**CLOUD COMPUTING PRACTICAL 02 – PaaS (Platform as a Service)**

**RHEA MENON | A029**

**PaaS**

Platform as a Service (PaaS) provides a runtime environment. It allows programmers to easily create, test, run, and deploy web applications. You can purchase these applications from a cloud service provider on a pay-as-per use basis and access them using the Internet connection. In PaaS, back end scalability is managed by the cloud service provider, so end- users do not need to worry about managing the infrastructure.

PaaS includes infrastructure (servers, storage, and networking) and platform (middleware, development tools, database management systems, business intelligence, and more) to support the web application life cycle.

Example: Google App Engine, Force.com, Joyent, Azure.

1. Programming languages: PaaS providers provide various programming languages for the developers to develop the applications. Some popular programming languages provided by PaaS providers are Java, PHP, Ruby, Perl, and Go.

2. Application frameworks: PaaS providers provide application frameworks to easily understand the application development. Some popular application frameworks provided by PaaS providers are Node.js, Drupal, Joomla, WordPress, Spring, Play, Rack, and Zend.

3. Databases: PaaS providers provide various databases such as ClearDB, PostgreSQL, MongoDB, and Redis to communicate with the applications.

4. Other tools: PaaS providers provide various other tools that are required to develop, test, and deploy the applications.

**Elastic Beanstalk**

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.

Elastic Beanstalk is a fully managed service provided by AWS that makes it easy to deploy and manage applications in the cloud without worrying about the underlying infrastructure. First, create an application and select an environment, configure the environment, and deploy the application.

**AWS Elastic Beanstalk Components**

* **Application:**Elastic Beanstalk directly takes in our project code. So Elastic Beanstalk application is named the same as your project home directory.
* **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.
* **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. For example, in the case of web applications AWS will regularly, as scheduled by the developers, ping the application to check if the response is status code 200 and if the application is running as expected. Health check responses:
  + **Red:** The application failed all health tests.
  + **Yellow:** The application failed some of the health tests.
  + **Grey:**The application is updating.
  + **Green:**The application passed the health check successfully.
* **Isolated:**All environments within a single application are isolated from each other (independent of each others’ running states). Needless to say, two different applications are also isolated.
* **Scalability:**Using Auto-Scaling within Elastic Beanstalk makes the application dynamically scalable.
* **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.
* **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.
* **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.
* **Automatic Provisioning:**Elastic Beanstalk takes away the burden of choosing the right services and configuring their security groups to work together.
* **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**

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. With IAM, you can centrally manage permissions that control which AWS resources users can access. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

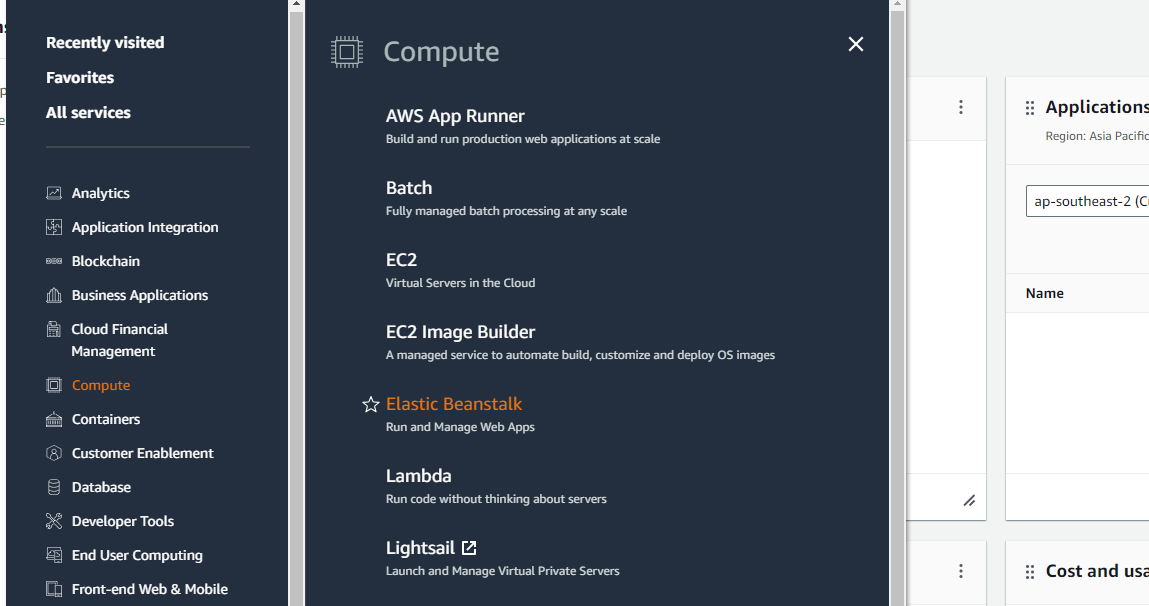
When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account.

Identity management solutions generally perform two tasks:

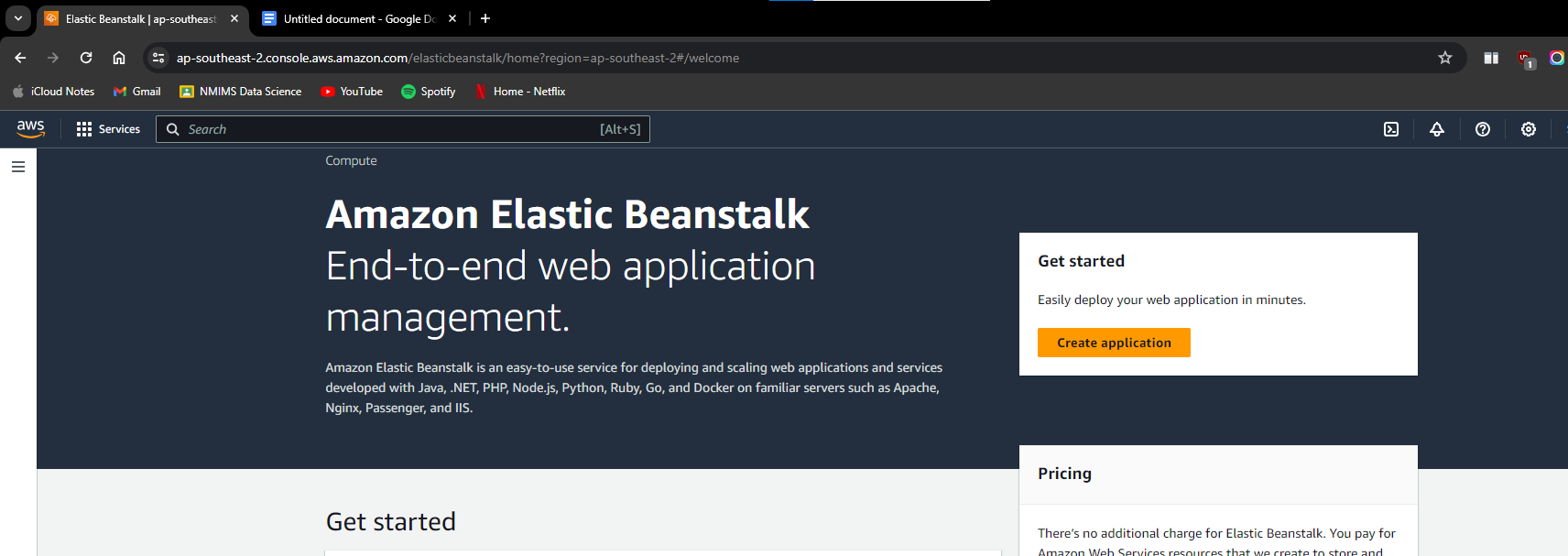
1. IAM confirms that the user, software, or hardware is who they say they are by authenticating their credentials against a database. IAM cloud identity tools are more secure and flexible than traditional username and password solutions.
2. Identity access management systems grant only the appropriate level of access. Instead of a username and password allowing access to an entire software suite, IAM allows for narrow slices of access to be portioned out, i.e. *editor*, *viewer*, and *commenter* in a content management system.

**Creating a Server (Tomcat) Application using Elastic Beanstalk**

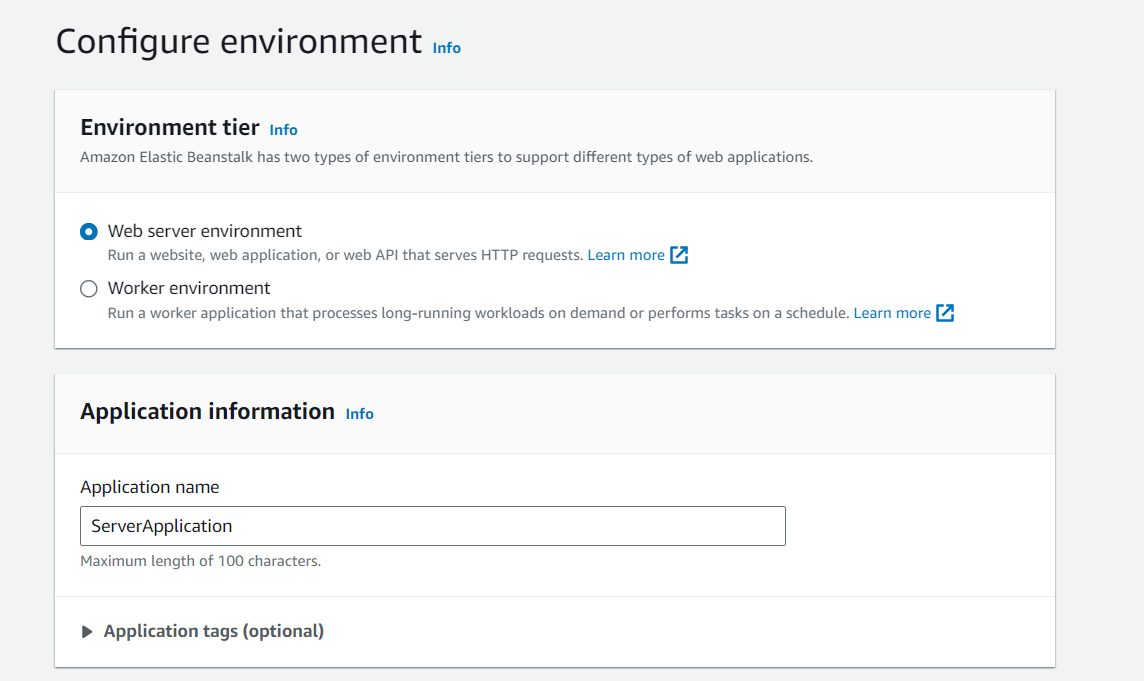
Go to AWS Console Home > Compute > Elastic Beanstalk.

