  
Practical-2 Platform as a service using AWS.

SAP ID: 86092300041

**Date:-23/01/2024**

**Platform as a Service:**Platform as a Service (PaaS) is a cloud computing service model that provides a ready-made platform allowing customers to develop, deploy, and manage applications without dealing with the complexities of underlying infrastructure.AWS PaaS solutions, such as Elastic Beanstalk and App Runner, automate infrastructure provisioning, scaling, and maintenance, allowing developers to focus on writing code without dealing with the complexities of managing servers. These services provide integrated development tools, support for various programming languages, scalability, and managed services like databases, all while following a pay-as-you-go pricing model.

**Here are some key characteristics and components of PaaS in AWS:**

1. **Application Hosting Environment:  
   a.** PaaS provides a ready-made environment for hosting applications. AWS offers services like AWS Elastic Beanstalk, which allows developers to upload their application code, and the platform takes care of provisioning and managing the underlying infrastructure.
2. **Automated Deployment:  
   a.** PaaS offerings typically include automated deployment and scaling capabilities. AWS services like AWS Elastic Beanstalk and AWS App Runner automate the deployment process, making it easier to handle updates and changes to applications.
3. **Managed Services:  
   a.** AWS PaaS offerings often include managed services for databases, messaging, caching, and other common application components. For example, Amazon RDS (Relational Database Service) is a managed database service that can be leveraged within a PaaS environment.
4. **Development Frameworks:  
   a.** PaaS platforms often support multiple programming languages and frameworks, allowing developers to choose the tools they are most comfortable with. AWS Elastic Beanstalk supports various programming languages, including Java, Python, Node.js, and more.
5. **Scalability:  
   a.** PaaS solutions in AWS are designed to scale easily. They automatically handle the provisioning and scaling of resources based on application demand. This ensures that the application can handle varying levels of traffic without manual intervention.
6. **Integrated Development Tools:  
   a.** PaaS offerings typically come with integrated development tools that facilitate the development and debugging process. AWS provides tools like AWS CodeBuild, AWS CodePipeline, and AWS CodeDeploy for continuous integration and continuous deployment (CI/CD) workflows.
7. **Cost Management:  
   a.** PaaS services often follow a pay-as-you-go pricing model, allowing users to pay only for the resources they consume. This can lead to cost savings as users do not need to invest in and maintain their own infrastructure.
8. **Security and Compliance:  
   a.** AWS PaaS offerings incorporate security features and compliance measures to help protect applications and data. AWS manages the security of the underlying infrastructure, while users are responsible for securing their application code and data.

**Advantages of PAAS:**

1. **Simplified Development:** PaaS abstracts the complexities of infrastructure management, allowing developers to focus on coding and application logic rather than worrying about hardware, networking, and other underlying components. This simplification accelerates the development process.
2. **Faster Time-to-Market:** PaaS provides pre-configured development frameworks, tools, and services, reducing the time needed to set up and manage infrastructure. This results in quicker deployment of applications, enabling faster time-to-market for new features or products.
3. **Scalability:** PaaS platforms often include automated scaling capabilities. This means applications can easily handle varying levels of traffic, and resources can be automatically provisioned or de-provisioned based on demand. It ensures optimal performance without manual intervention.
4. **Cost-Efficiency:** PaaS typically follows a pay-as-you-go pricing model, allowing users to pay only for the resources they consume. This can result in cost savings, as organizations do not need to invest heavily in infrastructure or worry about overprovisioning.
5. **Flexibility and Portability:** PaaS supports various programming languages and frameworks, offering flexibility to developers. It also provides a level of portability, allowing applications to be easily moved between different cloud providers or environments.

**Amazon Elastic Beanstalk:**

Amazon Elastic Beanstalk is a web infrastructure management service. It handles deployment and scaling for web applications and services.Elastic Beanstalk can automatically manage setup, configuration, scaling and provisioning for other AWS services. AWS Elastic Beanstalk is an AWS-managed service for web applications. Elastic Beanstalk is a pre-configured EC2 server that can directly take up your application code and environment configurations and use it to automatically provision and deploy the required resources within AWS to run the web application. Unlike EC2 which is Infrastructure as a service, Elastic Beanstalk is a Platform As A Service (PAAS) as it allows users to directly use a pre-configured server for their application. Of course, you can deploy applications without ever having to use elastic beanstalk but that would mean having to choose the appropriate service from the vast array of services offered by AWS, manually provisioning these AWS resources, and stitching them up together to form a complete web application. Elastic Beanstalk abstracts the underlying configuration work and allows you as a user to focus on more pressing matters.

**Components of Amazon ElasticBeanStalk:**

1. **Application:** Elastic Beanstalk directly takes in our project code. So the Elastic Beanstalk application is named the same as your project home directory.
2. **Application Environments:** Users may want their application to run on different environments like DEV, UAT, and PROD. You can create and configure different environments to run applications on different stages.
3. **Environment Health:** One of the most lucrative features of running applications on AWS or most of the other cloud platforms is automated health checks. AWS runs automatic health checks on all EC-2 deployments (Elastic Beanstalk is a managed EC-2 service) which can be monitored from the AWS console.
4. **Elastic Load Balancing:** All the web requests to the application are not directly relayed to application instances. They first hit the Elastic Load Balancer (ELB), which, as the name suggests, balances the load across all the application instances.
5. **Language support:** Elastic Beanstalk supports the applications developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.
6. **Pricing:** There is no extra charge for using Elastic Beanstalk. Users are only required to pay for the services and resources provisioned by Elastic Beanstalk Service.
7. **Automatic Provisioning:** Elastic Beanstalk takes away the burden of choosing the right services and configuring their security groups to work together.
8. **Impossible to Outgrow:** AWS claims that since Elastic Beanstalk uses the Auto Scaling feature it can, in theory, handle any amount of internet traffic.

**IAM:**

IAM stands for Identity and Access Management. In the context of Amazon Web Services (AWS), IAM refers to the service that allows you to manage access to AWS resources securely. IAM enables you to control who (authentication) can do what (authorization) in your AWS environment.

**Here are key aspects of AWS Identity and Access Management (IAM):**

**Users and Groups:**

**Users:** Represent individuals or entities that interact with AWS services. Each user has a unique set of security credentials.

**Groups:** Users can be organized into groups, and permissions can be assigned to groups, making it easier to manage access.

**Roles:** IAM roles define a set of permissions for making AWS service requests. Roles are not associated with a specific user or group but can be assumed by users, applications, or services when needed.

**Policies:** IAM policies are JSON documents that define permissions. They can be attached to users, groups, or roles, specifying what actions are allowed or denied on what resources.

**Access Keys:** IAM provides access keys (access key ID and secret access key) for programmatic access to AWS services. These keys are often used by developers and applications.

**Multi-Factor Authentication (MFA):** IAM supports MFA, an additional layer of security that requires users to provide a second form of authentication (such as a code from a virtual or hardware MFA device) in addition to their password.

**Identity Federation:** IAM allows you to integrate with external identity providers, such as Active Directory or social identity providers, to grant temporary access to AWS resources.

**Resource-Level Permissions:** IAM policies can define permissions not only at the service level but also at the resource level. This allows fine-grained control over access to specific AWS resources.

**IAM Roles for EC2 Instances:** IAM roles can be assigned to EC2 instances, allowing applications running on those instances to securely access AWS resources without embedding credentials in the code.

**Policy Conditions:** IAM policies can include conditions that must be met for the policy to be in effect. Conditions can be based on factors such as the time of day, the source IP address, or the use of MFA.

