**Detailed Schedule and Steps**

**9:00 AM–10:30 AM: Write Requirements and Sketch Architecture (~1.5 hours)**

**Goal**: Define the project scope and visualize the multi-cloud setup.

**Steps**:

1. **Open a Text Editor** (e.g., VS Code, Notepad):
   * Create a file named requirements.docx or requirements.md.
2. **Write Project Overview** (~20 minutes):
   * **Title**: "WordPress Student Portal: Multi-Cloud Secure Deployment"
   * **Objective**: Deploy a WordPress portal for course listings and user logins across AWS (EC2, Elastic Beanstalk) and Azure (VM, App Service), secured with IAM, Azure AD, Splunk SOAR, and in-depth testing.
   * **Application**: WordPress with plugins (LearnPress for courses, MiniOrange for Azure AD).
   * **Scope**:
     + IaaS: AWS EC2 (WordPress backend), Azure VM (MySQL).
     + PaaS: AWS Elastic Beanstalk (frontend), Azure App Service (REST API).
     + Security: IAM, RBAC, SSL/TLS, VPC/NSG, CloudTrail, Azure Monitor, Splunk Community, SOAR playbooks.
     + Testing: Nmap, WPScan, OpenVAS for vulnerabilities.
   * **Constraints**: 12 days, AWS Free Tier, Azure for Students, no coding.
3. **Detail Components** (~20 minutes):
   * **AWS**:
     + EC2: t2.micro, Amazon Linux 2, WordPress backend.
     + Elastic Beanstalk: PHP platform, WordPress frontend.
     + S3: Backup wp-content, encrypted with KMS.
   * **Azure**:
     + VM: Standard\_B1s, Ubuntu 20.04, MySQL.
     + App Service: F1 tier, WordPress REST API.
     + Blob Storage: Backups, encrypted with SSE.
   * **Security**:
     + Authentication: Azure AD SSO for WordPress.
     + Network: VPC (AWS), NSG (Azure) to restrict access.
     + Logging: CloudTrail, Azure Monitor, Splunk Community.
     + Automation: Splunk SOAR for login failure alerts.
     + Testing: Nmap (ports), WPScan (WordPress), OpenVAS (CVEs).
4. **Sketch Architecture** (~30 minutes):
   * Use a tool like Draw.io (free, [diagrams.net](https://app.diagrams.net/)) or paper.
   * Draw:
     + Users → AWS CloudFront → Elastic Beanstalk (WordPress frontend).
     + Elastic Beanstalk ↔ EC2 (WordPress backend) via S3.
     + EC2/App Service ↔ Azure VM (MySQL).
     + App Service (WordPress API) serving REST endpoints.
     + Security layers: IAM, Azure AD, VPC/NSG, Splunk/SOAR.
   * Save as architecture.png or sketch and photograph.
5. **Review and Save** (~10 minutes):
   * Ensure document covers WordPress, clouds, security, SOAR, and testing.
   * Save to local folder (e.g., ~/Projects/wordpress-portal/docs/).
   * Commit to reviewing on Day 12 for report.

**Output**:

* requirements.docx or requirements.md (~1 page).
* architecture.png or sketch.

**Commands that are used in Day 1**

Below is a concise list of **commands** used on **Day 1** for setting up your WordPress student portal project, as outlined in the detailed plan. These commands cover tool installation (Terraform, AWS CLI, Azure CLI, Git, Nmap, WPScan), account configuration, and GitHub repository initialization. They are specific to a **Linux/macOS** environment (for Windows, minor adjustments may apply, e.g., using curl or GUI installers). I’ve excluded non-command actions like writing documents or browsing to ensure focus on executable commands.

aws configure

az login

wget https://releases.hashicorp.com/terraform/1.5.7/terraform\_1.5.7\_linux\_amd64.zip

unzip terraform\_1.5.7\_linux\_amd64.zip

sudo mv terraform /usr/local/bin/

terraform version

terraform init

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws sts get-caller-identity

aws s3 ls

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

az account show

az vm list

sudo apt install git

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

git version

git status

sudo apt install nmap

nmap --version

nmap localhost

sudo apt install ruby

gem install wpscan

wpscan --version

wpscan --url https://wordpress.org --no-update

git clone https://github.com/<your\_username>/wordpress-security-portal.git

cd wordpress-security-portal

mkdir docs infra screenshots

git add .

git commit -m "Initial setup: requirements, architecture, README"

git push origin main

**Day 2:** I created IAM policies which are EC2 policy , S3 policy and Elastic Beanstalk Policy.

Which this policies are grouped into the single IAM users group.

**EC2 policy:**

 Name: StudentEC2Access.

 Description: “Allows EC2 management for WordPress backend.”

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ec2:Describe\*",

"ec2:StartInstances",

"ec2:StopInstances",

"ec2:RunInstances",

"ec2:TerminateInstances"

],

"Resource": "\*"

}

]

}

**S3 Policy**:

 Name: StudentS3Access.

 Description: “Allows S3 access for WordPress backups.”

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:PutObject",

"s3:GetObject",

"s3:ListBucket",

"s3:DeleteObject"

],

"Resource": [

"arn:aws:s3:::student-portal-wp-content",

"arn:aws:s3:::student-portal-wp-content/\*"

]

}

]

}

**Elastic Beanstalk Policy**:

 Name: StudentBeanstalkAccess.

 Description: “Allows Beanstalk management for WordPress frontend.”

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"elasticbeanstalk:CreateApplication",

"elasticbeanstalk:CreateEnvironment",

"elasticbeanstalk:UpdateEnvironment",

"elasticbeanstalk:Describe\*"

],

"Resource": "\*"

}

]

}

**Create docs/iam\_setup.md in repo folder:**

# AWS IAM Setup

- Group: StudentPortalAdmins

- Policies:

- StudentEC2Access: EC2 management.

- StudentS3Access: S3 bucket access.

- StudentBeanstalkAccess: Beanstalk management.

- User: student-user, attached to StudentPortalAdmins.

- Tested: ec2:Describe, s3:List, elasticbeanstalk:Describe.