gradle-project
2   |-- build.gradle  (Groovy Build Script)
3   |-- settings.gradle
4   |-- src
5     |   |-- main
6     |     |   |-- java
7     |     |   |-- resources
8     |   |-- test
9     |     |   |-- java
10    |     |   |-- resources
11
```

4 Build and Run a Simple Java Application

Step 1: Modify `build.gradle` (Groovy DSL)

```
1 plugins {
2   id 'application'
3 }
4
5 repositories {
6   mavenCentral()
7 }
8
9 dependencies {
10   testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
11 }
12
13 application {
14   mainClass = 'com.example.Main'
15 }
16
```

Step 2: Create `Main.java` in `src/main/java/com/example`

```

1 package com.example;
2
3 public class Main {
4     public static void main(String[] args) {
5         System.out.println("Hello from Gradle!");
6     }
7 }
```

Step 3: Build and Run the Project

- In IntelliJ IDEA, open the **Gradle tool window** (View → Tool Windows → Gradle).
- Click Tasks > application > run.
- Or run from terminal:

```

1 gradle run
2
```

5 Hosting a Static Website on GitHub Pages

Step 1: Create a `/docs` Directory

- Create `docs` inside the **root folder** (not in `src`).
- Add your **HTML, CSS, and images** inside `/docs`.

Step 2: Modify `build.gradle` to Copy Website Files (**This is optional**)

```

1 task copyWebsite(type: Copy) {
2     from 'src/main/resources/website'
3     into 'docs'
4 }
5
```

Step 3: Commit and Push to GitHub

```

1 git add .
2 git commit -m "Deploy website using Gradle"
3 git push origin main
4
```

Step 4: Enable GitHub Pages

- Go to **GitHub Repo** → **Settings** → **Pages**.
- Select the `/docs` **folder** as the source.

Your website will be hosted at:

```

1 https://yourusername.github.io/repository-name/
2
```

6 Testing the Website using Selenium & TestNG in IntelliJ IDEA

Step 1: Add Selenium & TestNG Dependencies in build.gradle

```

1 dependencies {
2     testImplementation 'org.seleniumhq.selenium:selenium-java:4.28.1' // use the latest stable
3         version
4     testImplementation 'org.testng:testng:7.4.0' // use the latest stable version
5 }
6 test {
7     useTestNG()
8 }
9

```

Step 2: Write a Test Script (src/test/java/org/test/WebpageTest.java)

```

1 package org.test;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.chrome.ChromeDriver;
5 import org.testng.Assert;
6 import org.testng.annotations.AfterTest;
7 import org.testng.annotations.BeforeTest;
8 import org.testng.annotations.Test;
9
10 import static org.testng.Assert.assertTrue;
11
12 public class WebpageTest {
13     private static WebDriver driver;
14
15     @BeforeTest
16     public void openBrowser() throws InterruptedException {
17         driver = new ChromeDriver();
18         driver.manage().window().maximize();
19         Thread.sleep(2000);
20         driver.get("https://sauravsharkar-codersarcade.github.io/CA-GRADLE/");
21     }
22
23     @Test
24     public void titleValidationTest(){
25         String actualTitle = driver.getTitle();
26         String expectedTitle = "Tripillar Solutions";
27         Assert.assertEquals(actualTitle, expectedTitle);
28         assertTrue(true, "Title should contain 'Tripillar'");
29     }
30
31
32     @AfterTest
33     public void closeBrowser() throws InterruptedException {
34         Thread.sleep(1000);
35         driver.quit();
36     }
37
38
39 }
40

```

Step 3: Run the Tests

- Open the **Gradle tool window** in IntelliJ.
- Click **Tasks > verification > test**. “**Recommended**”
- Or run from terminal:

```
1 gradle test // Fails sometimes due to terminal issues
2
```

7 Packaging a Gradle Project as a JAR

Step 1: Modify `build.gradle` for JAR Packaging

```
1 plugins {
2     id 'java'
3     id 'application'
4 }
5
6 application {
7     mainClass = 'com.example.Main'
8 }
9 jar {
10     manifest {
11         attributes 'Main-Class': 'com.example.Main' // This tells Java where to start execution
12     }
13 }
14
```

Step 2: Build and Package the JAR

```
1 gradle jar
2
```

The JAR file will be generated in `build/libs/`.

Step 3: Run the JAR

```
1 java -jar build/libs/<my-gradle-project>.jar
2
3 Expected output:
4 Hello from Gradle!
5
```

8 Gradle Lifecycle & Common Commands

Task	Command	Description
Initialize Project	<code>gradle init</code>	Creates a new Gradle project.
Compile Code	<code>gradle build</code>	Compiles the project.
Run Application	<code>gradle run</code>	Runs the application.

Clean Build Files	<code>gradle clean</code>	Deletes old build files.
Run Tests	<code>gradle test</code>	Executes unit tests.
Generate JAR	<code>gradle jar</code>	Packages the project into a JAR.
Deploy to GitHub Pages	<code>git push origin main</code>	Pushes website to GitHub.

9 Conclusion

Gradle provides a **fast, flexible, and scalable build automation system**.

We covered:

- 1 Project Setup in IntelliJ IDEA
- 2 Building & Running a Java Application
- 3 Hosting a Static Website on GitHub Pages
- 4 Testing with Selenium & TestNG
- 5 Packaging as a JAR

 Now you are **ready to use Gradle for build automation!** 

★ Gradle Build Lifecycle: A Simple & Colorful Guide

Gradle follows a **flexible, task-based lifecycle**, unlike Maven's fixed phases. The build lifecycle in Gradle is divided into three key stages:

1. Initialization Phase

👉 What happens here?

- Determines which projects are part of the build (for multi-project builds).
- Creates an instance of each project.

👉 Example Command:

```
1 gradle help
2
```

(Shows project details and verifies Gradle setup.)

2. Configuration Phase

👉 What happens here?

- Gradle loads and executes the `build.gradle` file.
- It **configures** tasks but does **not** execute them yet.

👉 Example Command:

```
1 gradle tasks
2
```

(Lists all available Gradle tasks in the project.)

3. Execution Phase

💡 What happens here?

- Gradle executes the **tasks requested by the user**.
- Dependencies are resolved dynamically.
- Supports **incremental builds** (only modified files are compiled).

💡 Example Command:

```
1 gradle build
2
```

(Builds the project by compiling and packaging files.)

⌚ Key Gradle Tasks & Commands

Task 🛠	Command	Description
🧹 Clean	gradle clean	Deletes previous build files.
🛠 Compile	gradle compileJava	Compiles the Java source code.
📦 Package (JAR/WAR)	gradle jar	Packages the project into a JAR file.
.TestCheck	gradle test	Runs unit tests.
🚀 Run Application	gradle run	Executes the main Java application.
📦 Dependency Resolution	gradle dependencies	Displays dependency tree.
▶ Parallel Execution	gradle build --parallel	Speeds up builds by running tasks in parallel.

🎨 Gradle Lifecycle Visualized

```
1 Initialization → Configuration → Execution
2 (Project setup)   (Tasks loaded)   (Tasks executed)
3
```

✓ **Gradle is flexible** – tasks can be **customized or skipped** based on project needs. Unlike Maven, Gradle allows **on-demand task execution** rather than following a strict lifecycle.

🌟 Conclusion

⌚ Gradle is powerful because:

- ✓ It **only executes what's necessary** (faster builds).

- It supports parallel execution for large projects.
- It gives developers more control over task execution.

With Gradle, you can **automate builds efficiently** and **improve performance** for Java, Kotlin, and Android projects.



Maven vs. Gradle: A Detailed Comparison

Feature	Maven	Gradle
Build Script Language	XML (<code>pom.xml</code>)	Groovy/Kotlin (<code>build.gradle</code> or <code>build.gradle.kts</code>)
Performance	Slower due to full rebuilds	Faster with incremental builds and parallel execution
Dependency Management	Uses Maven Central & local repository	Supports Maven, Ivy, and custom repositories
Configuration Style	Declarative (XML-based, verbose)	Declarative + Imperative (more concise, script-based)
Ease of Use	More structured, but verbose	More flexible, but has a learning curve
Incremental Builds	No support (always rebuilds everything)	Supports incremental builds (only recompiles changed files)
Build Performance	Slower, builds from scratch	Faster due to task caching and incremental execution
Customization & Extensibility	Limited custom scripts, plugin-based	Highly customizable with dynamic tasks and scripts
Multi-Project Builds	Complex and slower	Native support, optimized for large projects
IDE Support	Supported by IntelliJ IDEA, Eclipse, VS Code	Supported by IntelliJ IDEA, Eclipse, VS Code
Dependency Resolution	Resolves dependencies sequentially	Parallel dependency resolution (faster)
Popularity	Older and widely used in enterprise projects	Gaining popularity, especially in Android & Kotlin projects
Default Lifecycle Phases	3 lifecycle phases (clean, build, site)	More flexible task execution model
Learning Curve	Easier for beginners due to structured XML	Slightly steeper due to script-based configuration
Best Suited For	Stable Java projects , enterprises, and legacy codebases	Modern Java, Android, and Kotlin projects needing high performance

Which One Should You Choose?

-  **Use Maven** if you need **stability, structured builds, and an easier learning curve**.
-  **Use Gradle** if you want **faster builds, better performance, and flexibility** for complex projects.

 **Final Verdict:** If you're working on a simple Java project, **Maven** is a good choice. However, for modern, large-scale, or Android projects, **Gradle** is the better option due to its **speed, flexibility, and scalability**. 

 Here is a **KOTLIN DSL** version for Gradle “This is Optional”

Gradle Kotlin Build Script (build.gradle.kts)

Here's a **simple Gradle build script** written in **Kotlin DSL** (`build.gradle.kts`) for a Java application. 

build.gradle.kts (Kotlin DSL for Gradle)

```

1 // Apply necessary plugins
2 plugins {
3     kotlin("jvm") version "1.9.10" // Kotlin plugin for JVM
4     application // Enables the `run` task
5 }
6
7 // Define project properties
8 group = "com.example"
9 version = "1.0.0"
10 application.mainClass.set("com.example.MainKt") // Define the main class
11
12 // Repositories for dependencies
13 repositories {
14     mavenCentral() // Fetch dependencies from Maven Central
15 }
16
17 // Dependencies
18 dependencies {
19     implementation(kotlin("stdlib")) // Standard Kotlin library
20     testImplementation("org.junit.jupiter:junit-jupiter:5.9.1") // JUnit 5 for testing
21 }
22
23 // Enable JUnit 5 for tests
24 tasks.test {
25     useJUnitPlatform()
26 }
27
28 // JAR packaging
29 tasks.jar {
30     manifest {
31         attributes["Main-Class"] = "com.example.MainKt"
32     }
33 }
34

```

📌 Main.kt (Inside src/main/kotlin/com/example/)

```
1 package com.example  
2  
3 fun main() {  
4     println("Hello, Gradle with Kotlin DSL! 🎉")  
5 }  
6
```

📌 Running the Project

💻 Run the application

```
1 gradle run  
2
```

📦 Build the project

```
1 gradle build  
2
```

🧹 Clean the build files

```
1 gradle clean  
2
```

📁 Generate a JAR file

```
1 gradle jar  
2
```



🌟 Why Use Kotlin DSL for Gradle?

- ✓ **Type-Safety** - Catch errors at compile-time.
- ✓ **Better IDE Support** - IntelliJ IDEA provides autocomplete and suggestions.
- ✓ **Interoperability** - Works seamlessly with Java and Kotlin projects.

This script sets up a **basic Kotlin/Java project** with **JUnit 5 support**, **dependency management**, and a **runnable JAR file!** 🎉



CA - Experiment 3 Part 2 : GRADLE KOTLIN DSL WORKFLOW || IntelliJ Idea

Here's a **detailed and well-formatted** documentation Gradle Kotlin DSL setup, including explanations, code snippets, and color-coded syntax (for reference when viewing in an IDE).

⚠ Important Note : This experiment is only required if **students** or **VTU** asks you to show **GRADLE** with **KOTLIN DSL**, or else you can skip this.

Gradle Kotlin DSL: Setting Up & Building a Kotlin Project in IntelliJ IDEA

1 Setting Up the Gradle Project

Step 1: Create a New Project

1. Open **IntelliJ IDEA**.
2. Click on **File > New > Project**.
3. Select **Gradle** as the build system.
4. Choose **Kotlin** as the language.
5. Select **Gradle Kotlin DSL** (it will generate `build.gradle.kts`).
6. Name your project (e.g., `MVNGRDKOTLINDMO`).
7. Set the **JDK** (use **JDK 17.0.4**, since that's your version).
8. Click **Finish**.

2 Understanding `build.gradle.kts`

After creating the project, the default `build.gradle.kts` file looks like this:

```
1 import org.jetbrains.kotlin.gradle.tasks.KotlinCompile
2
3 plugins {
4     kotlin("jvm") version "1.8.10" // Use latest stable Kotlin version
5     application
6 }
7
8 group = "org.example"
9 version = "1.0-SNAPSHOT"
10
11 repositories {
12     mavenCentral()
```

```

13 }
14
15 dependencies {
16     implementation(kotlin("stdlib")) // Kotlin Standard Library
17     testImplementation("org.junit.jupiter:junit-jupiter-api:5.8.2")
18     testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:5.8.2")
19 }
20
21 tasks.test {
22     useJUnitPlatform()
23 }
24
25 tasks.withType<KotlinCompile> {
26     kotlinOptions.jvmTarget = "17" // Match with your JDK version
27 }
28
29 application {
30     mainClass.set("MainKt") // Update this if using a package
31 }
32

```

3 Creating the Main Kotlin File

Now, create your `Main.kt` file inside `src/main/kotlin/`.

If you're using a package (e.g., `org.example`), it should look like:

```

1 package org.example
2
3 fun main() {
4     println("Hello, Gradle with Kotlin DSL!")
5 }
6

```

If you're **not** using a package, remove the `package` line and ensure `mainClass.set("MainKt")` in `build.gradle.kts`.

4 Building and Running the Project

Build the Project

COMMIT TO ACHIEVE

```

1 ./gradlew build
2

```

Run the Project

```

1 ./gradlew run
2

```

5 Packaging as a JAR

To run the project without IntelliJ, we need a **JAR file**.

Step 1: Create a Fat (Uber) JAR

Modify `build.gradle.kts`:

```

1 tasks.register<Jar>("fatJar") {
2     archiveClassifier.set("all")
3     duplicatesStrategy = DuplicatesStrategy.EXCLUDE
4     manifest {
5         attributes["Main-Class"] = "MainKt"
6     }
7     from(configurations.runtimeClasspath.get().map { if (it.isDirectory) it else zipTree(it) })
8     with(tasks.jar.get() as CopySpec)
9 }
10

```

Step 2: Build the Fat JAR

```

1 ./gradlew fatJar
2

```

Step 3: Run the Fat JAR

```

1 java -jar build/libs/MVNGRDKOTLINDEMO-1.0-SNAPSHOT-all.jar
2

```

6 Common Gradle Commands

Command	Description
./gradlew build	Builds the project
./gradlew run	Runs the application
./gradlew test	Runs the tests
./gradlew clean	Cleans the build directory
./gradlew fatJar	Creates a runnable JAR

7 Additional Features COMMIT TO ACHIEVE

Adding Custom Gradle Tasks

Example: A simple task that prints "Hello, Gradle!"

```

1 tasks.register("hello") {
2     doLast {
3         println("Hello, Gradle!")
4     }
5 }
6

```

Run it with:

```

1 ./gradlew hello
2

```

8 | Troubleshooting & Fixes

1. `java.lang.NoClassDefFoundError: kotlin/jvm/internal/Intrinsics`

 Fix: Use a Fat JAR (`fatJar`) to include Kotlin dependencies.

2. `mainClass Not Found`

 Fix: Ensure `mainClass.set("MainKt")` or use the correct package (`org.example.MainKt`).

3. `Gradle Wrapper Missing`

 Fix: Generate it using:

```
1 gradle wrapper  
2
```

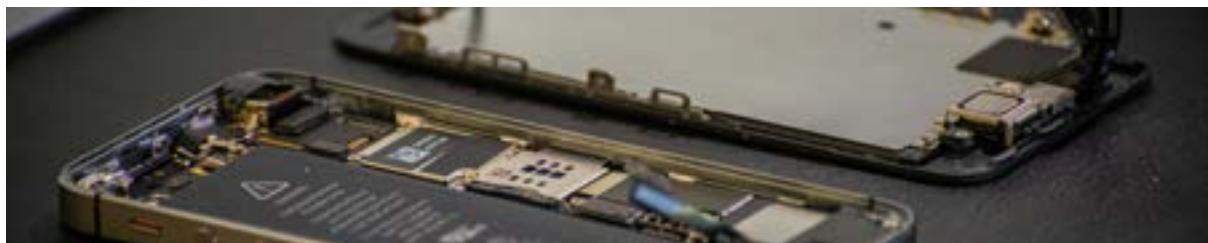
 You're all set! 🎉

This guide ensures you have a **fully working** Gradle project with Kotlin DSL, dependency management, custom tasks, and a runnable JAR. 😊



CODERS ARCADE

COMMIT TO ACHIEVE



CA - Experiment 4 - Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

Part 1: Create and Build a Java Application with Maven

Step 1: Create a Maven Project in IntelliJ IDEA

1. Open IntelliJ IDEA

- Launch **IntelliJ IDEA** and click on **File → New → Project**. 🌟

2. Select Maven

- In the **New Project** window, choose **Maven** from the options on the left.
- Check **Create from archetype** and select **maven-archetype-quickstart**.
- Click **Next**. 🚀

3. Enter Project Details

- GroupId:** com.example
- ArtifactId:** MVNGRDLDemo
- Click **Next** and then **Finish**. 🎉

4. Wait for IntelliJ to Load Dependencies

- IntelliJ will automatically download the Maven dependencies, so just relax for a moment. 😊

Step 2: Update `pom.xml` to Add Build Plugin

To compile and package your project into a `.jar` file, you need to add the **Maven Compiler** and **Jar** plugins. 🛠

1. Open the `pom.xml` file. 📄

2. Add the following inside the `<project>` tag:

```
1 <build>
2   <plugins>
3     <!-- Compiler Plugin -->
4     <plugin>
5       <groupId>org.apache.maven.plugins</groupId>
6       <artifactId>maven-compiler-plugin</artifactId>
7       <version>3.8.1</version>
8       <configuration>
9         <source>1.8</source>
```

```

10          <target>1.8</target>
11      </configuration>
12  </plugin>
13
14  <!-- Jar Plugin -->
15  <plugin>
16      <groupId>org.apache.maven.plugins</groupId>
17      <artifactId>maven-jar-plugin</artifactId>
18      <version>3.2.0</version>
19      <configuration>
20          <archive>
21              <manifest>
22                  <mainClass>com.example.App</mainClass>
23              </manifest>
24          </archive>
25      </configuration>
26  </plugin>
27 </plugins>
28 </build>
29

```



Step 3: Build and Run the Maven Project

1. Open IntelliJ IDEA Terminal

Press **Alt + F12** to open the terminal.

2. Compile and Package the Project

Run the following commands to build the project:

```

1 mvn clean compile
2 mvn package
3

```



3. Locate the JAR File

After running the above, your `.jar` file will be located at:

```

1 D:\Idea Projects\MVNGRDLDEMO\target\MVNGRDLDEMO-1.0-SNAPSHOT.jar
2

```



4. Run the JAR File

To run the generated JAR file, use:

```

1 java -jar target\MVNGRDLDEMO-1.0-SNAPSHOT.jar
2

```

Part 2: Migrate Maven Project to Gradle

Step 1: Initialize Gradle in Your Project

1. Open Terminal in IntelliJ IDEA

Make sure you're in the project directory:

```

1 cd "D:\Idea Projects\MVNGRDLDEMO"

```

2

2. Run Gradle Init Command

Execute the following command to migrate your Maven project to Gradle:

```
1 gradle init --type pom
2
```

This command will convert your Maven `pom.xml` into a Gradle `build.gradle` file. 

Step 2: Review and Update `build.gradle`

1. Open `build.gradle` in IntelliJ IDEA.
2. Ensure the following configurations are correct:

```
1 plugins {
2     id 'java'
3 }
4
5 group = 'com.example'
6 version = '1.0-SNAPSHOT'
7
8 repositories {
9     mavenCentral()
10 }
11
12 dependencies {
13     testImplementation 'junit:junit:4.13.2'
14 }
15
16 jar {
17     manifest {
18         attributes(
19             'Main-Class': 'com.example.App'
20         )
21     }
22 }
```



CODERS ARCADE

COMMIT TO ACHIEVE

Step 3: Build and Run the Gradle Project

1. Clean and Build the Project

To clean and build your Gradle project, run:

```
1 gradle clean build
2
```

2. Run the Generated JAR File

Now, run the generated JAR file using:

```
1 java -jar build/libs/MVNGRDLDemo-1.0-SNAPSHOT.jar
2
```

🎉 Conclusion

Congratulations! You have successfully:

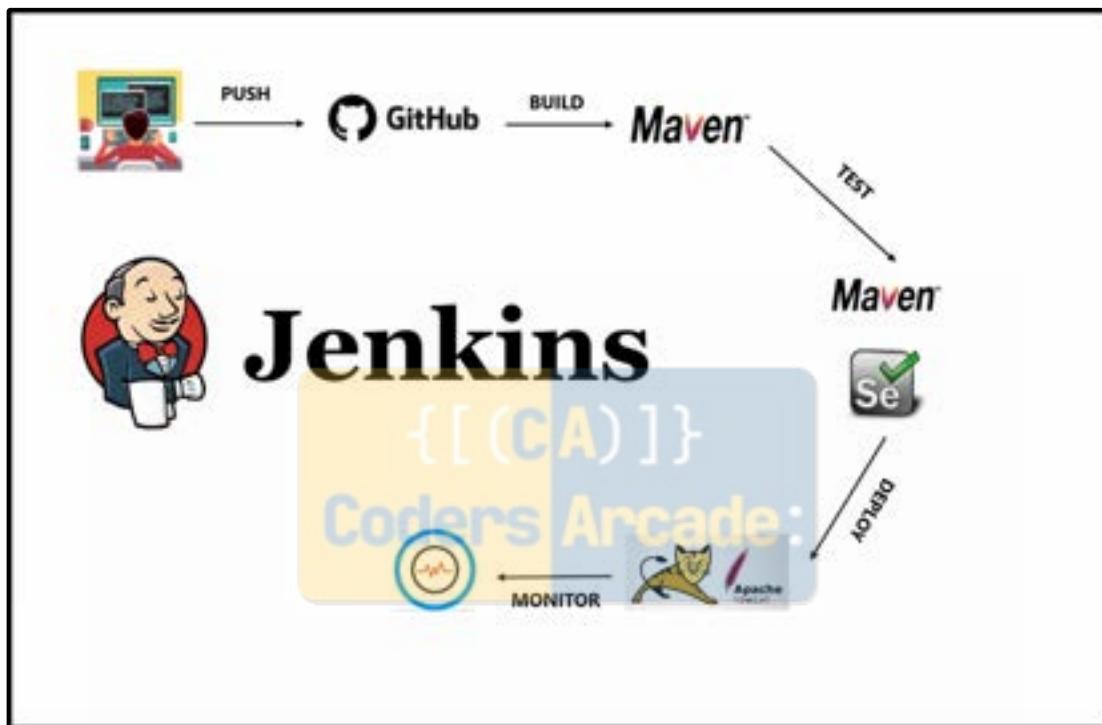
1. **Created a Maven project** in IntelliJ IDEA. 🎉
2. **Built and packaged** it into a JAR file using Maven. 🚧
3. **Migrated** the project from Maven to Gradle. 📁
4. **Built and ran** the project using Gradle. 🚶



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Jenkins Notes & Documentation



- What is Jenkins
- What is CI & CD
 - Continuous Integration, Delivery & Deployment
- Installation
 - System Requirements
- Installation on Windows
- Installation on Mac
- Jenkins Configuration
 - How to change Home Directory
 - How to setup Git on Jenkins
- Create the first Job on Jenkins
- How to connect to Git Remote Repository in Jenkins (GitHub)
- How to use Command Line in Jenkins CLI
- How to create Users + Manage + Assign Roles
- Jenkins Pipeline Beginner Tutorial
- Running Selenium Automation Tests from GitHub via Jenkins CI Agent :
 - There are two ways in which we can perform Selenium Automation tests from GitHub via Jenkins

CODERS ARCADE

COMMIT TO ACHIEVE

What is Jenkins

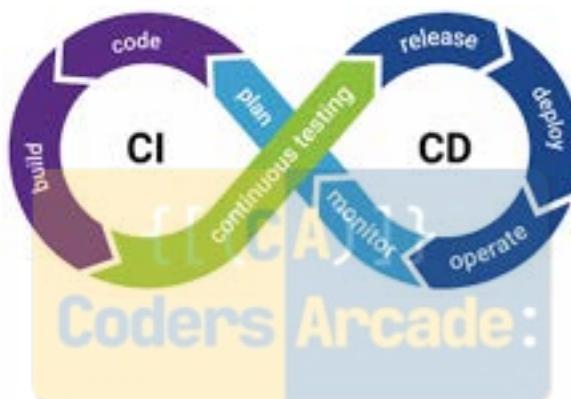
- Jenkins is a CI CD tool
- Free & Open Source
- Written in Java

What is CI & CD

Watch This Video To Get Detailed Idea About CI/CD Pipeline : [YouTube DevOps CI CD Pipeline || Simple & Detailed Explanation](#)

Continuous Integration, Delivery & Deployment

CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.



Installation

System Requirements

Memory 256 MB of RAM

Disk Space Depends on your projects

OS Windows, Mac, Ubuntu, Linux

Java 8 or 11 (JDK or JRE)

Installation on Windows

Watch This Video To Seamlessly Install Jenkins : [YouTube Jenkins Installation - Step by Step Guide](#)

Step 1 : Check Java is installed

Step 2 : Download Jenkins.war file

Step 3 : Goto cmd prompt and run command

`java -jar jenkins.war --httpPort=8080`

Step 4 : On browser goto <http://localhost:8080>

Step 5 : Provide admin password and complete the setup

Installation on Mac

Homebrew

Installation on Linux

[YouTube How to install Jenkins on Amazon AWS EC2 Linux | 8 Steps](#)

[YouTube 2. Jenkins Tutorials: How to install Jenkins on Ubuntu 22.04](#)

Jenkins Configuration

How to change Home Directory

Step 1: Check your Jenkins Home > Manage Jenkins > Configure System

Step 2 : Create a new folder

Step 3 : Copy the data from old folder to new folder

Step 4 : Create/Update env variable JENKINS_HOME

Step 5 : Restart Jenkins

`jenkins.xml`

`JENKINS_HOME`

How to setup Git on Jenkins

Step 1 : Goto Manage Jenkins > Manage Plugins

Step 2 : Check if git is already installed in Installed tab

Step 3 : Else goto Available tab and search for git

Step 4 : Install Git

Step 5 : Check git option is present in Job Configuration

Create the first Job on Jenkins

How to connect to Git Remote Repository in Jenkins (GitHub)

Step 1 : Get the url of the remote repository

Step 2 : Add the git credentials on Jenkins

Step 3 : In the jobs configuration goto SCM and provide git repo url in git section

Step 4 : Add the credentials

Step 5 : Run job and check if the repository is cloned

How to use Command Line in Jenkins CLI

Faster, easier, integration

Step 1 : start Jenkins

Step 2 : goto Manage Jenkins - Configure Global Security - enable security

Step 3 : goto - <http://localhost:8080/cli/>

Step 4 : download jenkins-cli jar. Place at any location.

Step 5 : test the jenkins command line is working