**Implement paas using elastic beanstalk for the following:**

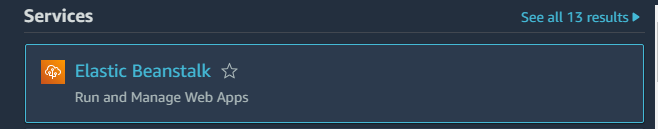
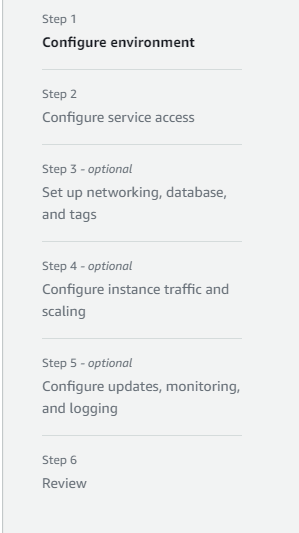
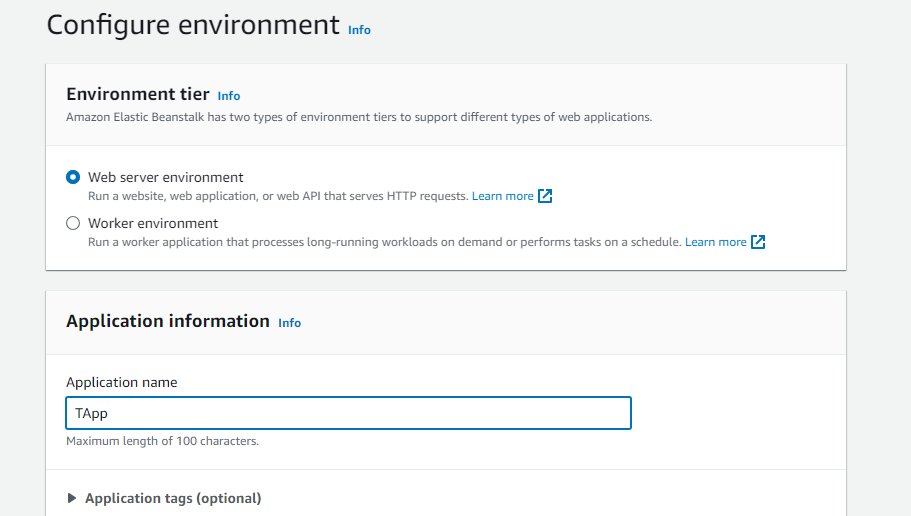
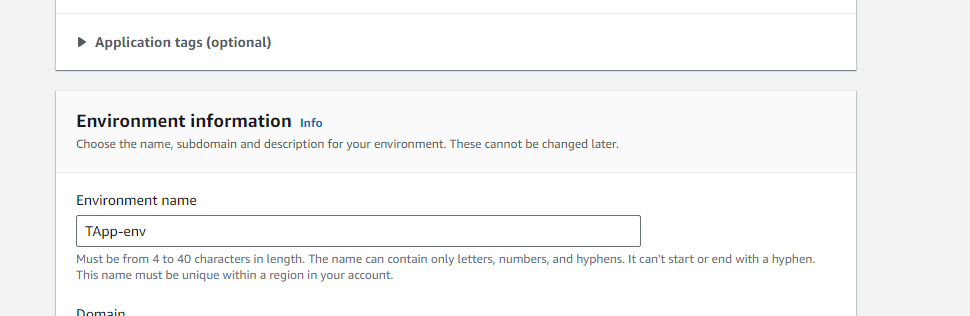
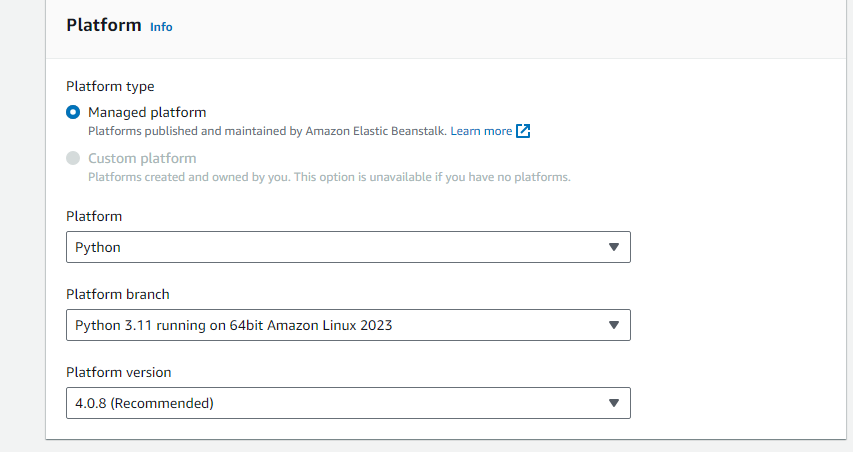
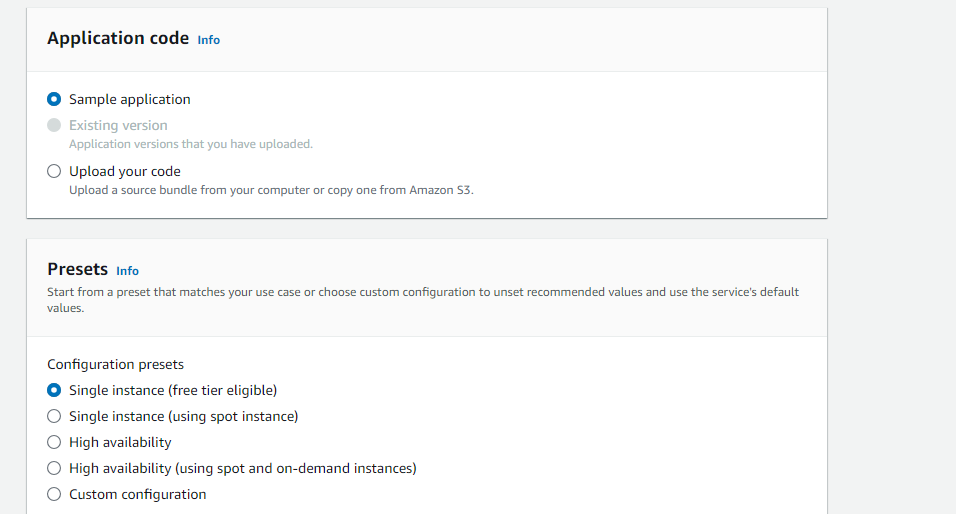
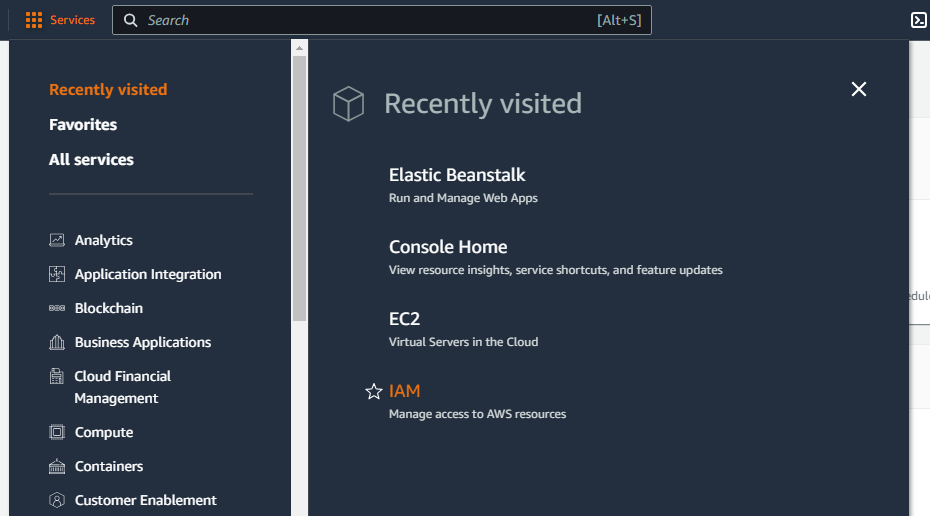
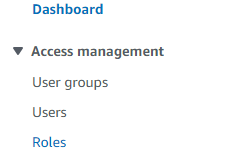
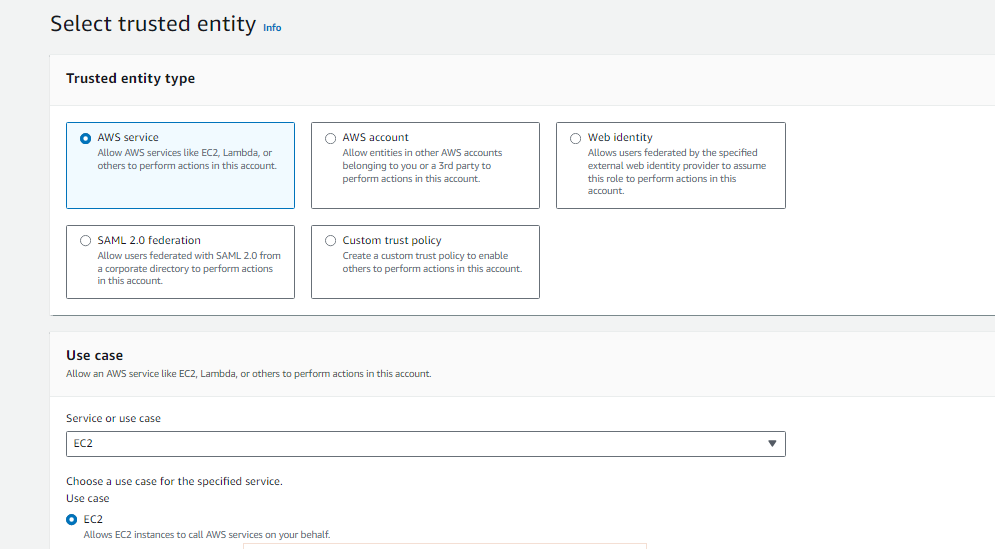
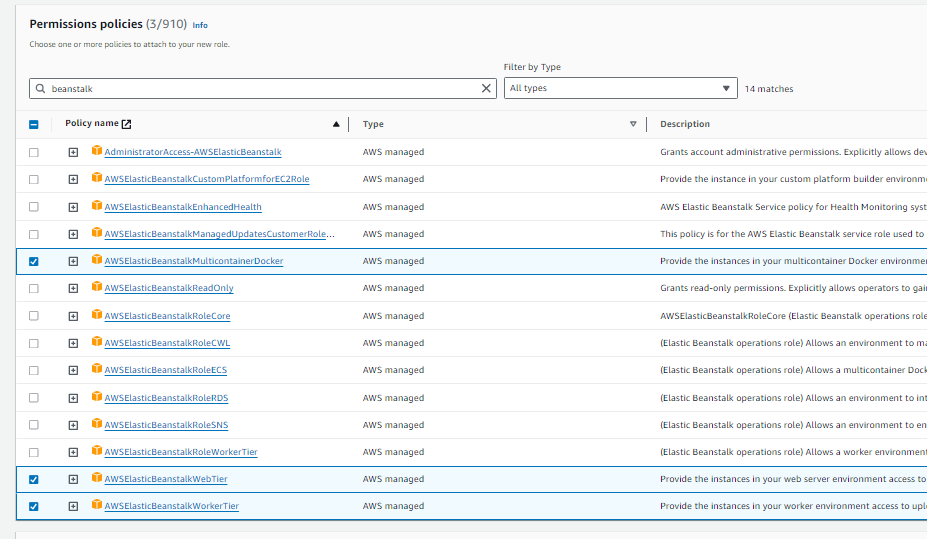