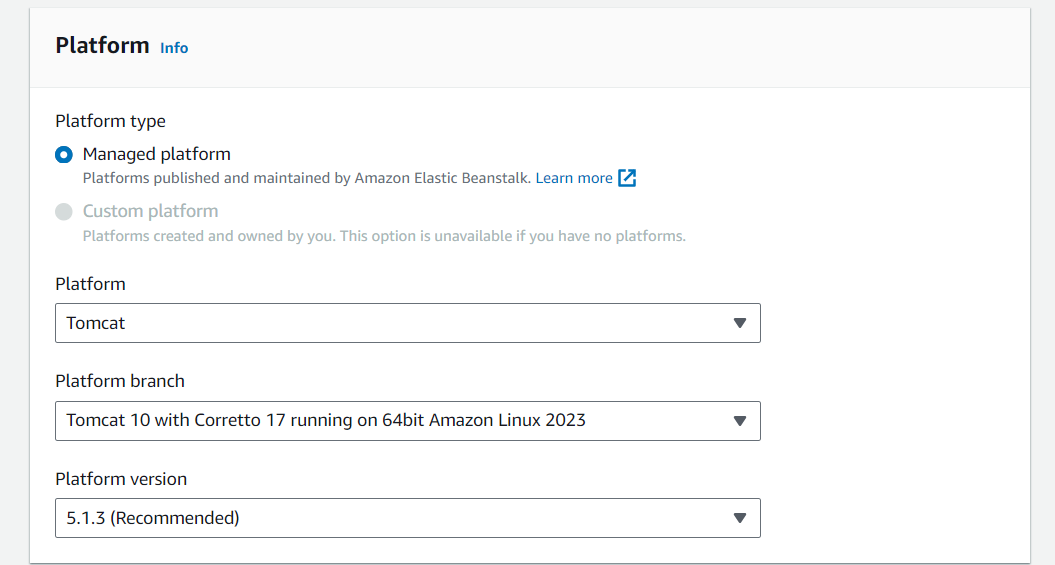


Click on Create application.

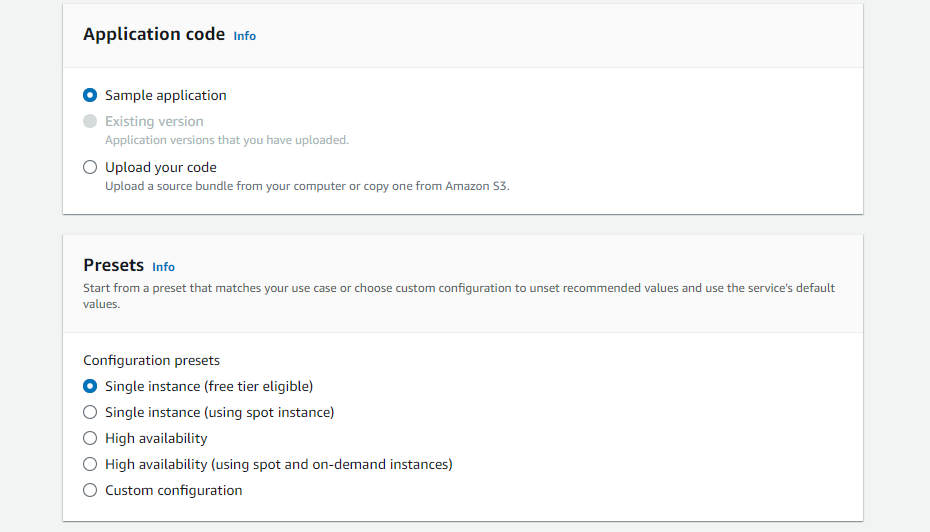


Give a name to the application and change the managed platform to Tomcat.

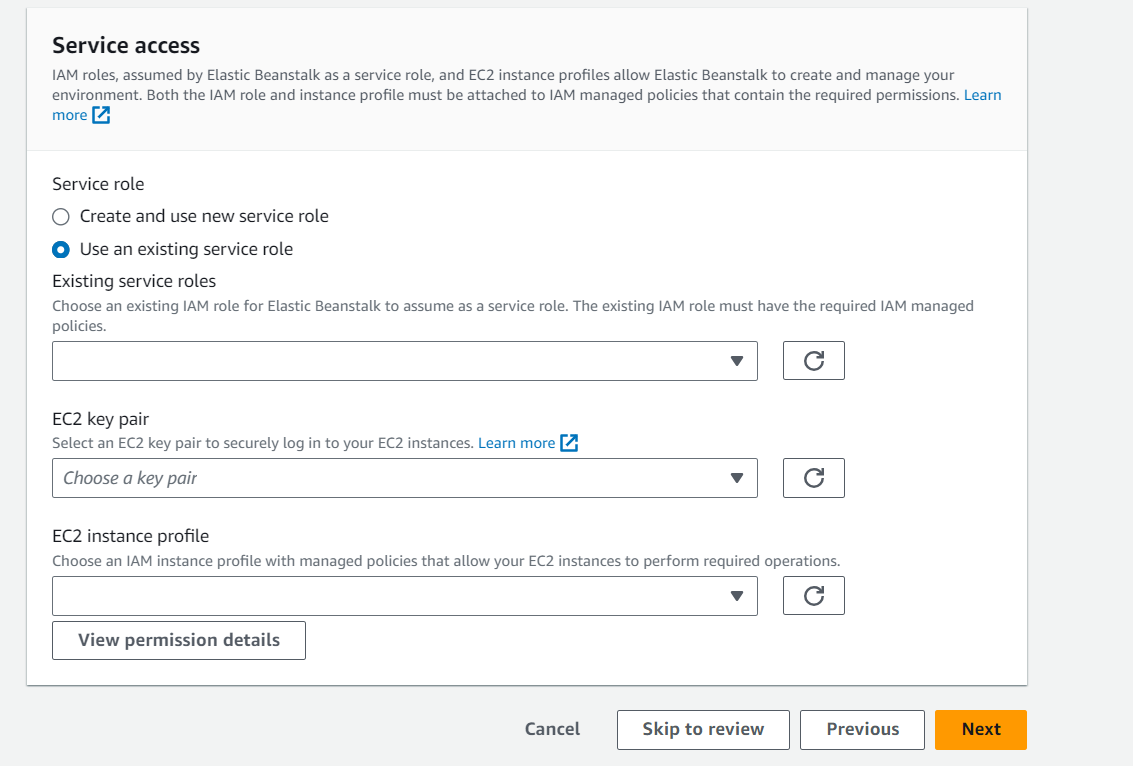




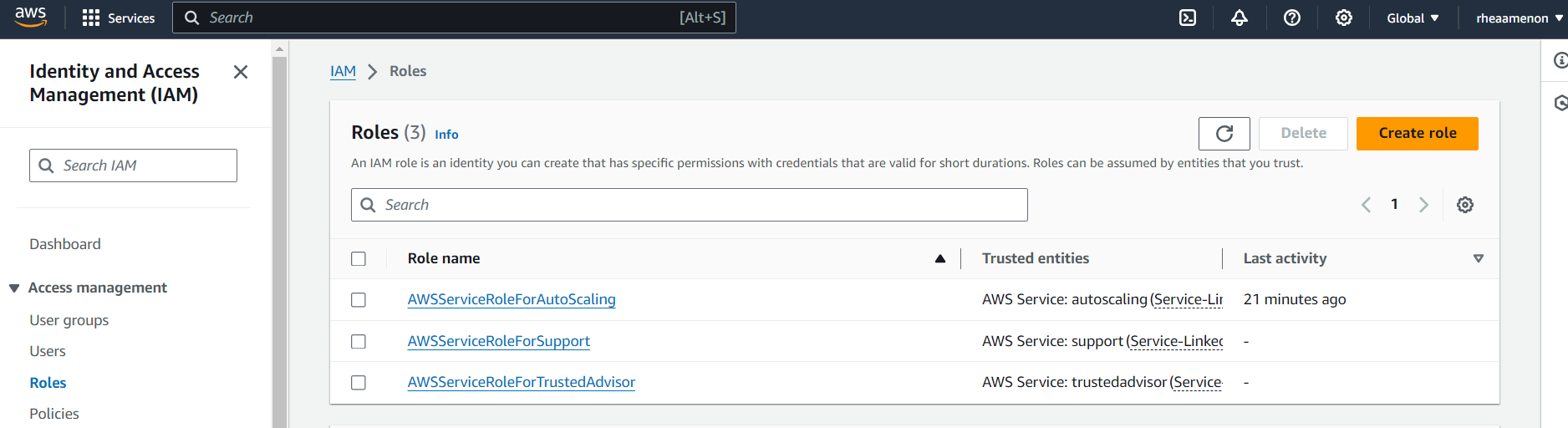
Use a sample application code and click on Next.



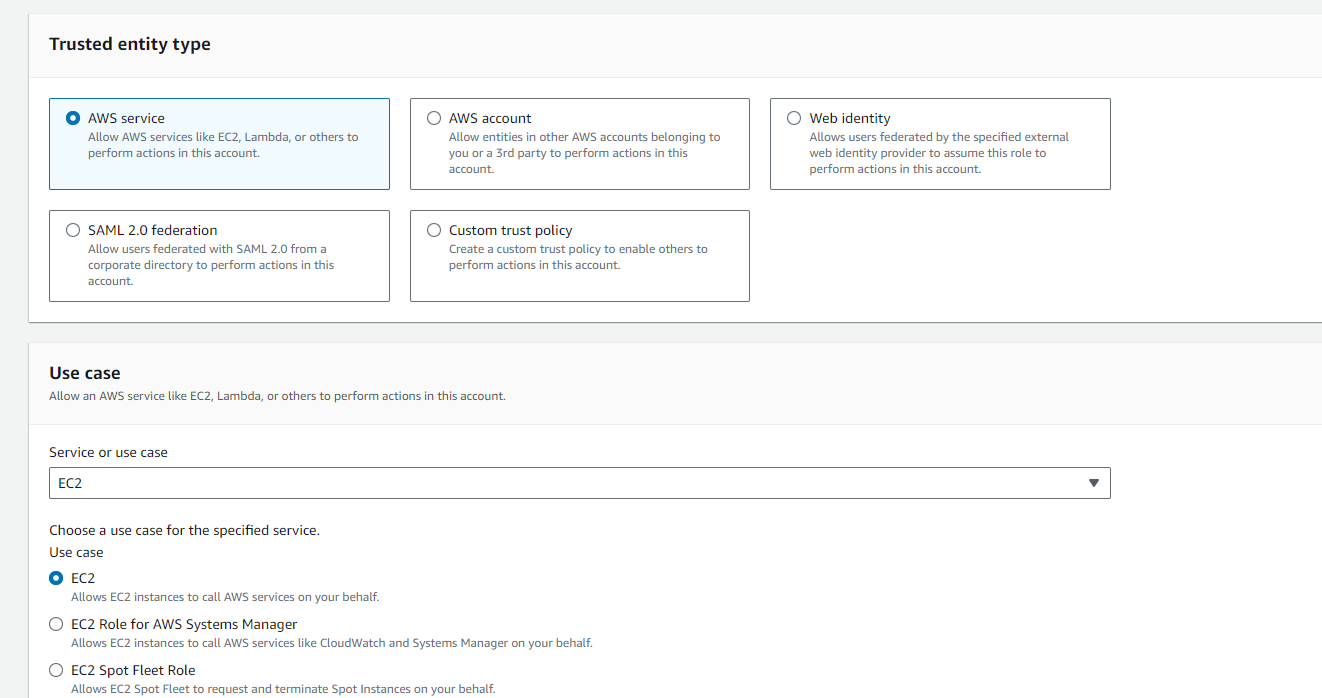
Use an existing service role, and for the service role open IAM on a new tab.