```
java -jar jenkins-cli.jar -s http://localhost:8080 /help --username <userName> --password <password>
```

How to create Users + Manage + Assign Roles

How to create New Users

How to configure users

[How to create new roles](#)

[How to assign users to roles](#)

[How to Control user access on projects](#)

Step 1 : Create new users

Step 2 : Configure users

Step 3 : Create and manage user roles Role Based Authorization Strategy Plugin - download - restart jenkins

Step 4 : Manage Jenkins - Configure Global Security - Authorization - Role Based Strategy

Step 5 : Create Roles and Assign roles to users

Step 6 : Validate authorization and authentication are working properly

Jenkins Pipeline Beginner Tutorial

[How to create Jenkinsfile](#)

- [What is pipeline](#)
- [What is jenkins pipeline](#)
- [What is jenkinsfile](#)
- [How to create jenkinsfile](#)

[Build > Deploy > Test > Release](#)

[Jenkinsfile : Pipeline as a code](#)



[Step 1 : Start Jenkins](#)

[Step 2 : Install Pipeline Plugin](#)

[Step 3 : Create a new job](#)

[Step 4 : Create or get Jenkinsfile in Pipeline section](#)

[Step 5 : Run and check the output](#)

[Jenkins Pipeline](#)

[How to get jenkinsfile from Git SCM](#)

CODERS ARCADE

[Step 1 : Create a new job or use existing job \(type : Pipeline\)](#)

[Step 2 : Create a repository or GitHub](#)

[Step 3 : Add Jenkinsfile in the repo](#)

[Step 4 : Under Jenkins job > Pipeline section > Select Definition Pipeline script from SCM](#)

[Step 5 : Add repo and jenkinsfile location in the job under Pipeline section](#)

[Step 6 : Save & Run](#)

COMMIT TO ACHIEVE

[Jenkins Pipeline](#)

[How to clone a git repo using Jenkinsfile](#)

[Show Live Demonstration.](#)

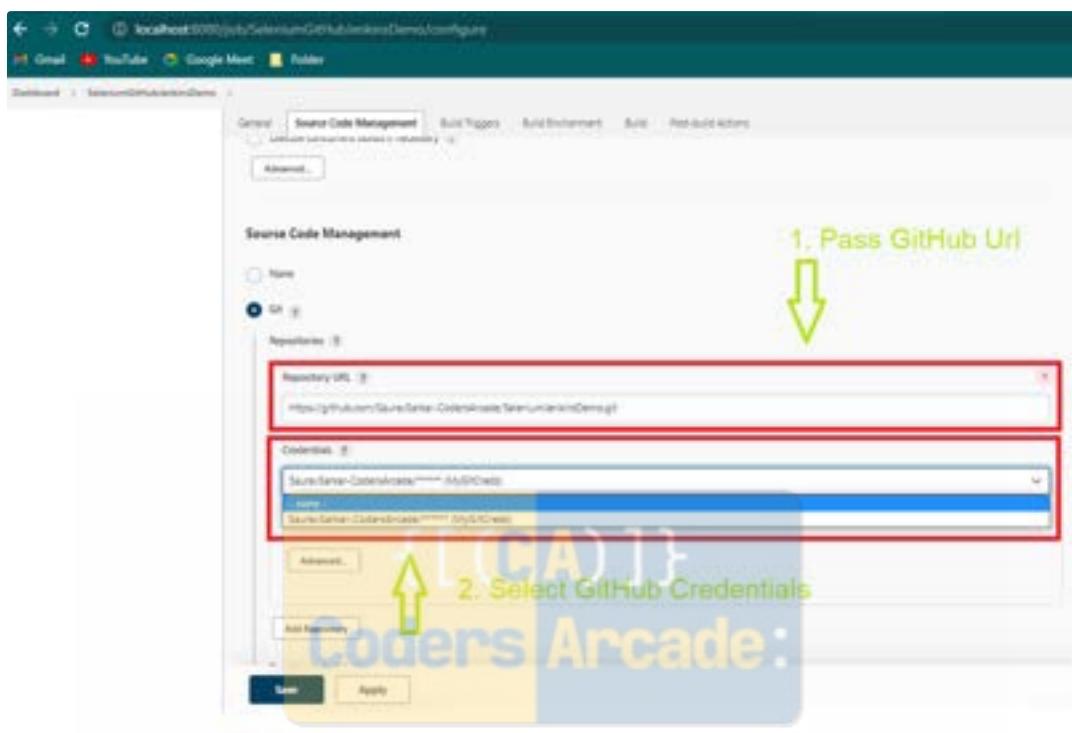
Running Selenium Automation Tests from GitHub via Jenkins CI Agent :

[There are two ways in which we can perform Selenium Automation tests from GitHub via Jenkins](#)

- Via [Git Credentials](#)
- Via [GitHub access token](#)
- The steps are shown with images.
- In the build, you have to execute the maven commands like: [mvn clean test, etc.](#)

Method 1:

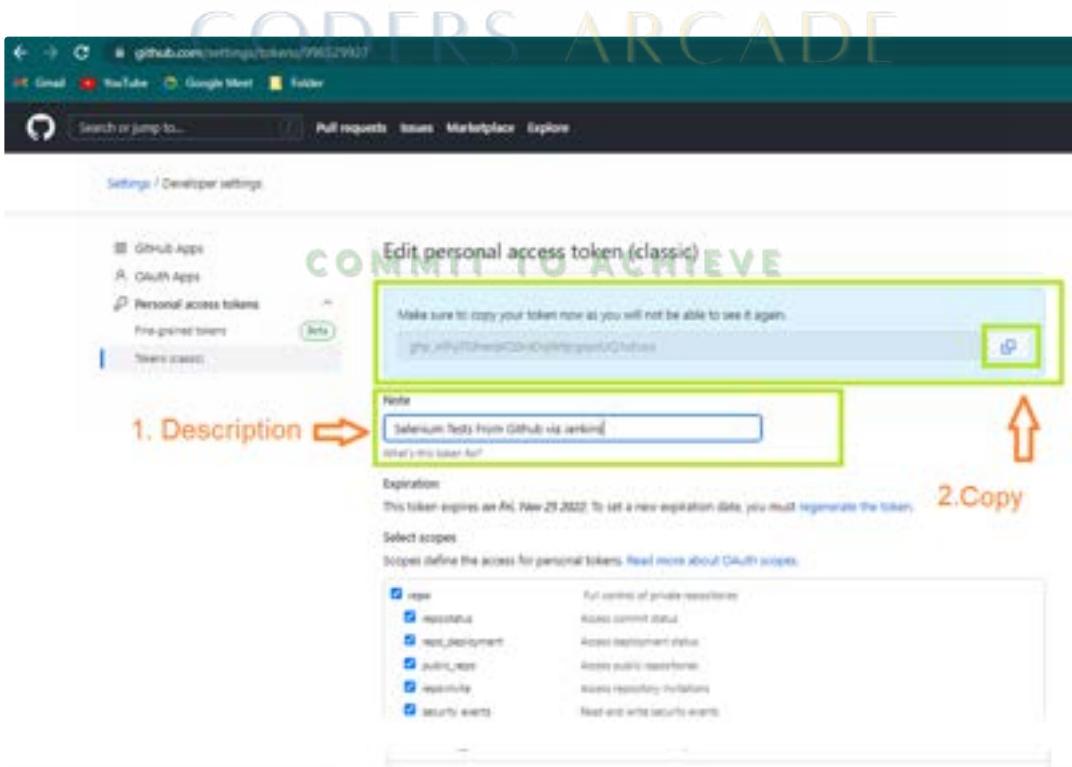
- Via **Git Credentials**



GitHub + Jenkins With Git Credentials

Method 2:

- Via **GitHub Access Token**



GitHub + Jenkins Via Access Token



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Experiment 5 - Introduction to Jenkins: What is Jenkins?, Installing Jenkins on Local or Cloud Environment, Configuring Jenkins for First Use

Experiment 5: Introduction to Jenkins

Objective

To understand the fundamentals of **Jenkins**, install it on a local or cloud environment, and configure it for first-time use.

1. What is Jenkins?

Jenkins is an **open-source automation server** used for:

-  **Continuous Integration (CI)** – Automatically testing and integrating code changes
-  **Continuous Deployment (CD)** – Automating application deployment
-  **Building Pipelines** – Managing end-to-end software development workflows
-  **Plugin-Based Extensibility** – Supporting tools like Maven, Gradle, Ansible, Docker, and Azure DevOps

Why Use Jenkins?

- ✓ Automates builds and tests
- ✓ Reduces manual intervention
- ✓ Improves software quality
- ✓ Works with multiple tools and platforms

2. Installing Jenkins

Jenkins can be installed using multiple methods:

-  **Windows Installer (.msi)** – Recommended for Windows
-  **Linux Package Manager** – Best for Linux Users
-  **Jenkins WAR File** – Universal method using Java

We'll cover all three approaches.

2.1 Prerequisites

-  **Java 11 or 17** is required for Jenkins.

Check Java version:

```
1  java -version
2
```

If Java is not installed, download it from:

 [Download the Latest Java LTS Free](#)

2.2 Installing Jenkins on Windows (MSI Installer) – Recommended

Step 1: Download Jenkins

 [Download and deploy](#)

Choose **Windows Installer (.msi)** for an easy setup.

Step 2: Install Jenkins

- 1 Run the downloaded **.msi** file.
- 2 Follow the installation wizard.
- 3 Select **Run Jenkins as a Windows Service** (recommended).
- 4 Choose the **installation directory** (default: `C:\Program Files\Jenkins`).
- 5 Click **Install** and wait for the setup to complete.

Step 3: Start Jenkins

- 1 Open **Services** (`services.msc`) and ensure **Jenkins** is running.
- 2 Open a web browser and go to:

```
1 http://localhost:8080
2
```



Step 4: Unlock Jenkins

- 1 Find the initial **Admin Password** in:

```
1 C:\Program Files\Jenkins\secrets\initialAdminPassword
2
```

- 2 Copy the password and paste it into the Jenkins setup page.

Step 5: Install Recommended Plugins

Jenkins will prompt you to install plugins. Click "**Install Suggested Plugins**".

Step 6: Create Admin User

- 1 Set up a **Username**, **Password**, and **Email**.
- 2 Click **Save and Finish**.

 **Jenkins is now ready!** 🎉

Access it anytime at:

```
1 http://localhost:8080
2
```

COMMIT TO ACHIEVE

2.3 Installing Jenkins on Linux (Ubuntu/Debian)

Step 1: Add Jenkins Repository

```
1 wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
2 sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
   /etc/apt/sources.list.d/jenkins.list'
3
```

Step 2: Install Jenkins

```
1 sudo apt update
```

```

2 sudo apt install jenkins -y
3

```

Step 3: Start Jenkins

```

1 sudo systemctl start jenkins
2 sudo systemctl enable jenkins
3

```

Step 4: Access Jenkins

Find the **initial password** in:

```

1 sudo cat /var/lib/jenkins/secrets/initialAdminPassword
2

```

Then open Jenkins in a browser:

```

1 http://localhost:8080
2

```

2.4 Installing Jenkins Using WAR File (Works on Any OS)

This method allows you to run Jenkins without installing it as a service.

Step 1: Download the Jenkins WAR File

 Download from: [Download and deploy](#)

Choose **Generic Java Package (.war)**.

Step 2: Run Jenkins Using Java

Navigate to the folder where the **.war** file is downloaded and run:

```

1 java -jar jenkins.war --httpPort=8080
2

```

 This will start Jenkins on port **8080**.

Step 3: Open Jenkins in Browser

Go to:

```

1 http://localhost:8080
2

```

Step 4: Unlock Jenkins & Setup

Follow the **same steps** as the Windows/Linux installation:

- ✓ Find the initial password
- ✓ Install plugins
- ✓ Create an admin user

 **Jenkins is now running without installation!**

To stop Jenkins, press **CTRL + C** in the terminal.

3. Configuring Jenkins for First Use

3.1 Understanding the Jenkins Dashboard

After logging in, you will see:

- **New Item** → Create Jobs/Pipelines
- **Manage Jenkins** → Configure System, Users, and Plugins
- **Build History** → View previous builds
- **Credentials** → Store secure authentication details

3.2 Installing Additional Plugins

Jenkins supports **plugins** for various tools like Maven, Gradle, Docker, and Azure DevOps.

- To install a plugin:
- 1 Go to **Manage Jenkins** → **Manage Plugins**
 - 2 Search for the required plugin
 - 3 Click **Install without Restart**

3.3 Setting Up Global Tool Configuration

Configure Java, Maven, and Gradle in Jenkins:

- 1 Go to **Manage Jenkins** → **Global Tool Configuration**
- 2 Add paths for:
 - **JDK** (C:\Program Files\Java\jdk-17)
 - **Maven** (C:\Maven\apache-maven-<version>)
 - **Gradle** (C:\Gradle\gradle-<version>)
- 3 Click **Save**

Assessment Questions

- 1 What is **Jenkins** used for in DevOps?
- 2 Explain the **difference** between CI and CD in Jenkins.
- 3 How do you **install Jenkins** on Windows, Linux, and using the WAR file?
- 4 Where can you find the **initial Jenkins password** after installation?
- 5 What are some **essential Jenkins plugins** for automation?

Important Note

You can use the below links to easily install **Jenkins** & understand **CI/CD Pipelines** in **DevOps**.

- **Jenkins Installation Video** – Follow this step-by-step guide for a **seamless Jenkins setup**: [Jenkins Installation - Step by Step Guide](#)
- **Understanding CI/CD in DevOps** – Learn how Jenkins fits into the **CI/CD pipeline**: [DevOps CI CD Pipeline](#)

Ensure you have **Java installed** before setting up Jenkins. If you face any issues, check the **Jenkins logs** for troubleshooting. ✓

CA - Experiment 6 - Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests

Experiment 6: Continuous Integration with Jenkins

Objective

To set up a Continuous Integration (CI) pipeline in Jenkins, integrate it with Git, and run Selenium Java tests using Maven.

Prerequisites

Before proceeding, ensure the following:

- Jenkins is installed and running on your system. If not, refer to [Experiment 5].
- Git is installed and configured in Jenkins. (Verify with `git --version`).
- Maven is installed and configured in Jenkins. (Check with `mvn -version`).
- Selenium Maven Project is ready with test cases (`src/test/java`).
- Project is stored in two places:
 - Locally on your system (e.g., `D:\Idea Projects\MVNGRDLDEMO`).
 - Pushed to GitHub with a valid repository link.
- Jenkins has access to the GitHub repository (via credentials).

1. Configuring Jenkins & Git Integration

Step 1: Verify Git Installation in Jenkins

1. Open Jenkins Dashboard → Manage Jenkins → Global Tool Configuration.
2. Under Git, verify the installation path (e.g., `C:\Program Files\Git\bin\git.exe`).
3. Click Save.

Step 2: Add GitHub Credentials in Jenkins

1. Navigate to Manage Jenkins → Manage Credentials.
2. Select Global credentials (unrestricted) → Click Add Credentials.
3. Choose Username with password or SSH Key, provide details, and click OK.

2. Running a Selenium Java Test from a Local Maven Project

Step 1: Create a New Jenkins Job

1. Go to Jenkins Dashboard → Click New Item.

2. Enter a project name → Select **Freestyle Project**.

3. Click **OK**.

Step 2: Configure the Build Step

1. Scroll to **Build** → Click **Add build step** → **Execute Windows Batch Command**.

2. Enter the following commands (**ensure correct navigation to project directory**):

```
1 cd D:\Idea Projects\MVNGRDLDEMO
2 mvn test
3
```

3. Click **Save** → Click **Build Now** to execute the test.

3. Running Selenium Tests from a GitHub Repository via Jenkins

Step 1: Set Up a New Jenkins Job for GitHub Project

1. Go to **Jenkins Dashboard** → Click **New Item**.

2. Enter a project name → Select **Freestyle Project**.

3. Click **OK**.

Step 2: Configure Git Repository in Jenkins

1. Under **Source Code Management**, select **Git**.

2. Enter your GitHub repository URL (e.g., <https://github.com/your-repo-name.git>).

3. Select the **Git credentials** configured earlier.

Step 3: Add Build Step for Maven

1. Scroll to **Build** → Click **Add build step** → **Execute Windows Batch Command**.

2. Enter the Maven test command:

```
1 mvn test
2
```

3. Click **Save**.

COMMIT TO ACHIEVE

Step 4: Trigger the Build

1. Click **Build Now** to fetch the code from GitHub and execute the Selenium tests.

2. Check the **Console Output** to verify test execution.

Important Notes

👉 **Prerequisites are crucial!** Make sure Jenkins, Git, Maven, and Selenium projects are set up correctly before proceeding.

👉 **Always navigate to the project directory** before running `mvn test` from a local system.

👉 **Use webhooks** in GitHub to automatically trigger builds when new code is pushed.

👉 **Configure email notifications** in Jenkins for build status updates.

🔗 **Jenkins & Git Integration Video:** [To Be Added - Version 1.2]

🔗 **Running Selenium Tests in Jenkins:** [To Be Added - Version 1.2]



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Experiment 7 - Configuration Management With Ansible

Experiment 7: Configuration Management with Ansible

Note:

This experiment has been created as a slide deck and is not included in this manual to preserve its original format and visual content.

You can **view and download the full PDF for Experiment 7** from our official DevOps resources folder on Google Drive using the link below:

 [Access Experiment 7 - Configuration Management with Ansible \(PDF\)](#)

 This Drive also contains all supporting files and resources used throughout the DevOps lab sessions.

CODERS ARCADE

COMMIT TO ACHIEVE



CA - Experiment 8 - Practical Exercise: Set Up a Jenkins CI Pipeline for a Maven Project, Use Ansible to Deploy Artifacts Generated by Jenkins.

This is a **detailed, step-by-step guide** to set up everything from **WSL installation** to **Jenkins Freestyle Project execution** with **Maven, Ansible, SSH setup (without password)**, and **remote deployment**.

Step-by-Step Guide: Jenkins + Maven + Ansible CI/CD on Windows WSL

◆ Overview

In this guide, we will: **Install WSL (Ubuntu on Windows)**

Set up Java, Maven, Git, and Ansible

Configure SSH connection between WSL & Remote Server

Set up Jenkins (running on Windows) and integrate it with SSH & Ansible

Execute a Jenkins Freestyle Project to deploy JAR files

Step 1: Install WSL (Windows Subsystem for Linux)

1.1 Enable WSL

COMMIT TO ACHIEVE

1 Open PowerShell as Administrator and run:

```
1 wsl --install
2
```

2 Reboot your system if required.

1.2 Install Ubuntu on WSL

1 Open Microsoft Store and install **Ubuntu 22.04 LTS**.

2 Launch **Ubuntu** and create a **username** and **password**.

3 **Update packages** inside Ubuntu:

```
1 sudo apt update && sudo apt upgrade -y
2
```

🟡 Step 2: Install Required Tools

📌 2.1 Install Java (JDK 17)

```
1 sudo apt install openjdk-17-jdk -y
2 java -version
3
```

Output should show openjdk version "17.0.x"

📌 2.2 Install Maven

```
1 sudo apt install maven -y
2 mvn -version
3
```

Ensure Maven is installed.

📌 2.3 Install Git

```
1 sudo apt install git -y
2 git --version
3
```



📌 2.4 Install Ansible

```
1 sudo apt update
2 sudo apt install ansible -y
3 ansible --version
4
```

CODERS ARCADE

🟡 Step 3: Set Up SSH Connection (No Password Required)

📌 3.1 Generate SSH Key

Run this inside **WSL (Ubuntu)**:

```
1 ssh-keygen -t rsa -b 4096
2
```

Press Enter for default options.

💡 Why do we write **4096** ? **ssh-keygen -t rsa -b 4096**

The **4096** in the command:

```
1 ssh-keygen -t rsa -b 4096
2
```

specifies the **key length** in bits.

👉 Explanation of the Command:

- ssh-keygen** → The command to generate an SSH key pair.

- `-t rsa` → Specifies the **type** of key to generate (RSA).
- `-b 4096` → Specifies the **bit length** (i.e., key size).

💡 Why Use 4096 Bits?

✓ Stronger Security:

- A **4096-bit key** is significantly more secure than the default **2048-bit key**.
- It provides better resistance against brute-force attacks.

✓ Long-Term Security:

- As computing power increases, **longer keys** ensure protection for a longer time.
- Many security professionals recommend **4096-bit RSA** for high-security environments.

✓ SSH Best Practices:

- While **2048-bit** is considered secure, **4096-bit** provides extra security without much performance loss.
- Modern systems can handle **4096-bit RSA** efficiently.

💡 When Should You Use 2048 Instead of 4096?

- If performance is a concern (e.g., on embedded devices or old systems).
- If you need faster SSH authentication times.
- However, for **most servers and personal use**, **4096-bit is preferred**.

💡 Conclusion:

Using 4096 makes your SSH keys stronger and more future-proof, ensuring better security for your remote connections. 

3.2 Copy Public Key to Remote Server

```
1 ssh-copy-id ronnie@172.27.158.79
2
```

If prompted, enter the **remote user password**.

COMMIT TO ACHIEVE

💡 Finding the IP Address of the Remote Machine!

The IP address 172.27.158.79 is the **private IP address** of your remote machine (where Ansible is running). Let's break down how you obtained it:

💡 How to Find the IP Address of the Remote Machine?

Since you are running **Windows WSL** and SSH-ing into a remote Linux machine (or another system), you likely found this IP address using one of the following methods:

💡 Method 1: Using `ip a` or `ifconfig` on the Remote Linux Machine

- 1 Log in to the **remote machine** (where Ansible is deployed).
- 2 Run the following command:

```
1 ip a
```

2

OR

```
1 ifconfig
2
```

3 Look for an entry under `eth0` or `ensXXX` (depending on the network interface).

- Example output:

```
1 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   default qlen 1000
2     inet 172.27.158.79/24 brd 172.27.158.255 scope global dynamic eth0
3
```

- Here, `inet 172.27.158.79` is the **IP address** of the machine.

Method 2: Using `hostname -I`

Run the following command on the remote machine:

```
1 hostname -I
2
```

This will return the machine's IP address, such as:

```
1 172.27.158.79
2
```

Method 3: Checking IP on WSL (for Windows Machines)

If you want to find your WSL (Windows Subsystem for Linux) instance's IP, run:

```
1 ip route | grep default
2
```

Example output:

```
1 default via 192.168.1.1 dev eth0 proto dhcp src 172.27.158.79 metric 100
2
```

Here, `172.27.158.79` is the **WSL machine's IP**.

How Was It Used in Jenkins?

- The IP address was used in **Jenkins SSH deployment**:

```
1 scp "C:/Users/Saurav/.jenkins/workspace/Maven-Ansible-Deploy/target/*.jar"
      ronnie@172.27.158.79:/home/ronnie/deployment/
2 ssh ronnie@172.27.158.79 "ansible-playbook /home/ronnie/ansible-deploy/deploy.yml"
3
```

- `ronnie@172.27.158.79` means:

- `ronnie` → The **username** on the remote machine.

- 172.27.158.79 → The **IP address** of the remote machine.

• Why Not Use `localhost` Instead?

- `localhost` refers to **your own system**, but your **Ansible machine** is on a separate system.
- If the remote machine is on the same network, you **must use its actual IP address** for SSH and SCP.

• What If the IP Changes?

If your remote machine **restarts** or reconnects, the IP may change (if it's dynamic).

 To get a **static IP**, configure a **static IP address** in the network settings or use a **DNS name**.

✓ Summary:

- The **IP address** 172.27.158.79 was obtained by checking the remote machine's network configuration (`ip a` or `hostname -I`).
- It is used for **SSH connection from Jenkins** to copy files and run the Ansible playbook.
- If it changes, you can find the new IP using the same commands and update your Jenkins job accordingly. 

3.3 Test SSH Login

```
1 ssh ronnie@172.27.158.79
2
```

 If it logs in **without asking for a password**, SSH is correctly set up.



Step 4: Install and Configure Jenkins

4.1 Install Jenkins on Windows

- 1 Download Jenkins MSI from [Jenkins Official Site](#).
- 2 Install Jenkins and start it as a **Windows Service**.
- 3 Open Jenkins Web UI at:

```
1 http://localhost:8080
2
```

4.2 Unlock Jenkins

- 1 Copy the **initial admin password** from:

```
1 C:\Users\Saurav\.jenkins\secrets\initialAdminPassword
2
```

- 2 Paste it into the Jenkins setup page.

4.3 Install Required Plugins

- Go to **Manage Jenkins** → **Manage Plugins** and install:
 -  **Maven Integration Plugin**

SSH Agent Plugin

🟡 Step 5: Create the Jenkins Freestyle Project

📌 5.1 Create a New Job

- 1 Go to Jenkins Dashboard → New Item
- 2 Select Freestyle Project → Name it **Maven-Ansible-Deploy**
- 3 Click **OK**

🟢 Step 6: Configure Jenkins Build Steps

📌 6.1 Set Up Source Code Management (Git)

- 1 Under **Source Code Management**, select Git
- 2 Enter your repository URL:

```
1 https://github.com/your-username/MVN-ANS-JEN-CICD.git
2
```

📌 6.2 Add Maven Build Step

- 1 Go to Build → Add Build Step → Execute Windows Batch Command
- 2 Paste this command:

```
1 cd %WORKSPACE%
2 mvn clean package
3
```

📌 6.3 Add SSH Deployment Step

- 1 Go to Build → Add Build Step → Execute shell script on remote host using SSH
- 2 Set **SSH site as:**

```
1 ronnie@172.27.158.79:22
2
```

3 Enter the command to transfer and deploy JAR:

```
1 scp "C:/Users/Saurav/.jenkins/workspace/Maven-Ansible-Deploy/target/*.jar"
ronnie@172.27.158.79:/home/ronnie/deployment/
2 ssh ronnie@172.27.158.79 "ansible-playbook /home/ronnie/ansible-deploy/deploy.yml"
3
```

Save the job.

ⓘ Steps to add **SSH Credentials** in **Jenkins**.

👉 How to Add SSH in Jenkins for Remote Deployment

To enable **Jenkins** to connect to a remote machine via **SSH**, follow these step-by-step instructions:

Step 1: Install SSH Plugin in Jenkins

- 1 Open **Jenkins** (<http://localhost:8080>).
- 2 Go to **Manage Jenkins** → **Manage Plugins**.
- 3 Click on the **Available** tab and search for "**SSH Pipeline Steps**" or "**Publish Over SSH**".
- 4 Select **Publish Over SSH** and click **Install Without Restart**.
- 5 Once installed, go back to the **Dashboard**.

Step 2: Add SSH Key to Remote Server (Already done for remote machine

Since your Jenkins job runs SSH commands on the remote machine (`ronnie@172.27.158.79`), you need to add the **public key** to the remote server.

- 1 Copy the public key to the remote machine using:

```
1 ssh-copy-id ronnie@172.27.158.79
2
```

OR (if `ssh-copy-id` is not installed)

```
1 cat ~/.ssh/id_rsa.pub | ssh ronnie@172.27.158.79 "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
2
```

- 2 Verify SSH login without a password:

```
1 ssh ronnie@172.27.158.79
2
```

If successful, Jenkins can now SSH into the remote server without needing a password.

Step 4: Configure SSH in Jenkins (Optional

- 1 Go to **Manage Jenkins** → **Manage Credentials**.
- 2 Under **Stores scoped to Jenkins**, click **(global)**.
- 3 Click **Add Credentials**.
- 4 Choose **Kind: SSH Username with Private Key**.
- 5 Set the following:

- **Username:** `ronnie`
- **Private Key:** Choose **Enter directly**, then paste the contents of:

```
1 cat ~/.ssh/id_rsa
2
```

- **Passphrase:** Leave it empty. **6** Click **OK**.

Step 5: Add SSH Host in Jenkins

- 1 Go to **Manage Jenkins** → **Configure System**.
- 2 Scroll down to **Publish Over SSH**.
- 3 Click **Add** under **SSH Servers**.
- 4 Fill in the details:

- **Name:** RemoteServer
- **Hostname:** 172.27.158.79
- **Username:** ronnie
- **Remote Directory:** /home/ronnie/deployment
- **Use Password Authentication, or use a different key:** Unchecked **[5]** Click **Test Configuration** to verify the connection.

Step 6: Configure Jenkins Job to Use SSH

[1] Open your Jenkins Freestyle Job (Maven-Ansible-Deploy).

[2] Click **Configure**.

[3] Under **Build Steps**, add:

- Execute shell script on remote host using SSH.

[4] Enter the SSH command to copy and deploy the JAR file:

```
1 scp "C:/Users/Saurav/.jenkins/workspace/Maven-Ansible-Deploy/target/*.jar"
ronnie@172.27.158.79:/home/ronnie/deployment/
2 ssh ronnie@172.27.158.79 "ansible-playbook /home/ronnie/ansible-deploy/deploy.yml"
3
```

[5] Click **Save**.

Step 7: Test the Setup

[1] Run the **Jenkins job**.

[2] Check the **Console Output** for SSH connection success.

[3] If everything works, Jenkins should:

- Copy the `.jar` file via SCP.
- Trigger the Ansible playbook to deploy the application.

 **Done!**  Your Jenkins pipeline now connects to the remote server over SSH for deployment. 

Step 7: Create Ansible Playbook for Deployment

7.1 Create Deployment Directory

On the **remote server**, run:

```
1 mkdir -p /home/ronnie/deployment
2
```

 The `-p` option in the `mkdir` command stands for "**parents**", and it ensures that:

-  If the parent directories in the given path **do not exist**, they will be created automatically.
-  If the directory **already exists**, no error will be thrown.

Example:

```

1 mkdir -p /home/ronnie/deployment
2

```

- ➊ If `/home/ronnie` already exists, only `deployment` will be created.
- ➋ If neither `/home/ronnie` nor `deployment` exists, both will be created.
- ➌ If `deployment` already exists, the command does **nothing** (no error).

This is useful to **avoid errors** when creating directories in scripts or automation! 

7.2 Write Ansible Playbook

1 Create the playbook:

```

1 nano /home/ronnie/ansible-deploy/deploy.yml
2

```

2 Paste the following YAML script:

```

1 ---
2 - hosts: localhost
3   tasks:
4     - name: Ensure deployment directory exists
5       file:
6         path: /home/ronnie/deployment
7         state: directory
8         mode: '0755'
9
10    - name: Copy the JAR file to the deployment folder
11      copy:
12        src: /home/ronnie/deployment/*.jar
13        dest: /home/ronnie/deployment/
14

```

3 Save and Exit (**CTRL + X**, then **Y**, then **Enter**).

7.3 Test Ansible Playbook