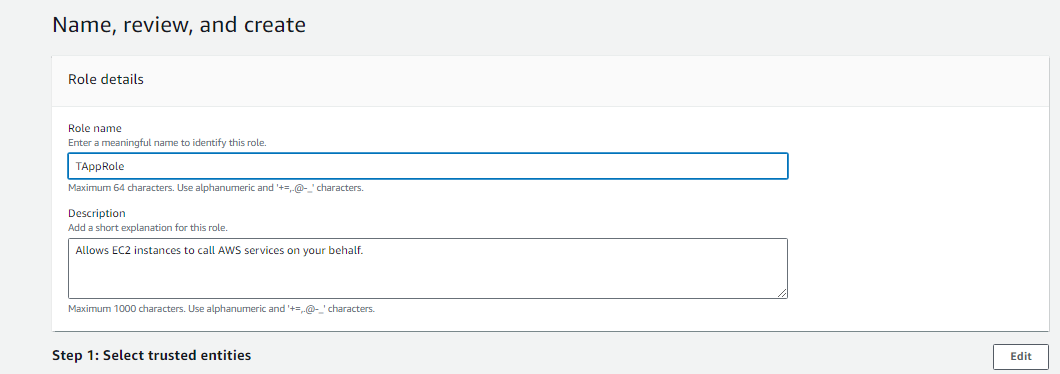
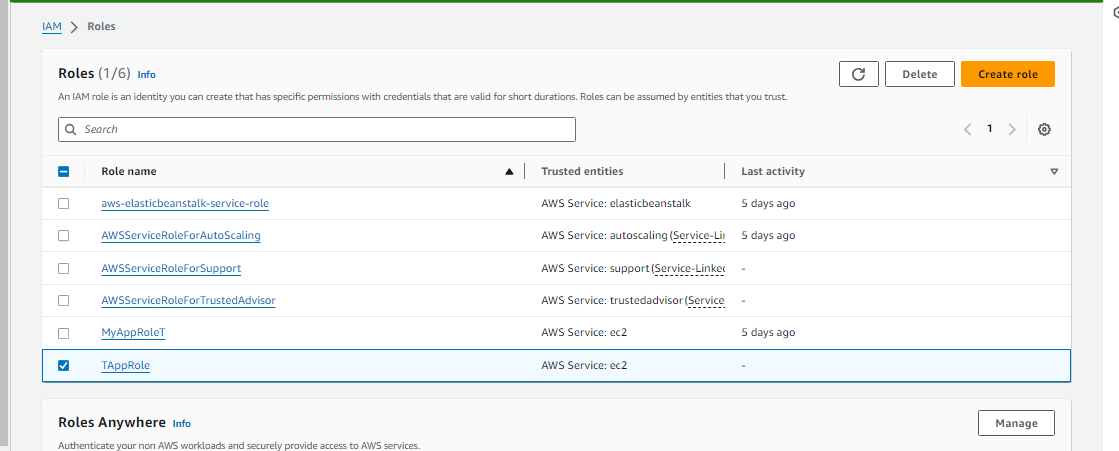
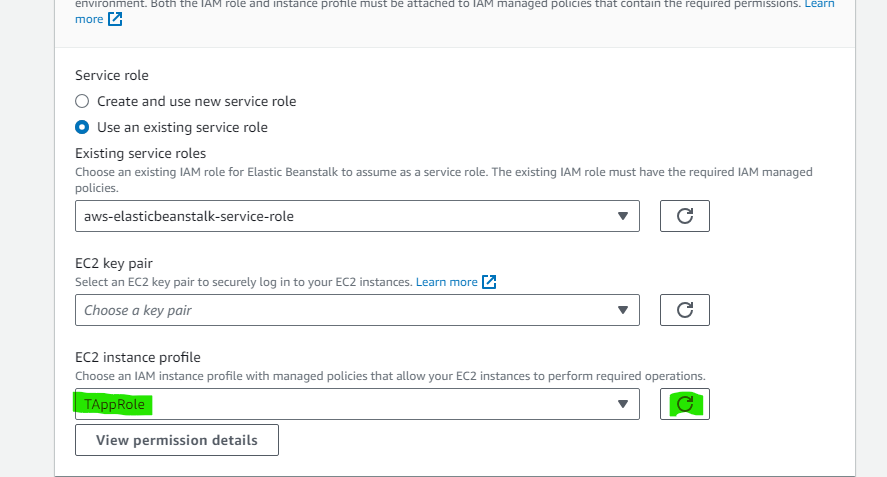
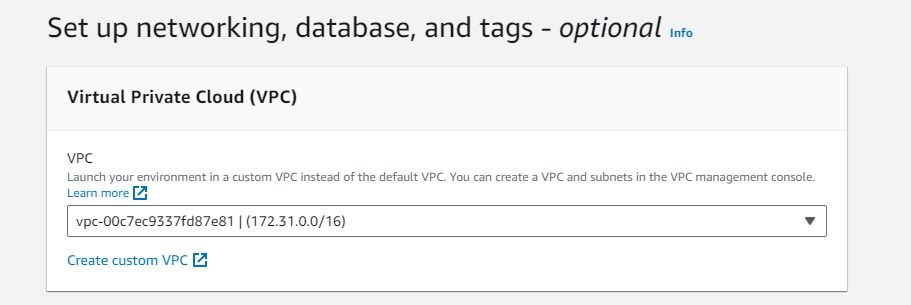
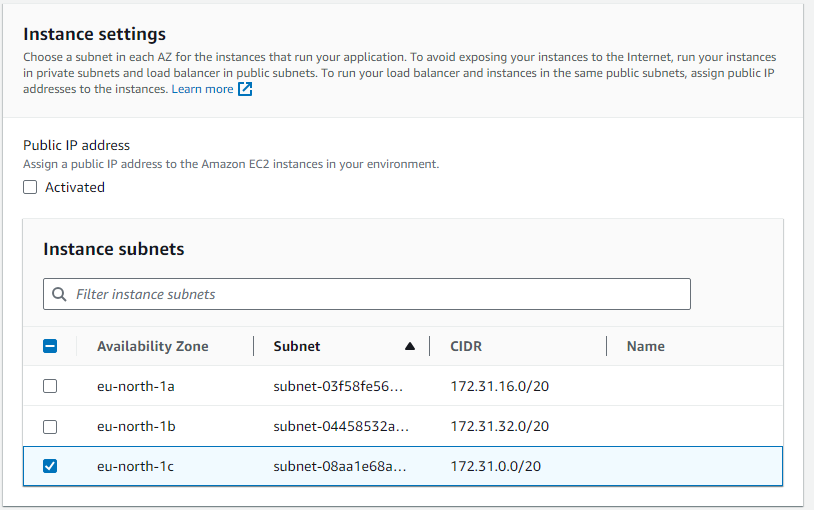
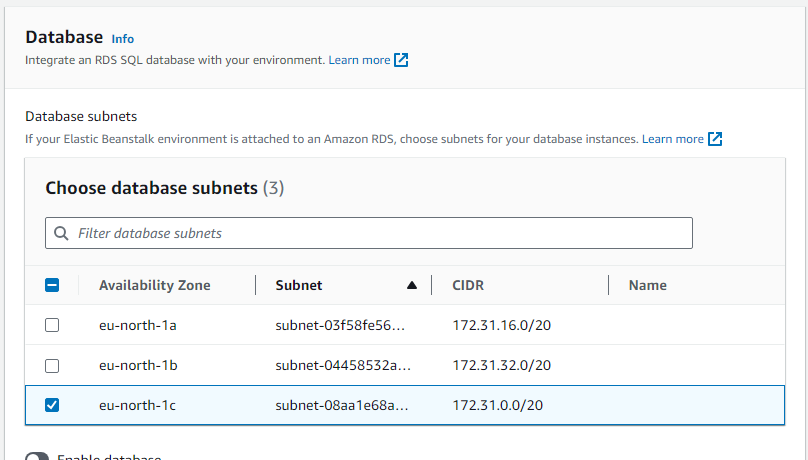
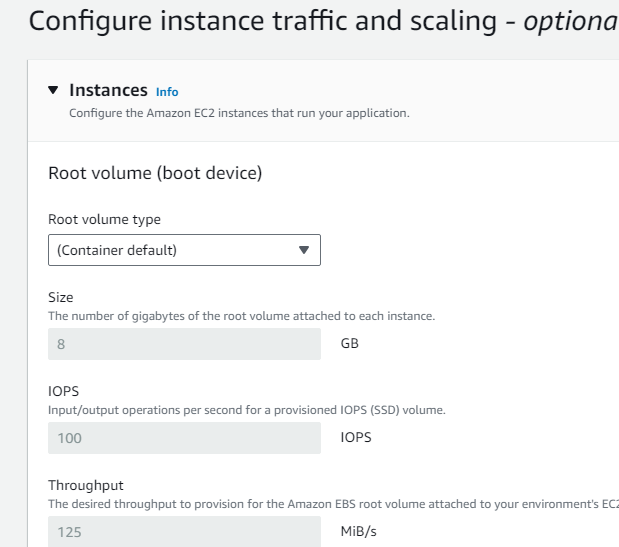
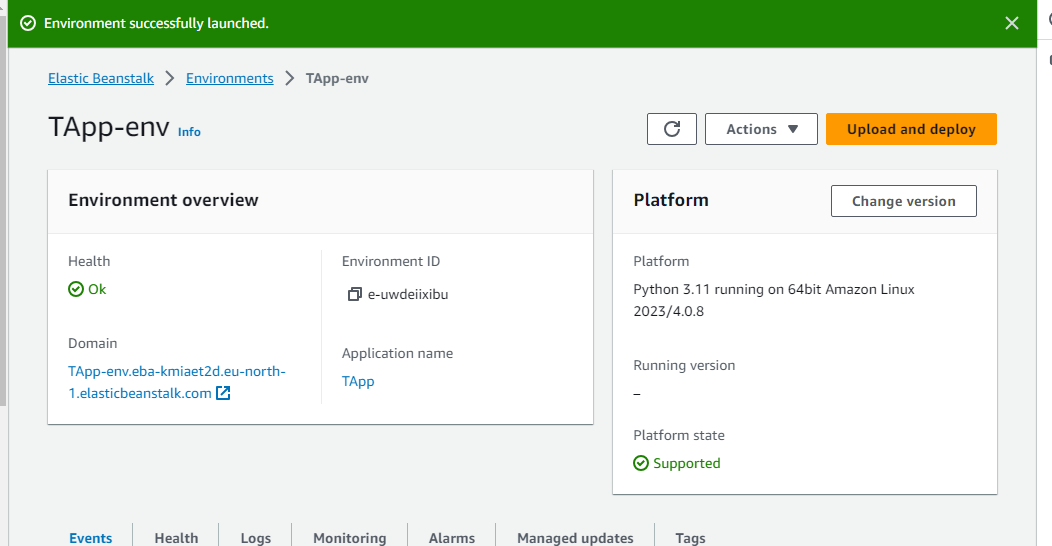
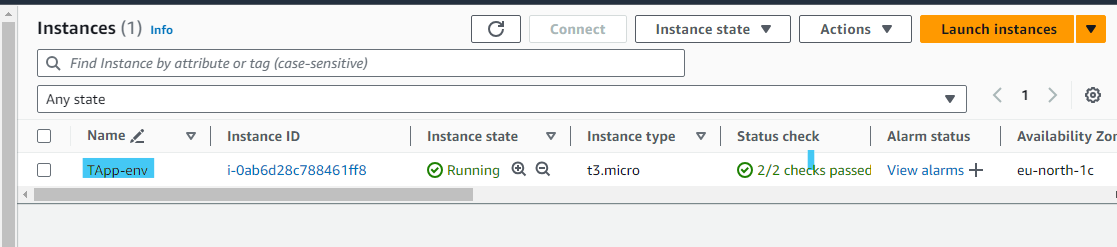
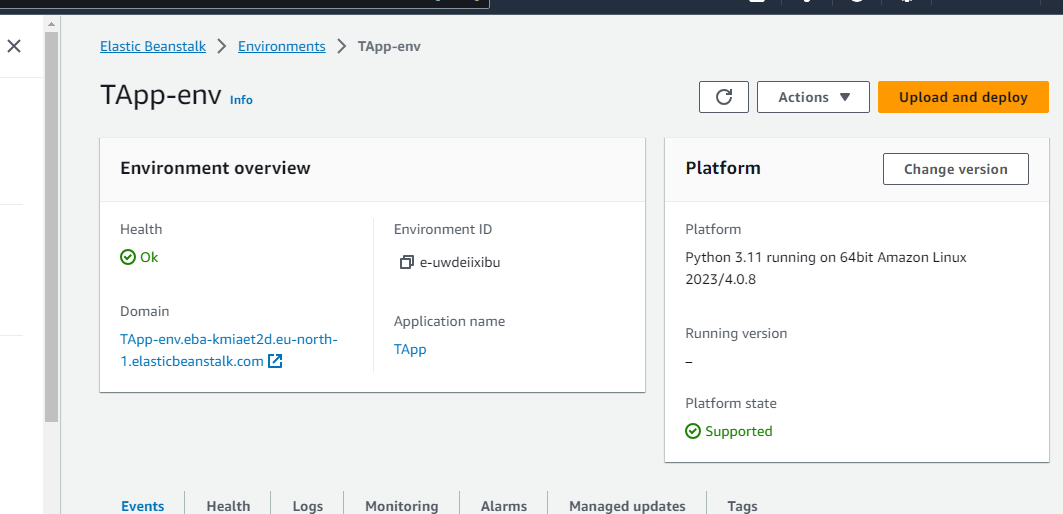
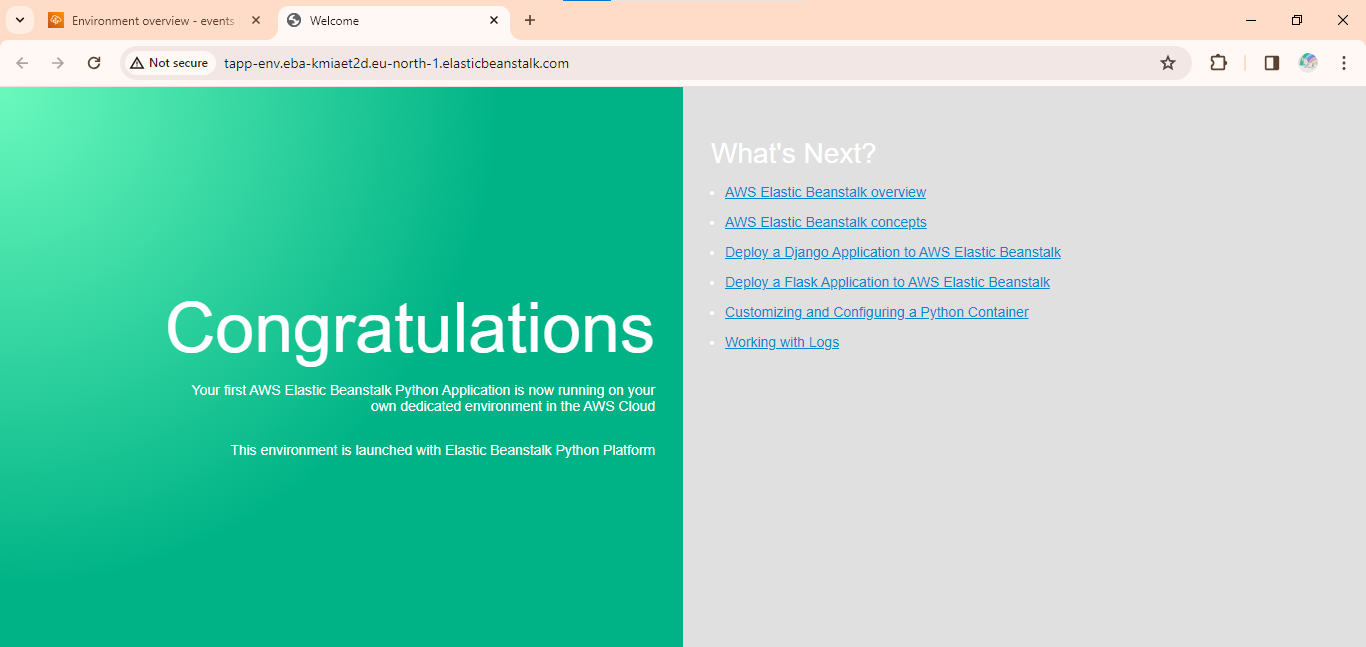
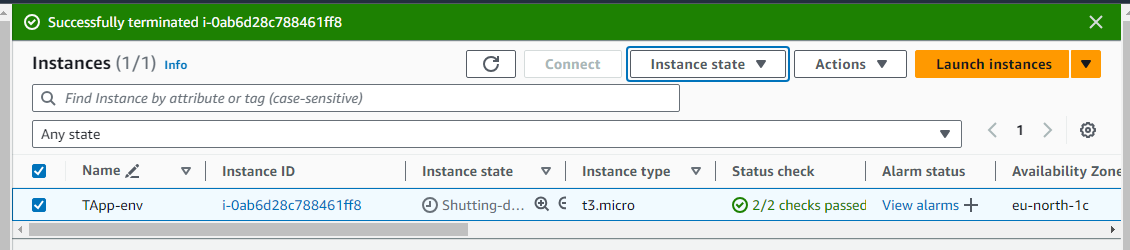
**1. Python**