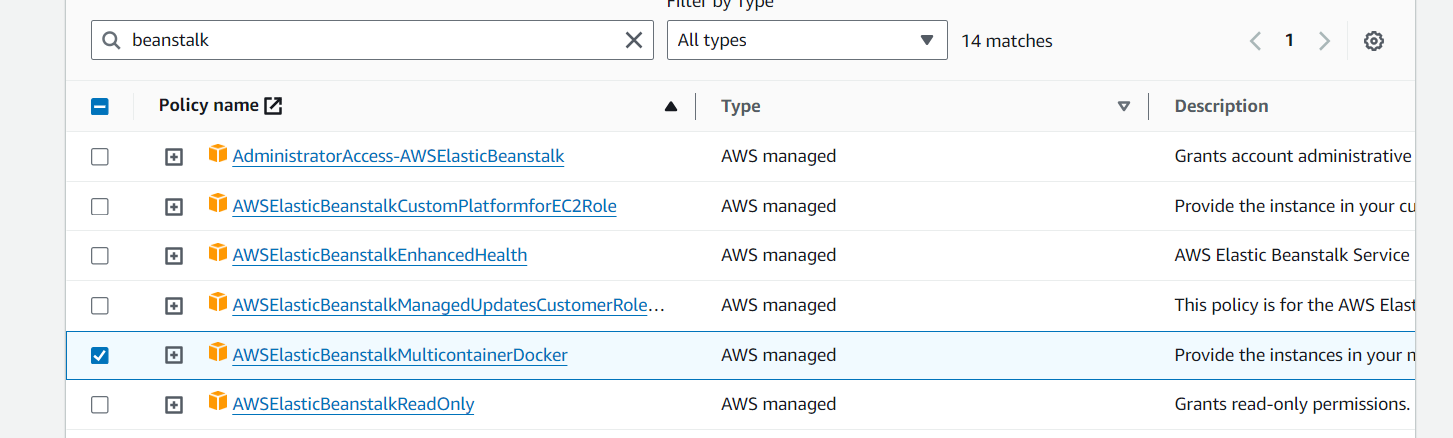
Go to Access management > Roles > Create role.

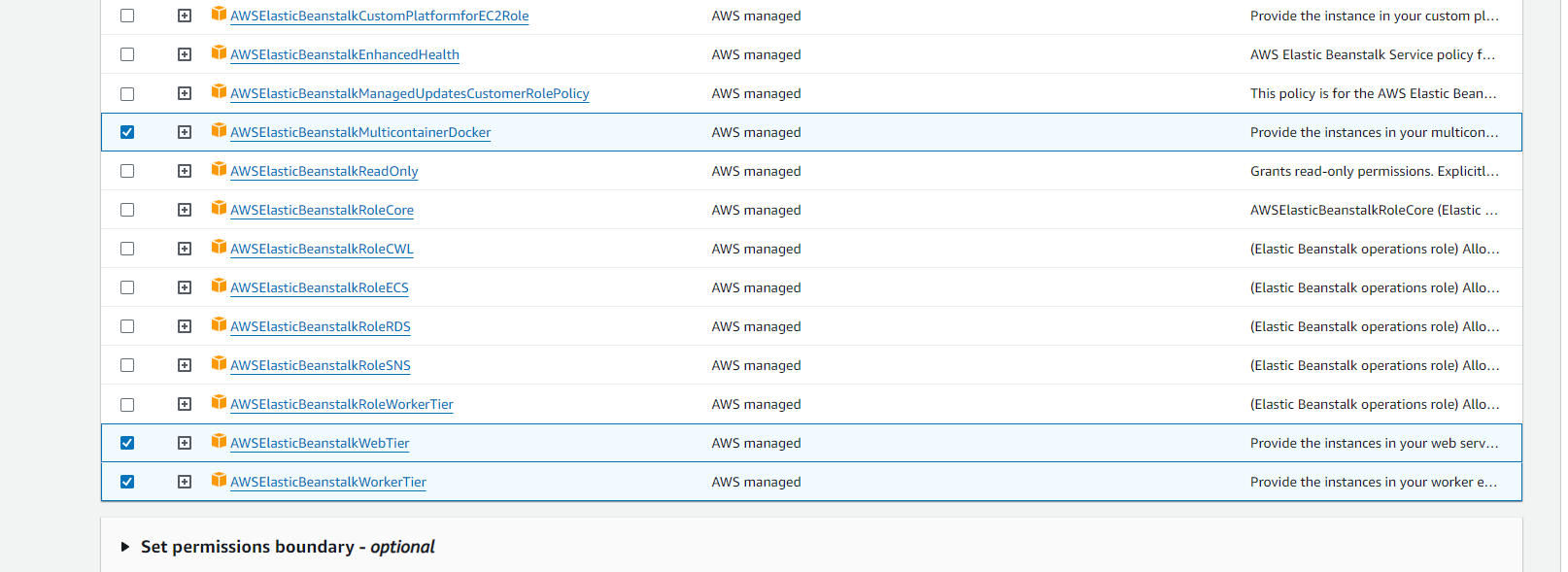


Choose the trusted entity type as AWS Service and the Use case as EC2.