```

1 ansible-playbook /home/ronnie/ansible-deploy/deploy.yml
2

```

Step 8: Run the Jenkins Build

1 Go to Jenkins Dashboard → Maven-Ansible-Deploy

2 Click Build Now 

3 Jenkins will:  Clone your GitHub repo

 Build the **Maven project**

 Deploy the **JAR file using SSH & Ansible**

4 Check remote deployment:

```

1 ls -l /home/ronnie/deployment
2

```

 You should see the **JAR file** inside `/home/ronnie/deployment/` !

Congratulations! You've Successfully Set Up CI/CD! 🏆

- ✓ WSL, Maven, Git, and Ansible installed
- ✓ SSH connection configured (passwordless login)
- ✓ Jenkins job triggers SSH deployment using Ansible
- ✓ Fully automated CI/CD pipeline is up & running 🎉

Additional Resources 📚

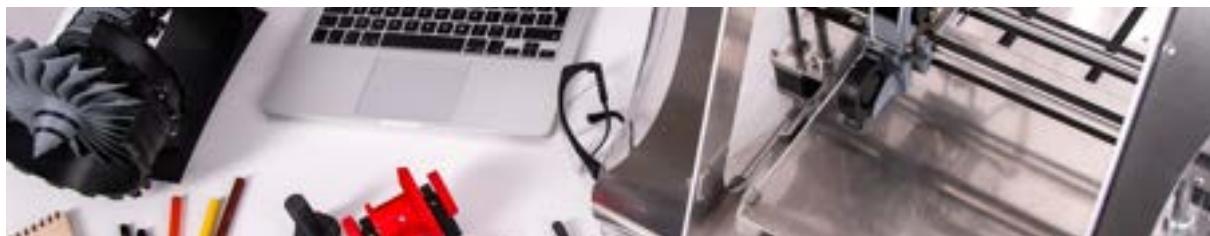
- Jenkins Documentation: [Jenkins User Documentation](#)
- Maven Documentation: [Official Maven Documentation](#)
- Ansible Documentation: [Ansible Official Documentation](#)

🎉 All the best! Happy DevOps-ing! 🎉🔥



CODERS ARCADE

COMMIT TO ACHIEVE



CA - Experiment 9 - Introduction To Azure DevOps

Here's the detailed and clean content for **Experiment 9: Introduction to Azure DevOps** – By Saurav Sarkar

Experiment 9: Introduction to Azure DevOps

Subject Code: BCSL657D

Course: DevOps Laboratory

Affiliated to: VTU (Visvesvaraya Technological University)

Objective

To introduce Azure DevOps, understand its core services, and set up an Azure DevOps **account** and **project** for DevOps workflow implementation.

Introduction to Azure DevOps

Azure DevOps is a set of development tools and services offered by Microsoft to support the entire software development lifecycle (SDLC). It helps in planning work, collaborating on code development, and building and deploying applications.

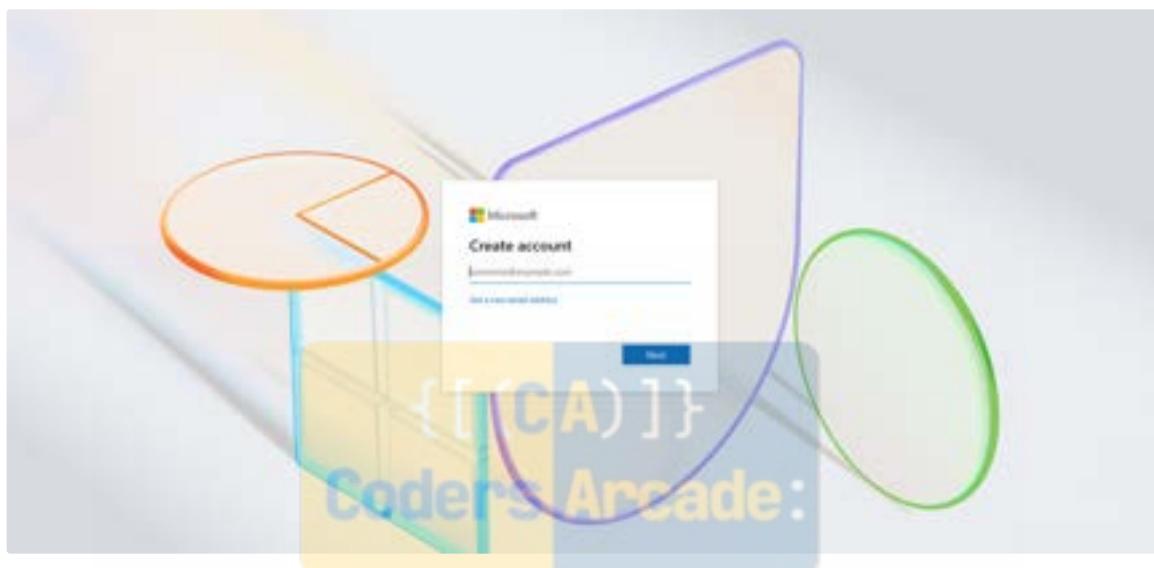
Key Services in Azure DevOps

Service Name	Description
Azure Boards	Agile planning, work item tracking, sprint planning, and dashboards.
Azure Repos	Git repositories for source code version control.
Azure Pipelines	CI/CD services for building, testing, and deploying applications.
Azure Artifacts	Package management for Maven, npm, NuGet, etc.
Azure Test Plans	Manual and exploratory testing tools.

🛠 Step-by-Step Procedure

✓ Step 1: Create a Microsoft Account (If Not Already)

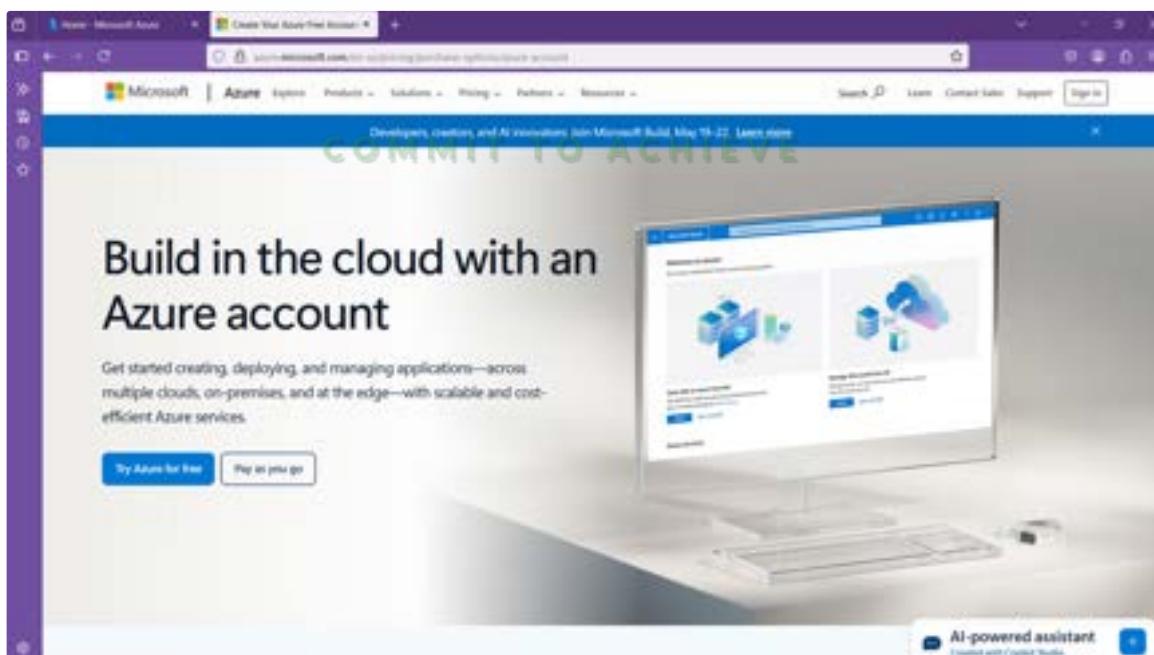
1. Go to <https://signup.live.com>
2. Create a new Microsoft account using your email ID.



Create Microsoft Account - (If You Don't Have One)

✓ Step 2: Sign in to Azure DevOps

1. Navigate to <https://dev.azure.com>
2. Log in with your Microsoft credentials.
3. Create A Free Azure Account with \$200 credit. (30 Days Trial Period)



Select The Try Azure For Free Option

4. Note : INR ₹2 will be deducted from your **Credit or Debit Card**, during subscription. It's mandatory for validation of **Azure Account**.

5. Initially you will not be able to deploy artefacts & won't be able to create build or release pipelines. You will need to disable some settings.

After the account creation, you will need to navigate to the home page : <https://portal.azure.com/#home>

6. On the search bar type " **Azure DevOps Organizations** " to go to the " [My Azure DevOps Organizations](#) " page as shown below & click on the link at the bottom left hand corner as shown.



CODERS ARCADE

[Click on My Azure DevOps Organizations Link](#)

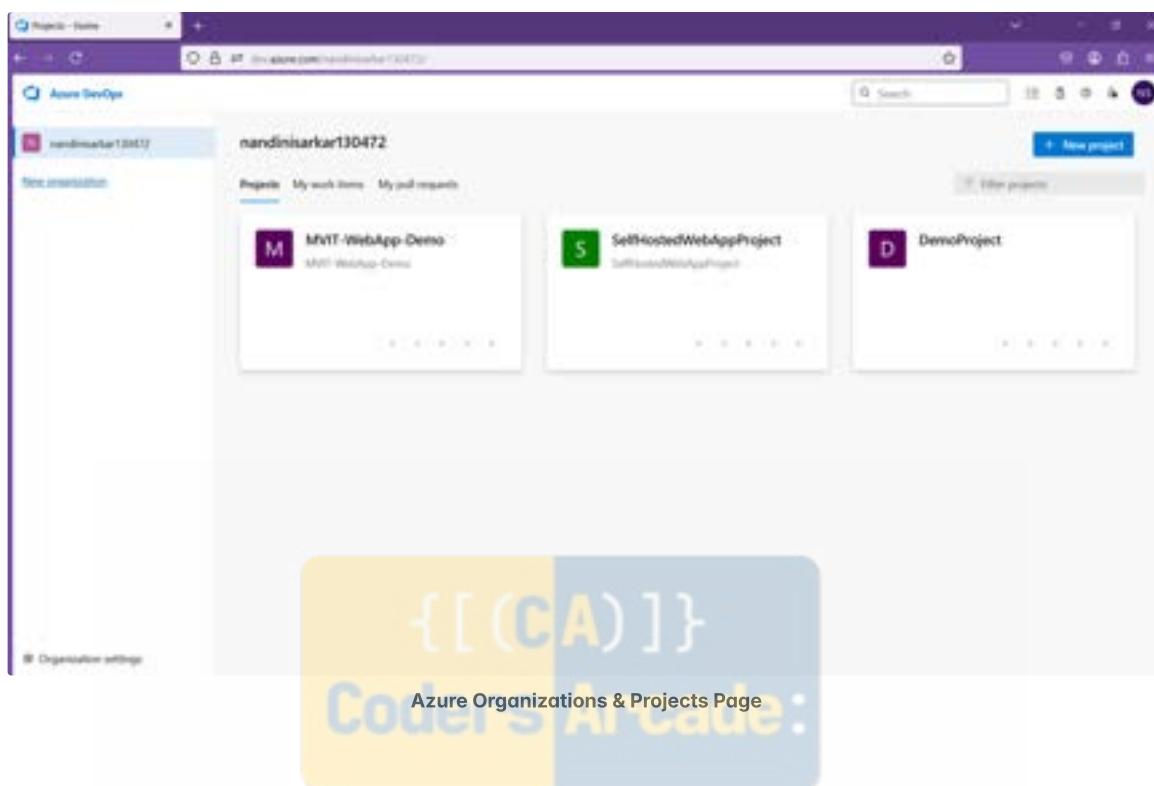
7. Once you have created the account, create a new organisation by clicking one the New Organisation Link as shown below on the left hand side of the screenshot.

8. Once created, you will need to remember the **Organization Name** or save it somewhere because it will be needed during **parallelism sign-up** for Microsoft. This helps in creating **agents** & also services like **web, db**, etc. You can see that the organisation name in our case is : <https://dev.azure.com/nandinisarkar130472/>

9. Initially there will be no projects under your Organisation. You can similarly create a **New Project** by clicking on the **New Project** link on the right hand side of the page as shown below on the screenshot.

10. Once created, all the **Projects** will reflect on the webpage as shown below in the snippets/screenshots eg **DemoProject**.

11. All the detailed steps are given in **Step 3** below after the screenshots.



✓ Step 3: Create a New Organization

1. Click on **New organization**
2. Choose a relevant name (e.g., `codersarcade-org`) and location (e.g., South India or East US)
3. Click **Continue**

✓ Step 4: Create a New Project

1. Click on **New Project**
2. Fill in:
 - **Project Name:** DevOps-VTU-Lab
 - **Description:** VTU Lab Manual Project for BCSL657D
 - **Visibility:** Private (recommended)
3. Click **Create**

CODERS ARCADE

COMMIT TO ACHIEVE

✓ Step 5: Signing - up for Parallelism

1. Parallelism is a feature which will help Azure DevOps users to start & use some important Azure services such as web servers, etc.
2. In the **Free Tier Accounts**, these features are not readily available.
3. So, we need to request Microsoft for some Free Tier Agents to perform our tasks.
4. **Please fill this form at least 5 days before you are planning to perform the experiments 9 till 12.**
5. **Generally, the parallelism service gets activated within 3 days but for safety, we will do it before 5 days of commencement of experiments 9 till 12.**
6. Go to this url: [Microsoft Forms](#) You will need your organisation name : Example :
<https://dev.azure.com/nandinisarkar130472/>
7. Fill up the details properly like **Name, Email, Organisation (Case Sensitive), Parallelism (Private)** & click **Submit** as shown below:

Azure DevOps Parallelism Request

This form is for users to request increased parallelism in Azure DevOps.

Please consider that it could take 4-5 business days to process the request. We are working on improving this process at the moment. Sorry for the inconvenience.

* Required

- What is your name? *

-> Your Name Here

- What is your email address? *

-> Your mail id here...!!!

- What is the name of your Azure DevOps Organization? *
(E.g. for <https://myorganization.visualstudio.com> or <https://dev.azure.com/myorganization> link formats - organization name would be 'myorganization')

-> This has to be correct. It's "Case Sensitive"

- Are you requesting a parallelism increase for Public or Private projects? *

Private -> Select Private
 Public

-> Submit

Never give out your password. [Report Abuse](#)

COMMIT TO ACHIEVE

✓ Step 5: Check for Parallelism Access

- After filling this form it is important to visit the **My Organisations** → **Organisation settings** page to keep checking if we have **Parallelism Access** or not.
- Here's the link → https://dev.azure.com/nandinisarkar130472/_settings/organizationOverview
- For your case it will be your organisation name that you have created earlier:
- On the **Organization Settings** sidebar, scroll down to select the **Parallel Jobs** option as shown:
- You should be able to see **Free Tier 1 parallel job upto 1800 mins/mo.** as shown in the screenshot below :
- After you get access, you will be able to perform the further experiments.
- Till then, explore the different pages & different options that are available on your **Azure Account**.

The screenshot shows the 'Organization Settings' page for the user 'nandinisarkar130472'. The left sidebar includes links for General, Overview, Projects, Users, Billing, Global notifications, Usage, Extensions, Microsoft Entra, Security, Security overview, Policies, Permissions, and Boards.

Private projects:

- Microsoft-hosted:** This should be Free tier. (Free tier: 1 parallel job up to 1000 mins/mo)
- Self-hosted:** 1 Parallel jobs
- Free parallel jobs:** 1
- Visual Studio Enterprise subscribers:** 0
- Monthly purchases:** 0 Change

Currently 0/1800 minutes are consumed

Only happens after Parallelism Access.

Public projects:

- Microsoft-hosted:** 0 Parallel jobs
- Self-hosted:** Unlimited Parallel jobs

Parallelism Access: Coders Arcade: COMMIT TO ACHIEVE

Outcome

You have now successfully:

- Created and configured an Azure DevOps organization.
- Created your first project within Azure DevOps.
- Explored the primary services available in Azure DevOps.

Assessment Questions

- What is Azure DevOps? List its major components.
- How does Azure Pipelines help in DevOps workflows?
- What is the difference between Azure Boards and Azure Repos?
- Why do we use Azure Artifacts?
- What steps are involved in creating a new Azure DevOps project?

Additional Resources

- [Microsoft Azure DevOps Documentation](#)
- YouTube: [Coders Arcade Channel](#)
- LinkedIn: [Saurav Sarkar](#)



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Experiment 10 - Creating Build Pipelines in Azure DevOps

 **Building a Maven/Gradle Project with Azure Pipelines, Integrating Code Repositories (e.g., GitHub, Azure Repos), Running Unit Tests and Generating Reports.**

Objective

- Create a **CI/CD pipeline** in **Azure DevOps**.
- Build a **Maven/Gradle project** using **Azure Pipelines**.
- Integrate **code repositories** (GitHub, Azure Repos).
- Run **unit tests** and generate reports.

Introduction to Azure Pipelines

Azure Pipelines is a CI/CD service that automates:

- Code building** (Maven, Gradle, .NET, etc.).
- Testing** (JUnit, Selenium, etc.).
- Deployment** (Azure App Services, Kubernetes, VMs).

Why Use Azure Pipelines?

- ✓ **Cloud-based & Scalable** – No need for local build agents.
- ✓ **Multi-Platform Support** – Works with Linux, macOS, and Windows.
- ✓ **Integration with GitHub, Azure Repos** – Fetches code from repositories.
- ✓ **Automation** – Continuous Integration (CI) and Continuous Deployment (CD).

Prerequisites:

- A Maven Project - Dynamic Web Project** should be ready in **Eclipse**.
- The war file(Web Application Archive)** should be exported and placed in the **target** folder of the same project.
- If you want to view the Web Application, make sure you have Apache Tomcat (**10.1 or above**) installed in your system.
- Also, **Tomcat Server** should be configured in **Eclipse** to view the Web Application.
- The same project should be pushed to **GitHub** and a valid **url** should be available. This will be connected to **Azure Pipelines**.
- If you don't have a **Maven Project - Dynamic Web Project**, here is the link to my GitHub repository → [GitHub - SauravSarkar-CodersArcade/MyWebApp-AzureDevOps: MyWebApp-AzureDevOps Practice](https://github.com/SauravSarkar-CodersArcade/MyWebApp-AzureDevOps)

Just Fork it in your GitHub Account and you will be ready to go. Here are the steps for forking:

1. Open the **GitHub** repository: [GitHub - SauravSarkar-CodersArcade/MyWebApp-AzureDevOps: MyWebApp-Azu
reDevOps Practice](https://github.com/SauravSarkar-CodersArcade/MyWebApp-AzureDevOps)
2. Click the **Fork** button on the top right corner of the page.
3. Select your **GitHub** account as the destination where the forked repository should be created.
4. **GitHub** will create a copy of the repository in your own account.
5. Open your newly forked repository and copy the URL (either **HTTPS** or **SSH**).

6. Use this forked repository **URL** when connecting your **Azure DevOps Pipeline to GitHub**.

With all these steps completed, you will be ready to start with **DevOPs VTU Lab → Experiment 10**.

Here is a complete step by step guide from scratch to execute **DevOPs VTU Lab Experiment 10**:

1. The Azure Project created from **Experiment 9** should be ready. For example **Demo Project or Test Project**.

The screenshot shows the Azure DevOps interface for a project named 'DemoProject'. On the left, there's a sidebar with options like Overview, Summary, Dashboards, Wiki, Boards, Report, Pipelines, Test Plans, and Artifacts. The main content area is titled 'About this project' and has a section for 'Help others to get on board!' with a button to 'Add Project Description'. To the right, there's a 'Project stats' panel showing various metrics: 0% for 'Builds queued', 0% for 'Deployments succeeded', and 0% for 'Compliance status'. At the bottom, there's a 'Members' section with one user listed.

2. Go to organization settings and make sure these settings are off as shown:

Your Organization → Organization Settings → Settings → They are **On** by default.

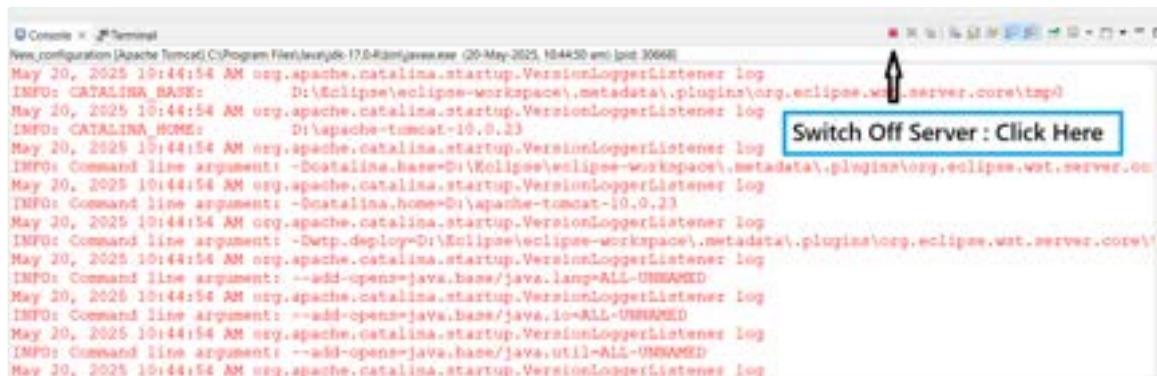
The screenshot shows the 'Project Settings' page for 'DemoProject' under 'Organization Settings'. The 'Settings' tab is selected. There are several options listed, each with a red arrow pointing to it and a callout box stating 'These two options should be off.' These options include:

- Limit variables that can be set at queue time
- Limit job authorization scope to current project for non-release pipelines
- Limit job authorization scope to current project for release pipelines
- Enable creation of classic build pipelines
- Enable creation of classic release pipelines

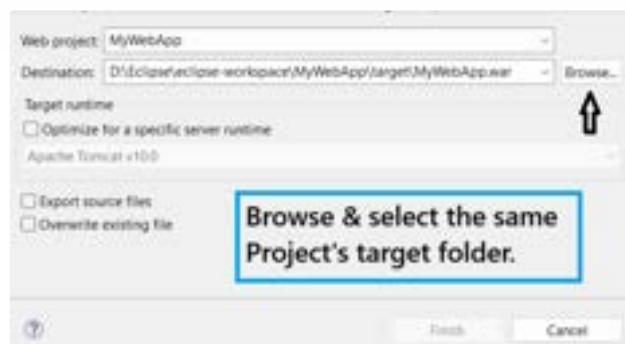
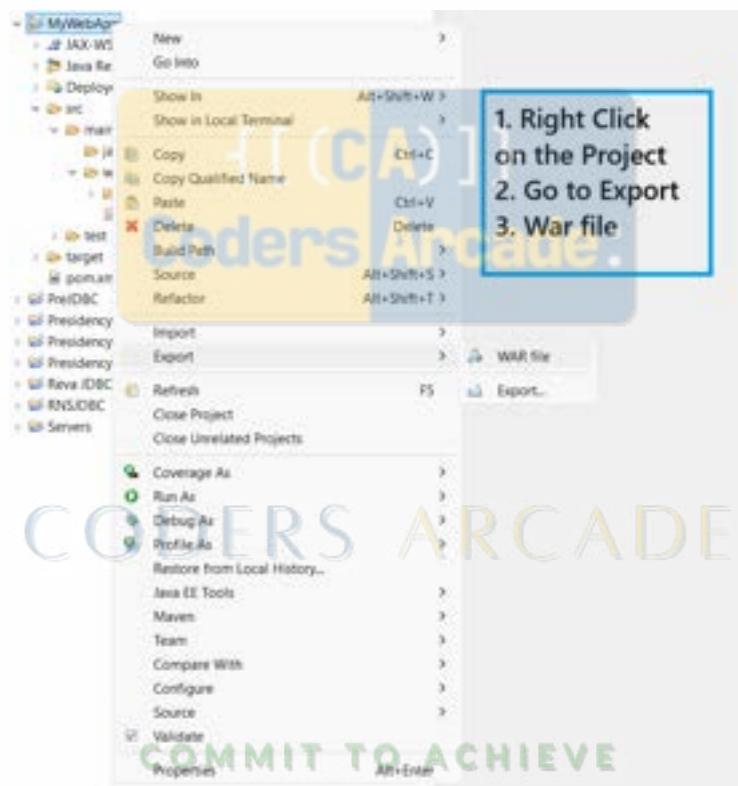
3. Go to your **Eclipse Maven Dynamic Web Project** & follow these steps:







```
Console < Terminal
New configuration [Apache Tomcat] C:\Program Files\Apache Software Foundation\Tomcat 10.0.23\bin\catalina.bat (20-May-2023, 10:44:54 AM)
INFO: Catalina HOME: D:\Eclipse\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
INFO: Catalina BASE: D:\Eclipse\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
INFO: Catalina HOME: D:\apache-tomcat-10.0.23
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
INFO: Command line arguments: -Dcatalina.home=D:\apache-tomcat-10.0.23
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
INFO: Command line arguments: -Dcatalina.base=D:\Eclipse\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
INFO: Command line arguments: --add-opens=java.base/java.lang=ALL-UNNAMED
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
INFO: Command line arguments: --add-opens=java.base/java.io=ALL-UNNAMED
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
INFO: Command line arguments: --add-opens=java.base/java.util=ALL-UNNAMED
May 20, 2023 10:44:54 AM org.apache.catalina.startup.VersionLoggerListener log
```



4. Make sure the same project is pushed to **GitHub**. If not, use this [URL](#) & **Fork** it using the steps as given in the beginning:

5. GitHub URL : [GitHub - SauravSarkar-CodersArcade/MyWebApp-AzureDevOps: MyWebApp-AzureDevOps Practice](https://github.com/SauravSarkar-CodersArcade/MyWebApp-AzureDevOps)

SauravSarkar-CodersArcade / MyWebApp-AzureDevOps Same Project Pushed To GitHub

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

MyWebApp-AzureDevOps

master · 8 branches · 8 tags · 24 commits

SauravSarkar-CodersArcade (set up CI with Azure Pipelines) · 1 month ago

- index Initial Commit, added index.json to the repository and verified... 1 month ago
- settings Initial Commit, added index.json to the repository and verified... 1 month ago
- src/main/webapp Updated index.json 1 month ago
- target Deleted the parent file 1 month ago
- changes Initial Commit, added index.json to the repository and verified... 1 month ago
- project Initial Commit, added index.json to the repository and verified... 1 month ago
- azure-pipelines.yml Set up CI with Azure Pipelines 0 days ago
- package Initial Commit, added index.json to the repository and verified... 1 month ago

About

MyWebApp-AzureDevOps Practice

- Activity
- 8 stars
- 1 watching
- 8 forks

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)

6. Go back to your Azure Project & follow the given steps:

Azure DevOps TestProject (last updated 2021-01-12)

TestProject

Overview Summary Wiki Boards Apps Pipelines Back to Your Azure Project -> In Your Organization

Click on Pipelines

Welcome to the project!

Project stats

No stats are available at this moment. Setup a service to see project activity.

COMMIT TO ACHIEVE

It will have no pipelines in the beginning, click on the button **Create Pipeline** to create your 1st Pipeline.



Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

Create Pipeline

Click Here

Connect Select Configure Review

New pipeline

Where is your code?

Azure Repos Git · YAML
Find private Git repositories, pull requests, and code review.

Bitbucket Cloud · YAML
Access by Atlassian.

Github · YAML Select this is our Source Code is on GitHub

Github Enterprise Server · YAML
The self-hosted version of Github Enterprise.

Other Git
Any generic Git repository.

Subversion
Centralized version control by Apache.

Use the classic editor to create a pipeline without YAML.

✓ Connect **Select** Configure Review

New pipeline

Select a repository

Filter by keywords My repositories

SauravSarkar-CodersArcade/RNSIT-DevOps Step: 2 This is my repository. Select your correct GitHub Repo.

SauravSarkar-CodersArcade/your-maven-project

SauravSarkar-CodersArcade/MyWebApp-AzureDevOps Step: 1

SauravSarkar-CodersArcade/BIET-Web

SauravSarkar-CodersArcade/BIET-JAVA-DSA

SauravSarkar-CodersArcade/MVN-ANS-IEN-CICD

SauravSarkar-CodersArcade/Modern-Periodic-Table-

SauravSarkar-CodersArcade/MVTF-DevOps-Bsp-2

SauravSarkar-CodersArcade/DevOps-Automation-Test

1. Initially, it might ask you to authenticate your GitHub Credentials.
 2. Give the correct credentials & validate.
 3. Once access is validated, select the correct repository.

New pipeline

Review your pipeline YAML

This is the heart of the Azure Build Pipeline : The **YAML File**

Variables Save and run ↗

Click Here