**2. Java**

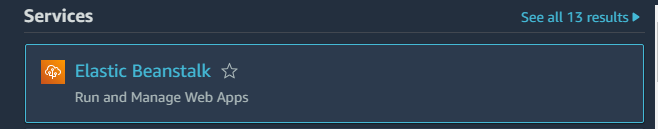
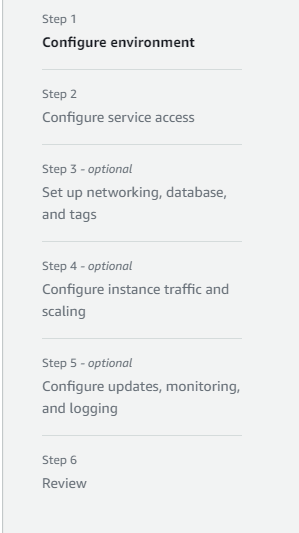
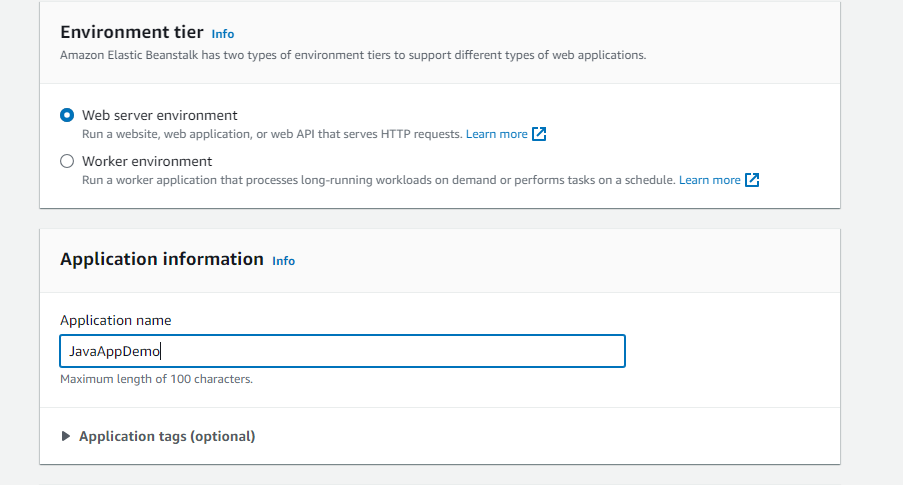
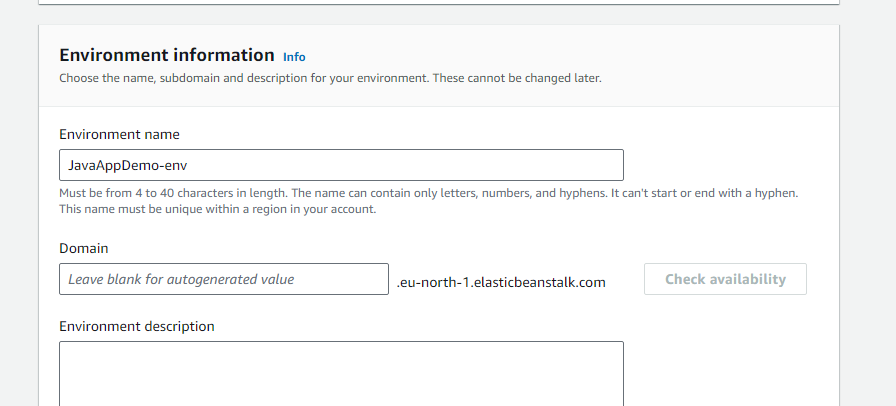
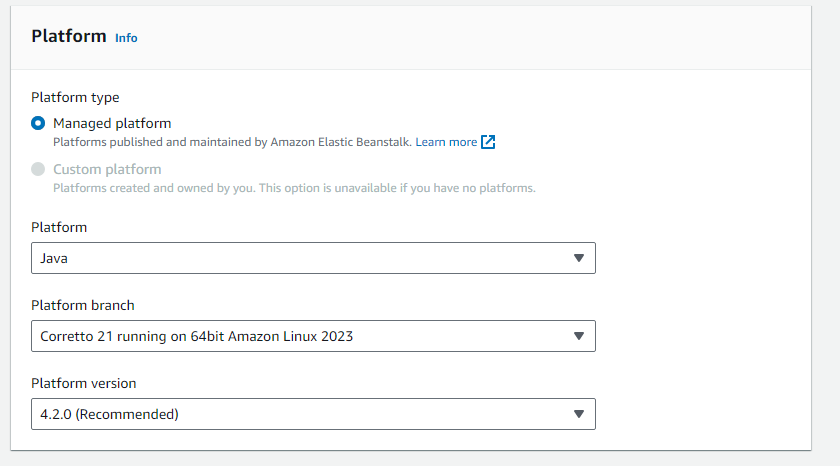
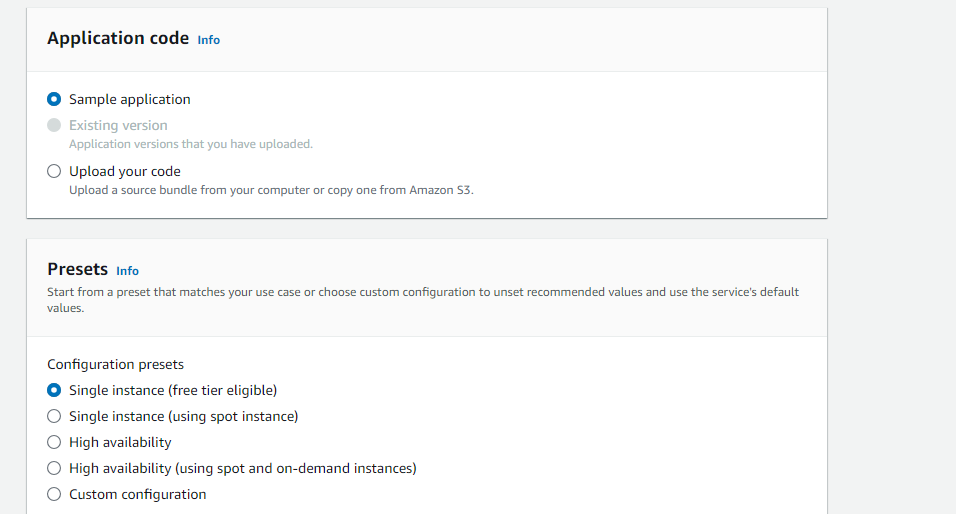
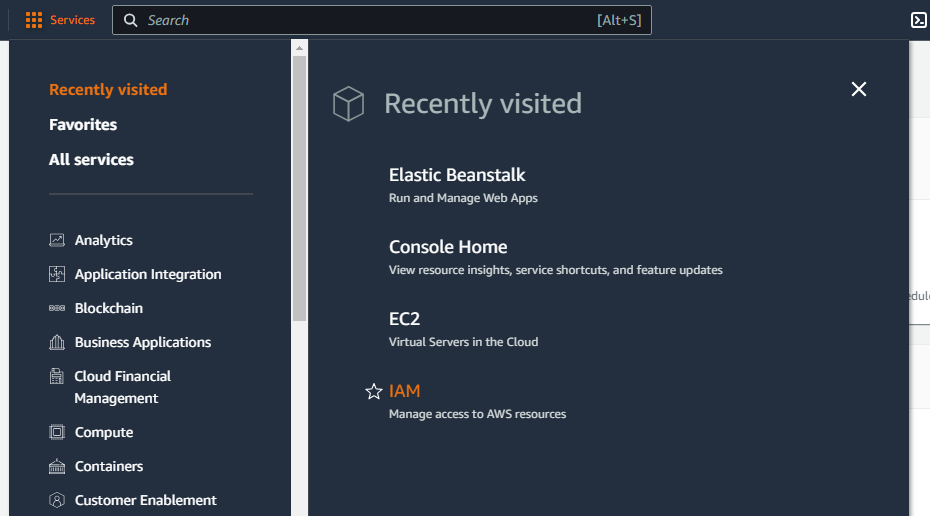
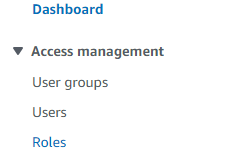
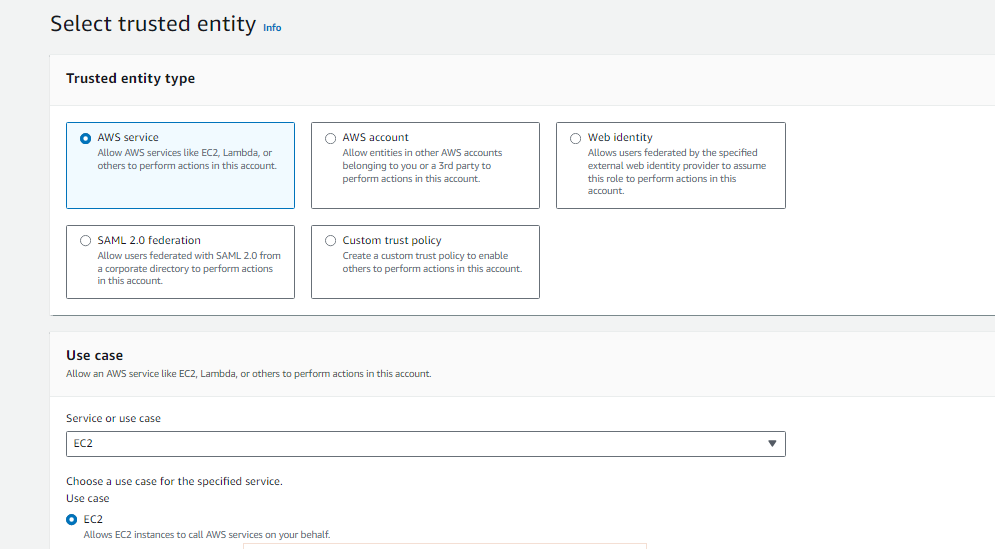
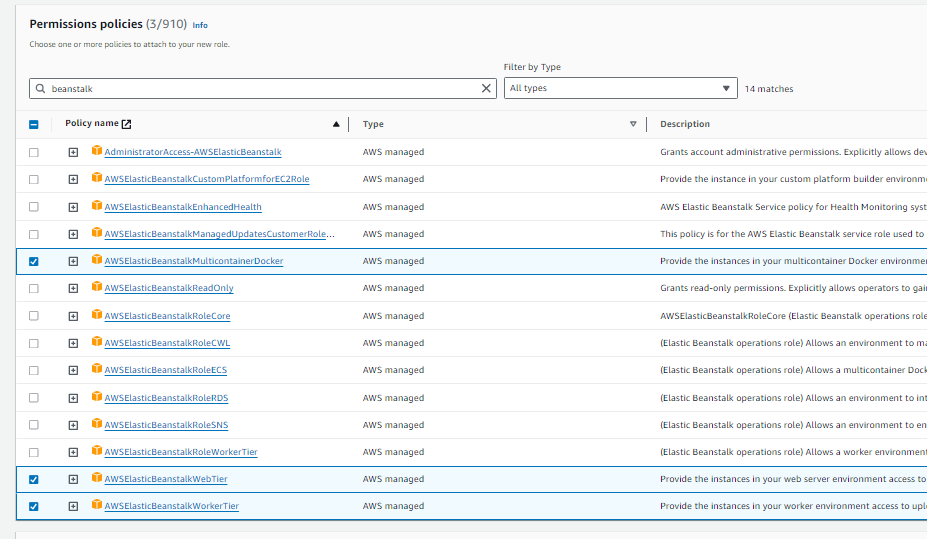
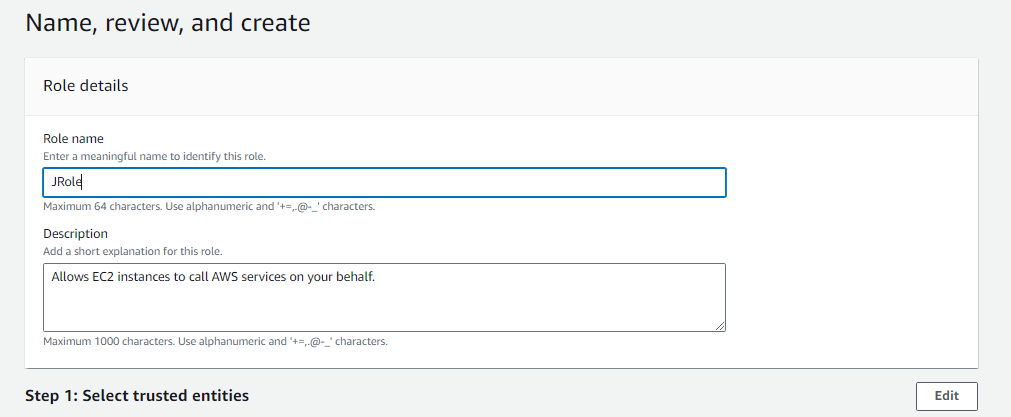
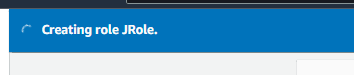