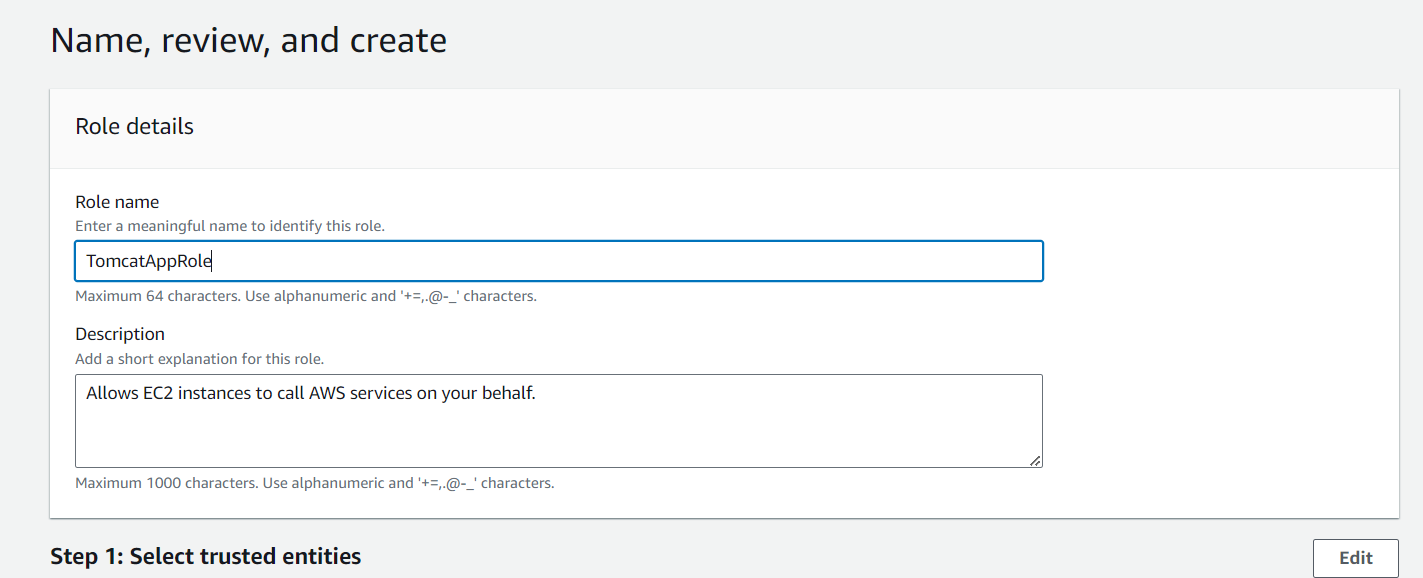


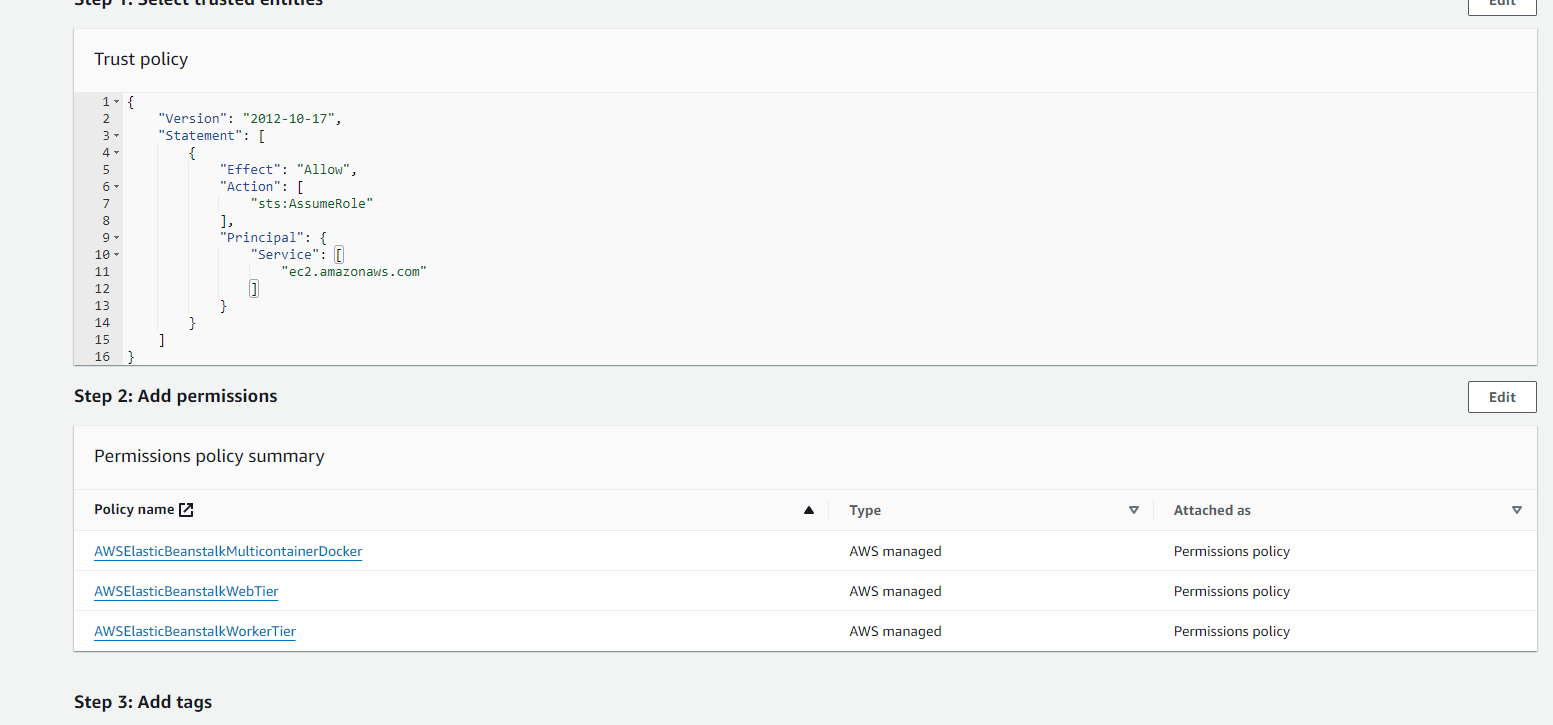
Search for Beanstalk in the search bar and select the following policies.

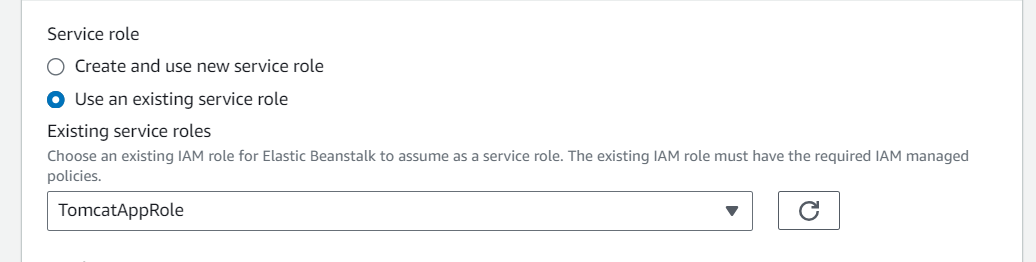




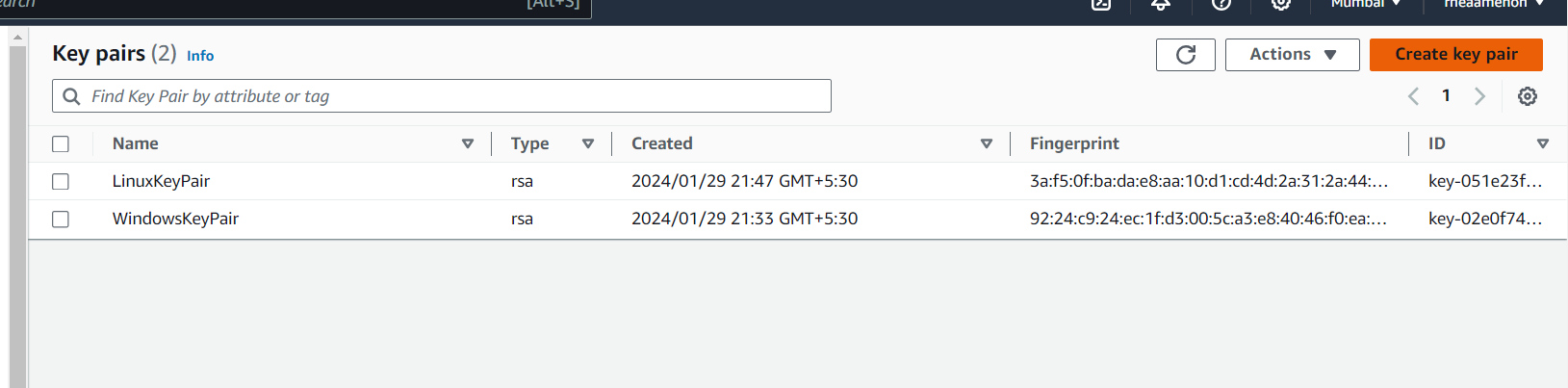
Name the role and create it. When you refresh the roles textbox in the previous tab, the created role will appear.



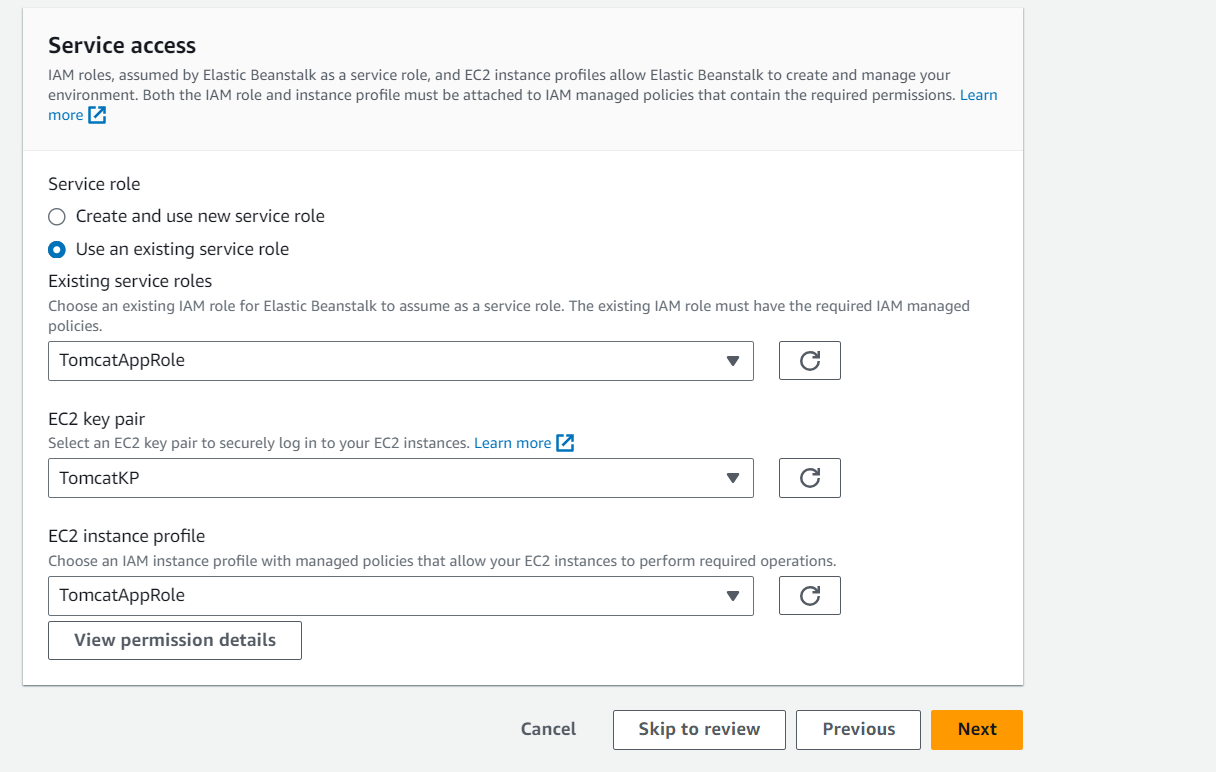




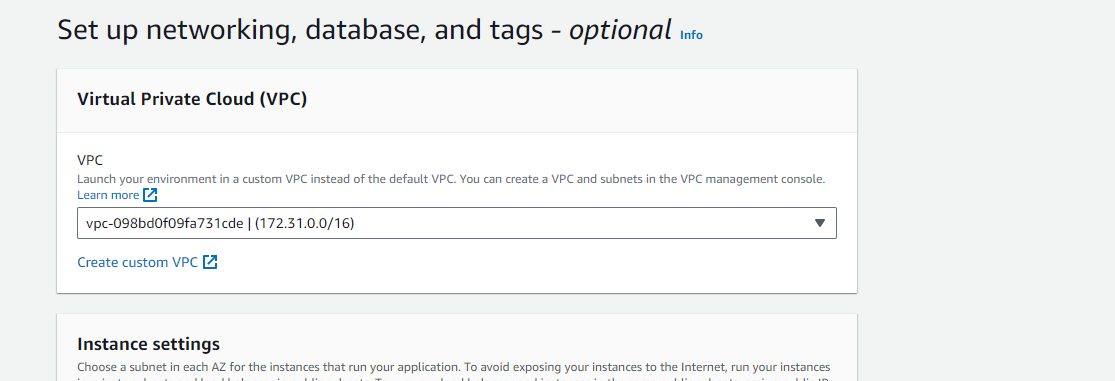
For Key Pairs: Console > All Services > EC2 > Key Pairs > Create key pair and save the .ppk file to the device.