```

1 # Maven
2 # Build your Java project and run tests with Apache Maven.
3 # Add steps that analyze code, save build artifacts, deploy, and more:
4 # https://docs.microsoft.com/azure/devops/pipelines/languages/java
5
6 trigger:
7   - master
8
9 pool:
10   vmImage: ubuntu-latest
11
12 stages:
13   - name: Deploy
14     tasks:
15       - task: Maven@0
16         inputs:
17           workspaceDirectory: 'pom.xml'
18           arguments: '-DskipTests'
19           javaVersionOptions: 'JavaVersion'
20           javaLanguageOptions: 'J7_8'
21           jdkArchitectureOptions: 'x64'
22           publishJUnitResults: true
23           testResultsFile: '**/surefire-reports/TEST-*.xml'
24           goals: 'package'
25
26

```

- All the tasks that we perform in the build pipeline are configured here in the **YAML File**.
- If you want to keep the same file, don't make any changes.
- Else, you can delete the contents from line number 23 and start the steps as shown in the next slides/screenshots.
- If you have deleted the contents, then click on the Show assistant link:

New pipeline

Review your pipeline YAML

Variables Save and run ↗

Click Here

New pipeline

Review your pipeline YAML

Variables Save and run ↗

1. Type Here ➡

2. Click Here ➡

```

1 # Maven
2 # Build your Java project and run tests with Apache Maven.
3 # Add steps that analyze code, save build artifacts, deploy, and more:
4 # https://docs.microsoft.com/azure/devops/pipelines/languages/java
5
6 trigger:
7   - master
8
9 pool:
10   vmImage: ubuntu-latest
11
12 stages:
13   - name: Deploy
14     tasks:
15       - task: Maven@0
16         inputs:
17           workspaceDirectory: 'pom.xml'
18           arguments: '-DskipTests'
19           javaVersionOptions: 'JavaVersion'
20           javaLanguageOptions: 'J7_8'
21           jdkArchitectureOptions: 'x64'
22           publishJUnitResults: true
23           testResultsFile: '**/surefire-reports/TEST-*.xml'
24           goals: 'package'
25
26

```

← Copy files ⓘ

Source Folder ⓘ

Contents ⓘ

"/**/war" ➡ **Type this**

← Copy files ⓘ

Source Folder ⓘ

Contents ⓘ

**/*.jar

Target Folder ⓘ ⏪ Click Here

Advanced

We need the path variable because we don't have any static location.

{[(CA)]}

Coders Arcade:

Source Folder ⓘ

Contents ⓘ

**/*.jar

Target Folder ⓘ

Target folder or UNC path files will copy to. You can use variables. Example: \$build.artifactstagingdirectory

Advanced

This is the path variable.

COMMIT TO ACHIEVE

← Copy files ⓘ

Source Folder ⓘ

Contents ⓘ

**/*.jar

Target Folder ⓘ

Target folder or UNC path files will copy to. You can use variables. Example: \$build.artifactstagingdirectory

Advanced

Copy this variable.



New pipeline **COMMIT TO ACHIEVE** Review your pipeline YAML

SauravSarkar-CodersArcade/MyWebApp-AzureDevOps / azure-pipelines.yml ⓘ

```

  - task: Maven@3
    inputs:
      mavenGoalFile: 'pom.xml'
      mavenOptions: '-Xmx3072m'
      javaHomeOption: 'JDKVersion'
      jdkVersionOption: '17.0'
      jdkArchitectureOption: 'x64'
      publishJUnitResults: true
      testResultsFiles: '**/surefire-reports/TEST-*_xml'
      goals: 'package'
  - task: CopyFiles@2
    inputs:
      ContentFile: '**/*.war'
      TargetFolder: '${{build.artifactstagingdirectory}}'

```

This task gets added.

New pipeline

Review your pipeline YAML

Variables

Save and run

SauravSarkar-CodersArcade/MyHelloApp-AzureDevOps / azure-pipelines.yml * ·

```

  tasks:
    - task: Maven@0
      inputs:
        mavenPomFile: "pom.xml"
        mavenOptions: "-DskipTests"
        javaHomeOptions: "JDKVersion"
        javaVersionOptions: "17.0"
        javaArchitectureOptions: "x64"
        publishBuildResults: true
        testResultsuffix: "/surefire-reports/TEST-.xml"
        goals: "package"
    - task: CopyFiles@0
      inputs:
        contents: "**/*.war"
        targetFolder: "$(build.artifactstagingdirectory)"

```

1. Now we need to Publish our Build Artifact (war) file.
 2. Go to the Assistant again.

Tasks

Q: Search tasks

- .NET Core**
Build, test, package, or publish a .NET application...
- Android signing**
Sign and align Android API files
- Art**
Build with Spotify Art
- App Center distribute**
Distribute app builds to testers and users via the...
- App Center test**
Test app packages with Visual Studio App Center
- Archive files**
Compress files into .7z, .tar.gz, or .zip
- ARM template deployment**
Deploy an Azure Resource Manager (ARM) template...
- Azure App Configuration Export**



.NET Core
Build, test, package, or publish a .NET application...

Index sources and publish symbols
Index your source code and publish symbols to a ...

npm
Install and publish npm packages, or run an npm ...

Publish build artifacts ← Click
Publish build artifacts to Azure Pipelines or a Win...

Publish code coverage results
Publish any of the code coverage results from a bu...

Publish Pipeline Artifacts
Publish (upload) a file or directory as a named arti...

Publish Pipeline Metadata
Handle Pipeline Metadata to be used by other t...

[← Publish build artifacts](#)

Path to publish *

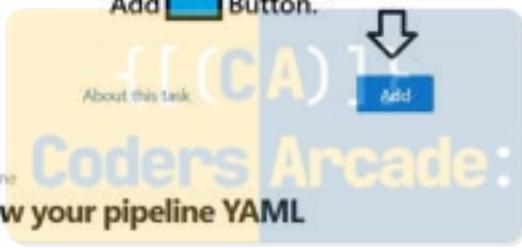
Artifact name *

Artifact publish location *

Max Artifact Size

Advanced

No changes, just click on Add Button.



Coders Arcade:
Review your pipeline YAML

Now pipeline

○ SauravSarkar-CodersArcade/MyWebApp-AzureDevOps / azure-pipelines.yml

```

13:   - task: Maven@3
14:     inputs:
15:       mavenXmlFile: 'pom.xml'
16:       mavenOptions: '-Dmaven.wagon.http.ssl.insecure=true'
17:       javaHomeOption: 'JDKVersion'
18:       jdkVersionOption: '17.0'
19:       jdkArchitectureOption: 'x64'
20:       publishJUnitResults: true
21:       testResultsFiles: '**/surefire-reports/TEST-*.xml'
22:       goals: 'package'
23:
24:   - task: CopyFiles@2
25:     inputs:
26:       Contents: "**/*.war"
27:       TargetFolder: $(build.ArtifactStagingDirectory)
28:
29:   - task: PublishBuildArtifacts@1
30:     inputs:
31:       PathToPublish: '$(build.ArtifactStagingDirectory)'
32:       ArtifactName: 'drop'
33:       publishLocation: 'Container'

```

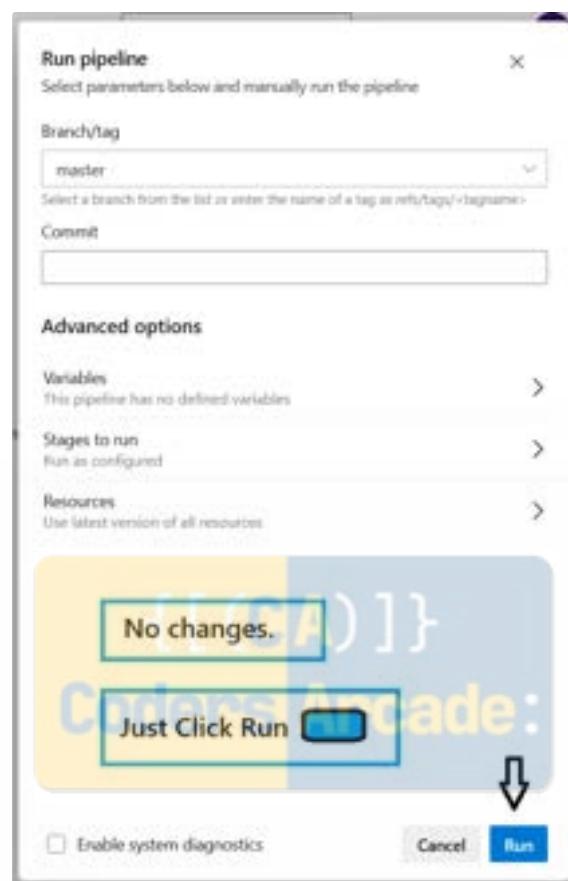
This gets added.

The screenshot shows the Azure DevOps Pipeline Editor interface. At the top, there's a header with 'Save Pipeline' (highlighted with a red arrow), 'Variables', and a 'Run' dropdown. Below the header is a 'Tasks' section with a search bar labeled 'Search tasks'. A list of tasks is displayed, including:

- .NET Core: Build, test, package, or publish a dotnet applicatio...
- Android signing: Sign and align Android APK files.
- Ant: Build with Apache Ant.
- App Center distribute: Distribute app builds to testers and users via Visua...
- App Center test: Test app packages with Visual Studio App Center.
- Archive files: Compress files into ZIP, RAR, GZIP, or ZIP...
- ARM template deployment: Deploy an Azure Resource Manager (ARM) template.

In the center, there's a large watermark for 'CODERS ARCADE' in blue and yellow. Below the tasks, there's a navigation bar with 'Raw', 'Branches', and 'Analytics' buttons, followed by a 'Run pipeline' button. A callout box with a red border and white text says 'Click any of the buttons to run the Pipeline.' There are also two red arrows pointing to the 'Run pipeline' button and the 'Run' dropdown in the header.

COMMIT TO ACHIEVE



#20250520.1 • Set up CI with Azure Pipelines

SauravSarkar/CodersArcade/MyWebApp-AzureDevOps

[Summary](#) [Code Coverage](#)

CODERS ARCADE

Manually run by Nandini Sarkar

Repository and version
SauravSarkar/CodersArcade/MyWebApp-AzureDevOps
master · 866e2d24

Time started and elapsed
Just now

Related
0 work items
0 artifacts

Tests and coverage
Get started

Jobs

A Job gets started.

COMMIT TO ACHIEVE

Your Azure Pipeline has started running.

Click Here

An arrow points from the 'A Job gets started.' message to the 'Click Here' button.

The screenshot shows a DevOps pipeline interface. On the left, a sidebar lists 'Jobs in run #20250520.1' under 'SauravSarkar' and 'CodersArcade.MyWebApp'. A large arrow points from the sidebar to the main content area. The main area shows a job titled 'Initialize job' with a status of 'Running'. The log pane displays the following steps:

```

1: Starting: Initialize job
2: Agent name: 'Hosted Agent' ← Blue arrow pointing here
3: Agent machine name: 'fv-az95-786'
4: Current agent version: '4.205.8'
5: Operating system
6: Runner Image
7: Runner Image Resolved
8: Current image version: '20250520.1.0'
9: Agent running as: 'root'
10: Prepare build directory...
11: Set build version...
12: Download all required tasks...
13: Downloading task: Maven (3.8.8)
14: Maven (3.8.8) version 3.8.8 (throughout) is incompatible.
15: Downloading task: SonarQube (3.0.1)
16: Downloading task: JenkinsfileParser (2.206.8)
17: Downloading task: PublishBuildArtifacts (7.347.1)

```

A callout box highlights step 2: 'Agent name: "Hosted Agent"' with the text: 'All commands & tasks from the YAML file will be executed step by step on an Ubuntu-image (VM Machine)'.

The screenshot shows a DevOps pipeline interface with a sidebar listing 'Jobs in run #20250520.1' under 'SauravSarkar' and 'CodersArcade.MyWebApp'. The main area shows several green circular icons indicating jobs are running. A large arrow points from the sidebar to the main content area. The main area shows a job titled 'Job' with a status of 'Running'. The log pane displays the following steps:

```

1: Pull: Agent Plugins
2: Image: ubuntu-latest
3: Queued: Job now [instance.socat(e), idle]
4: Agent: Hosted Agent
5: Started: Job was...
6: Duration: 2m
7: ...
8: The agent request is already running or has already completed.
9: Job preparation parameters
10: Job artifact produced
11: Job have command data...
12: Starting job...
13: Agent: Command Start: Detect Docker Container
14: Agent: Command End: Detect Docker Container
15: Agent: Command Start: Detect Docker Container
16: Agent: Command End: Detect Docker Container
17: Agent: Command End: Detect Docker Container
18: Finishing: Job

```

A callout box highlights step 8: 'The agent request is already running or has already completed.' with the text: 'The Job / Pipeline run continues. Keep checking the Logs.'

CODERS ARCADE

COMMIT TO ACHIEVE

The screenshot shows the 'Jobs in run #20250520.1' page from the Azure DevOps interface. The pipeline name is 'SauravSarkar-CodersArcade.MyWebApp-AzureDevOps'. A specific job named 'Job' is highlighted with a green checkmark icon. The log for this job shows the following steps:

- 1: Pull: Azure_Pipelines
- 2: Image: ubuntu-2004
- 3: Queued: Today at 11:10:46 (source_codeless.html)
- 4: Agent: Nested Agent
- 5: Started: Today at 11:11 am
- 6: Duration: 2ms
- 7: The agent request is already running or has already completed.
- 8: Job preparation parameters
- 9: ⚡ Artifact produced
- 10: Job live console data
- 11: Starting: Job
- 12: Azure_Command_Start / DetachWorkerContainer
- 13: Azure_Command_End / DetachWorkerContainer
- 14: Azure_Command_Start / DetectWorkerContainer
- 15: Azure_Command_End / DetectWorkerContainer
- 16: Finishing: Job

A callout box with the text "Look for errors in the logs if any." is positioned over the log output area.

In the top right corner of the main pane, there is a large yellow banner with the text "{{(CA)}}". Below it, another banner says "Coders Arcade". An arrow points from the word "Job" in the banner to the green checkmark icon on the left.

At the bottom of the page, there is a green banner with the text "COMMIT TO ACHIEVE".

On the left side of the main pane, there is a link labeled "Back to Pipeline".

A callout box with the text "If everything is green, the Job has run successfully." is positioned over the "COMMIT TO ACHIEVE" banner.

#20250520.1 - Set up CI with Azure Pipelines

This is the Build Pipeline that just executed.

Run new

This run is being retained as one of 3 recent runs by pipeline.

View retention leases

Summary Code Coverage

Manually run by Nandini Sarkar

Repository and version: SauravSarkar-CodersArcade/MyWebApp-AzureDevOps P master → R66e2d24

Time started and elapsed: Today at 11:31 am (27s)

Related: 0 work items 1 published

Tools and coverage: Get started

Warnings: 2

- Task 'Maven' version 3 (Maven@3) is deprecated.
- The Maven@3 task is deprecated, please use a newer version of the Maven task.

Jobs: Name Status Duration

Click Here

Artifacts

Published

Name	Status	Size
drop	1 KB	
target	1 KB	
MyWebApp.war	3 KB	

Our war file is deployed in the Azure Build Pipeline.

COMMIT TO ACHIEVE

The screenshot shows the Azure DevOps interface for a pipeline named '#20250520.1 - Set up CI with Azure Pipelines'. The pipeline has been manually run by Nandini Sarkar. It was triggered at 11:11 am today and completed 2 hours ago. The pipeline has 40 work items and 1 published artifact. There are 2 warnings: one about a deprecated Maven task and another about a deprecated Maven task. A message box says: 'This completes your Experiment 10 : Creating Build Pipeline in Azure Pipelines'.



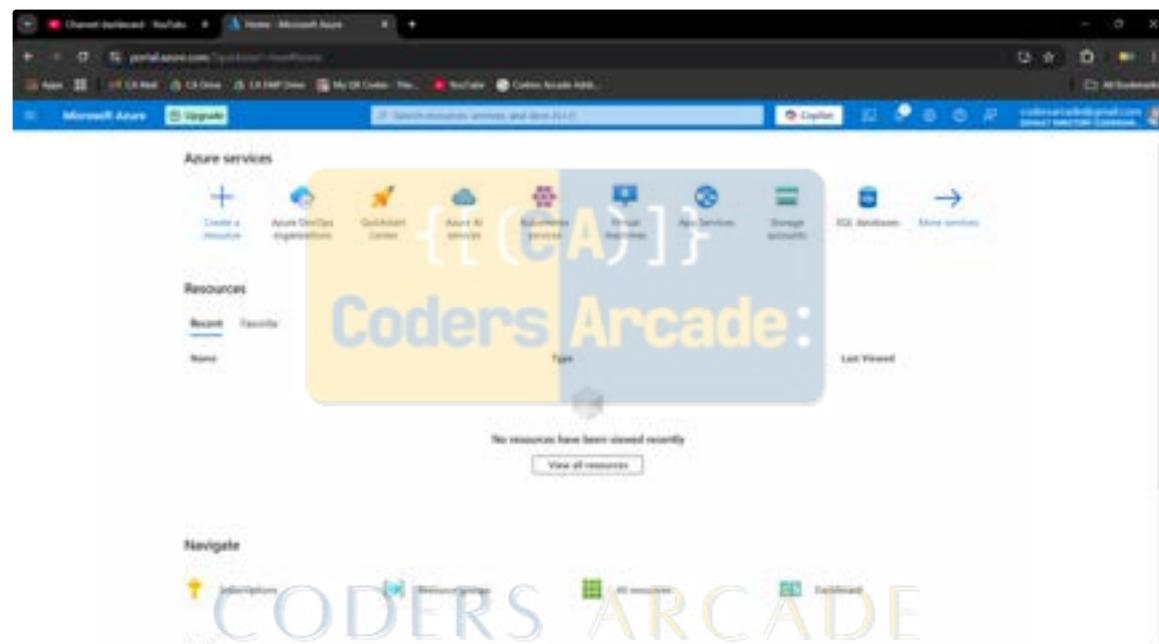
CODERS ARCADE

COMMIT TO ACHIEVE

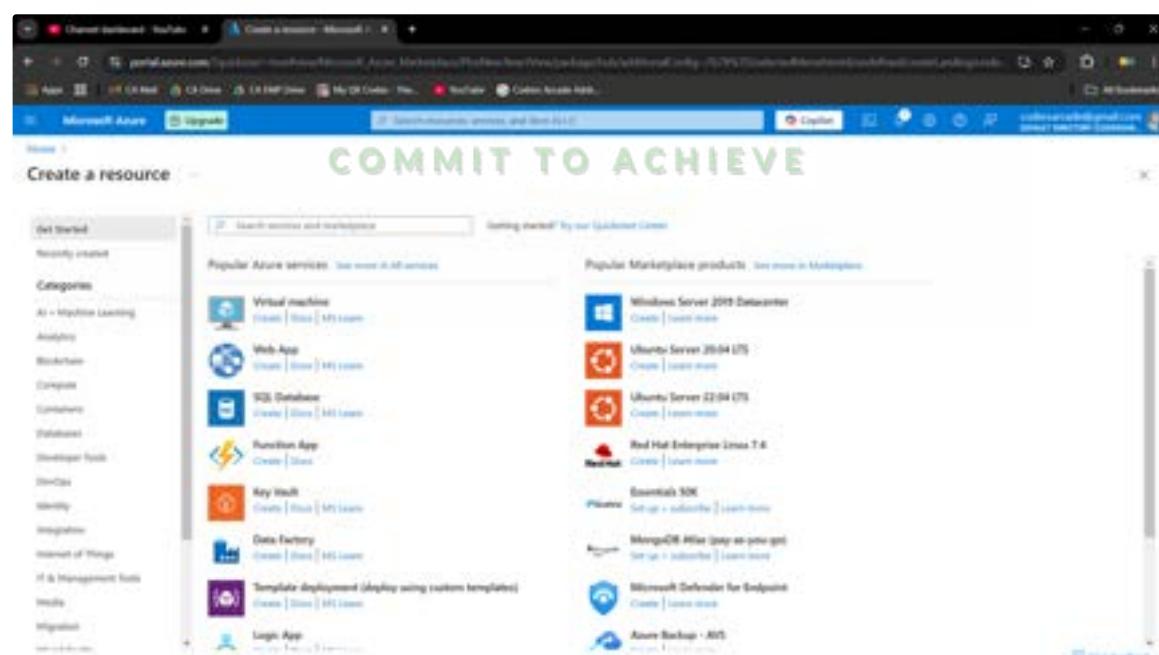
CA - Creating QA-Web-App & PROD-Web-App || Instructions For Exp 10 & 11

We will be creating two Web Apps for QA & Prod Deployment of Artifacts

Steps to create the QA-Web-App



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text 'Search services and Marketplace'. Below it, a large yellow banner displays the 'Coders Arcade' logo. On the left, a sidebar titled 'Azure services' includes icons for 'Create a resource', 'Azure DevOps integration', 'Quickstart Center', 'Alert AI insights', 'Feedback service', 'Storage accounts', 'Key vault', 'App Service', and 'More services'. The main content area shows a message: 'No resources have been viewed recently' with a 'View all resources' button. Below this, there's a 'Navigate' section with a 'CODERS ARCADE' logo.



The screenshot shows the 'Create a resource' wizard in the Microsoft Azure portal. The title bar says 'Create a resource - Microsoft Azure'. The left sidebar lists categories like 'Get Started', 'Recently created', and various service categories such as AI + Machine Learning, Analytics, Blockchain, Compute, Container, Database, Developer Tools, DevOps, Identity, Integration, Internet of Things, IT & Management Tools, Media, and Migration. The main panel has a 'Popular Azure services' section with icons for Virtual machine, Web App, SQL Database, Function App, Key Vault, Data Factory, and Template deployment. It also features a 'Popular Marketplace products' section with icons for Windows Server 2019 Datacenter, Ubuntu Server 20.04 LTS, Ubuntu Server 22.04 LTS, Red Hat Enterprise Linux 7.8, RHEL 8, CentOS 8.0, MongoDB Atlas (pay-as-you-go), and Microsoft Defender for Endpoint. A 'Getting started' link is visible at the top right.

Create Web App

Basic Database Deployment Networking Monitor + secure Tags Review + create

App Service lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Learn more about performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure management. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource Group *

Instance Details

Name Enter App name

Ensure unique default hostname. [More about this option](#)

Runtime * Node.js Common

Runtime stack *

Production profile * Local Azure

The screenshot shows the Microsoft Azure portal interface for creating a new web application. The top navigation bar includes 'Video demo - YouTube Studio', 'Create Web App - Microsoft App', 'Azure DevOps - Microsoft App', 'My Information', 'Settings - Parallel pipe (public)', and 'All Subscriptions'. The main title is 'Create Web App'. The 'Instance Details' section shows the name 'codersarcadeCA' and the option to 'Select unique default hostname (ex. https://codersarcade.net)'. The 'Runtime stack' is set to 'Node.js', and the 'Operating system' is 'Linux'. The 'Region' is 'West US (US1)'. Under 'Pricing plan', it says 'App Service plan pricing tier determines the location, features, cost and compute resources allocated to your app'. The 'App Service Plan (Canada Central)' dropdown shows 'Java', 'Java 21', 'Java 17', and 'Java 11'. The 'Review + create' button is at the bottom left.

This screenshot is identical to the one above, but it features a large yellow watermark in the center containing the text '{[(CA)]}'.

This screenshot is identical to the one above, but it features a large blue watermark in the center containing the text 'CODERS ARCADE'.

This screenshot is identical to the one above, but it features a large green watermark in the center containing the text 'COMMIT TO ACHIEVE'.

Create Web App

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Resource Group *

Instance Details

Name * https://codersarcadecentral-17.codersarcade.net

Ensure unique default hostname on. More about this setting

Runtime stack * Node Container

Platform features Docker GitHub

Pricing plan

App Service plan pricing tier Apache Tomcat 10.0

Usage Plan (Capacity Control) *

Review + Create

Create Web App

Instance Details

Name: codersarcade

Choose unique default hostname (ex. codersarcade.azurewebsites.net)

Runtime: Code Container

Runtime stack: .NET 7.0

Java with server stack: Apache Tomcat 10.1

Operating system: Linux

Region: Canada Central

Non-Regional app: App Service Plan? By a different region or contact your App Service Administrator.

Pricing plan:

App Service plan pricing tier determines the location, Azure cost and compute resources associated with your app. Learn more [↗](#)

Basic (Canada Central) New ASP .NET Core app (Windows)

Review + create Previous New - Database

Basic Database Deployment Networking Monitor + secure Tags Review + create

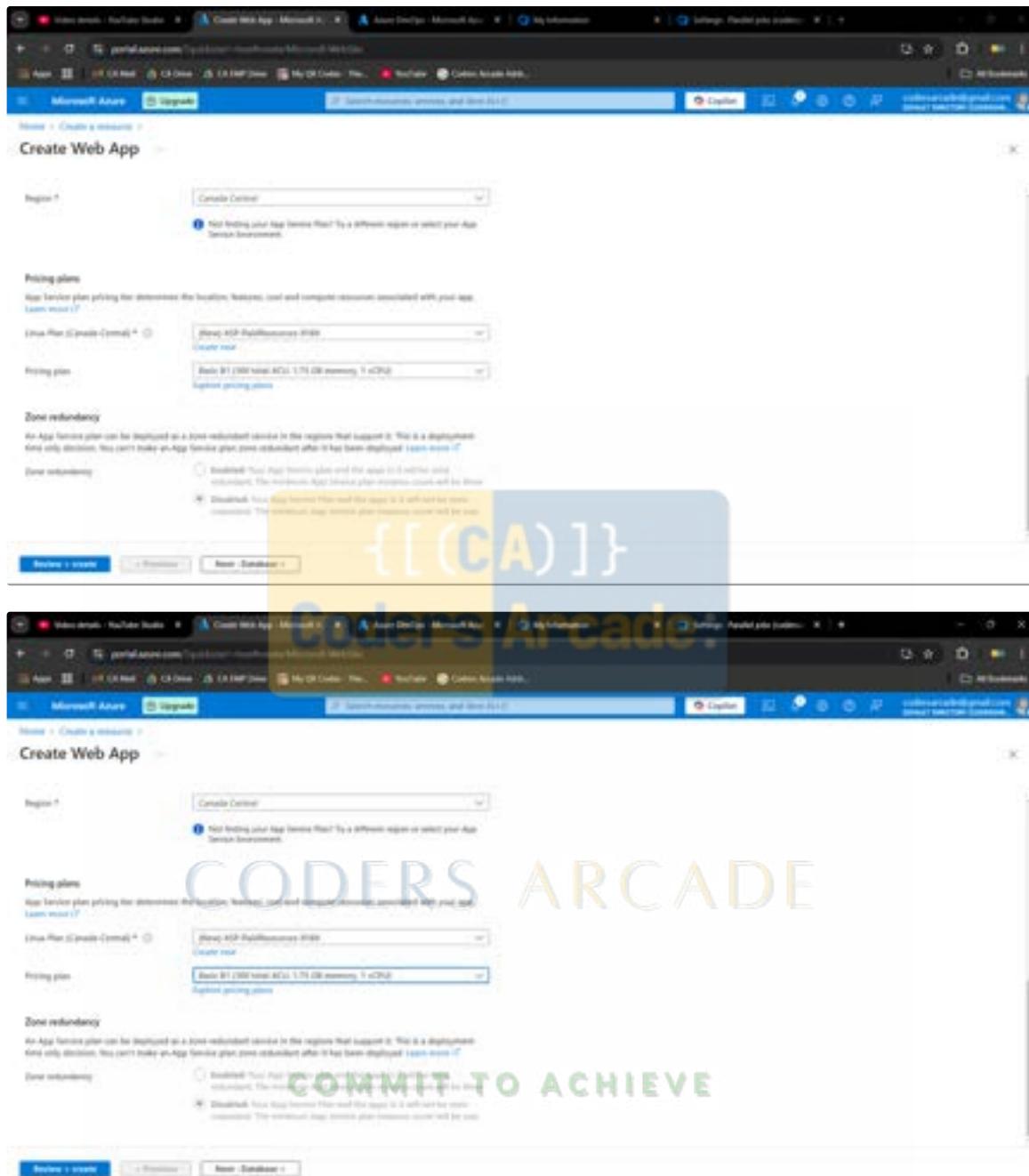
This instance is created; a new app and related networking resources will be created automatically. [Learn more ↗](#)

Create a Database

CODERS ARCADE

COMMIT TO ACHIEVE

Review + create Previous New - Deployment



Create Web App

Review + create

Details

Subscription	az790233-af50-47f0-bec1-f7f7f0d00000
Resource Group	codersarcade
Name	codersarcade
Secure unique domain (optional)	
Region	East US
Runtime stack	Java 17
Java web server stack	Apache Tomcat 10.1

App Service Plan (New)

Plan	AS-1
Region	West US 2
Compute Power	Standard
Memory	1.75 GB Memory

Monitor + secure (New)

Log type	AS-1
Logs	CloudWatch Metrics
Logs	CloudWatch Logs
Logs	CloudWatch Metrics
Logs	CloudWatch Metrics

The screenshot shows the Microsoft Azure portal interface for creating a new Web App. The app name is set to 'Coders Arcade'. The configuration includes:

- Kontena stack:** Java 17, Apache Tomcat 10.1
- App Service Plan (Standard):** Name: 'ASP-PacifcResource-WEB', Region: 'West Central', Size: 'Small', Total ACUs: 1.75, Memory: 1.75 GB
- Monitor + secure (None):** Application Insights: Enabled, Name: 'codersarcadeWeb', Region: 'Central US'
- Deployment:** Basic authentication: Disabled, Continuous deployment: Not enabled (Setup after app creation)

A large watermark with the text 'CODERS ARCADE' is overlaid on the screenshot.

Create Web App

Review + create

Details

Subscription	00000000-0000-0000-0000-000000000000
Resource Group	codersarcade
Name	codersarcade
Secure unique domain (optional)	
Region	East US
Runtime stack	Java 17
Java web server stack	Apache Tomcat 10.1

App Service Plan (New)

Deployment is in progress... Deployment to resource group 'codersarcade' is in progress.

Microsoft.Web-WebApp-Portal-817b4a27-ad23 | Overview

Deployment is in progress

Deployment name: Microsoft.Web-WebApp-Portal-817b4a27-ad23
Start time: 2023-07-07T23:48:00
Completion ID: 00000000-0000-0000-0000-000000000000

COMMIT TO ACHIEVE

The image shows two screenshots of the Microsoft Azure Portal, illustrating the deployment process of a web application.

Screenshot 1 (Deployment in Progress):

- Deployment Status:** Deployment is in progress.
- Deployment Details:** Deployment name: Microsoft.Web-WebApp-Portal-817b4a27-ad23, Start time: 2023/07/21 13:48:03, Correlation ID: 90394ab-4e29-4a07-99b1-4c10d1f00000.
- Feedback:** Give feedback, If I tell you about your experience with deployment.
- Right Panel:** Microsoft Defender for Cloud, Free Microsoft tutorials, Work with an expert.

Screenshot 2 (Deployment Complete):

- Deployment Status:** Deployment successful.
- Deployment Summary:** Deployment Microsoft.Web-WebApp-Portal-817b4a27-ad23 to resource group TestResource has succeeded.
- Next Step:** Manage deployment for your app (Recommended), Protect your app with authentication (Recommended).
- Feedback:** Give feedback, If I tell you about your experience with deployment.
- Right Panel:** Cost Management, Microsoft Defender for Cloud, Free Microsoft tutorials, Work with an expert.

Watermark: CODERS ARCADE COMMIT TO ACHIEVE

The image shows two separate screenshots of the Microsoft Azure Portal, both displaying the 'Overview' page for a 'Web App'.

Screenshot 1 (Top): Microsoft.Web-WebApp-Portal-817b4a27-ad23 | Overview

- Deployment Status:** Your deployment is complete.
- Deployment Summary:**
 - Deployment name: Microsoft.Web-WebApp-Portal-817b4a27-ad23
 - Subscription: Free Trial
 - Resource group: portalResourceGroup
 - Start time: 2023/07/21 13:48:03
 - Correlation ID: 93934ab-6235-4ad7-93b1-4c1d10f0e
- Deployment Details:**
 - Manage deployment for your app: Recommended
 - Protect your app with authentication: Recommended
- Next steps:**
 - Manage deployment for your app: Recommended
 - Protect your app with authentication: Recommended
- Give Feedback:** Give feedback
- Need assistance with deployment?** Need assistance with deployment?

Screenshot 2 (Bottom): sauravDemoCA | Overview

- Resource Group:** portalResourceGroup
- Subscription:** Canada Central
- Location:** Canada Central
- Default domain:** portalResourceGroup.sauvademo.ca (41.182.162.144)
- App Service Plan:** Standard (1 vCore)
- Health Check:** Enabled
- Logs:** Add logs
- Properties:**
 - Web app:** Name: sauravDemoCA, Publishing model: Code, Runtime stack: Java 11 (OpenJDK)
 - Deployment Center:** Deployment logs, Last deployment, Deployment provider: None
 - Application Insights:** Name: sauravDemoCA, Log type: Application, Log level: Error, Application insights provider: None
- Monitoring:** Metrics, Log Analytics, Application Insights
- Logs:** Download log file, View log file, Share on GitHub, Need assistance with logs?
- Logs (Logs):** Default log, Custom log
- Metrics:** Default metric, Custom metric
- Logs (Metrics):** Default metric, Custom metric
- Logs (Custom):** Add custom log
- Logs (Log Analytics):** Default log, Custom log
- Logs (Application Insights):** Application insights provider: None
- Logs (Metrics):** Default metric, Custom metric
- Logs (Custom):** Add custom metric
- Logs (Log Analytics):** Default log, Custom log
- Logs (Application Insights):** Application insights provider: None

The screenshot shows the Microsoft Azure portal interface. The main focus is on the 'sauravDemoCA' web app configuration page. Key details include:

- Resource group:** sauravDemoCA
- Status:** Running
- Location:** Canada Central
- Subscription:** Test Dell
- Subscribed At:** 07/06/2024
- Default domain:** sauravdemoca.scm.azurewebsites.net
- App Service Plan:** APP Standard (S1)
- Operating System:** Linux
- Health (Logs):** Not Configured

The 'Properties' tab is selected, showing the following details for the 'Web app' section:

- Name:** sauravdemoca
- Publishing model:** Code
- Business plan:** Java 11 Standard
- Deployment Center:** Deployment logs, Last deployment, Deployment provider
- Domains:** Default domain (sauravdemoca.scm.azurewebsites.net), Custom domains
- Application Insights:** Name: sauravdemoca, Region: Canada Central (Show More)

Below the configuration page, there is a promotional banner for Coders Arcade:

Hey, Java developers!
Your app service is up and running.
Time to take the next step and deploy your code.

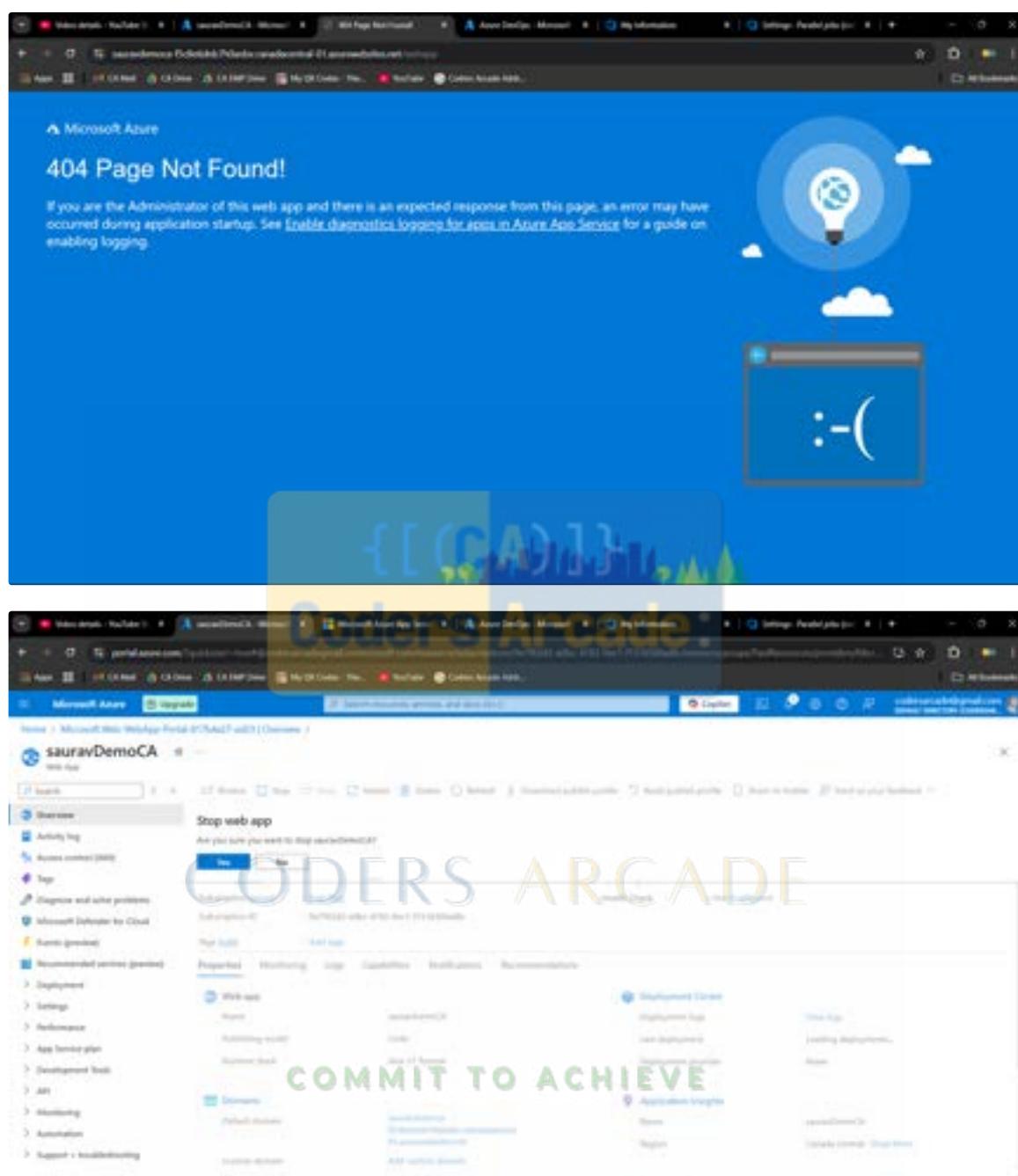
Have your code ready?
Use deployment center to get code published
from your client or setup continuous deployment.

Don't have your code yet?
Follow our quickstart guide and you'll
have a full app ready in 5 minutes or less.

Deployment Center **Quickstart**

Technical Information
Container image: AzureStandardJava
Java version: 17.0.18
Java home: /usr/lib/jvm/java-17-openjdk-amd64

COMMIT TO ACHIEVE



sauravDemoCA

Essentials

- Resource group: sauravDemoCA
- Status: Staged
- Location: Canada Central
- Subscription: India East
- Subscribed At: 2024-07-05 06:17:11 UTC

Properties

Web app	Deployment
Name: sauravDemoCA	Deployment logs
Publishing model: Code	User deployment
Business plan: Java 11 Standard	Deployment provider: None

Domains

- Default domain: sauravdemoca.azurewebsites.net
- Custom domains: Add custom domain

Application Insights

- Name: sauravDemoCA
- Region: Canada Central (View more)

Steps to create the PROD-Web-App

Azure services

- Cloud resources
- Azure DevTest Laboratories
- Quickstart Center
- AI services
- Container service
- Compute
- App Service
- Storage accounts
- SQL databases
- More services

Resources

Name	Type	Last Viewed
sauravDemoCA	App Service	9 minutes ago
sausage	Resource group	8 minutes ago

Commit To Achieve

Navigate

- Subscriptions
- Resource groups
- All resources
- Dashboard

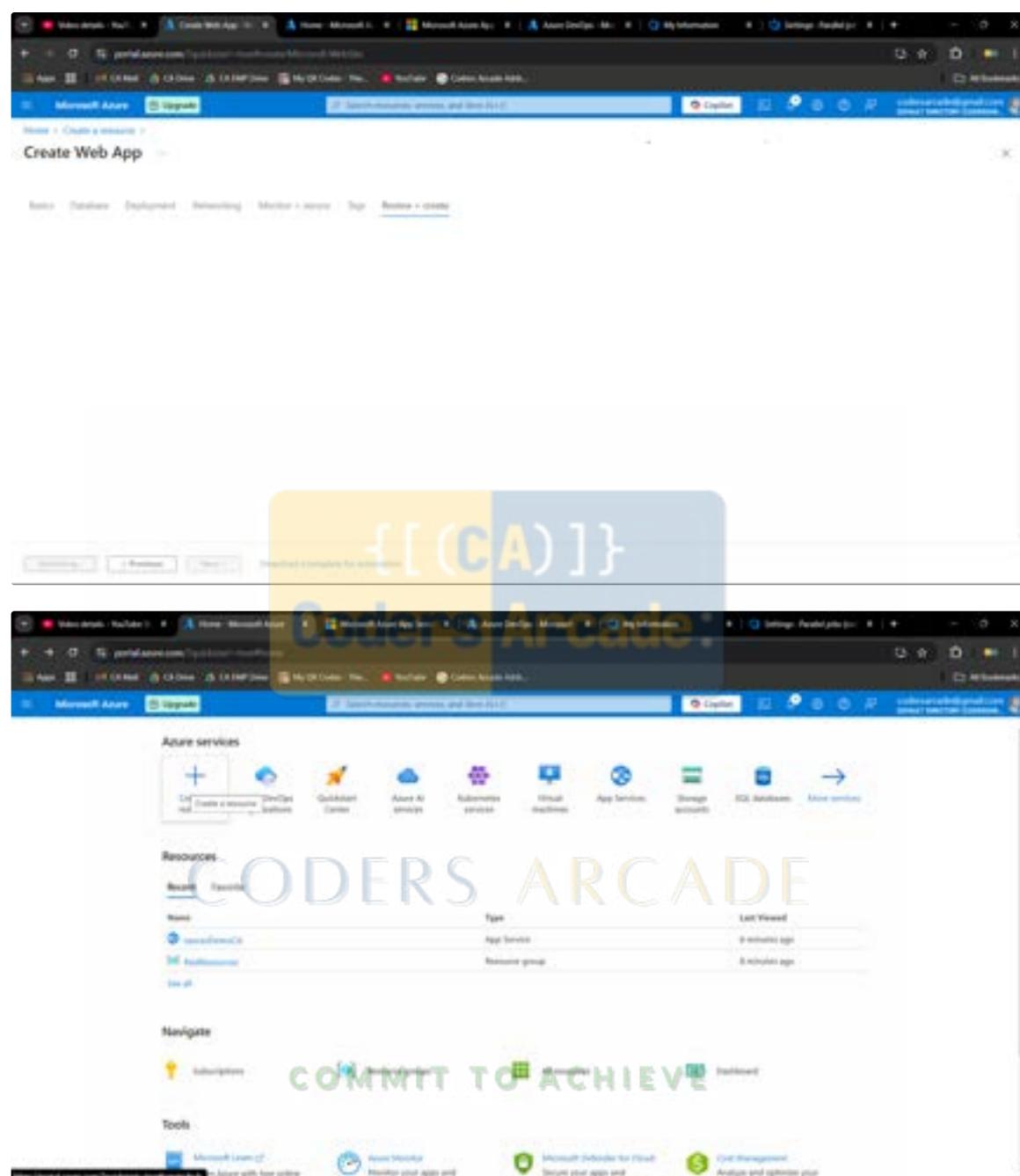
Tools

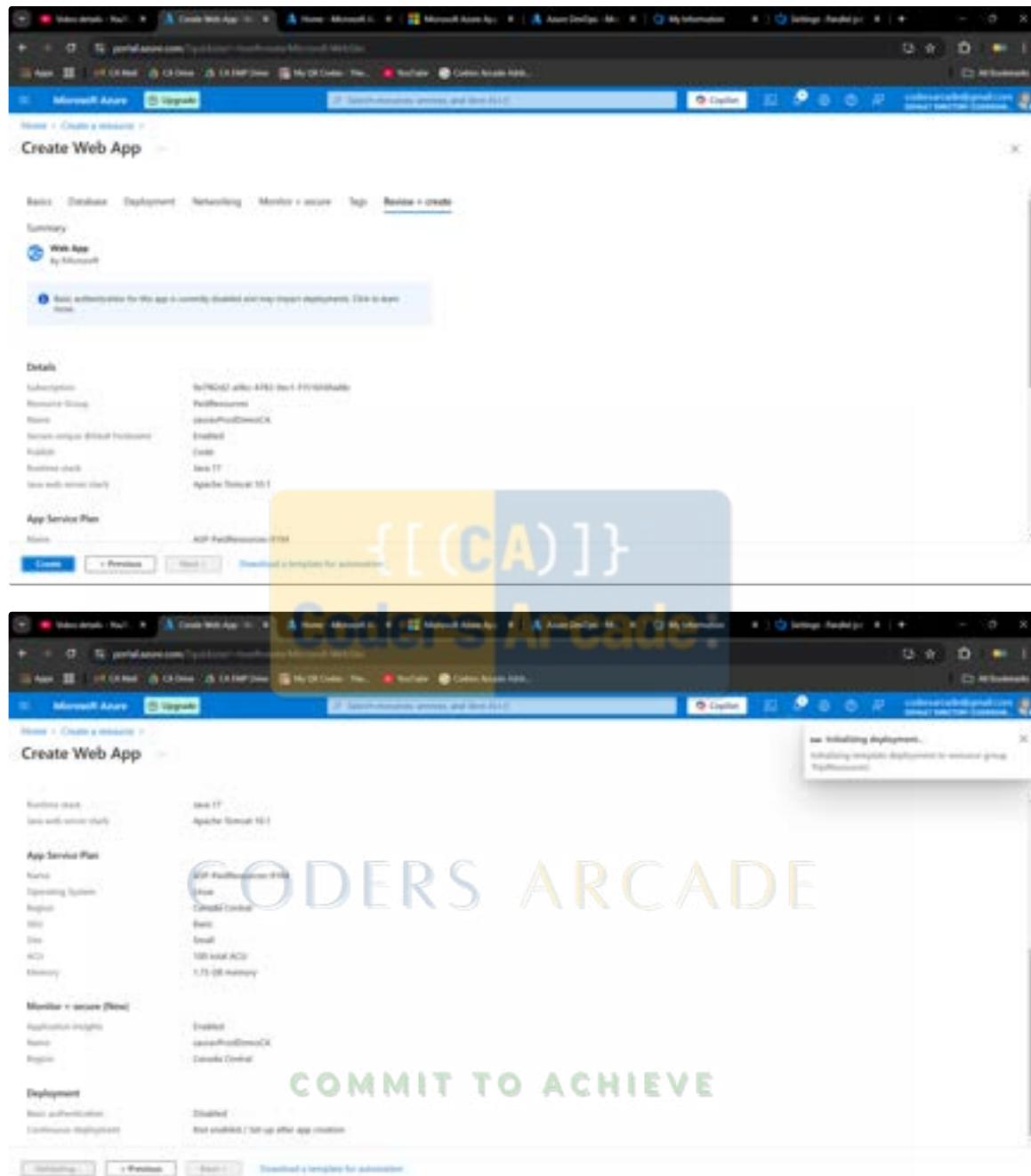
- Microsoft Learn
- Microsoft Defender for Cloud
- Microsoft Monitoring
- Microsoft Defender for Identity
- Cloud Management

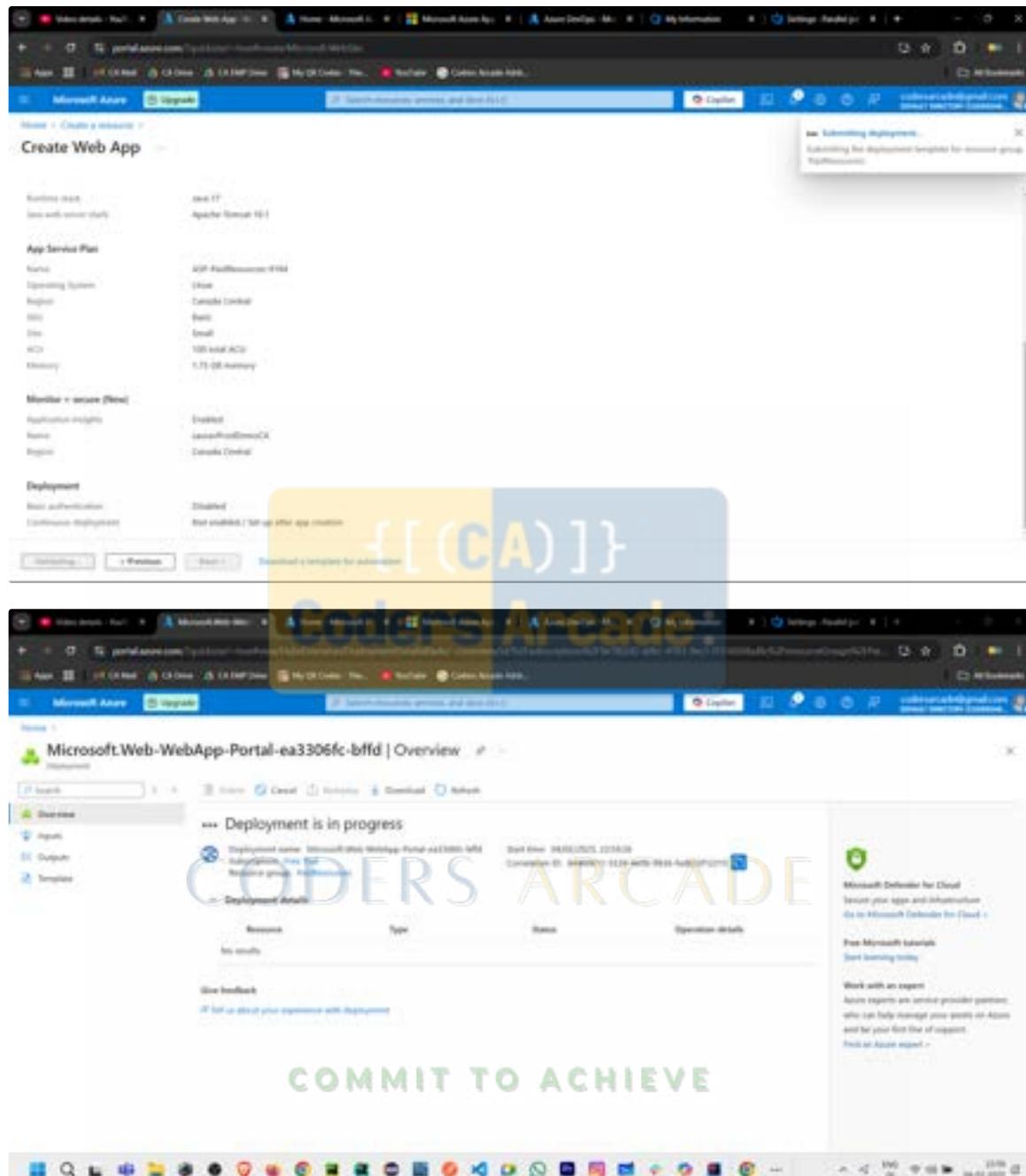
The image shows two screenshots of the Microsoft Azure portal side-by-side.

Top Screenshot: The main 'Create a resource' page. It features a sidebar on the left with 'Get Started' and 'Categories' sections. The 'Categories' section lists various Azure services like AI, Blockchain, Container, Database, Functions, Identity, Integration, Internet of Things, Monitoring & Management, Media, and Migration. The main area displays 'Popular Azure services' and 'Popular Marketplace products'. A specific product, 'Microsoft Defender for Endpoint', is highlighted with a yellow background and a callout bubble.

Bottom Screenshot: A detailed view of creating a 'Web App'. The form includes fields for 'Subscription' (selected: 'Free Trial'), 'Resource Group' (selected: 'codersarcade'), 'Name' (input: 'codersarcade'), 'Ensure unique default hostname (optional)', 'Publish' (radio button selected: 'Code'), 'Runtime stack' (selected: 'Node.js'), 'Default web server stack' (selected: 'Apache Tomcat 10.1'), 'Operating system' (radio button selected: 'Linux'), and 'Region' (selected: 'Canada Central'). At the bottom, there are buttons for 'Review + create' and 'Cancel'.







The screenshot shows the Microsoft Azure Portal with two tabs open:

- Microsoft.Web-WebApp-Portal-ea3306fc-bffd | Overview**: This tab displays deployment details. It shows a green checkmark indicating "Your deployment is complete" with a timestamp of "Start time: 2023/07/21 23:58:08" and a correlation ID of "84490711-1129-4676-9531-4d030920c". It also lists "Deployment details" and "Next steps" with links to "Manage diagnostics for your app" and "Protect your app with authentication". A "Give feedback" button and a "Tell us about your experience with deployment" link are present.
- sauravProdDemoCA | Overview**: This tab shows the properties of a web app named "sauravProdDemoCA". The properties include:
 - Resource group**: sauravProdDemoCA
 - Subscription**: Canada Central
 - Default domain**: sauravproddecomca.azurewebsites.net
 - App service plan**: Standard (S1) v2
 - Runtime stack**: Java 17 (Standard)
 - Deployment Center**: Deployment logs, Last deployment, Deployment provider: None
 - Application Insights**: Name: sauravProdDemoCA, Log type: Java Central (View more)

A large watermark for "CODERS ARCADE" and "COMMIT TO ACHIEVE" is overlaid across both screens.

sauravProdDemoCA

Essentials

- Resource group: sauravProd
- Status: Running
- Location: Canada Central
- Subscription: saurov
- Subscribed At: 2017-04-06T17:51:11Z

Properties

Web app

- Name: sauravProdDemoCA
- Publishing model: Business plan
- Code: Java EE Server

Deployment Center

- Deployment logs
- Last deployment
- Deployment provider: None

Domains

- Default domain: sauravproddemo.ca
- Custom domains: none

Application Insights

- Item: sauravProdDemoCA
- Region: Canada Central (View more)

Hey, Java developers!

Your app service is up and running. Time to take the next step and deploy your code.

Have your code ready? Use deployment center to get code published from your client or setup continuous deployment.

Don't have your code yet? Follow our quickstart guide and you'll have a full app ready in 5 minutes or less.

Deployment Center

Dashboard

Technical Information

- Container: Java 8
- Java version: 11.0.11
- Java home: /usr/lib/jvm/java-11-amazon-jdk11

COMMIT TO ACHIEVE

The screenshot shows two consecutive screenshots of the Microsoft Azure portal interface, specifically for a web application named "sauravProdDemoCA".

Screenshot 1: The user is prompted to stop the web app. A modal dialog box asks, "Are you sure you want to stop sauravProdDemoCA?" with "Yes" and "No" buttons. The background shows the Azure portal's navigation bar and the "Overview" tab of the web app's configuration page.

Screenshot 2: The modal has been closed, and a success message "Successfully stopped web app" is displayed at the top right of the screen. The message says "Successfully stopped web app sauravProdDemoCA". The background shows the same configuration page with the "Web app" section visible.

Now on the home page you can see the services that you have created.

The image shows two screenshots of the Microsoft Azure portal. The top screenshot displays the main Azure services dashboard with various service icons like Create a resource, Azure DevOps Organization, Quickstart Center, etc. Below this is a table of resources:

Name	Type	Last Viewed
surveyDemoCA	App Service	5 minutes ago
businesses	Resource group	2 minutes ago
surveyDemoCA	App Service	10 minutes ago

The bottom screenshot shows a detailed view of the 'surveyDemoCA' App Service. It includes a large preview image with the text '[[(CA)]] Coders Arcade: COMMIT TO ACHIEVE'. The 'Resource details' pane shows the following information:

Type	Last Viewed
App Service	5 minutes ago
Resource group	2 minutes ago
App Service	10 minutes ago