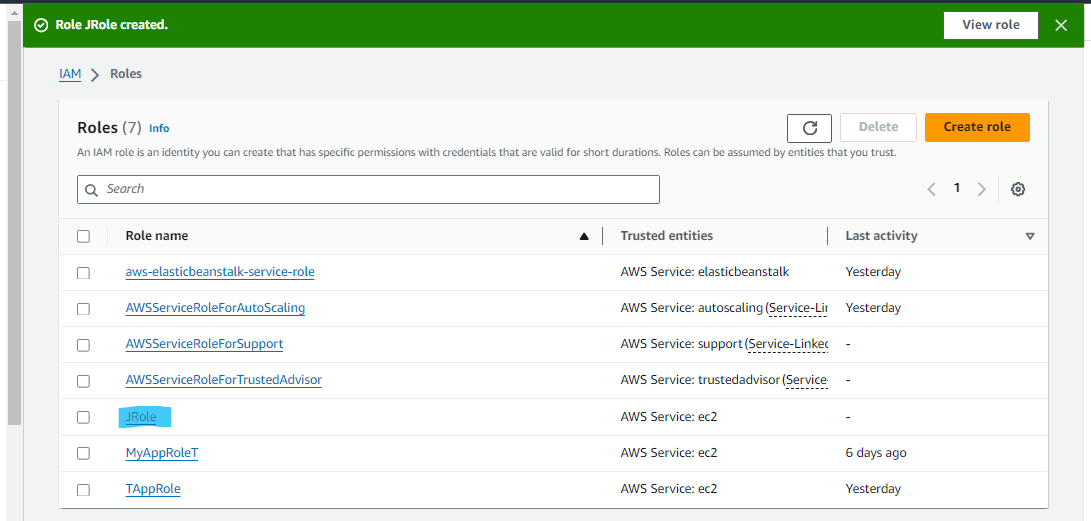
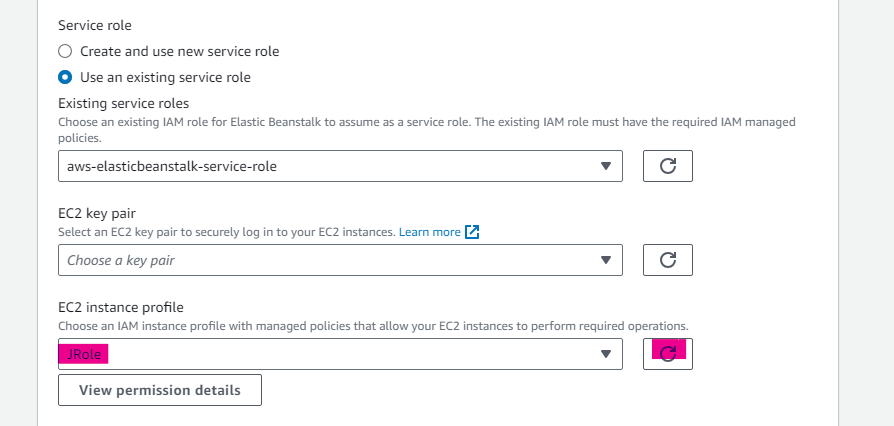
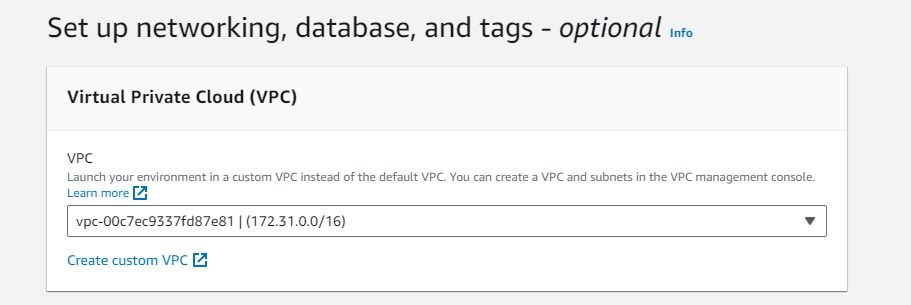
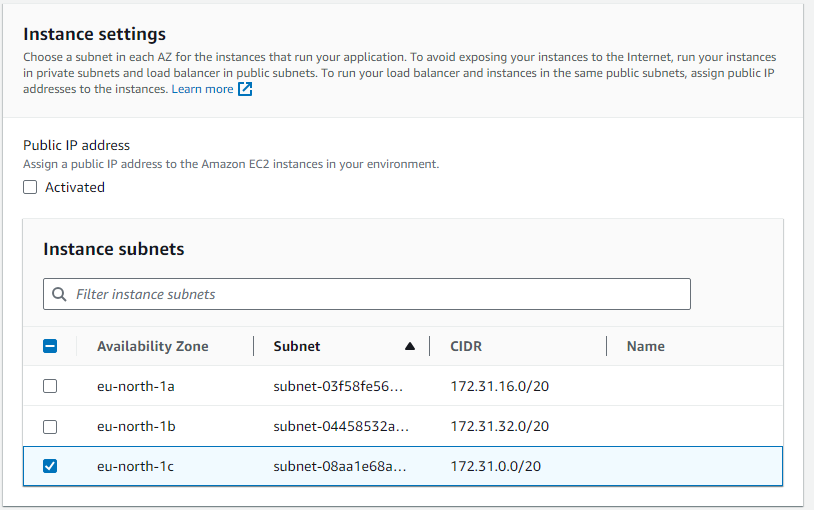
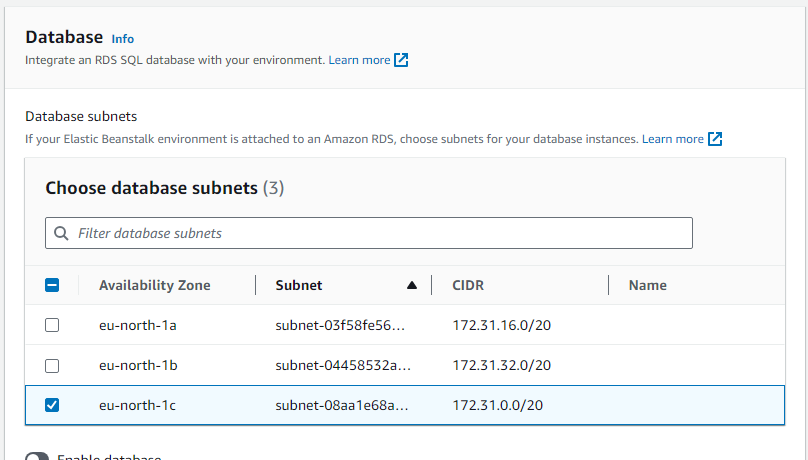
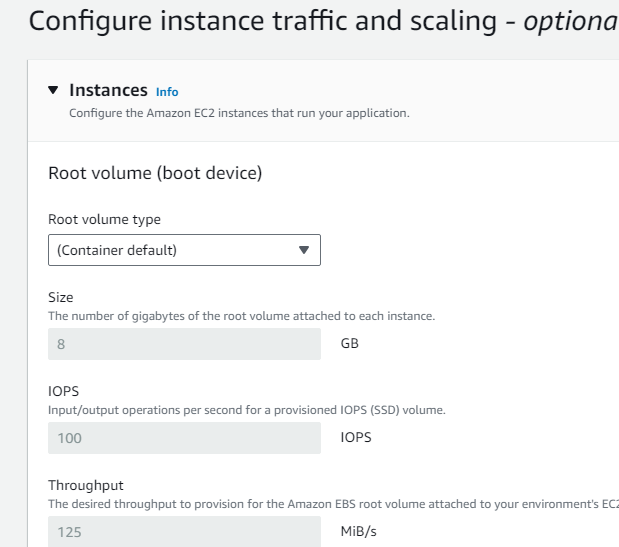
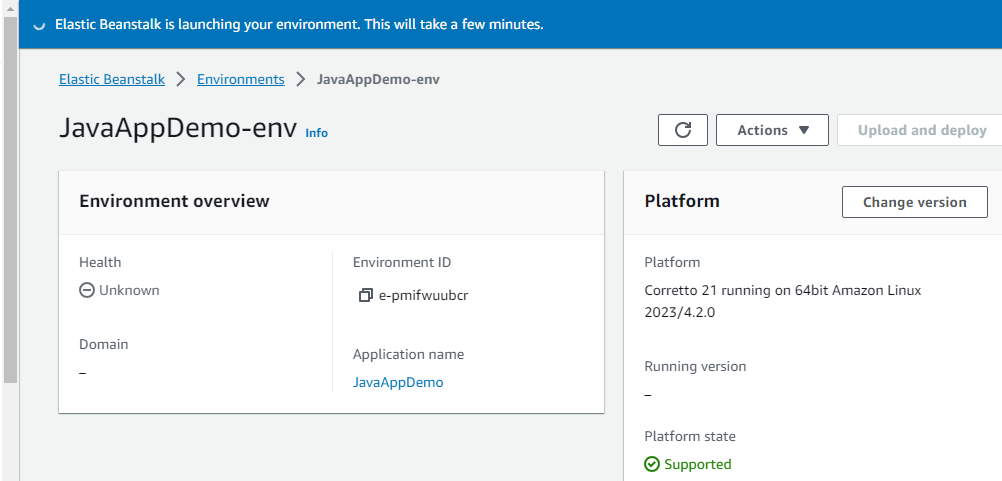
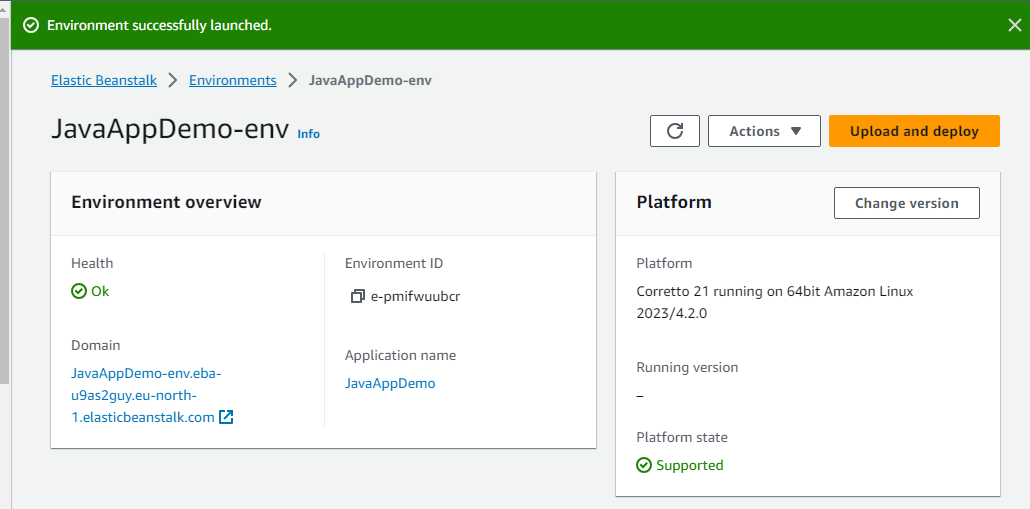
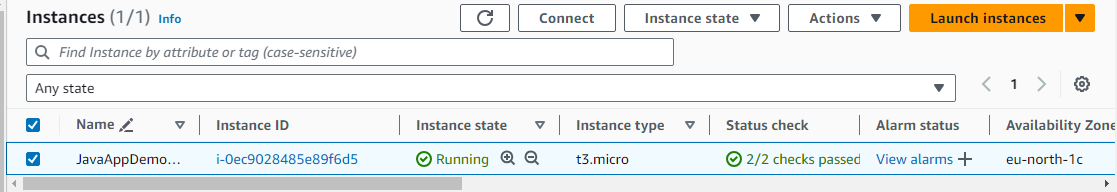
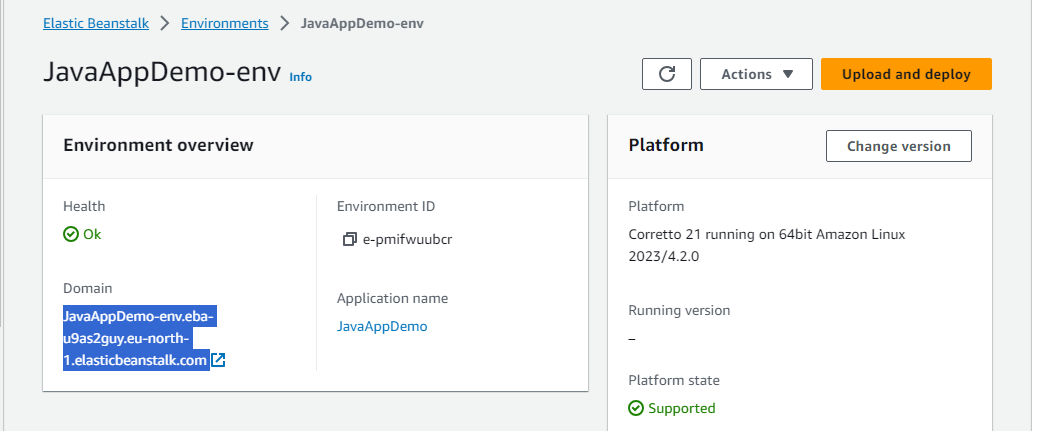
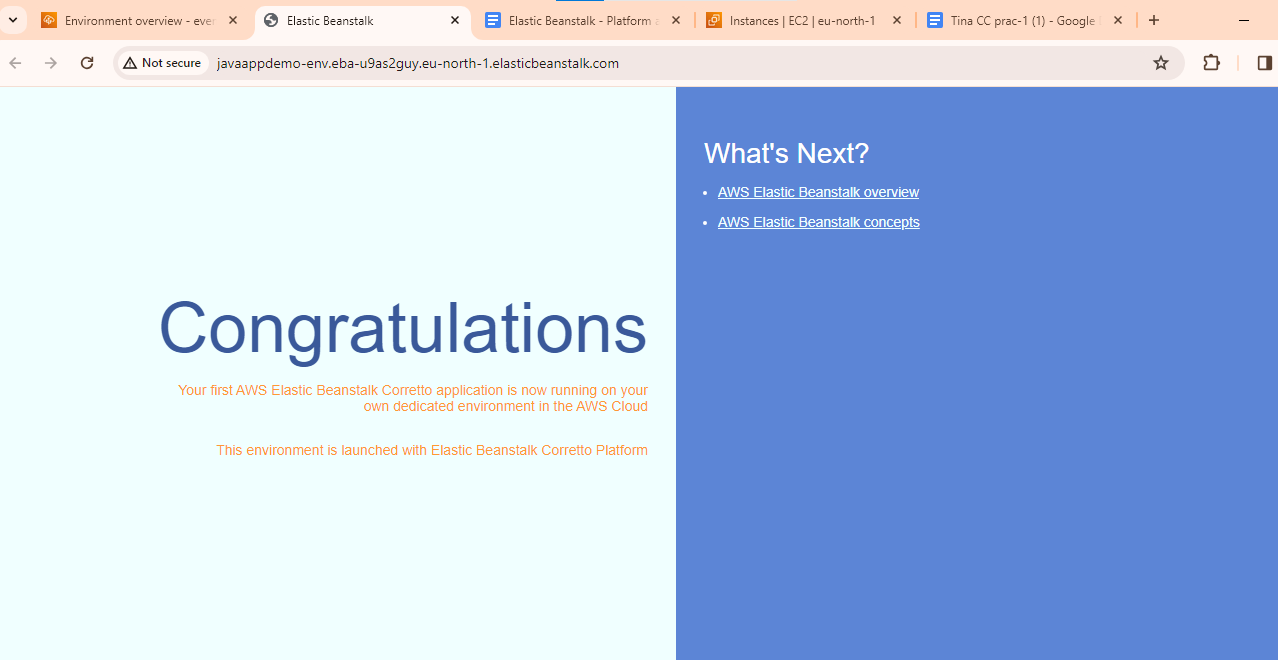
**3. Server**