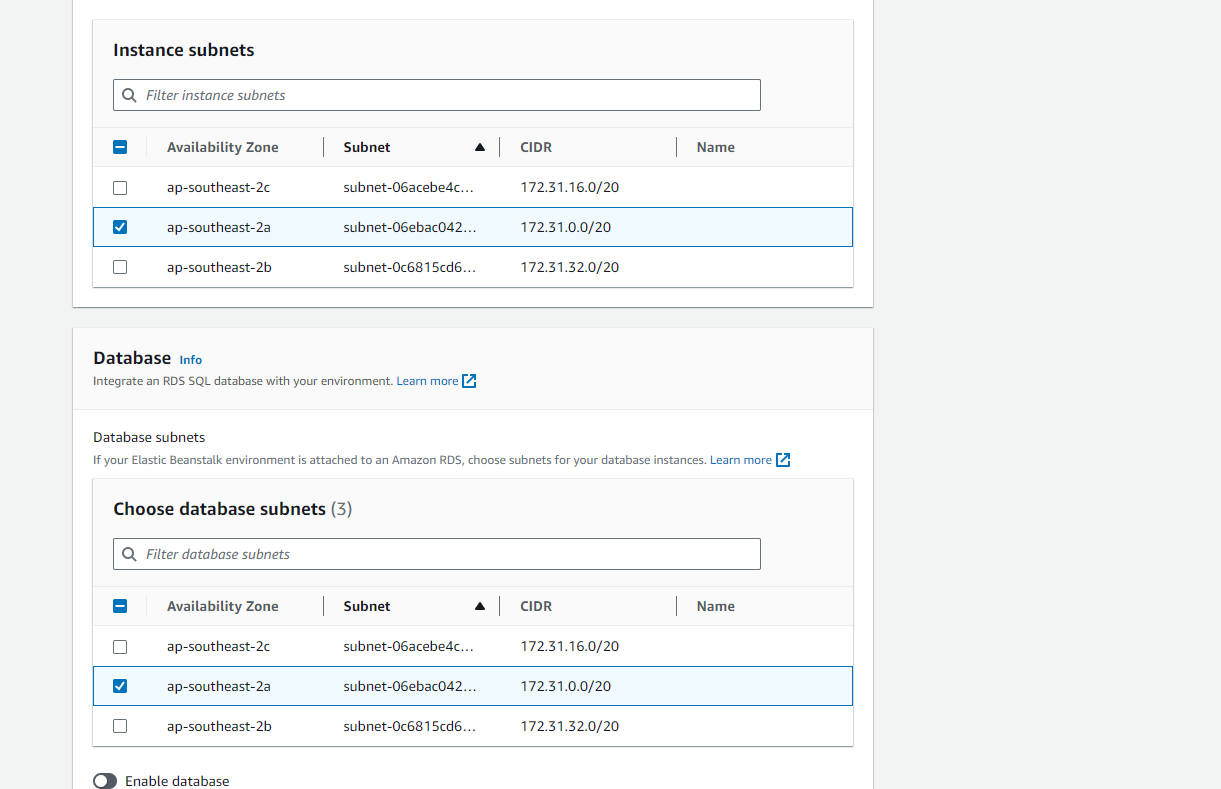
Select the following roles and keys.



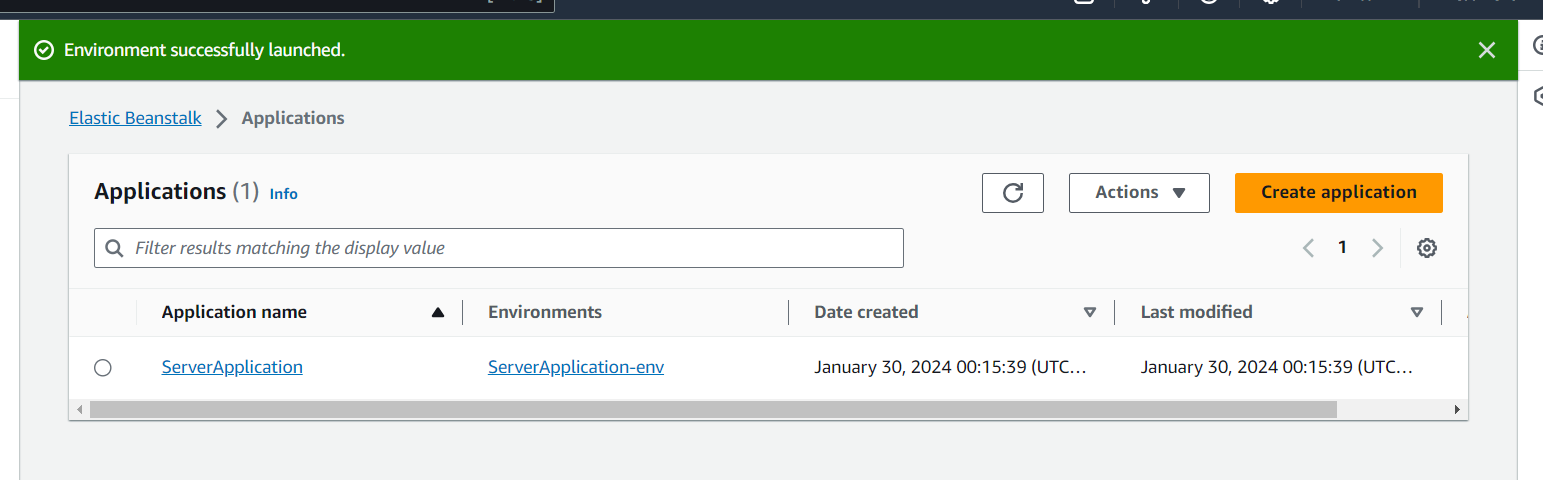
Select the VPC from the dropdown.



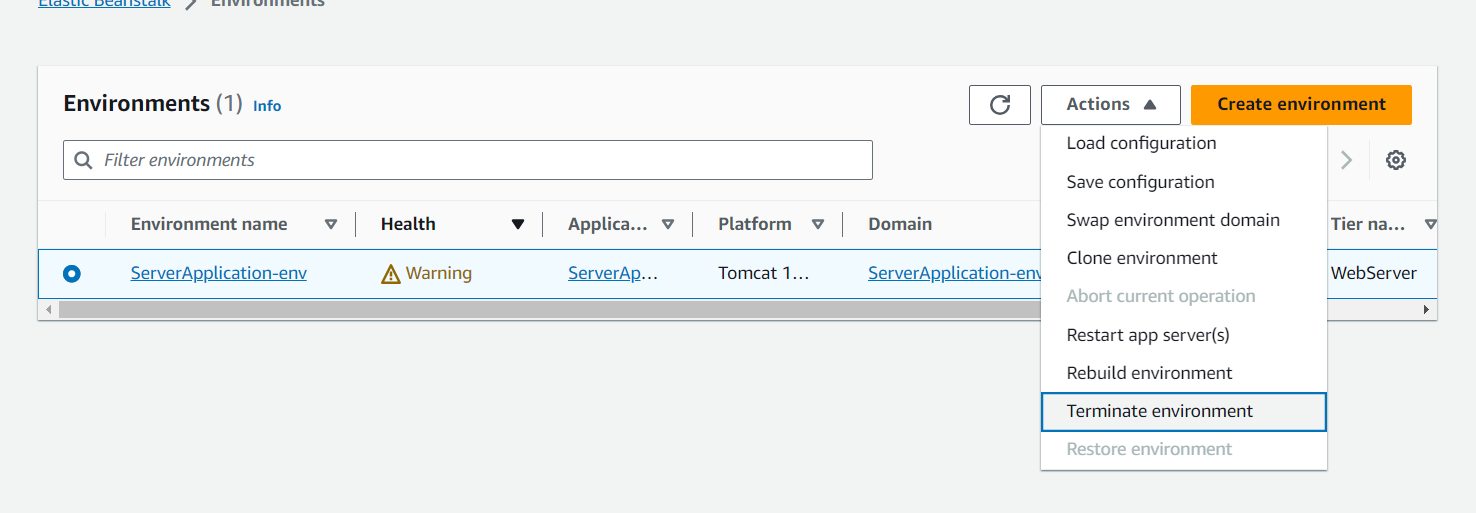
Select the corresponding subnets.



The environment is successfully launched.