Both screenshots show the same browser tabs at the top, including Video, Home, Microsoft, Microsoft, Microsoft, Microsoft, Microsoft, Microsoft, My Dev Codes, YouTube, and Coders Arcade App.

The screenshot shows two views of the Microsoft Azure portal:

- Top View (Resource Group Details):**
 - Resource Group:** sauravProdDemoCA
 - Status:** Running
 - Location:** Canada Central
 - Subscription:** Test Dell
 - Subscribed At:** 2024-01-05T11:11:11Z
 - Tags:** Add tags
 - Properties:** Web app, Deployment Center, Application Insights
 - Domains:** Default domain, Custom domains
- Bottom View (Azure Services Dashboard):**
 - Azure services:** Create a resource, Azure DevOps Pipelines, Quickstart Center, Azure AI services, Subscriptions, Virtual machines, App Service, Storage accounts, ECR artifacts, More services.
 - Resources:** sauravProdDemoCA (App Service), sauravTestCA (Virtual Machine), sauravTestCA (Storage account).
 - Resource details:** Type: App Service, Resource group: sauravProdDemoCA, Last Viewed: 10 minutes ago.

The screenshot shows two side-by-side views of the Microsoft Azure portal.

Top View: The Azure services blade is open, showing the 'Azure DevOps organizations' section. It includes a description of what Azure DevOps organizations offer, a 'View' button, and links to 'Virtual machines', 'App Service', 'Storage accounts', 'SQL Database', and 'More services'. On the left, there's a sidebar with 'Resources' and a list of recent resources including 'sauravDemoCA', 'sauravDevOpsOrg', and 'Ingestion'. Below the sidebar is a 'Navigate' section with 'Subscriptions', 'Resource groups', 'All resources', and 'Dashboards'.

Bottom View: A detailed view of a specific Azure resource, a 'Web app' named 'sauravDemoCA'. The properties pane shows the following details:

- Essentials:**
 - Resource group: sauravDemoCA
 - Subscription: sauravDemoCA
 - Location: Central US
 - Default domain: sauravdemo.cacentral.cloudapp.azure.com
 - App service plan: sauravDevOpsOrg (1 vCore)
 - Operating system: Linux
 - Health check: /api/healthcheck (optional)
- Properties:**
 - Name: sauravDemoCA
 - Publishing model: Code
 - Runtime stack: Java 11 (Optional)
 - Domains:
 - Default domain: sauravdemo.cacentral.cloudapp.azure.com
 - Add custom domain
- Deployment Center:**
 - Deployment logs: None
 - Last deployment: None
 - Deployment provider: None
- Application Insights:**
 - Name: sauravDemoCA
 - Region: Asia Pacific (Optional)
 - Log type: None

The background of the bottom view features a large watermark reading 'CODERS ARCADE' and 'COMMIT TO ACHIEVE'.

CA - Experiment 11 - Creating Release Pipelines - Azure App Services

Objectives of Experiment 11

1. Understand the power of **Release Pipelines** in Azure DevOps for seamless web app deployment 
2. Deploy the generated `.war` file from the build pipeline to the **QA App Service (demoCA)** using a release pipeline 
3. Configure **stages, tasks, and environments** in the release pipeline like a DevOps pro 
4. Link the **build artifacts** directly to the release pipeline for smooth integration 
5. Perform **automated testing** using **Java + Selenium WebDriver** to validate deployments 
6. Test the live QA deployment URL using a JUnit test case (`BrowserTest.java`) and check the page title 
7. Strengthen your understanding of continuous delivery by bridging **Build → Release → Test** steps seamlessly 

Optional: Integrate Azure Key Vault & Secrets in Experiment 11

You can enhance your DevOps pipeline security by using **Azure Key Vault** to manage secrets and fetch them securely during deployment. Here's how:

Step 1: Create Azure Key Vault

1. Go to the **Azure Portal**.
2. Search for **Key Vaults** in the top search bar and click **Create**.
3. Fill in the basic details:
 - **Name:** e.g., `myKeyVaultDemo`
 - **Region:** Same as your App Service
 - **Resource Group:** Use existing or create new
4. Click **Review + Create** → **Create**.

Step 2: Add Secrets to Key Vault

1. Open the created Key Vault → Go to **Secrets** → Click **+ Generate/Import**.
2. Enter:
 - **Name:** e.g., `dbPassword`
 - **Value:** your secret value (e.g., `MySecurePass123!`)
3. Click **Create**.

Step 3: Give Azure DevOps Access to Key Vault

1. Go to **Access Policies** in your Key Vault.
2. Click **+ Add Access Policy**.
3. Select these permissions under Secret permissions:
 - **Get**
 - **List**
4. Under **Principal**, search for `Project Collection Build Service (your_project_name)` and select it.
5. Click **Add** → **Save**.

Step 4: Link Key Vault in Azure DevOps Release Pipeline

1. Go to your **Release Pipeline** in Azure DevOps.

2. In the **Variables** tab → Click **Link secrets from an Azure Key Vault as variables**.
3. Select the correct Azure subscription and Key Vault.
4. Choose the secrets you want to import.
5. These secrets will now be available as variables using `$(secretName)` format in the pipeline tasks.

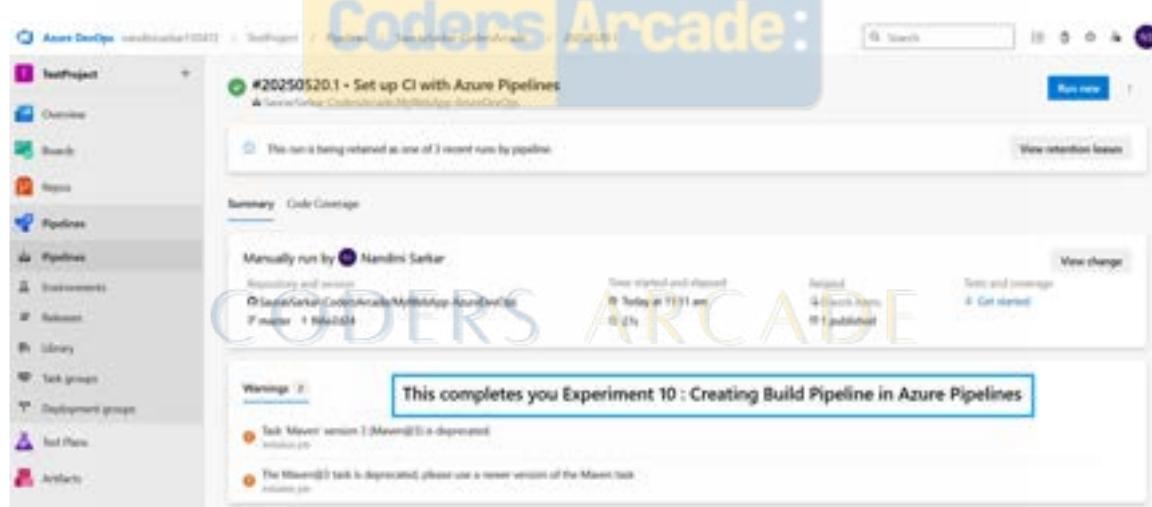
Note: This step is **optional**, but highly recommended for **production-grade** security practices.

Note:

For the **simple demo applications** used in our **Lab Experiments**, integrating Azure Key Vault is **not mandatory**. These labs are designed for educational purposes, focusing on understanding DevOps concepts and pipeline flows. However, in **real-world production environments**, using **Azure Key Vault** is highly recommended to securely manage secrets like passwords, API keys, and connection strings. 

Follow these steps for the execution of Experiment 11

1. Start from the Azure Build Pipeline that was created in the last experiment (**Experiment 10**).



The screenshot shows the Azure DevOps interface for a pipeline named '#20250520.1 - Set up CI with Azure Pipelines'. The pipeline has been triggered manually by 'Nandini Sarkar'. It includes a 'Code Coverage' step and a 'Run tests' step. A warning message at the bottom states: 'This completes your Experiment 10 : Creating Build Pipeline in Azure Pipelines'. The warning details two issues: 'Task: Maven version 3 (Maven@3) is deprecated' and 'The Maven@3 task is deprecated; please use a newer version of the Maven task'.

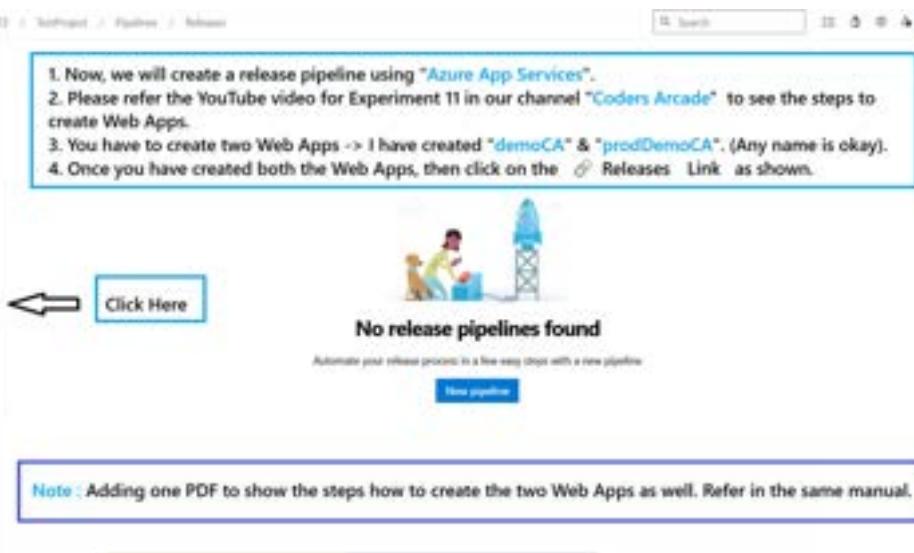
2. Now follow the steps as given below.

1. Now, we will create a release pipeline using "Azure App Services".

2. Please refer the YouTube video for Experiment 11 in our channel "Coders Arcade" to see the steps to create Web Apps.

3. You have to create two Web Apps -> I have created "demoCA" & "prodDemoCA". (Any name is okay).

4. Once you have created both the Web Apps, then click on the Releases Link as shown.



No release pipelines found

Automate your release process in a few easy steps with a new pipeline

[New pipeline](#)

Note: Adding one PDF to show the steps how to create the two Web Apps as well. Refer in the same manual.



This is the First Stage (QA)

1. Once you click on **Releases**, this screen will come up.
 2. We are deploying our war file to **Azure App Services**.



- To create the Web Apps you can refer the video for Experiment 11 or else the PDF section that I have attached along with the manual.
- Go to [Azure Home Page](#) to start the Web Apps that you created earlier in the last step.

Hi Nandini, see what more you can get from your Azure free account.

[View remaining credits](#) for my any service, or [Get more credits](#) included with your account. [About Azure credits](#) Before your 30 day trial is over, log in to continue to access your Azure account and monthly free services.

COMMIT TO ACHIEVE

- Take a free online course on Microsoft Learn
- Watch a demo and attend a live Q&A
- Start a project with Quickstart Center
- Explore support resources

Azure services

- Create a resource
- Azure DevOps organization
- Quickstart Center
- Azure AI Studio
- Kubernetes service
- Virtual machines
- App Services
- Storage accounts
- SQL databases
- More services

Make sure you navigate to Azure Home Page & Start the Resources (The two Web Apps)

Scroll Down and you will be able to see the resources that you have created earlier.

Resources

DemoCA

Name: DemoCA

Type: App Service

Kind: applinux

Location: Canada Central

Subscription: Azure subscription 1

Resource group: devops-training

Status: Stopped

App Type: Web App

Navigate

Subscriptions: CA)

Click here

Web App Started

Successfully started web app DemoCA

Successfully started web app ProdDemoCA

Essentials

Resource group (selected): DevOps-Training	Default domain: demoa.azuredashboards.net
Status: Stopped	App Service Plan: ASP-DevOpsTraining-WEB1 (S1)
Location (selected): Canada Central	Operating System: Linux
Subscription (selected): Azure subscription 1	Health Check: Current service health check status while app is in a "Stopped" state.

Properties

Web app	Name: DemoCA	Deployment log: demoa.azuredashboards.net
Publishing model:	Code	Last deployment: 2023-07-10T11:51:10Z
Runtime Stack:	Java 17 (Tomcat)	Deployment provider: VSTS

Logs

Capabilities

Notifications

Recommendations

Deployment Center

Application Insights

Domains

COMMIT TO ACHIEVE

ProdDemoCA

Click

Similarly, start the other Web App

Started

Successfully started web app ProdDemoCA

Essentials

Resource group (selected): DevOps-Training	Default domain: proddemo.azuredashboards.net
Status: Stopped	App Service Plan: ASP-DevOpsTraining-WEB1 (S1)
Location (selected): Canada Central	Operating System: Linux
Subscription (selected): Azure subscription 1	Health Check: Current service health check status while app is in a "Stopped" state.

Properties

Web app	Name: ProdDemoCA	Deployment log: proddemo.azuredashboards.net
Publishing model:	Code	Last deployment: 2023-07-10T11:51:10Z
Runtime Stack:	Java 17 (Tomcat)	Deployment provider: VSTS

Logs

Capabilities

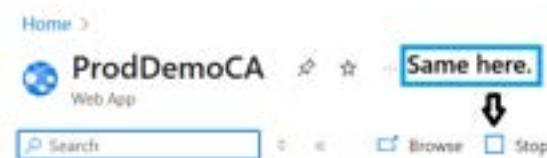
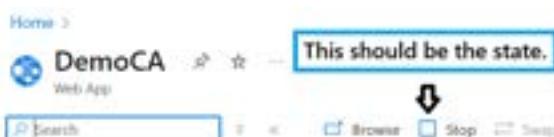
Notifications

Recommendations

Deployment Center

Application Insights

Domains



5. Go back to your release pipeline & follow the steps given below.

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add Stages | + Add

Click on the Stage 1 & rename it.

Stage
Stage 1
Properties Name and owners of the stage
Stage name Stage-1
Stage owner Nandini Sarkar

All pipelines > New release pipeline

Pipeline Tasks Variables Retention Options History

Artifacts | + Add Stages | + Add

COMMIT TO ACHIEVE

QA-Deployment
1. Rename

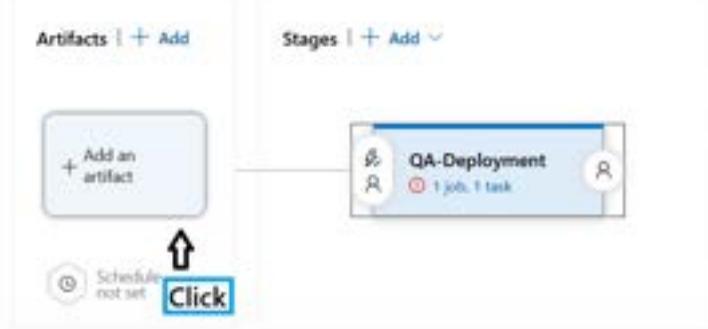
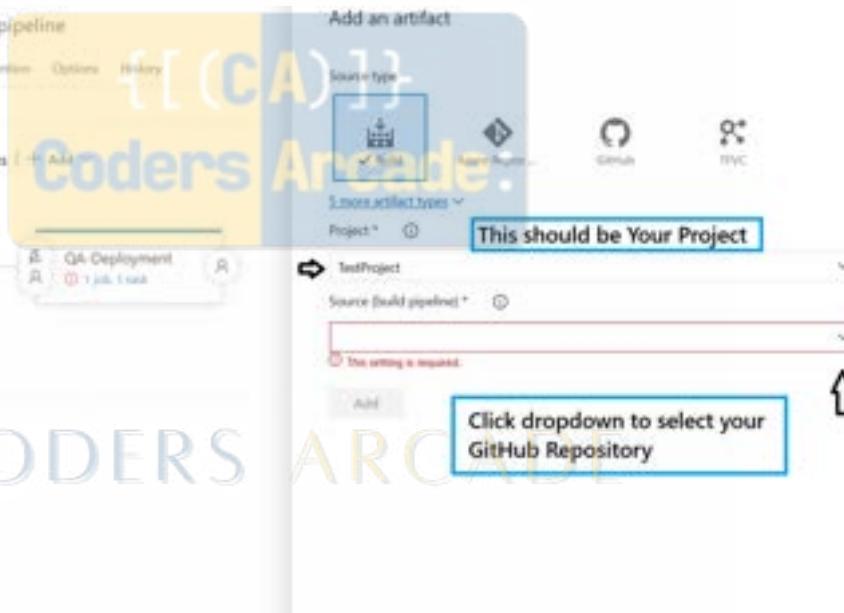
Stage QA-Deployment
Properties Name and owners of the stage
Stage name QA-Deployment
Stage owner Nandini Sarkar

3. ↗

6. You will need to add the Artifact (the war file) from Experiment 10 here in the release pipeline.

All pipelines > **New release pipeline**

Pipeline Tasks Variables Retention Options History

All pipelines > **New release pipeline**

CODERS ARCADE

COMMIT TO ACHIEVE

Add an artifact

Source type



Build



Azure Repos ...



GitHub



TFS

5 more artifact types ▾

Project *

TestProject

Source (build pipeline) *

SauravSarkar-CodersArcade.MyWebApp-AzureDevOps

Add

Select the repository.

Coders Arcade:

Add an artifact

Source type



Build



Azure Repos ...



GitHub



TFS

5 more artifact types ▾

Project *

TestProject

Source (build pipeline) *

SauravSarkar-CodersArcade.MyWebApp-AzureDevOps

Default version *

Latest

Source alias *

SauravSarkar-CodersArcade.MyWebApp-AzureDevOps

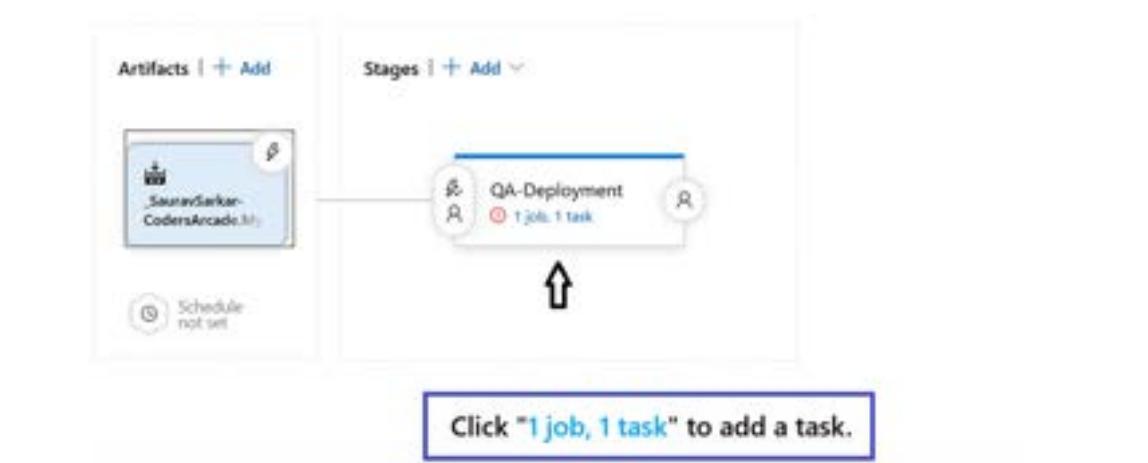
The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of SauravSarkar-CodersArcade.MyWebApp-AzureDevOps published the following artifacts: dosp.

Add



Click Add





Stage name: QA-Deployment

Parameters | [Unlink all](#)

Azure subscription: [Manage ID](#)

Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) Authorize | [O](#)

This field is linked to 1 setting in Deploy Azure App Service

App type: [Rb](#)

Web App on Windows

App service name: [Qb](#)

This setting is required.

Click to Authorize

**1. A pop-up window will show.
2. Enter correct Microsoft Credentials
3. Validate the authorization.**

Stage name: QA-Deployment

Parameters | [Unlink all](#)

Azure subscription: [Manage ID](#)

Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8)

Scoped to subscription 'Azure subscription 1'

App type: [Rb](#)

Web App on Windows

App service name: [Qb](#)

This setting is required.

Click Here

**1. Select the Operating System from the dropdown.
2. We created the two Web Apps on Linux Machines.
3. So, we will select "Web App on Linux".**

Stage name: QA-Deployment

Parameters | [Unlink all](#)

Azure subscription: [Qb](#) | [Manage](#)

Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) [v](#) [o](#)

(Optional) Scope to subscription: Azure subscription 1

App type: [Qb](#)

[Web App on Windows](#)  **Select this option**

[Web App on Linux](#)

[Web App for Containers \(Linux\)](#)

[Function App on Windows](#)

[Function App on Linux](#)

[Function App for Containers \(Linux\)](#)

[API App](#)

[Mobile App](#)

[[(CA)]]

Coders Arcade:

Stage name: QA-Deployment

Parameters | [Unlink all](#)

Azure subscription: [Qb](#) | [Manage](#)

Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) [v](#) [o](#)

(Optional) Scope to subscription: Azure subscription 1

App type: [Qb](#)

[Web App on Linux](#)  

This field is linked to 1 setting in 'Deploy Azure App Service'

App service name: [Qb](#) 

(Optional) This setting is required.

Startup command: [Qb](#) 

1. Select the "App service name"
2. For QA -> "demoCA" in my case. Choose yours.

Stage name: QA-Deployment

Parameters | Unlink all

Azure subscription: Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) | Manage

App type: Web App on Linux

App service name: DemoCA (This one.)

QA-Deployment

Parameters | Unlink all

Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) | Manage

App type: Web App on Linux

App service name: DemoCA (Selected)

Startup command: COMMIT TO ACHIEVE

QA-Deployment

Run on agent: Run on agent

Second Step

Deploy Azure App Service

Agent job ⓘ

No changes here

Display name *

Run on agent

Agent selection ^

Agent pool ⓘ | Pool information | Manage ⓘ

Hosted Windows 2019 with VS2019

Demands ⓘ

Name Condition Value

+ Add

Execution plan ^

Parallelism ⓘ

None Multi-configuration Multi-agent

Timeout *

Execution plan ^

Parallelism ⓘ

None Multi-configuration Multi-agent

Timeout *

Job cancel timeout *

Artifact download ^

SauravSarkar: CodersArcade.MyWebApp- Latest Selected all artifacts

Additional options ^

Allow scripts to access the OAuth token ⓘ

Run this job ⓘ

Only when all previous jobs have succeeded

QA-Deployment
Deployment process

Run on agent:
 Run on agent

Deploy Azure App Service ← **Third step** →

Azure App Service deploy

[B] Task version: 4.*

View XML Remove

Display name: Deploy Azure App Service

Connection-type: Azure Resource Manager

Azure subscription: Azure subscription 1 (5c3d4002-810b-4f3b-8e17-2d8be9773ac8)

(Copy to subscription) Your subscription?

App Service type: Web App on Linux

App Service name: DemoCA ← **Verify the name**

(Deploy to Slot or App Service Environment)

Package or folder: \${System.DefaultWorkingDirectory}/**.zip ← **Change**

Runtime Stack:

Startup command:

Post Deployment Action

Application and Configuration Settings

Control Options

Output Variables

COMMIT TO ACHIEVE
We are deploying an "war" file

Web App on Linux

App Service name *

Deploy to Slot or App Service Environment (1)

Package or folder * ← Change to "war" (1)

Runtime Stack (1)

Startup command (1)

Post Deployment Action (1)

Application and Configuration Settings (1)

Control Options (1)

Output Variables (1)

{[(CA)]}
Coders Arcade:
Click Save → Save Create Release View Release (1)

Azure App Service deploy (1)

[B] Task version: 4*

Display name *

Connection type * (1)

Azure Resource Manager (1)

Azure subscription * (1) Manage (1)

Azure subscription 1 (fc3de602-810b-4838-8ac7-2dfbe9713ad8)

Scoped to subscription: Azure subscription 1

App Service type * (1)

Web App on Linux

App Service name * (1)

Deploy to Slot or App Service Environment (1)

The screenshot shows the Azure DevOps interface for creating a new release pipeline. At the top, there's a 'Save' dialog box with fields for 'Folder' and 'Comment'. Below it, the main interface shows the creation of a 'New release pipeline' named 'Release-1'. The pipeline configuration includes a 'Run on agent' task and a 'Deploy Azure App Service' task. A watermark for 'CODERS ARCADE' is visible across the interface.

Save

Folder *

Comment

Click ok ➡ OK Cancel

All pipelines > **New release pipeline** Edit Name

Pipeline Tasks Variables New release pipeline Options History

QA-Deployment Deployment process

Run on agent Run on agent

Deploy Azure App Service (Azure App Service deploy)

All pipelines > **Release-1** Make it "Release 1"

Pipeline Tasks Variables Retention Options History

QA-Deployment Deployment process

Run on agent Run on agent

Deploy Azure App Service (Azure App Service deploy)

All pipelines > **Release-1**

Pipeline Tasks Variables Retention Options History

QA-Deployment Deployment process

Run on agent Run on agent

Deploy Azure App Service (Azure App Service deploy)

Save it ➡ Save View history View releases

Azure App Service deploy Task version: 4.0

Display name: Deploy Azure App Service

Save

Comment

Click ok ➡ OK Cancel

All pipelines > **Release-1** Click "All Pipelines"

Pipeline Tasks Variables Retention Options History

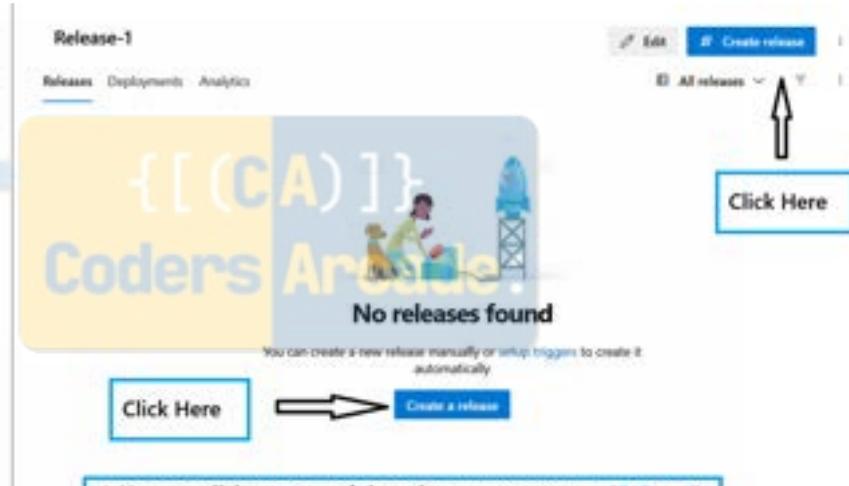
Q: Search all pipelines

+ New ↻

Name

✓ All pipelines

Release-1 ← **Check this**



Release-1

Releases Deployments Analytics

No releases found

You can create a new release manually or setup triggers to create it automatically.

Click Here → Create a release

1. You can click any one of these buttons to create a "Release".
2. Click to start the "Release Pipeline".

Create a new release

Release-1

Pipeline ^

Click on a stage to change its trigger from automated to manual.

QA-Deploy



QA Stage displayed here.

Stages for a trigger change from automated to manual. ⓘ

Artifacts ^

Select the version for the artifact sources for this release

Source alias:

Version:

_SauravSarkar-CodersArcade.My... 20250520.1

Release description

[[(CA)]]
Coders Arcade:

Create**Cancel**

Create a new release

Release-1

CODERS ARCADE

Pipeline ^

Click on a stage to change its trigger from automated to manual.

QA-Deploy

Stages for a trigger change from automated to manual. ⓘ

COMMIT TO ACHIEVE

Artifacts ^

Select the version for the artifact sources for this release

Source alias:

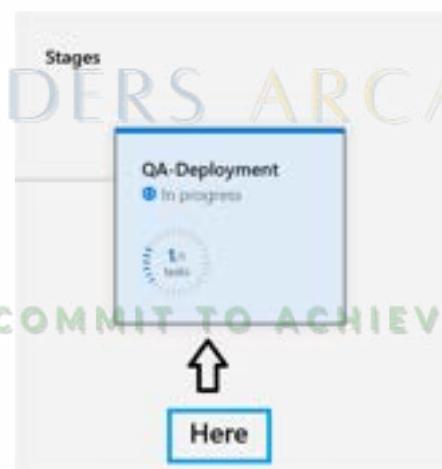
Version:

_SauravSarkar-CodersArcade.My... 20250520.1

Release description

Click to create the release.**Create****Cancel**

The screenshot shows a DevOps pipeline interface. At the top, a green banner displays the message: "Release-1 has been queued" with a left arrow icon, and "You should be able to see a new release named "Release-1" queued." Below this, the main interface shows a "Release-1" card with a "Click the "Release-1" Link" button. The interface includes tabs for "Releases", "Deployments", and "Analytics". A "Create release" button is also visible. The "Stages" section shows a "QA-Deployment" stage with a status of "Queued". A callout box points to this stage with the text: "You can see that your deployment has been queued in the "QA-Deployment" Stage." A large watermark for "CODERS ARCADE" and "COMMIT TO ACHIEVE" is overlaid on the bottom half of the screen.



The screenshot shows the Azure DevOps interface. At the top, there's a header with the text "Stages". Below it, a stage named "QA-Deployment" is shown as "In progress" with a progress bar at 1/1. Below the stage, there are three buttons: "Cancel", "Logs", and "Logs" (highlighted with a blue border). A large button labeled "Click to view logs" is also present. An upward arrow icon is positioned next to the logs button.

Run on agent

Pool: Hosted Windows 2019 with ... · Agent: Hosted Agent

Started: 20/5/2025, 11:46:51 am

7s

Initialize job - succeeded

Download artifact - _SauravSarkar-CodersArcade/MyWebApp-AzureDevOps - drop

5s

1s

Starting: Download artifact - _SauravSarkar-CodersArcade/MyWebApp-AzureDevOps - drop

Task : Download build artifacts
Description : Download files that were saved as artifacts of a completed build
Version : 0.387.1
Author : Microsoft Corporation
Help : <https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/download-build-artifacts>

1. These are the logs of the tasks that are running.
2. Everything green means the tasks are running without any errors.
3. Ignore the orange or yellow logs because they are just warnings.

COMMIT TO ACHIEVE

Run on agent:
Pool: Hosted Windows 2019 with ... · Agent: Hosted Agent

Started: 20/5/2025, 11:46:51 am
35s

- Initialize job + succeeded
- Download artifact - _SauravSarkar-CodersArcade/MyWebApp-AzureDevOps - drop - succeeded
- Deploy Azure App Service

Get service connection details for Azure App Service: "DemoCA" ← Our Web App Name

Deploy Azure App Service