**4. Node.js**

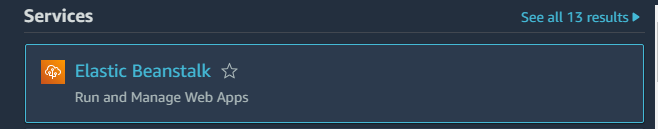
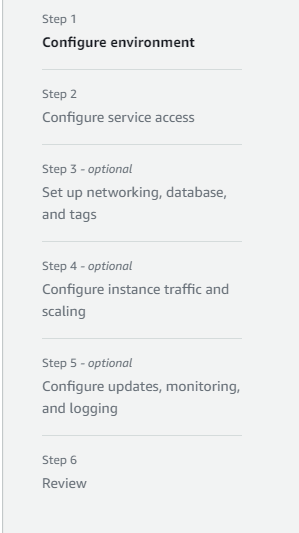
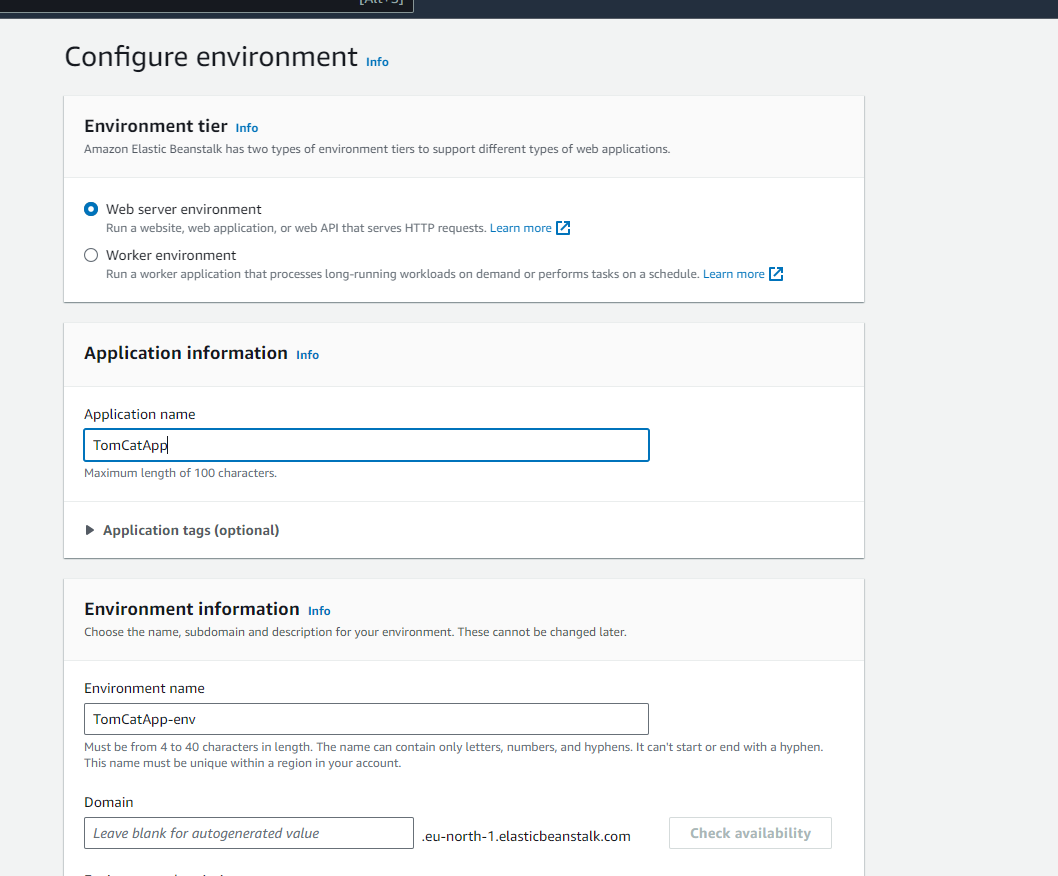
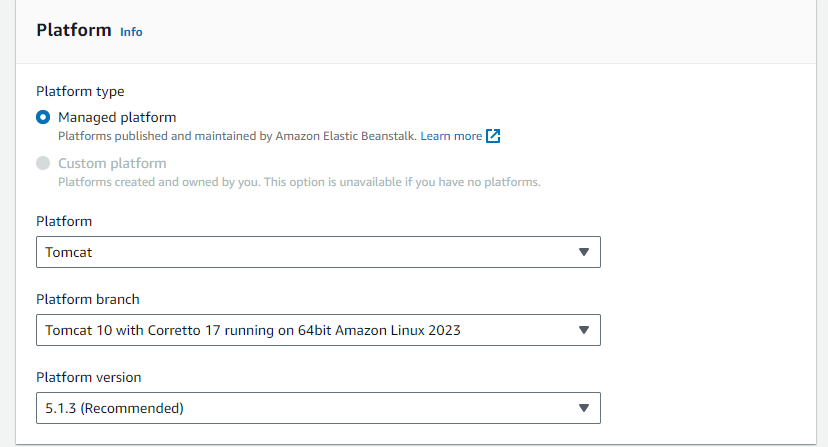
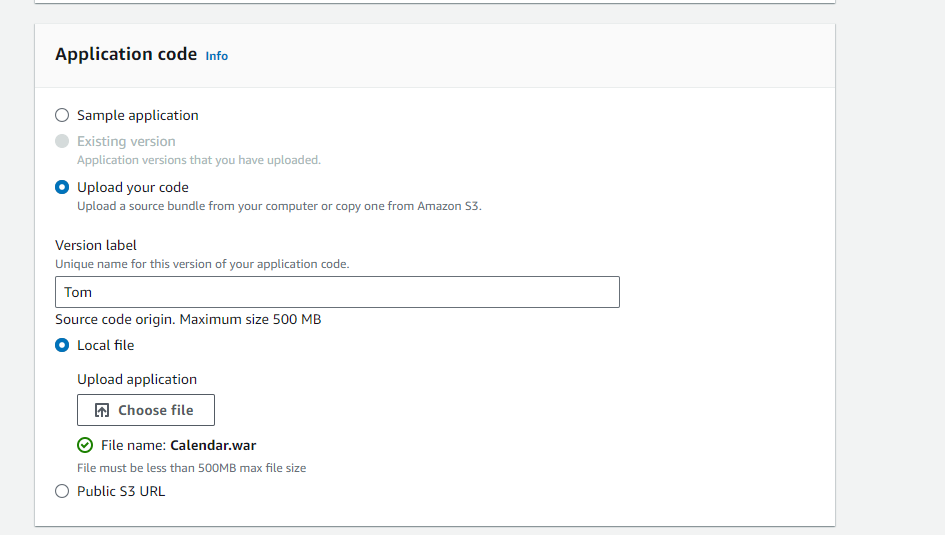
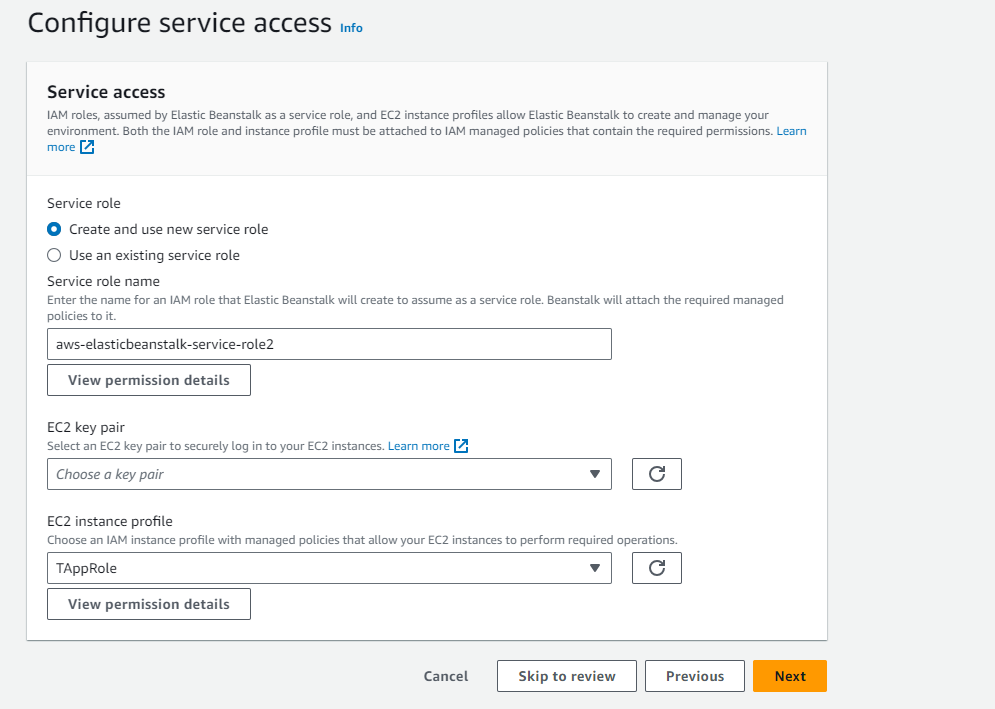
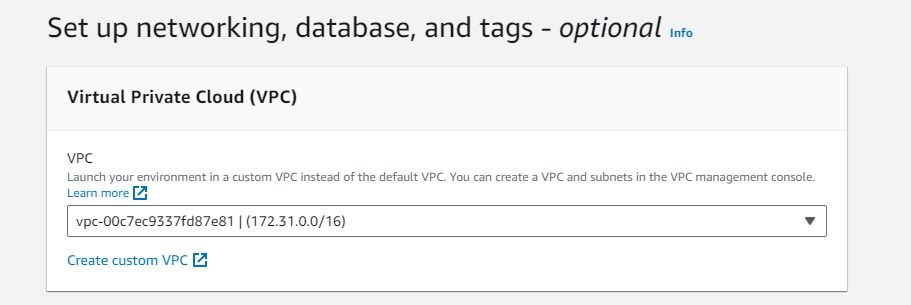
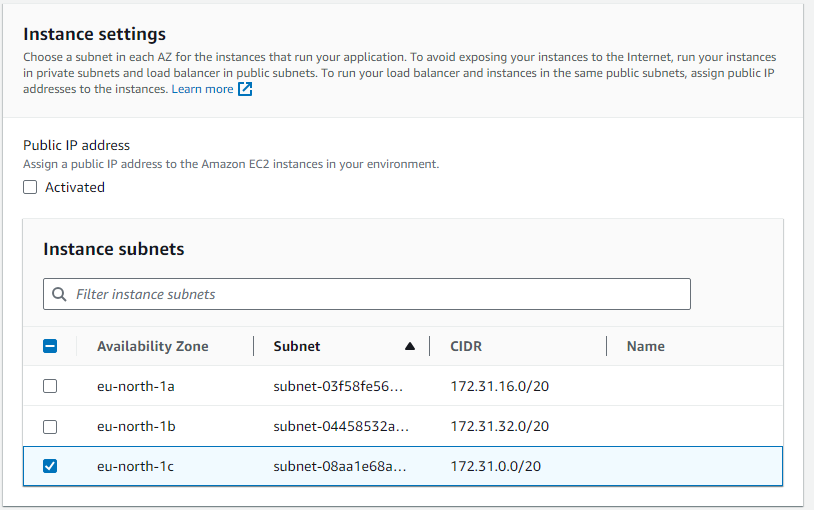