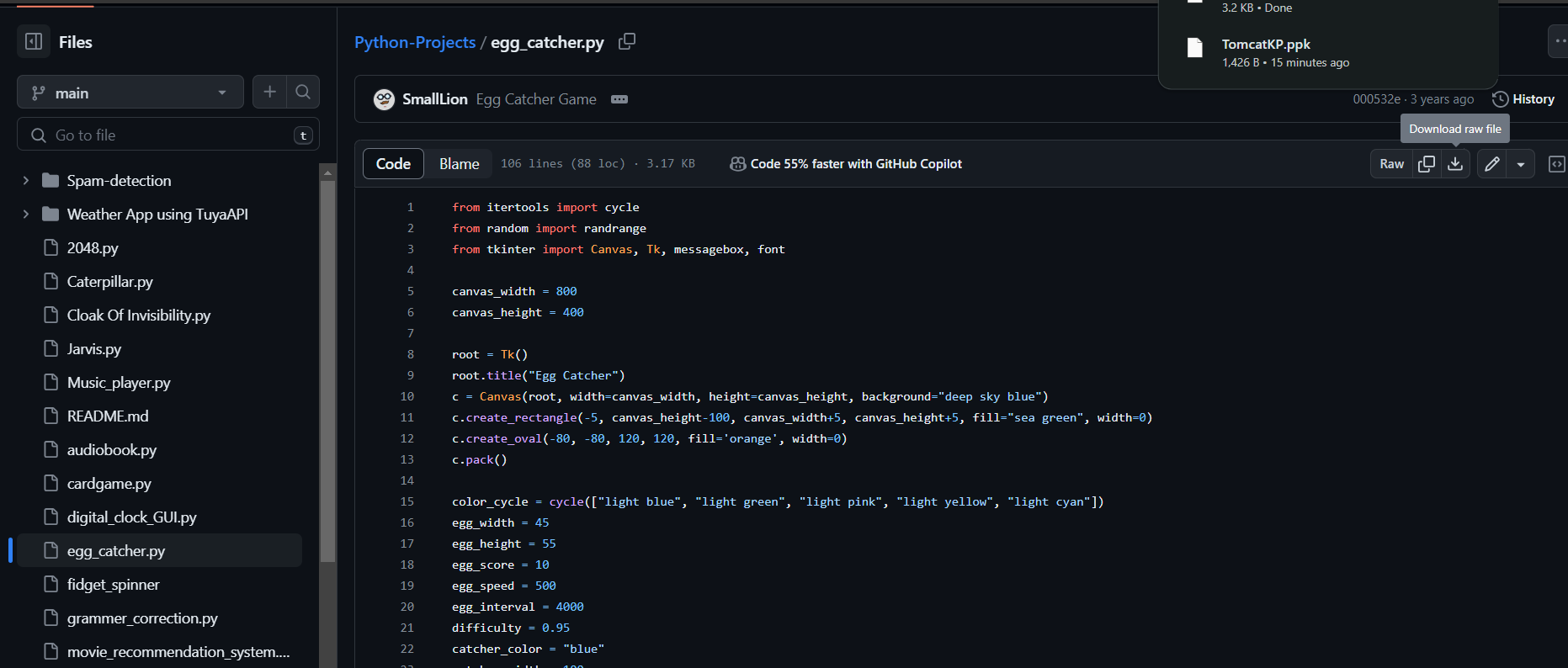


Terminate the environment

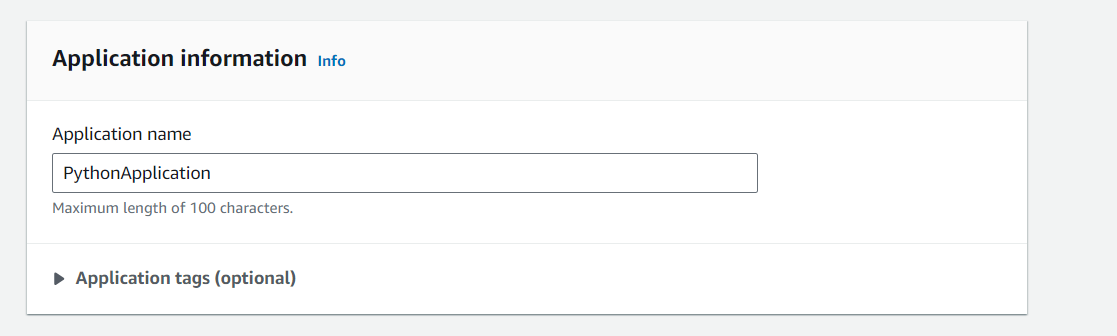


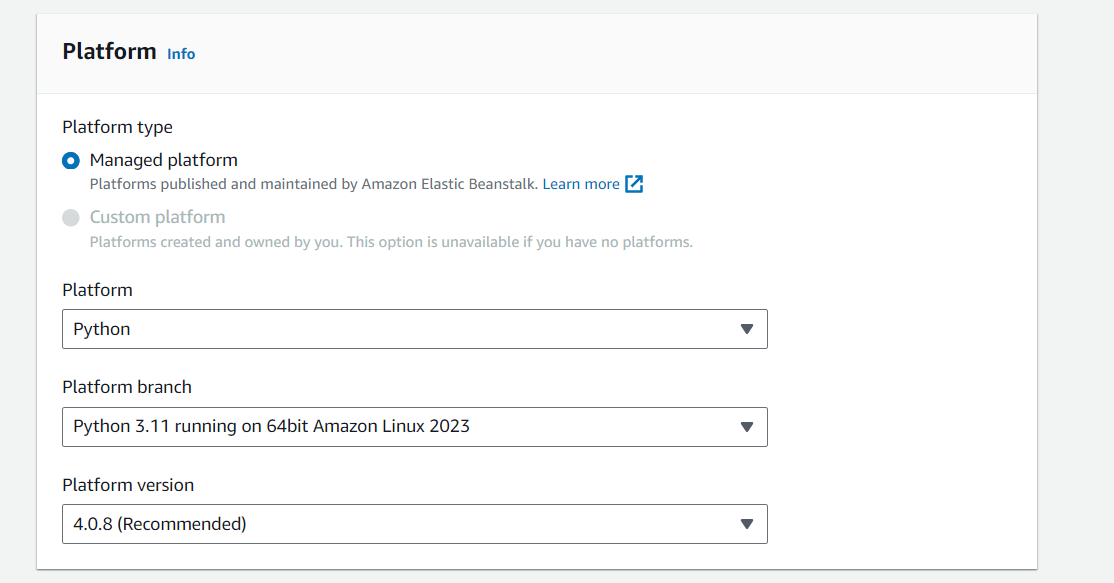
**Creating a Python (External code) Application using Elastic Beanstalk**

Follow the same steps as the previous platform application, but change the setting for the application code. First download a sample .py file.

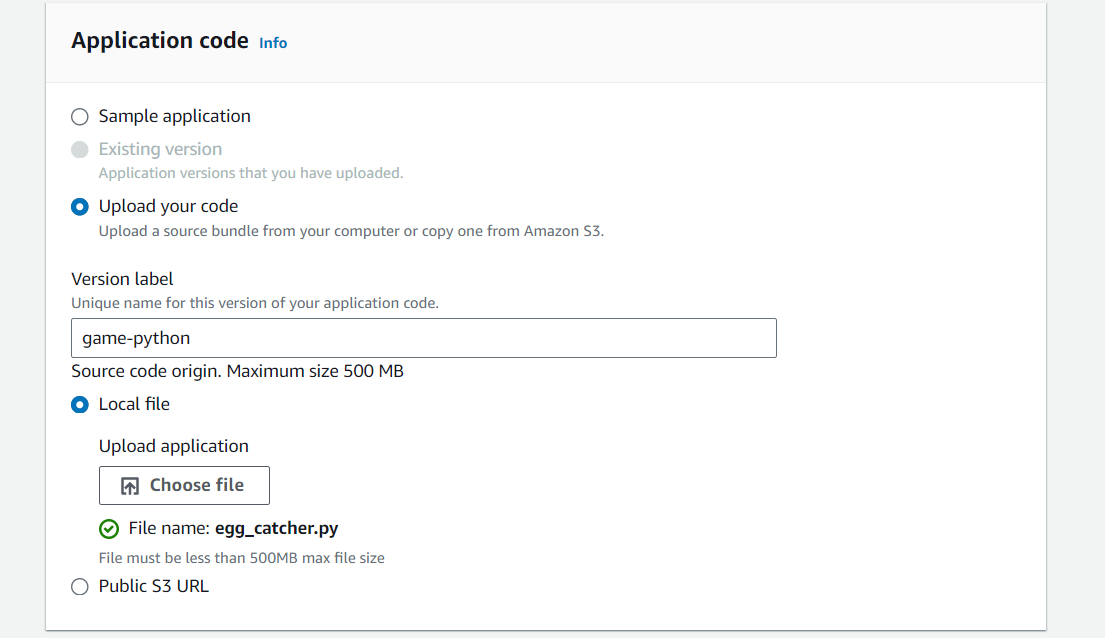


Follow the same steps for creation.

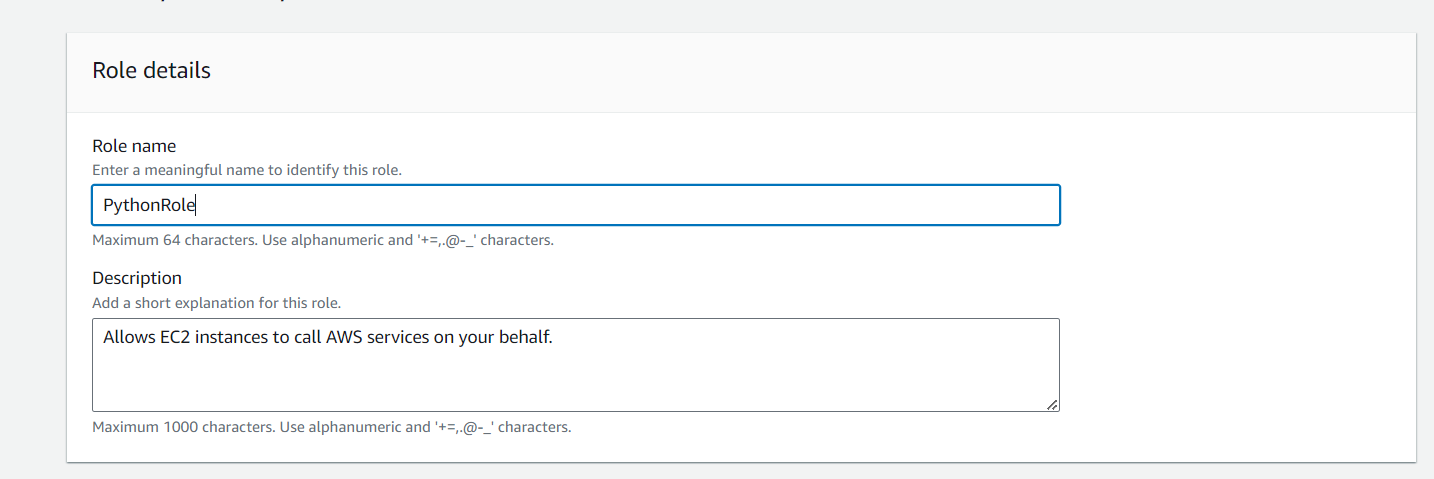




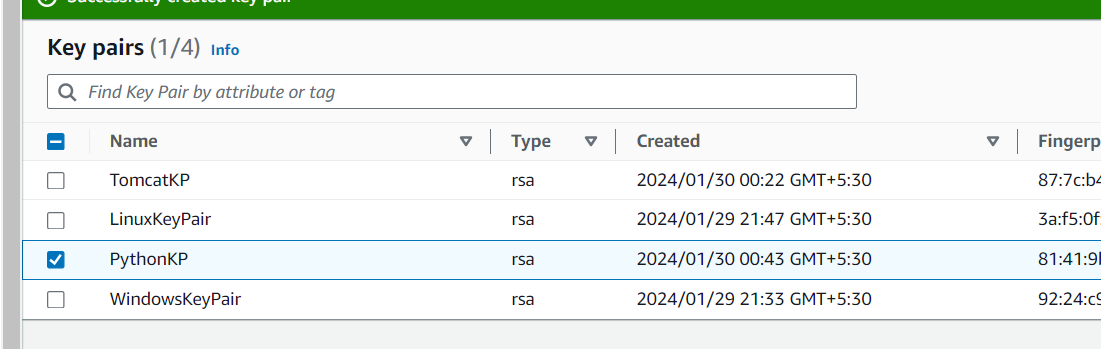
Change the Application code radio button from sample code to ‘Upload your code’ aqnd upload the .py file. Give it a unique version name.

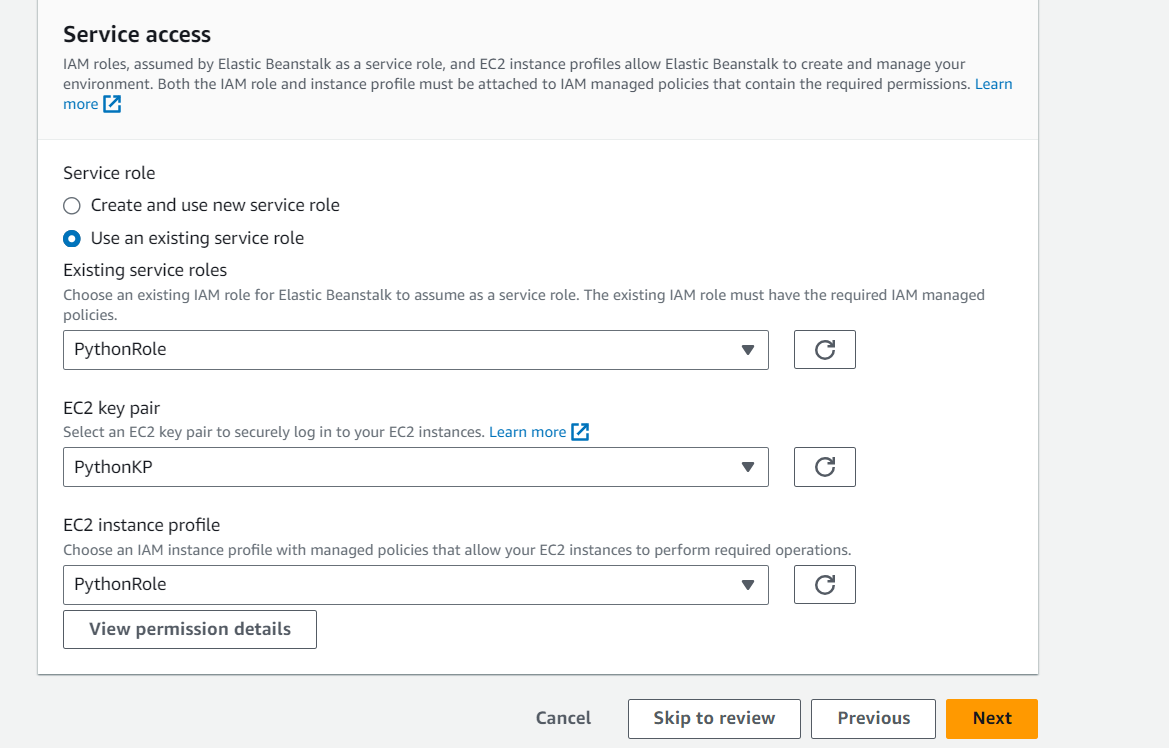


New role for the python application.