```

[1] 2025-05-20T06:17:01.004934Z ##[section]Starting: Deploy Azure App Service
[2] 2025-05-20T06:17:01.013151Z Task : Azure App Service Deploy
[3] 2025-05-20T06:17:01.015189Z Description : Deploy to Azure App Service a web, mobile, or API app using Docker, Java, .NET, .NET Core, Node.js, PHP, Python, or IIS
[4] 2025-05-20T06:17:01.015190Z Version : 4.254.0
[5] 2025-05-20T06:17:01.015190Z Author : Microsoft Corporation
[6] 2025-05-20T06:17:01.015190Z Help : https://aka.ms/appservice/troubleshooting
[7] 2025-05-20T06:17:01.015190Z
[8] 2025-05-20T06:17:01.015190Z Get service connection details for Azure App Service: "DemoCA"
[9] 2025-05-20T06:17:01.015190Z Package deployment using ZIP Deploy initiated.
[10] 2025-05-20T06:17:01.015190Z Deployment logs can be viewed at https://democa-f2fpgbgxcpgnhzed.scm.canadacentral-01.azurewebsites.net/api/deployments/2129475-9881-4
[11] 2025-05-20T06:17:01.015190Z Successfully deployed and package to App Service.
[12] 2025-05-20T06:18:09.568000Z Successfully updated App Service configuration details
[13] 2025-05-20T06:18:09.568000Z Successfully added release annotation to the Application Insight : DemoCA
[14] 2025-05-20T06:18:13.008000Z Successfully updated deployment history at https://democa-f2fpgbgxcpgnhzed.scm.canadacentral-01.azurewebsites.net/api/deployments/1
[15] 2025-05-20T06:18:13.008000Z App Service Application URL: https://democa-f2fpgbgxcpgnhzed.canadacentral-01.azurewebsites.net
[16] 2025-05-20T06:18:13.7151790Z ##[section]Finishing: Deploy Azure App Service

```

This is what we need. Our Application URL

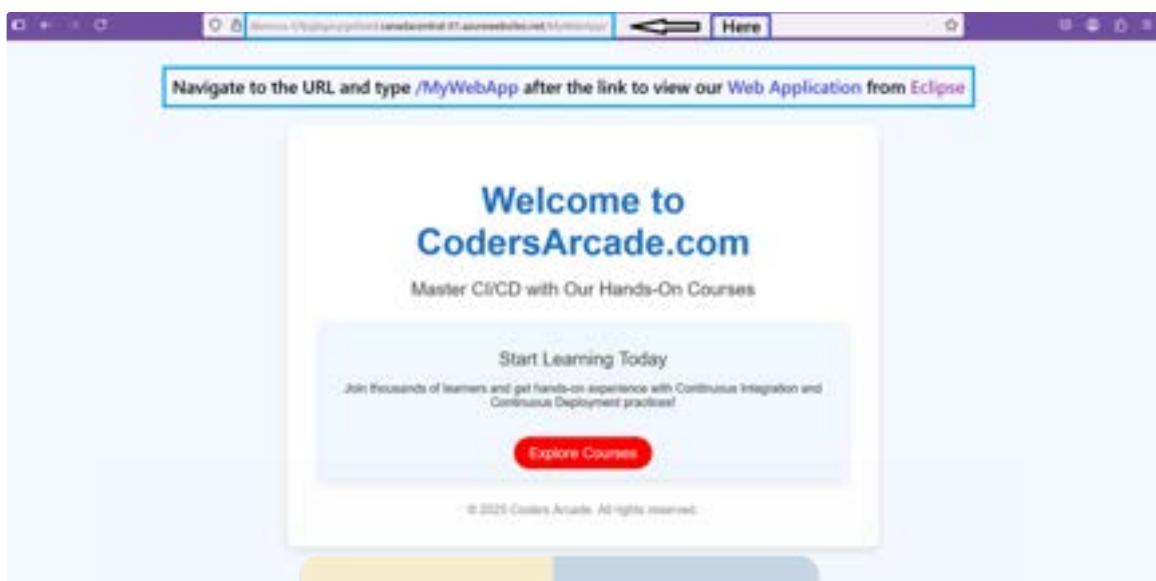
You need to copy this URL

COMMIT TO ACHIEVE

2025-05-20T06:18:04.011171Z Successfully updated app service configuration details.
 2025-05-20T06:18:09.568000Z successfully added release annotation to the Application Insight : DemoCA
 2025-05-20T06:18:13.008000Z Successfully updated deployment history at https://democa-f2fpgbgxcpgnhzed.scm.canadacentral-01.azurewebsites.net
 2025-05-20T06:18:14.0087790Z App Service Application URL: https://democa-f2fpgbgxcpgnhzed.canadacentral-01.azurewebsites.net ← This one

ment History at https://democa-f2fpgbgxcpgnhzed.scm.canadacentral-01.azurewebsites.net : https://democa-f2fpgbgxcpgnhzed.canadacentral-01.azurewebsites.net

Azure App Service



7. Copy the Application URL link & navigate to Your IntelliJ Idea IDE where you have written a simple Test Case in Maven using JUnit Framework to Test & Validate the title of the same URL.

8. Here is the GitHub Link for the Maven Project for the Simple Automation Test. [GitHub - SauravSarkar-CodersArcade/DevOps-Automation-Test: DevOps-Automation-Test](#)

9. Now after writing the test, follow the steps as given below:

```

File Edit View Navigate Code Refactor Build Run Tools Git Window Help Automation-Azure-DevOps - BrowserTest.java
Automation-Azure-DevOps src test java BrowserTest
Project Automation-Azure-DevOps D:\Idea Projects\ Automation-Azure-DevOps pom.xml (Automation-Azure-DevOps) BrowserTest.java
src idea src main test java BrowserTest target pom.xml External Libraries Scratchers and Consoles
import org.junit.Assert;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
// SauravSarkar-CodersArcade
public class BrowserTest {
    @SauravSarkar CoderArcade
    @Test
    public void getData() {
        System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("https://democse-17.firebaseioapp.com/.netcentral-31.azurewebsites.net/index.html");
        String actualTitle = driver.getTitle();
        System.out.println(actualTitle);
        String expectedTitle = "Coders Arcade - CI/CD Learning";
        Assert.assertEquals(actualTitle, expectedTitle);
        driver.close();
    }
}

```

Paste the URL Here

↓

Write a Java - Selenium test case to validate the title of the App URL Web Page.
Will provide the GitHub Link.

```

Automation-Azure-DevOps - src/test/java/BrowserTest.java
Automation-Azure-DevOps - pom.xml (Automation-Azure-DevOps) - BrowserTest.java
Project - src - Automation-Azure-DevOps - src - test - java - BrowserTest
src - target
pom.xml
External Libraries
Scratches and Cossides

import org.junit.Assert;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BrowserTest {
    @Test
    public void getData() {
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.google.com");
        String actualTitle = driver.getTitle();
        String expectedTitle = "Google Search - Edge Learning";
        Assert.assertEquals(actualTitle, expectedTitle);
        driver.quit();
    }
}

Run: BrowserTest.getData - Test Successful
Tests passed: 1 | Failed: 0 | Skipped: 0
BrowserTest - 2 sec 700 ms
"C:\Program Files\Java\jdk-17.0.4\bin\java.exe" ...
Hello Guys
May 28, 2025 11:52:07 AM org.openqa.selenium.devtools.CdpVersionFinder findN
WARNING: Unable to find an exact match for CDP version 136, returning the c
Coders Arcade - CI/CD Learning - Title Validated

```

Your release pipeline "Release-1" has succeeded.

Deployment process: Run on agent - succeeded

Run on agent	Started	Duration
Pool: Hosted Windows 2019 with ... - Agent: Hosted Agent	2025-05-28T11:48:51Z	~ 1m 27s
Initialize job		5s
Download artifact - SauravSarker-CodersArcade.MyWebApp-AzureDevOps - drop - succeeded		4s
Display Azure App Service - succeeded		~ 1m 16s
Finalize Job - succeeded		~ 1s

The screenshot shows the Azure DevOps interface for a Release Pipeline named 'Release-1'. Under the 'Deployment process' section, there is a 'Run on agent' step with the status 'succeeded'. The pipeline details show a 'Hosted Windows 2019 with... - Agent: Hosted Agent' and a start time of '2025-05-20T11:46:51'. The deployment log lists four steps: 'Initialize job - succeeded', 'Download artifact - SauravSarkar-CodersArcade/MyWebApp-AzureDevOps - drop - succeeded', 'Deploy Azure App Service - succeeded', and 'Finalize job - succeeded'. A callout box highlights two notes: '1. This completes your DevOps VTU Lab Experiment Number 11' and '2. If you want to add Azure Key Vault & Manage Secrets in the pipeline, you can follow it from the steps in the beginning. But not recommended for simple demo.'

Final Summary – Experiment 11: Creating a Release Pipeline and Automating Tests using Azure DevOps

- Set up a **Release Pipeline** in Azure DevOps to deploy the `.war` file (generated from the Build Pipeline) into the **QA Server (demoCA)**.
- Connected the **Build Pipeline artifacts** with the Release Pipeline to automate deployment.
- Configured a deployment stage that targets an **Azure App Service** acting as the **QA environment**.
- Performed **automation testing** using Java + Selenium WebDriver after successful deployment.
- Updated the test script to include your **App Service URL** in `driver.get("your_app_service_url");`.
- Verified the app's deployment by checking the **webpage title** and basic UI elements.
- Understood how **QA environments** are used to validate changes before promoting to production.
- Explored the **optional** use of **Azure Key Vault** to securely manage secrets like passwords or tokens.
- **Note:** For simple demo apps in lab experiments, using Key Vault is **not mandatory** but is a **best practice** in real-world scenarios.

COMMIT TO ACHIEVE

CA - Experiment 12 - Final Practical Exercise and Wrap-Up

Build and Deploy a Complete DevOps Pipeline, Discussion on Best Practices and Q&A

Objectives of Experiment 12

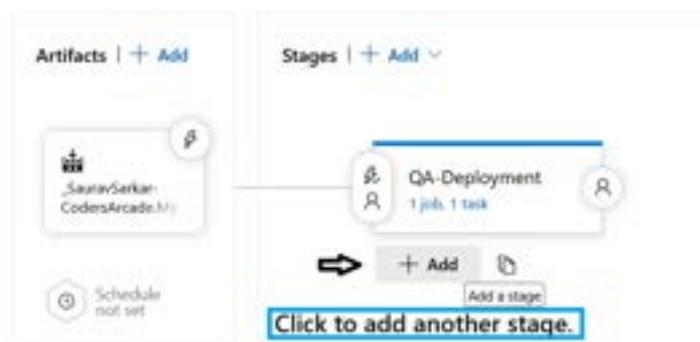
1. Complete the **final stage of the CI/CD pipeline** by deploying the `.war` file to the **Production Server (prodDemoCA)** using Azure DevOps Release Pipeline
2. Set up and configure a **production-ready release pipeline** connected to the build artifacts
3. Deploy the web application to the **Azure App Service (prodDemoCA)** as the production environment
4. Perform **post-deployment validation** using Java-Selenium automation with **JUnit test cases**
5. Update the automation script with your **production app URL** for live testing
6. Enable the **Continuous Deployment Trigger (CD trigger)** to automate deployment on every GitHub push
7. Observe automatic **Release creation** (e.g., **Release-4**) right after build completion and artifact publishing
8. Learn how to handle **multi-stage pipelines** for QA and Production environments effectively
9. Ensure App Services (`demoCA` and `prodDemoCA`) are running during release execution
10. Experience the **full DevOps workflow**: Code → Build → Release → Deploy → Test, all done automatically

Here are the steps to follow for the Final Practical Exercise in DevOps VTU Experiment 12

1. You need to navigate to your Project & the Release Pipeline that you created in Experiment 11 as shown below:
2. Now you can follow these steps for Experiment 12 as shown below:

The screenshot shows the Azure DevOps interface with the following details:

- Left Sidebar:** Shows the project structure with "TestProject" selected. Other items include "Overview", "Builds", "Repos", "Pipelines", "Environments", "Releases", "Library", "Task groups", "Deployment groups", "Test Plans", and "Artifacts".
- Top Navigation:** Shows the URL "Azure DevOps /mainlinekafe/10413", the project name "TestProject", and tabs for "Pipelines" and "Releases".
- Main Area:**
 - Pipeline View:** Shows a pipeline with a "GitHub Deployment" step.
 - Release Pipeline:** Shows a "Release-1" pipeline with a single "GitHub Deployment" step. A callout box says: "Navigate to the same Release Pipeline that was successfully executed in the Experiment 11."
 - Environment View:** Shows an environment named "prodDemoCA" with a status of "Running". A callout box says: "Note: The same Web Apps (`demoCA` & `ProdDemoCA`) (in my case), choose yours accordingly, should be started and in running state."



Stage
Prod-Deployment

Properties ↗
Name and owners of the stage

Stage name: Prod-Deployment Rename the stage

Stage owner: Nandini Sarkar

1. The same steps will be repeated but this time, the Web App will be "prodDemoCA" for the "Prod Deployment".
2. This is what happens in the industry.
3. First QA-Deployment-> Quality Assurance
4. Then Prod-Deployment-> Production Release

Coders Arcade:

Prod-Deployment
① 1 job, 1 task

View stage tasks

Add the tasks

Prod-Deployment ← Step 1 → **COMMIT TO ACHIEVE**

Run on agent: Use this agent

Deploy Azure App Service: Some settings need attention

Stage name: Prod-Deployment

Parameters:

Azure subscription:

This setting is required. Select & Authorize ↑

App type:

Web App on Windows ← Change to Linux ↑

App service name:

This setting is required.

Stage name: Prod-Deployment

Parameters | [Unlink all](#)

Azure subscription: Azure subscription 1 (fc30e602-810b-4838-8ac7-2dfbe9713ad8) | [Manage](#)

App type: Web App on Linux

App service name: DemoCA, ProdDemoCA, MvIT, QA-Environment, DevOps-RNSIT

Select the other Web App

"ProdDemoCA" in my case.
Choose yours from the dropdown.

Run on agent: Step 2

Deploy Azure App Service (Azure App Service deploy)

Should look exactly like this screen.

COMMIT TO ACHIEVE

Agent job ①

No changes here

Display name *

Run on agent

Agent selection ^

Agent pool ② | [Pool Information](#) | [Manage](#) ③

Hosted Windows 2019 with VS2019 ④ ⑤

Demands ⑥

Name	Condition	Value
+ Add		

Execution plan ^

Parallelism ⑦

None Multi-configuration Multi-agent

Timeout ⑧

All pipelines > ⑨ Release-1

Pipeline Tasks - Variables Retention Options History

Prod-Deployment Deployment process

Run on agent Run on agent ⑩

⑪ Deploy Azure App Service ⑫ Step 3 ⑬

⑭ COMMIT TO ACHIEVE

Azure App Service deploy ⑮

Task version: 4.1 ⑯

Display name *

Display Azure App Service ⑰

Connection type ⑱

Azure Resource Manager ⑲

Azure subscription ⑳ ⑳ Manage ⑳

Azure subscription 1 (b33a602-810b-4818-8ac7-2d85e971a0ff) ⑳

Selected for subscription: [View subscription](#) ⑳

App Service type ⑳

Web App on Linux ⑳ ⑳ Verify ⑳

App Service name ⑳ ⑳ Verify ⑳

Deploy to Slot or App Service Environment ⑳

Package or folder ⑳

Web App on Linux

App Service name * ProdDemoCA

Deploy to Slot or App Service Environment (?)

Package or folder * \$(System.DefaultWorkingDirectory)/**/*.war Change ...

Runtime Stack (?) ... O

Startup command (?) ...

Post Deployment Action (?) ...

Application and Configuration Settings (?) ...

Control Options (?) ...

Output Variables (?) ...

Save → Save ... View releases ...

Because we are deploying "war" file.

Web App on Linux

App Service name * ProdDemoCA

Deploy to Slot or App Service Environment (?)

Package or folder * \$(System.DefaultWorkingDirectory)/**/*.war ...

All pipelines > Release-1 Save Create release View releases ...

Pipeline Today Variations Retention Options History

COMMIT TO ACHIEVE

Artifacts | + Add Stages | + Add =

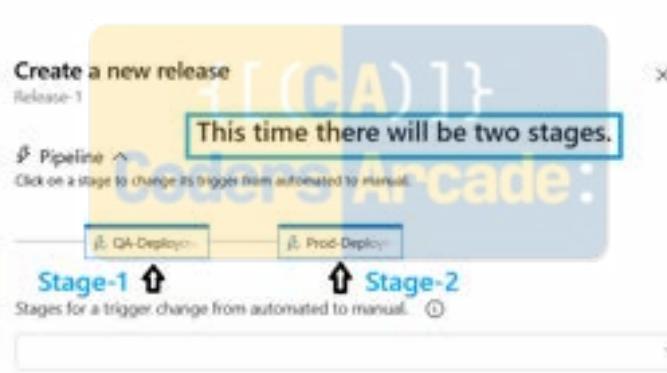
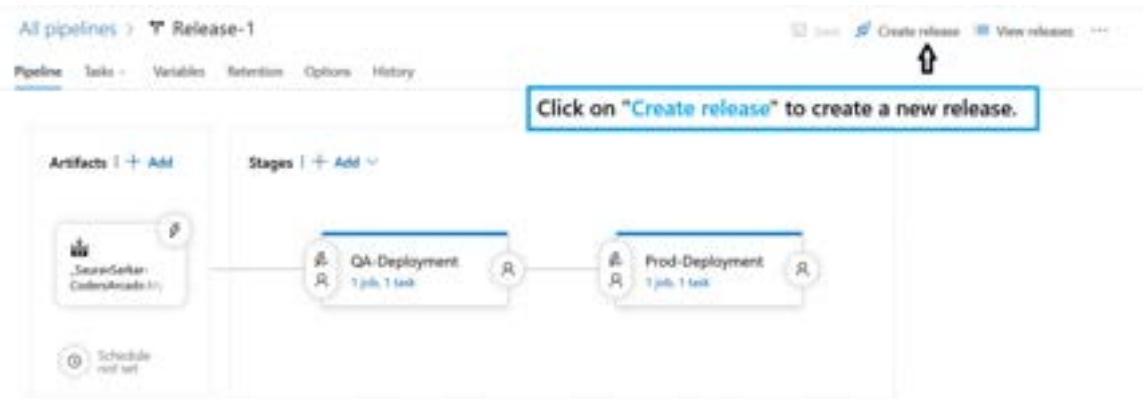
```

graph LR
    A["Source: SauravSarkar - CodersArcade.sln"] --> B["QA-Deployment  
1 job, 1 task"]
    B --> C["Prod-Deployment  
1 job, 1 task"]

```

S SauravSarkar Schedule not set

1. This is how your "Release Pipeline" should look like after adding the tasks.
 2. The initial task from Experiment 11 -> QA-Deployment.
 3. The latest task from Experiment 12 -> Prod-Deployment.



CODERS ARCADE

Artifacts ^

Select the version for the artifact sources for this release

Source alias	Version
SauravSarkar-CodersArcade.MY...	20250520.1

Release description:

COMMIT TO ACHIEVE

Click here to create and start the Release.

Create **Cancel**

All pipelines > **Release-1**

Here Release: Release-2 has been created

You should be able to view this new release "Release-2" created.

Click "Release-2" to View Logs

Artifacts | + Add

Schedule not set

Stages | + Add

QA-Deployment 1 job, 1 task

Prod-Deployment 1 job, 1 task

Release-1 > Release-2

Pipeline Variables History Deploy Refresh Edit ...

Release

Manually triggered by Nandini Sarkar 20/5/2025, 12:31 pm

Actions

QA-Deployment 0 Queued Waiting in Hosted Windows

Prod-Deployment 0 Not deployed

The two stages will start in this new release "Release-2".

COMMIT TO ACHIEVE

Release-1 > Release-2 > QA-Deployment | Succeeded

Pipeline Tasks Variables Logs Tests Deploy Cancel Refresh Download all logs Edit ...

Deployment process Succeeded

Run on agent

Pool: Hosted Windows 2019 with ... - Agent: Hosted Agent Started 20/5/2025, 12:31:13 pm — 53s

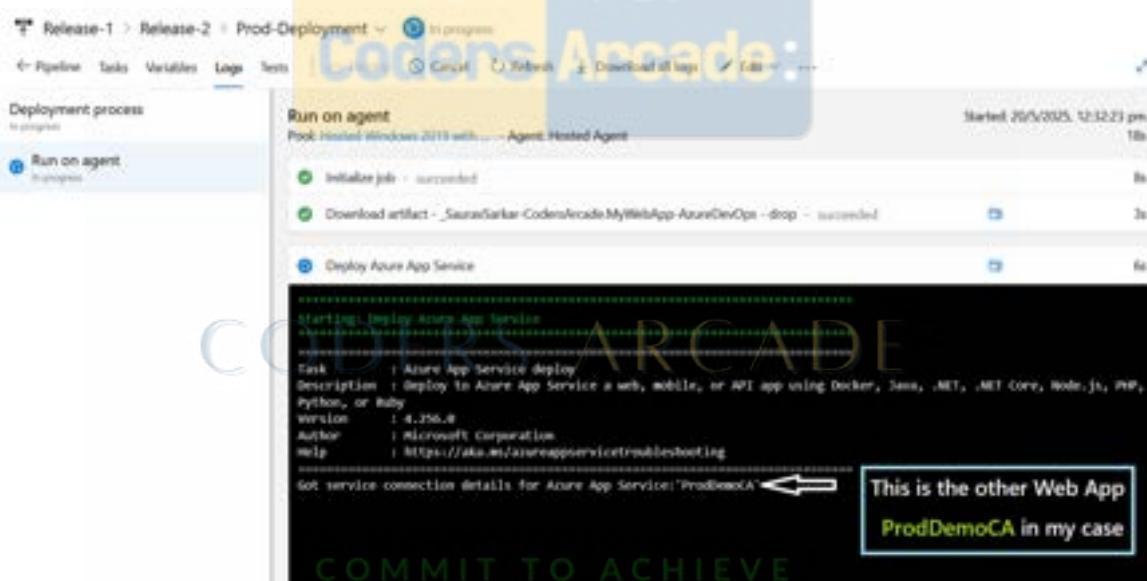
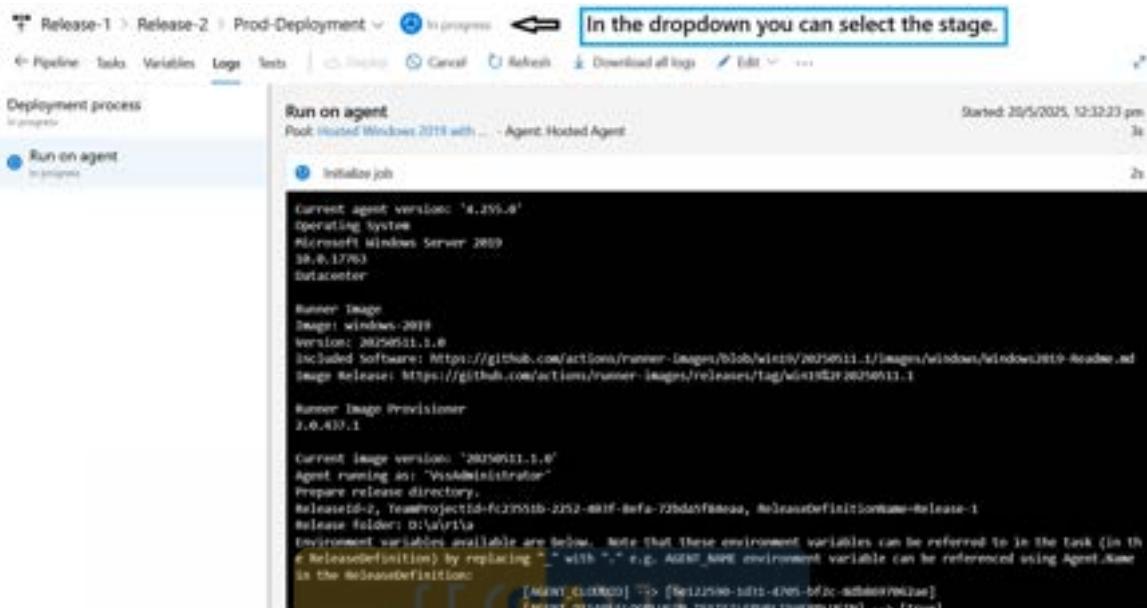
Initialize job - succeeded 8s

Download artifact - _SauravSarkar-CodersArcade/MyWebApp-AzureDevOps - drop - succeeded 2s

Deploy Azure App Service - succeeded 41s

Finalize Job - succeeded <1s

The two stages "QA-Deployment" and "Prod-Deployment" will complete one by one. Click to view the logs. All green means the stages have completed successfully.



Run on agent

Pool: Hosted Windows 2019 with... · Agent: Hosted Agent

Started: 20/5/2025, 12:32:23 pm
1m 25s

- Initialize job - succeeded 8s
- Download artifact - _SauravSarkar-CodersArcade.MyWebApp-AzureDevOps - drop - succeeded 3s
- Deploy Azure App Service 1m 13s

```
Starting: Deploy Azure App Service

Task      : Azure App Service deploy
Description : Deploy to Azure App Service a web, mobile, or API app using Docker, Java, .NET, .NET Core, Node.js, PHP, Python, or Ruby
Version   : 4.256.0
Author    : Microsoft Corporation
Help      : https://aka.ms/azureappservice/troubleshooting

Got service connection details for Azure App Service: 'ProdDemoCA'
Package deployment using WAR Deploy initiated.
Deploy logs can be viewed at https://produdemoca-dhepfleif3fthz9.scm.canadacentral-01.azurewebsites.net/api/deploymentLogs/0ba597dd-e881-4800-af46-25bf23902036/log
Successfully deployed web package to App Service.
```

Wait for the App Service URL

Deploy Azure App Service

Previous task | Next task | X

```
2025-05-28T07:04:19+2937564Z :section|starting: Deploy Azure App Service
2025-05-28T07:04:19+2937564Z Task      : Azure App Service deploy
2025-05-28T07:04:19+2937564Z Description : Deploy to Azure App Service a web, mobile, or API app using Docker, Java, .NET, .NET Core
2025-05-28T07:04:19+2937564Z Version   : 4.256.0
2025-05-28T07:04:19+2937564Z Author    : Microsoft Corporation
2025-05-28T07:04:19+2937564Z Help      : https://aka.ms/azureappservice/troubleshooting
2025-05-28T07:04:19+2937564Z
2025-05-28T07:04:19+2937564Z :section|get service connection details for Azure App Service: 'ProdDemoCA'
2025-05-28T07:04:19+3139814Z Package deployment using WAR Deploy initiated.
2025-05-28T07:04:19+3139814Z Deploy logs can be viewed at https://produdemoca-dhepfleif3fthz9.scm.canadacentral-01.azurewebsites.net/
2025-05-28T07:04:19+3139814Z Successfully deployed web package to App Service.
2025-05-28T07:04:19+3139814Z Successfully updated app service configuration details
2025-05-28T07:04:19+3139814Z Successfully added release annotation to the Application Insight : ProdDemoCA
2025-05-28T07:04:19+3139814Z Successfully updated deployment history at https://produdemoca-dhepfleif3fthz9.scm.canadacentral-01.azurewebsites.net
2025-05-28T07:04:19+3139814Z App service Application URL: https://produdemoca-dhepfleif3fthz9.canadacentral-01.azurewebsites.net
2025-05-28T07:04:19+3139814Z :section|finishing: Deploy Azure App Service
```

This is our App Service Application URL
You can see this time it's **produdemoca**
Copy this URL -> In Any Browser