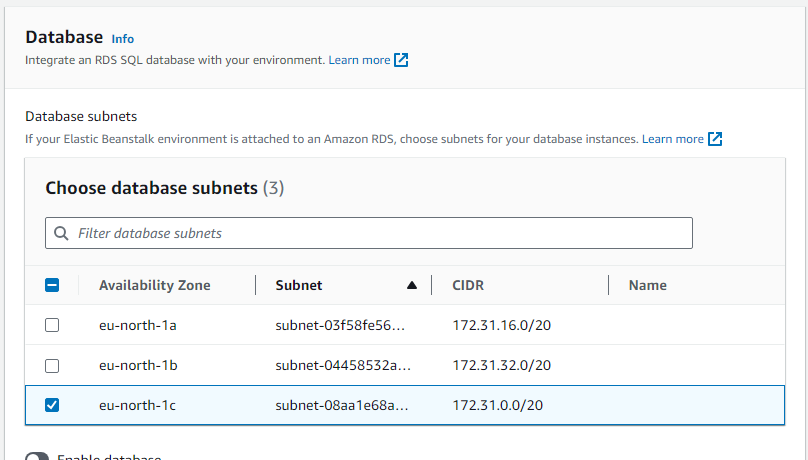
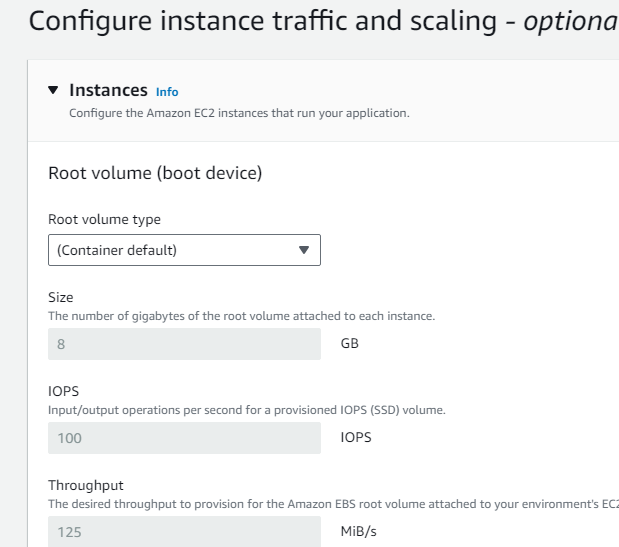
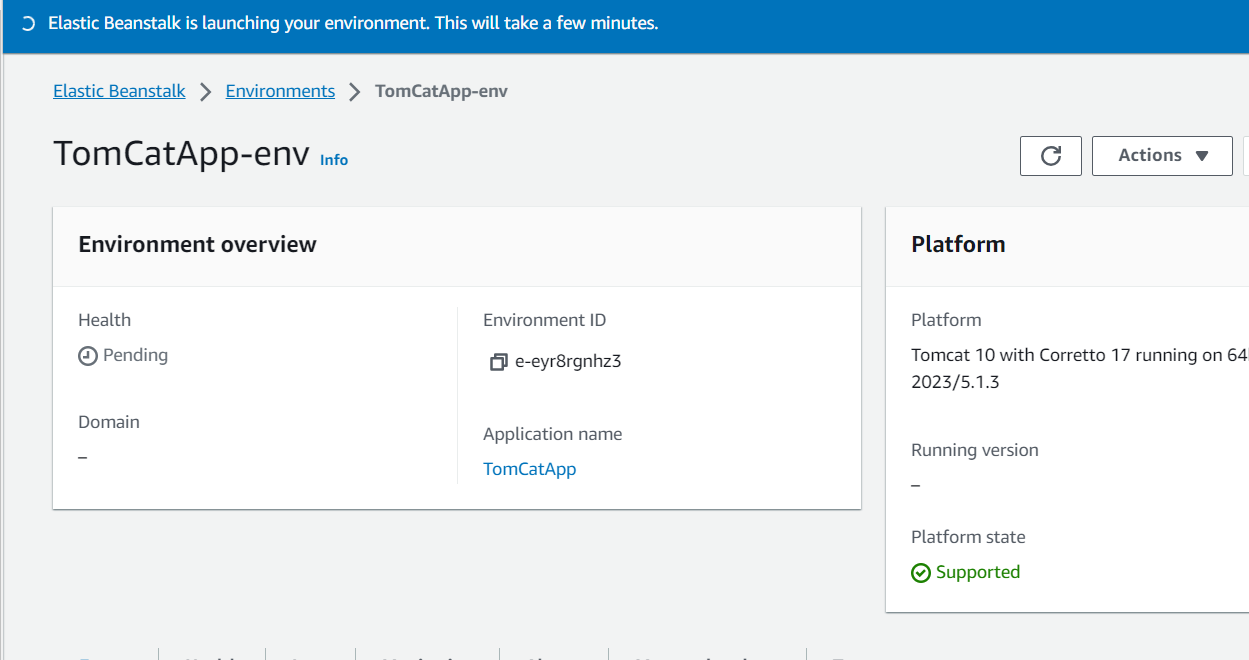
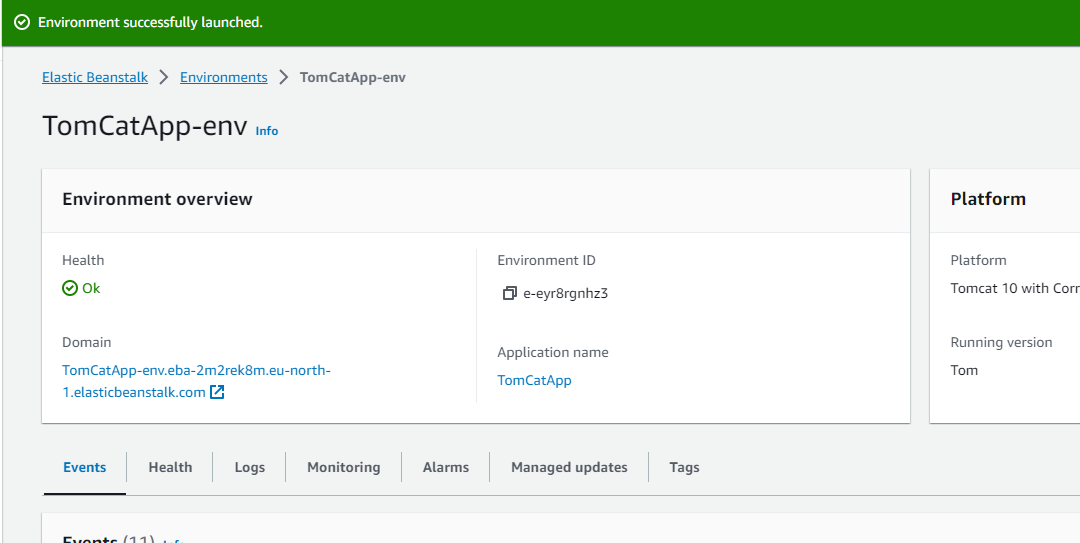
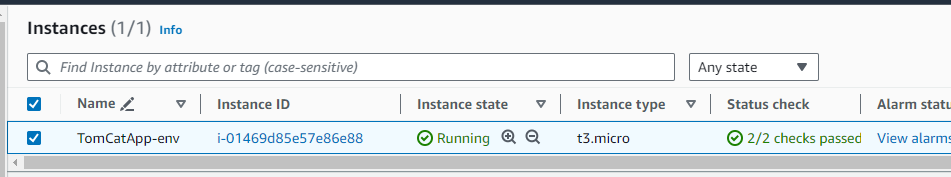
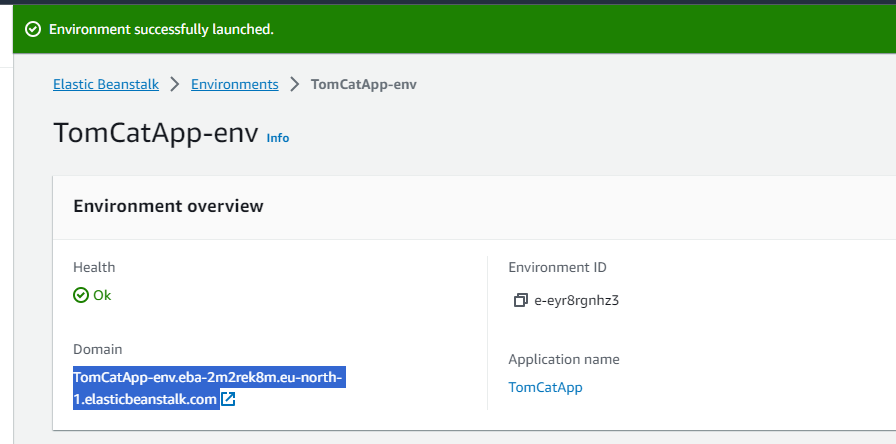
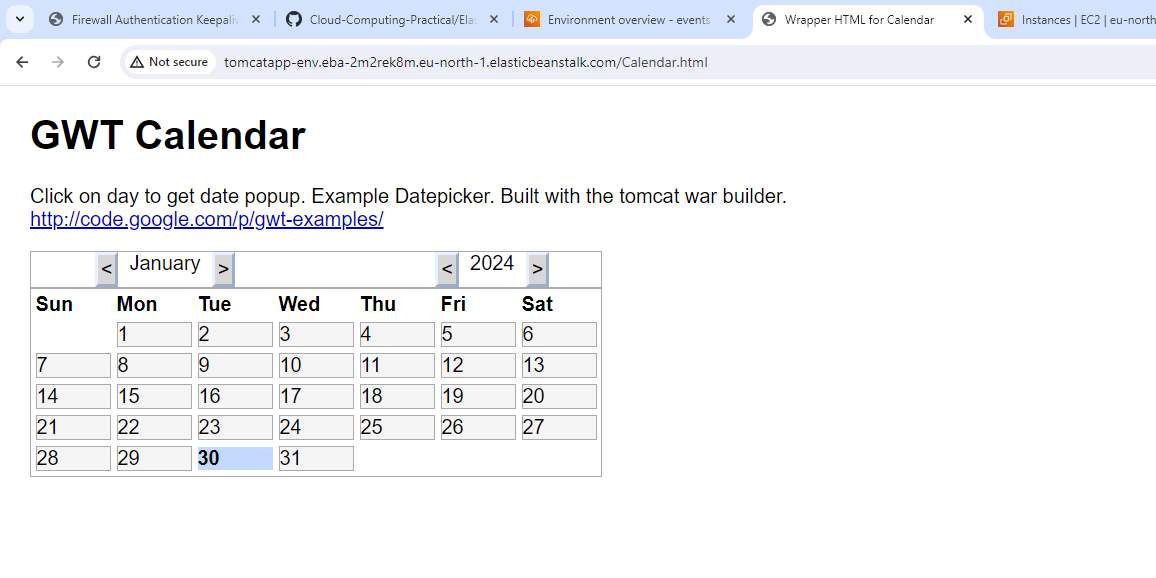
**Steps : For Python**