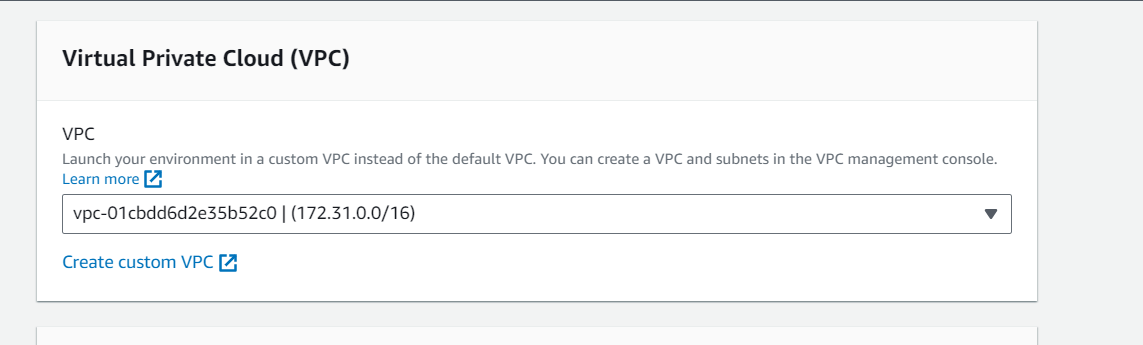


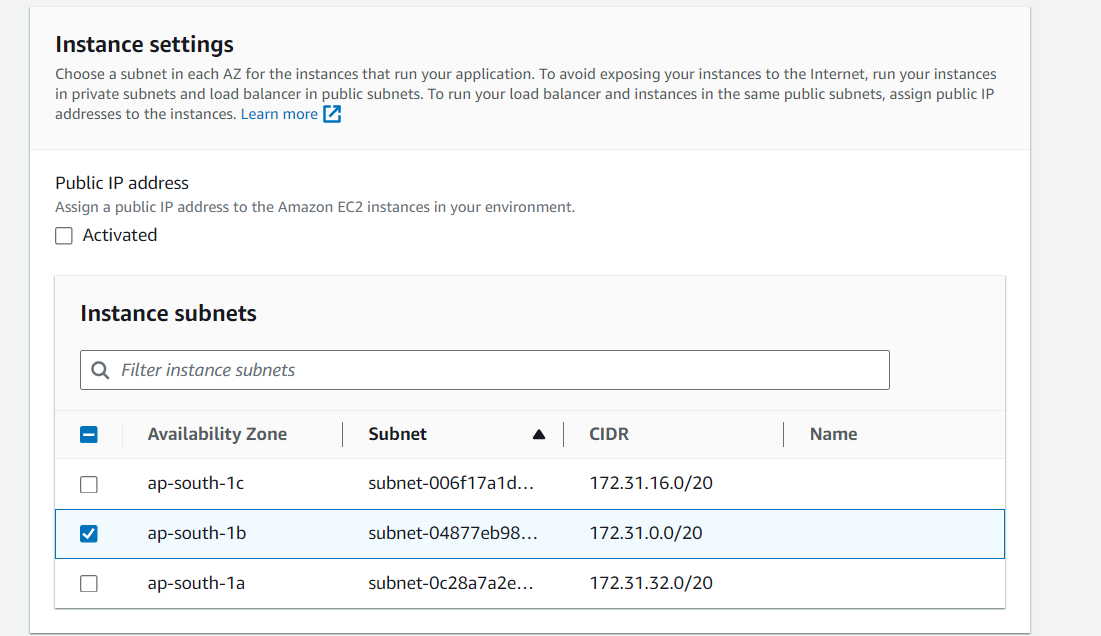
New Key pair

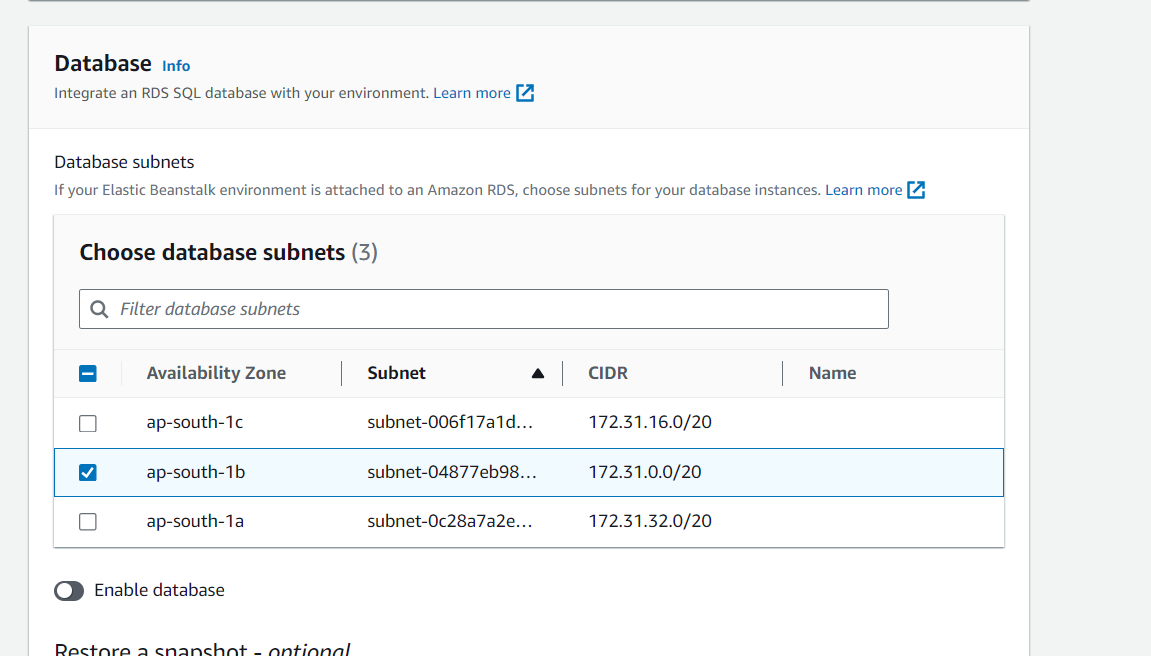


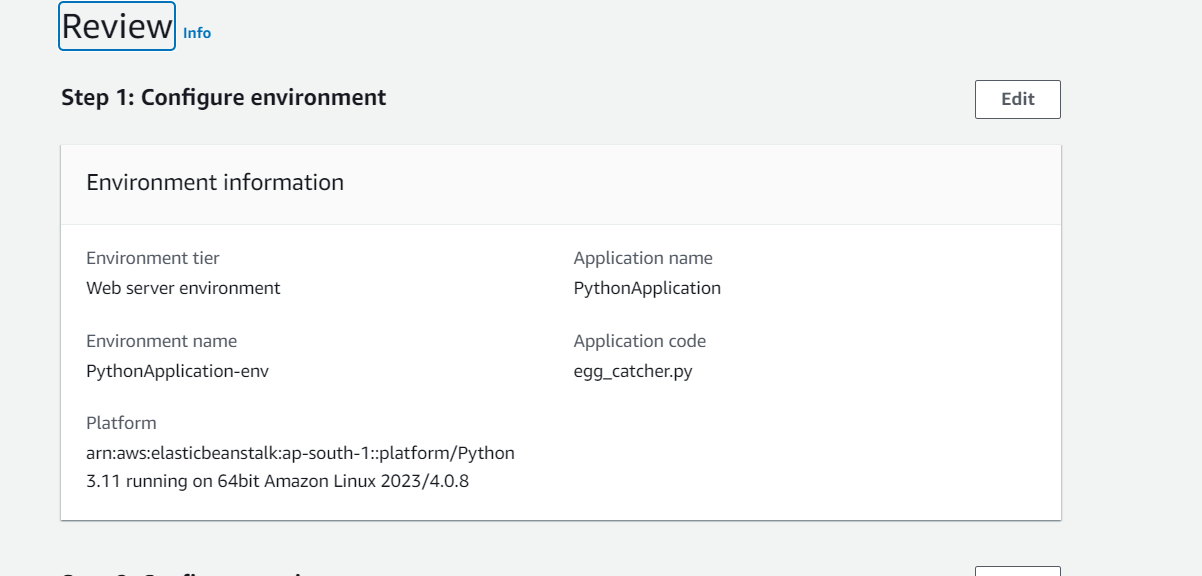


VPC Settings same as Server Application

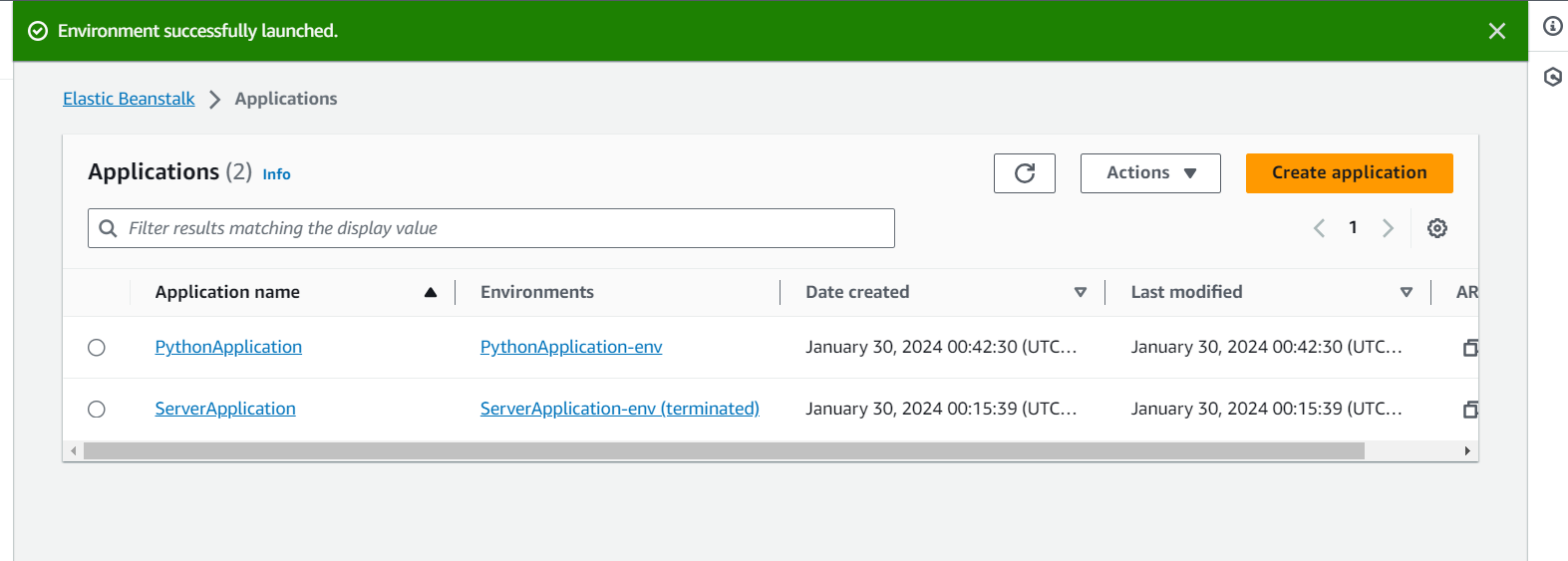






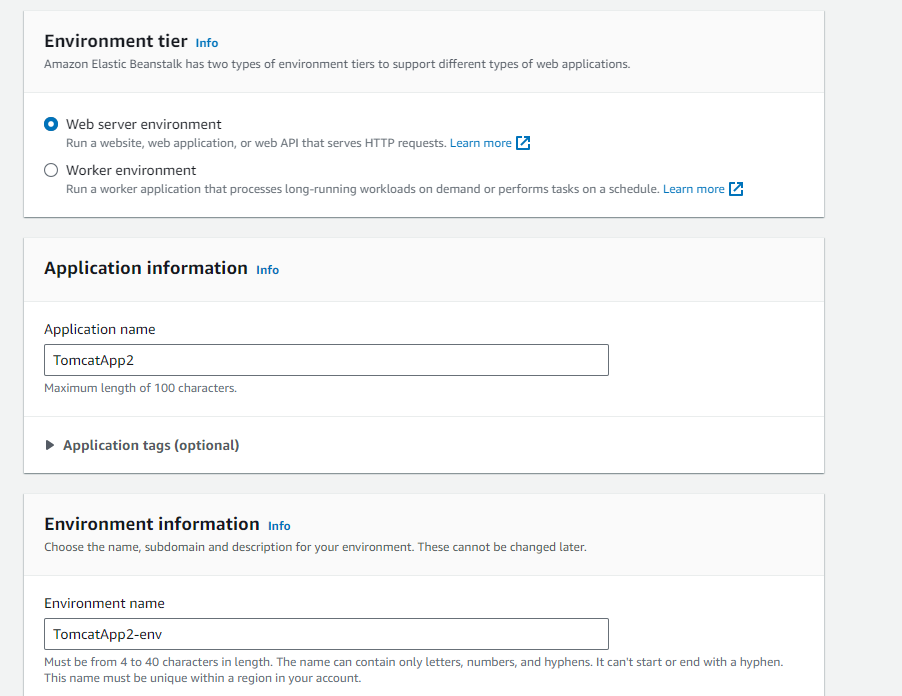


The environment is created.