```
2025-05-28T07:04:27+6875235Z Successfully added release annotation to the Application Insight : ProdDemoCA
2025-05-28T07:04:31+4098946Z Successfully updated deployment history at https://produdemoca-dhepfleif3fthz9.scm.canadacentral-01.azurewebsites.net
2025-05-28T07:04:32+4609541Z App Service Application URL: https://produdemoca-dhepfleif3fthz9.canadacentral-01.azurewebsites.net
2025-05-28T07:04:33+8408248Z :section|finishing: Deploy Azure App Service
```



The App Service Application URL should display the Web Application "index.jsp" from the Eclipse Project

Welcome to CodersArcade.com
Master CI/CD with Our Hands-On Courses
Start Learning Today
Join Thousands of learners and get hands-on experience with Continuous Integration and Continuous Deployment practices!
Explore Courses

©2025 Coders Arcade. All rights reserved.

Release-1 > Release-2 > Prod-Deployment ✓ Succeeded ← Complete CI-CD Pipeline Has Succeeded

← Pipeline Tasks Variables Logs Tests → Deploy Artifacts Download all logs Edit ⋮

Deployment process succeeded

Run on agent

Pool: Hosted Windows 2019 with Agent: Hosted Agent

Started 20/5/2025, 12:32:23 pm — 2m 10s

- Initialize job - succeeded
- Download artifact - _SauravSarkar-CodersArcade.MyWebApp-AzureDevOps-drop - succeeded
- Deploy Azure App Service - succeeded
- Finalize job - succeeded

Now, every time we push our changes from Local to GitHub, this pipeline should be automatically triggered so that all changes are reflected immediately in the App Service Application URL to the public/clients/end-users.

For that we need to enable the "Continuous Deployment Trigger".

Release-1 > Release-2

Pipeline Variables History + Deploy ⋮ Cancel Refresh Edit ⋮ ⋮

This was manual

Release ↓

Manually triggered by Nandini Sarkar on 20/5/2025, 12:30 pm

Artifacts

_SauravSarkar-CodersA... 20250520.1 master

Stages

Edit release Edit pipeline

QA-Deployment ✓ Succeeded on 20/5/2025, 12:32 pm

Prod-Deployment ✓ Succeeded on 20/5/2025, 12:34 pm

COMMIT TO ACHIEVE

The screenshot shows the Azure DevOps 'All pipelines' view. On the left, there's a sidebar with a search bar and a tree view showing 'All pipelines' (selected) and 'Release-1'. The main area displays two releases: 'Release-2' (2025/05/20 1:37 master) and 'Release-1' (2025/05/20 1:37 master). Below the releases is a button labeled 'Go to your Release Pipeline'. The URL is https://dev.azure.com/SauravSarkar-CodersArcade/_apis/pipelines?view=grid&releaseId=1.

Release-1 > Release-2

Pipeline Variables History + Deploy Cancel Refresh Edit ...

Release Stages

Release-2

Manually triggered by Nandini Sarkar on 20/5/2025, 12:30 pm

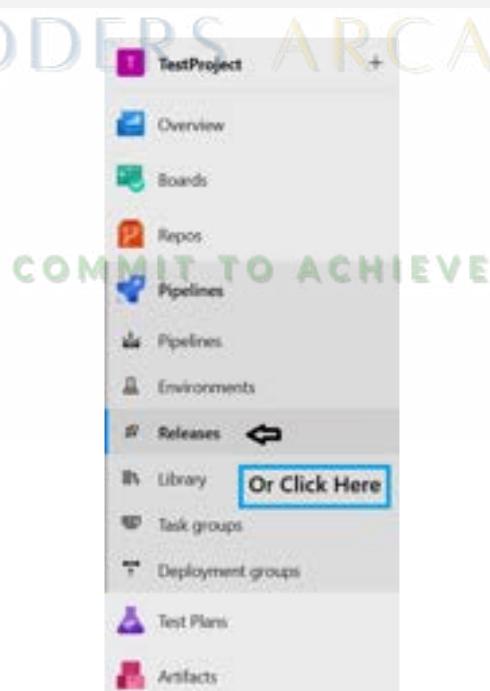
Artifacts SauravSarkar-CodersA... 20250520.1 master

Stages

QA-Deployment [Succeeded] on 20/5/2025, 12:32 pm

Prod-Deployment [Succeeded] on 20/5/2025, 12:34 pm

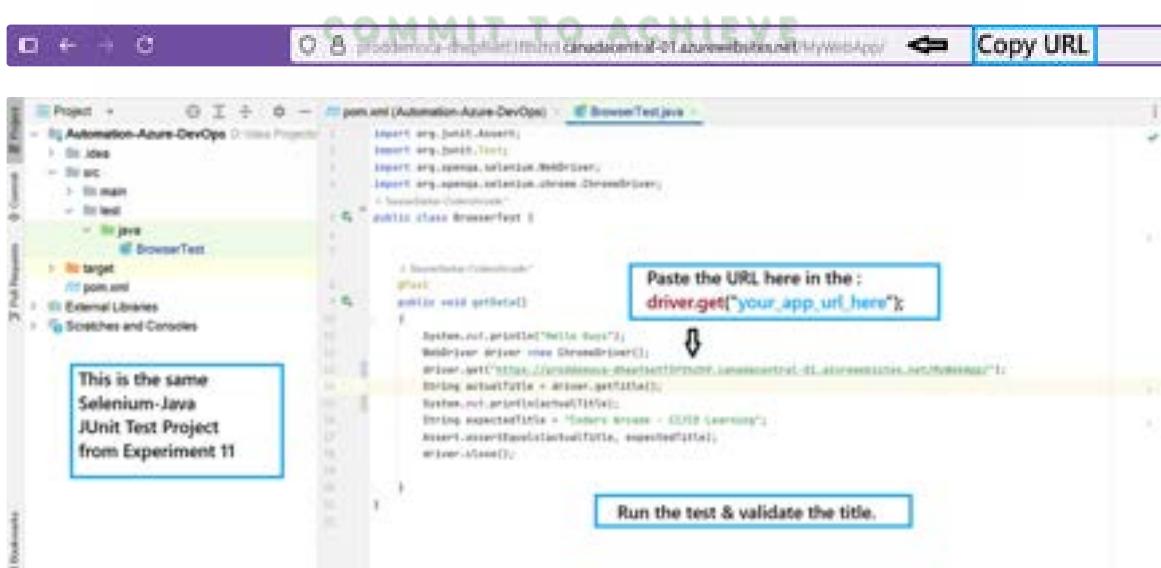
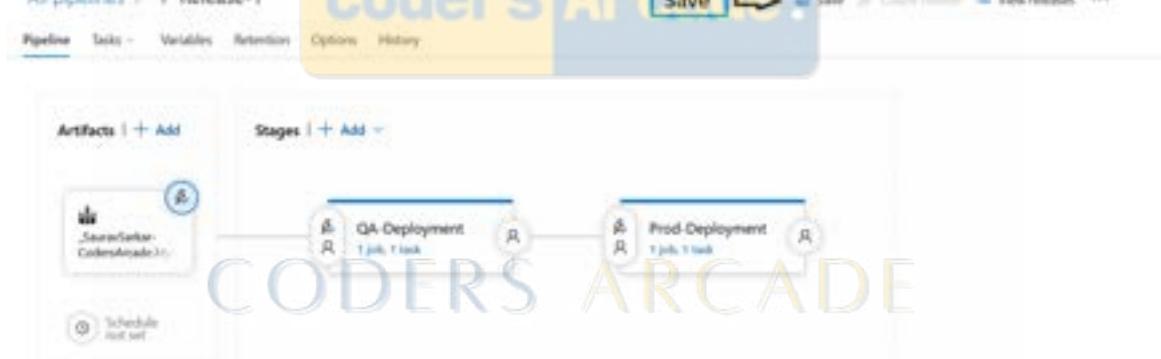
Click Edit Pipeline





This screenshot shows the "Continuous deployment trigger" configuration in the Azure DevOps Pipeline settings. It includes sections for "Artifacts" and "Stages". The "Stages" section shows a flow from "QA-Deployment" to "Prod-Deployment". A large yellow box contains the text "Continuous deployment trigger" and "Build: _SauravSarkar-CodersArcade:MyWebApp-AzureDevOps". Below this, there's a toggle switch labeled "Disabled" with a blue box around the "Enable This" text. A black arrow points from the "Disabled" switch to the "Enable This" text. Another blue box highlights the "Continuous deployment trigger" section. At the bottom, there's a "Pull request trigger" section with a "Disabled" toggle switch and a note about enabling it.

COMMIT TO ACHIEVE





```
<div class="container">
  <h1>Welcome to CodersArcade.com</h1>
  <h2>Master CI/CD with Our Hands-On Courses</h2>
  <div class="cta">
    <a href="https://www.youtube.com/c/codersarcade" class="button">Start Learning Today!!!!</a> ← Change something
    <p>Join thousands of learners and get hands-on experience with
      <a href="https://www.youtube.com/c/codersarcade" class="button">Start Learning Today!!!!!!</a> ← Like this
    </p>
    <footer>
      <p>© 2025 Coders Arcade. All rights reserved.</p>
    </footer>
  </div>
</div>
```

```
Saurav@Saurav MINION4 /d/Eclipse/eclipse-workspace/MyWebApp
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   target/m2e-wtp/web/resources/META-INF/maven/com.codersarcade/MyWebApp/pom.properties

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   target/m2e-wtp/web/resources/META-INF/maven/com.codersarcade/MyWebApp/pom.properties
    modified:   target/m2e-wtp/web/resources/META-INF/maven/com.codersarcade/MyWebApp/pom.xml

Saurav@Saurav MINION4 /d/Eclipse/eclipse-workspace/MyWebApp (master) ← See the modifications. Add them
$ COMMIT TO ACHIEVE
```

```
Saurav@Saurav MINION4 /d/Eclipse/eclipse-workspace/MyWebApp
$ git add .
Warning: in the working copy of 'src/main/webapp/index.jsp', LF will be replaced by CRLF the next time Git touches it
Saurav@Saurav MINION4 /d/Eclipse/eclipse-workspace/MyWebApp (master)
$ git commit -m "Latest Commit- Changed the index.jsp file for validation in deployment- Continuous Trigger"
[master c043bc2] Latest Commit- Changed the index.jsp file for validation in deployment- Continuous Trigger ← Observe the commit message
 2 files changed, 2 insertions(+), 2 deletions(-)

Saurav@Saurav MINION4 /d/Eclipse/eclipse-workspace/MyWebApp (master)
$ git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 16 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (14/14), 1.00 KiB | 1.00 MiB/s, done.
Total 14 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
```

Commit your changes & push to GitHub

Pipelines

Recent: All | Run

Back in your Build Pipeline you will see a Job queued.

New pipeline | Filter pipelines

Recently run pipelines

Pipeline Last run

SauravSarkar-CodersArcade.MyWebApp... #20250520.2 • Latest Commit- Changed the index.jsp file for validation in deployment- Continuous Trigger Just now Now

← SauravSarkar-CodersArcade.MyWebApp-AzureDevOps

Edit | Run pipeline | ↻

Runs | Branches | Analytics

Description Stage

#20250520.2 • Latest Commit- Changed the index.jsp file for validation in deployment- Continuous Trigger Just now This has started ↻

Click ↻ 2 Individual CI by 2 master 24m ago

#20250520.1 • Set up CI with Azure Pipelines Manually triggered to 2 master 1 hour ago ↻ 24 ago 21s

← #20250520.2 • Latest Commit- Changed the index.jsp file for validation in deployment- Continuous Trigger ↻

Cancel ↻

Summary | Code Coverage

Individual CI by SauravSarkar-CodersArcade ↻ View change

Repository and version SauravSarkar-CodersArcade/MyWebApp-AzureDevOps ↻

P master 1 c040bc20 ↻

Time started and elapsed Just now ↻

Related 0 work items ↻

Tests and coverage A Get started ↻

Jobs

Name

Job Click ↻ Queued ↻

← CODERS ARCADE ↻

+ Jobs in run #20250520.2

SauravSarkar-CodersArcade/MyWebApp-AzureDevOps

Job

Job COMMIT TO ACHIEVE ↻

Pool: Azure Pipelines ↻

Image: alpine:latest ↻

Queued: Just now [https://dev.azure.com/cod... ↻]

The agent request is not running because all potential agents are running other requests. Current position in queue: 3 ↻ See preparation parameters ↻

The Job with all tasks in build pipeline in Experiment 10 that we had created earlier to Copy war File & Publish Build Artifacts is under progress.

View the logs for errors.

Note: This is the Build Pipeline. This will complete first. Then the Release Pipeline will start automatically. This is because of the Continuous Deployment Trigger ↻

Job

Pipeline: Apache-Pipeline
Image: ubuntu-latest
Started: Today at 12:49 pm (server_localtime, 3m)
Agent: Hosted Agent
Started: Just now
Duration: 2m

The agent request is already running or has already completed.
Job preparation parameters:
1 Job(s) produced
Job 1: Sync Command Start: /detectdectionContainer
Sync Command End: /detectdectionContainer
Sync Command Start: /detectdectionContainer
Sync Command End: /detectdectionContainer
Finishing: Job

You can see that the Build Pipeline has Succeeded ✓

Now go to Releases

Release-1

Release-3 →
2025/05/20, 12:51:58 pm
QA-Deployment... Prod-Dep...

Release-2
2025/05/20, 12:30:59 pm
QA-Dep... Prod-Dep...

Release-1
2025/05/20, 11:46:21 am
QA-Dep...

You can see a new Release named "Release-3" has started automatically. Click on the "Release-3" link to view the status and logs.

Release-1 > Release-3

Stages

Continuous deployment
by Microsoft Visual Studio, 2025/05/20, 11:11 pm
Anil's
SauravSarkar-Coder... 27 minutes

QA-Deployment: Succeeded
2025/05/20, 10:57 pm

Prod-Deployment: In progress
Deploy Azure App Service
2025/05/20, 10:57 pm

QA-Deployment Succeeded
Prod-Deployment Running

This time it is a Continuous Deployment Trigger.

Release-1 > Release-3

Pipeline Variables History + Deploy ⚡ Cancel ⚡ Refresh ⚡ Edit ⚡

Release	Stages
Continuous deployment for Microsoft Visual Studio... on 20/5/2025, 12:11 pm Artifacts SauravSarkar-Co... 20/5/2025 1 Master	QA-Deployment Succeeded on 20/5/2025, 12:11 pm Prod-Deployment Succeeded on 20/5/2025, 12:11 pm

Both the stages will succeed.

This is a complete DevOps CI-CD Pipeline Automation with Complete Deployment Trigger Enabled.

Pipelines Recent All Runs New pipeline Filter pipelines

Recently run pipelines

Pipeline	Last run
SauravSarkar-CodersArcade.MyWebApp...	#20250520.3 • Updated War file 1 Individual CI for 1 master 1 statistics

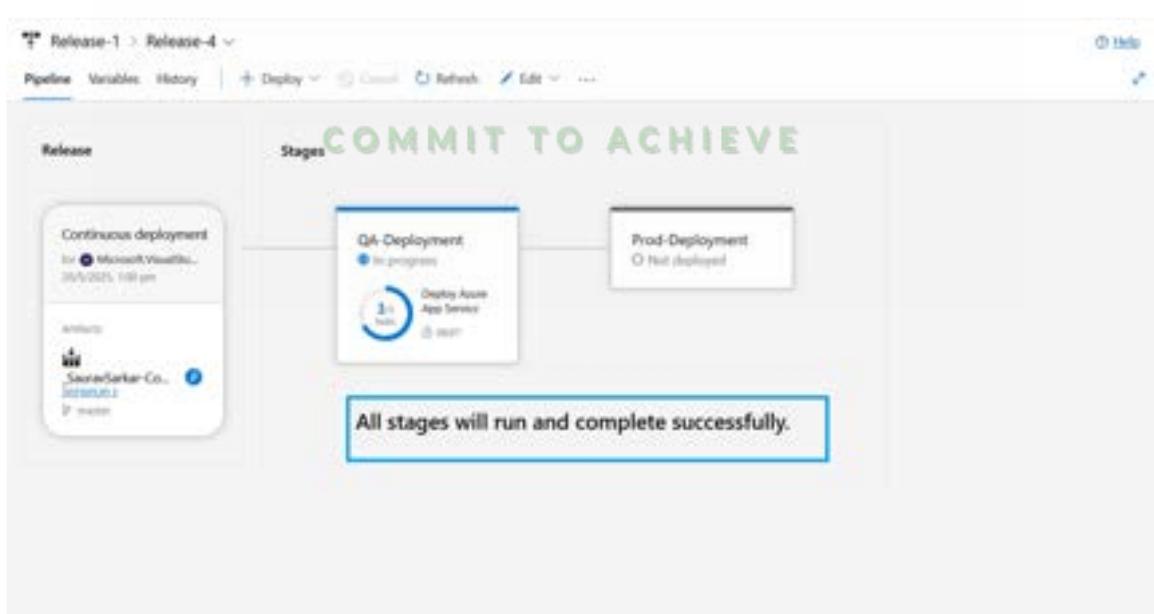
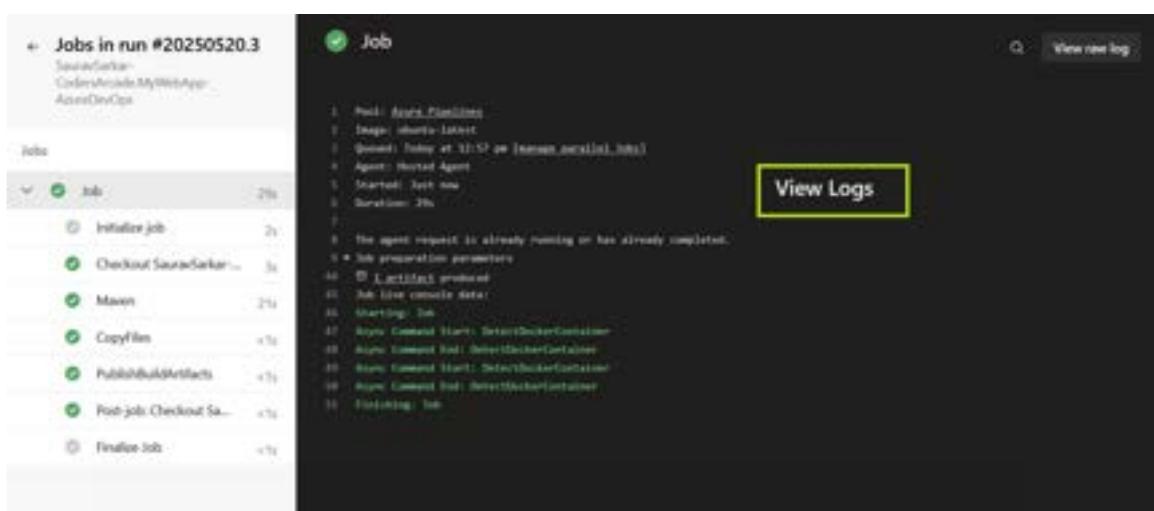
Change triggers a new Job in Pipeline ↪ Just now ↪

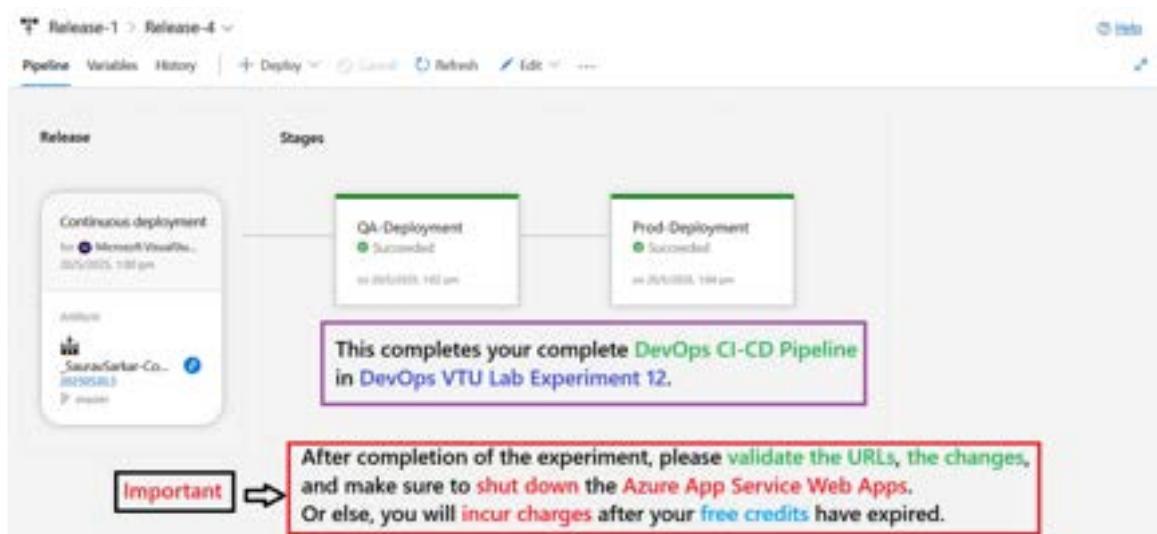
CODERS ARCADE

SauravSarkar-CodersArcade.MyWebApp-AzureDevOps

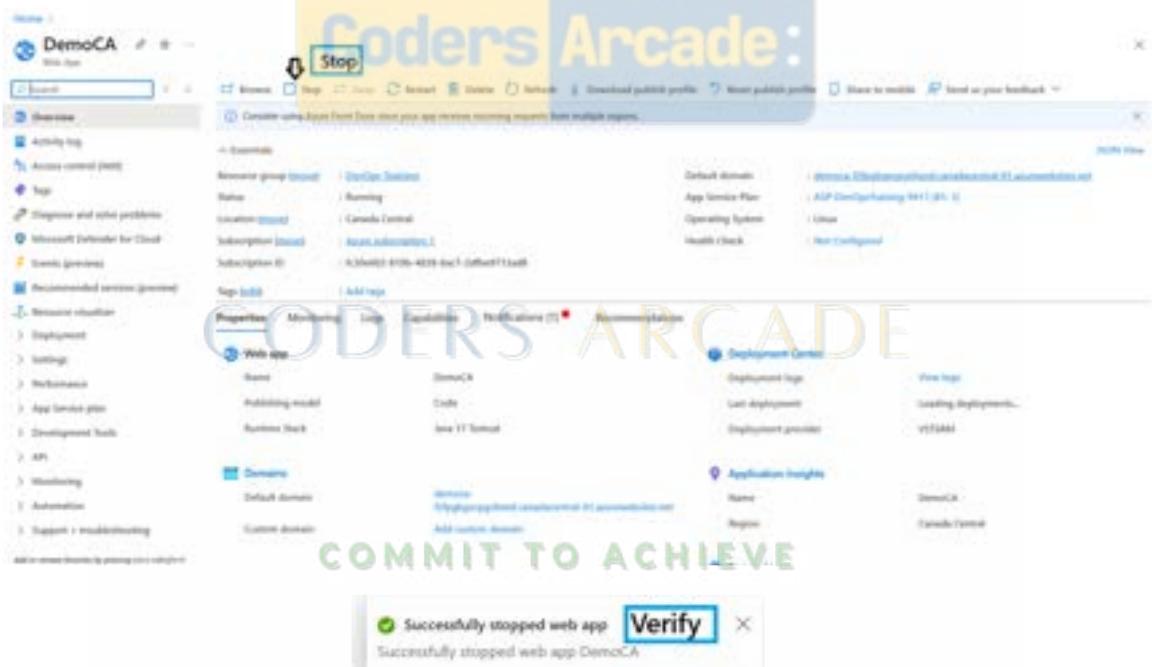
Runs Branches Analytics Note: If the changes are not reflected, run mvn clean package, then push the changes.

Description	Steps	Run
#20250520.3 • Updated War file 1 Individual CI for 1 master 1 statistics	Latest Job ↪	Just now
#20250520.2 • Latest Commit: Changed the index.jsp file for validation in...	1 Individual CI for 1 master 1 statistics	10m ago 0:26
#20250520.1 • Set up CI with Azure Pipelines Manually triggered by SauravSarkar 1 Pipeline 1	1 Pipeline	2h ago 0:27s



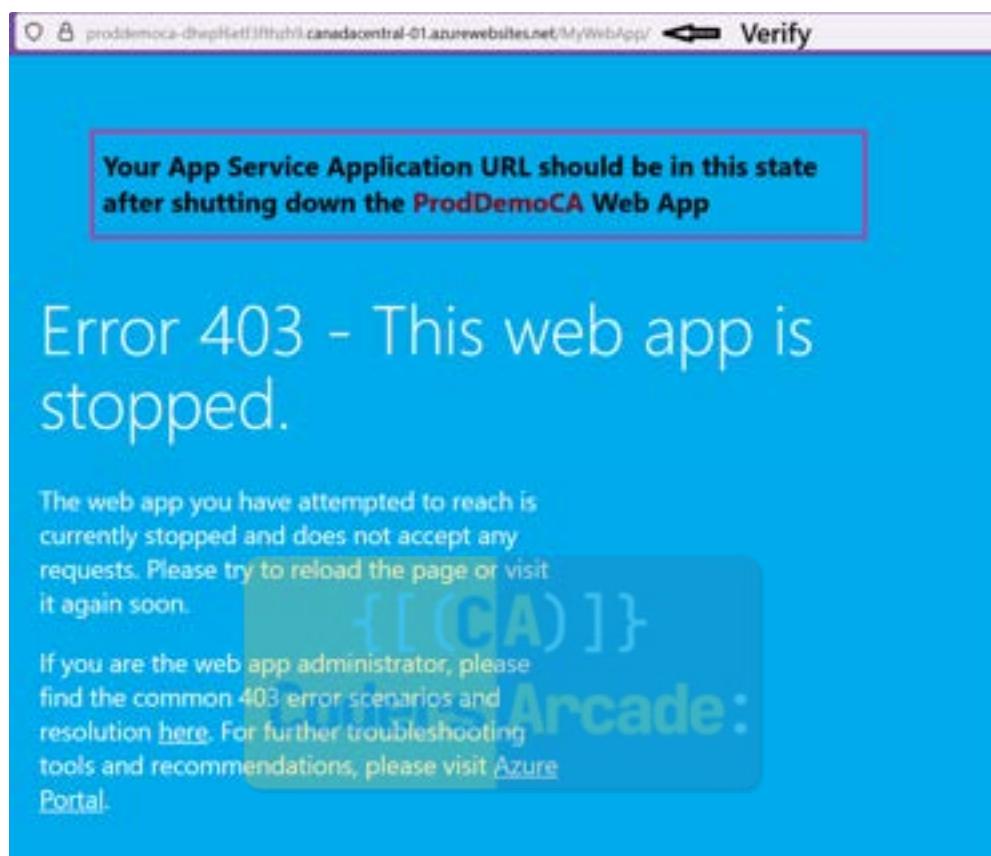


3. Now go to portal.azure.com/#home and stop the two Web Apps as shown below.
4. This is mandatory because otherwise you will incur charges after the free credits have expired [Important]



The screenshot shows the Azure portal's 'ProdDemoCA' web app overview page. The 'Stop' button is highlighted with a red arrow. A modal window at the bottom left displays the message: "Successfully stopped web app." with a green checkmark icon and a blue 'Verify' button.

The screenshot shows a browser window with the URL <https://prodemo-ca-fifpgbgcpgnhted.canadacentral-01.azurewebsites.net/>. The page displays an error message: "Your App Service Application URL should be in this state after shutting down the demoCA Web App". Below this, there is a large blue banner with the text: "Error 403 - This web app is stopped." and a message for administrators about common 403 error scenarios and resolution.



Final Summary – Experiment 12: Full CI/CD Pipeline with Azure DevOps (Production Deployment)

- Completed the **CI/CD pipeline** by configuring a **Release Pipeline** for deploying the final `.war` file to the **Production Server** (`prodDemoCA` Web App on Azure).
- Used the same `.war` artifact generated in the Build Pipeline (Experiment 10), now pushed to **Production** via a separate **release stage**.
- Deployed to the **prodDemoCA Azure App Service**, simulating a real-world Production environment.
- Copied the generated **App Service URL** and updated it inside the `driver.get("your_app_service_url");` method in the automation script.
- Reused the existing `BrowserTest.java` file and ran it with **JUnit** to perform a **simple title validation** — confirming successful deployment.
- Enabled the **Continuous Deployment Trigger** () so that every time you push code to GitHub, it:
 - Triggers the Build Pipeline
 - Publishes artifacts
 - Automatically starts a new **Release** (e.g., Release-4 after Release-3)
 - Deploys to the Production Server
 - Completes the automation test
- Validated how any code change (once pushed) triggers the **full CI/CD cycle** from build → release → deployment → test without manual intervention.
- Final reminder: Ensure the **Azure Web Apps** (`demoCA` and `prodDemoCA`) are started before releases get triggered, otherwise deployment will fail.

This wraps up the **complete DevOps CI/CD journey** across Experiments 10, 11, and 12 — from code build to production deployment with automation testing, all powered by Azure DevOps. 🚀💻🔁



CODERS ARCADE

COMMIT TO ACHIEVE

CA - Thank You Message From Coders Arcade

Thank You, Coders Arcade Family!

What an incredible journey it's been! 

From setting up Maven & Gradle builds, mastering Jenkins pipelines, diving into Ansible for configuration management, and finally reaching the finish line with full-scale CI/CD pipelines using Azure DevOps — we've covered it all, **step by step**, together. 

These past few weeks have been packed with:

-  Building real-world-ready projects
-  Automating end-to-end pipelines
-  Deploying apps in cloud environments
-  Running Selenium-based automation tests
-  Learning through hands-on experiments, not just theory

And honestly... it wouldn't have been possible without **your amazing support** — the views, the comments, the love, the shares. 

You kept us going. You kept us creating. 

 Yes, the **Cloud Theory** part (like basics of cloud models, services, scaling, pricing, etc.) is still pending — we had to **speed up the experiment uploads** for your syllabus timelines. But don't worry! We'll be **posting those theory videos soon** in the **same playlist**, so make sure to **stay tuned**.

 From the entire Coders Arcade team — **THANK YOU** for being a part of this amazing DevOps Lab Series.

 If you haven't already, please:

 **Subscribe** to the channel

 **Hit the bell icon** so you never miss an update

 Drop your feedback or questions in the comments — we read every one of them!

This is just the beginning. Bigger and better content is on the way — not just for VTU but for everyone who wants to **learn DevOps the right way**. 

Till then...

 **Commit To Achieve.** 

— Team Coders Arcade