1. After Logging into AWS, Go to Services -> Elastic beanstalk  
   
2. Click on Create Application.
3. Steps are given:  
   
4. Fill in Application name in Configure env:  
     
   
5. Select the platform: Python  
   
6. Rest - Default:  
   
7. Next
8. Create New Role in IAM:  
     
   1. Go to IAM (In Services)  
     
   2. Select Roles:   
     
   3. IAM > Roles > Create Role  
   4. Entity- AWS and Use Case- EC2  
     
   5. Permission policies to be selected:  
   AWSElasticBeanstalkMulticontainerDocker  
   AWSElasticBeanstalkWebTier  
   AWSElasticBeanstalkWorkerTier  
     
     
     
     
     
     
   6.   
     
     
     
   7. Tap on Create Role  
   8. New Role is created  
   
9. In Configure Access Service > EC2 instance profile > Use an existing Service role> Click on Refresh > Select The new role (TAppRole)  
   
10. Click on Next
11. Select a network  
    
12. Select the same IP address as you selected in step 11 for Instance subnets and database  
      
    
13. Click on Next
14. For Configure Instance and Scaling everything will be default  
    
15. For Configure updates also everything will be default
16. It will Show Review , Click on Submit
17. Now you will see it is getting launched
18. The web app is successfully launched!  
    
19. Now Go to Instances in Services > EC2 , click on refresh and you will see the instance is running  
    
20. Go to Elastic beanstalk window and click on domain  
    
21. Now , Elastic beanstalk server is running on your own dedicated environment  
    
22. Terminate the Instance.  
    

**Steps : For Java**

1. Login into your AWS account
2. Go to Services -> Elastic beanstalk  
   
3. Click on Create Application.
4. Steps are given:  
   
5. Fill in Application name in Configure env:  
     
   
6. Select the platform: Java  
   
7. Rest - Default:  
   
8. Next
9. Create New Role in IAM:  
     
   1. Go to IAM (In Services)  
     
   2. Select Roles:   
     
   3. IAM > Roles > Create Role  
   4. Entity- AWS and Use Case- EC2  
     
     
   5. Permission policies to be selected:  
   AWSElasticBeanstalkMulticontainerDocker  
   AWSElasticBeanstalkWebTier  
   AWSElasticBeanstalkWorkerTier  
     
     
     
     
     
     
   6.   
     
     
     
   7. Tap on Create Role  
     
   8. New Role is created  
     
   
10. In Configure Access Service > EC2 instance profile > Use an existing Service role> Click on Refresh > Select The new role (JRole)  
    
11. Click on Next
12. Select a network  
    
13. Select the same IP address as you selected in step 11 for Instance subnets and database  
      
    
14. Click on Next
15. For Configure Instance and Scaling everything will be default  
    
16. For Configure updates also everything will be default
17. It will Show Review , Click on Submit
18. Now you will see it is getting launched  
    
19. The web app was successfully launched!  
    
20. Now Go to Instances in Services > EC2 , click on refresh and you will see the instance is running  
    
21. Go to Elastic beanstalk window and click on domain  
    
22. Now , Elastic beanstalk server is running on your own dedicated environment  
    
23. Terminate the Instance.

**Steps : For TomCat**

1. Login into your AWS account
2. Go to Services -> Elastic beanstalk  
   
3. Click on Create Application.
4. Steps are given:  
   
5. Fill in Application name in Configure env:  
   
6. Select the platform: Tomcat  
   
7. Download an Application code: Calendar.war from github and upload it, Label it as Tom  
   
8. Create and Use new service role and just rename Service role name as “aws-elasticbeanstalk-service-role2 “ , For Instance profile use any previous profile or create new. Here we have used a previous profile.  
   
9. Click on Next
10. Select a network  
    
11. Select the same IP address as you selected in step 11 for Instance subnets and database  
      
    
12. Click on Next
13. For Configure Instance and Scaling everything will be default  
    
14. For Configure updates also everything will be default
15. It will Show Review , Click on Submit
16. Now you will see it is getting launched  
    
17. The web app was successfully launched!  
    
18. Now Go to Instances in Services > EC2 , click on refresh and you will see the instance is running  
    
19. Go to Elastic beanstalk window and click on domain  
    
20. Now , TomCat WebApp is running on your own dedicated environment  
    
21. Terminate the Environment and Instance.