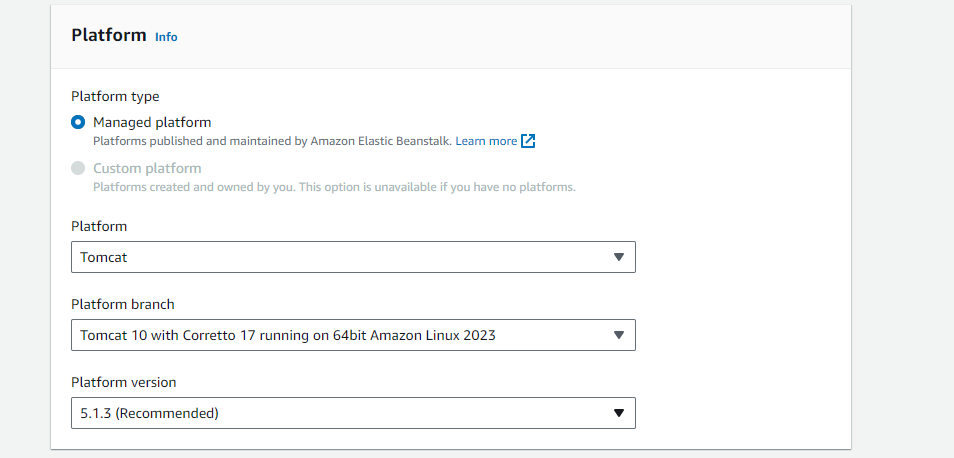


**Configuring a Tomcat Application**

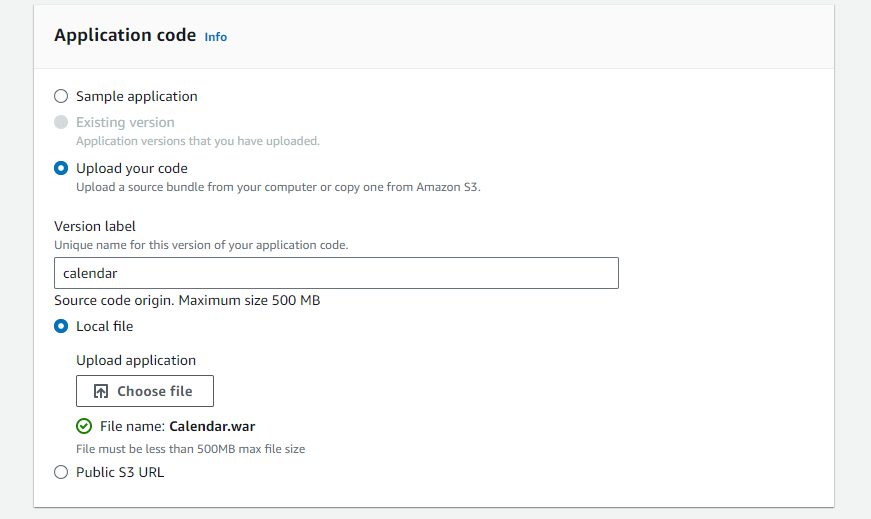
Create a new Web Server environment and give it a name.



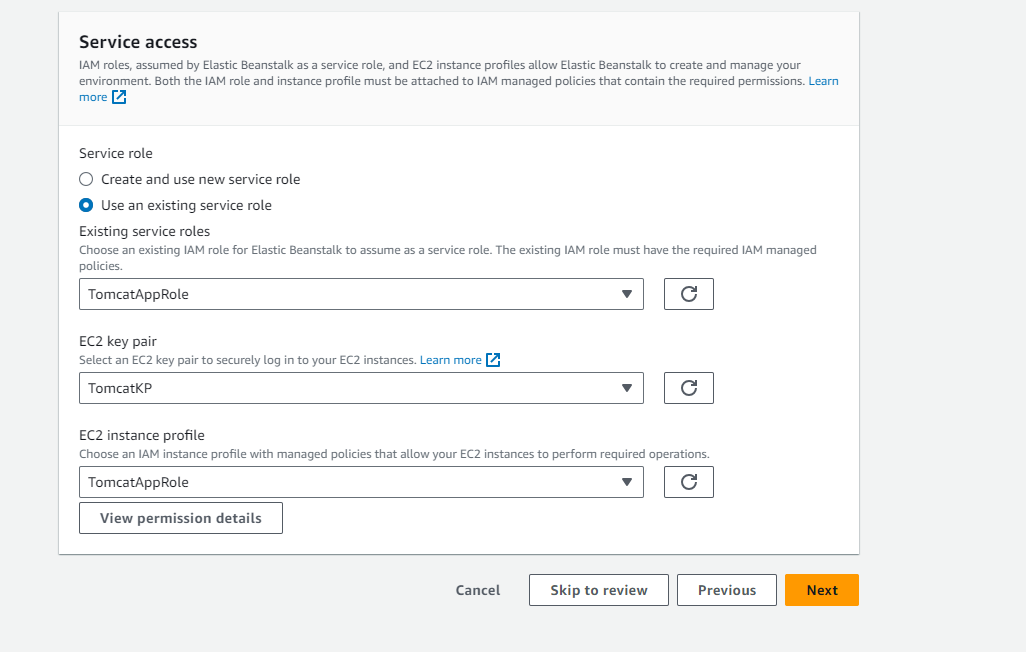
Change the platform details to Tomcat.



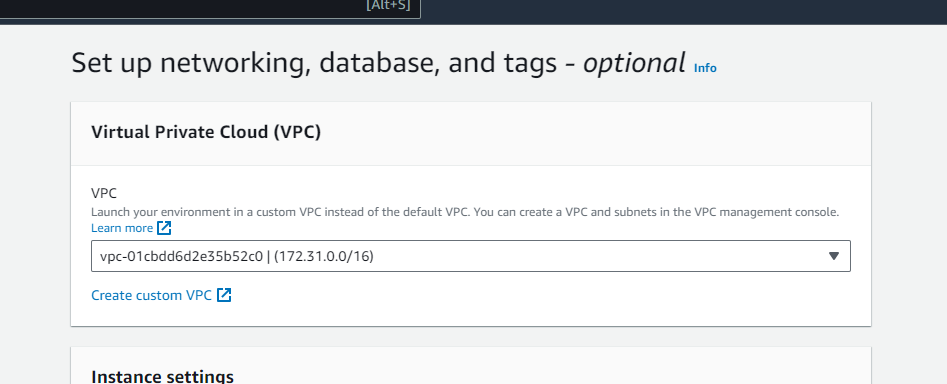
Upload the calendar.war file to the application.

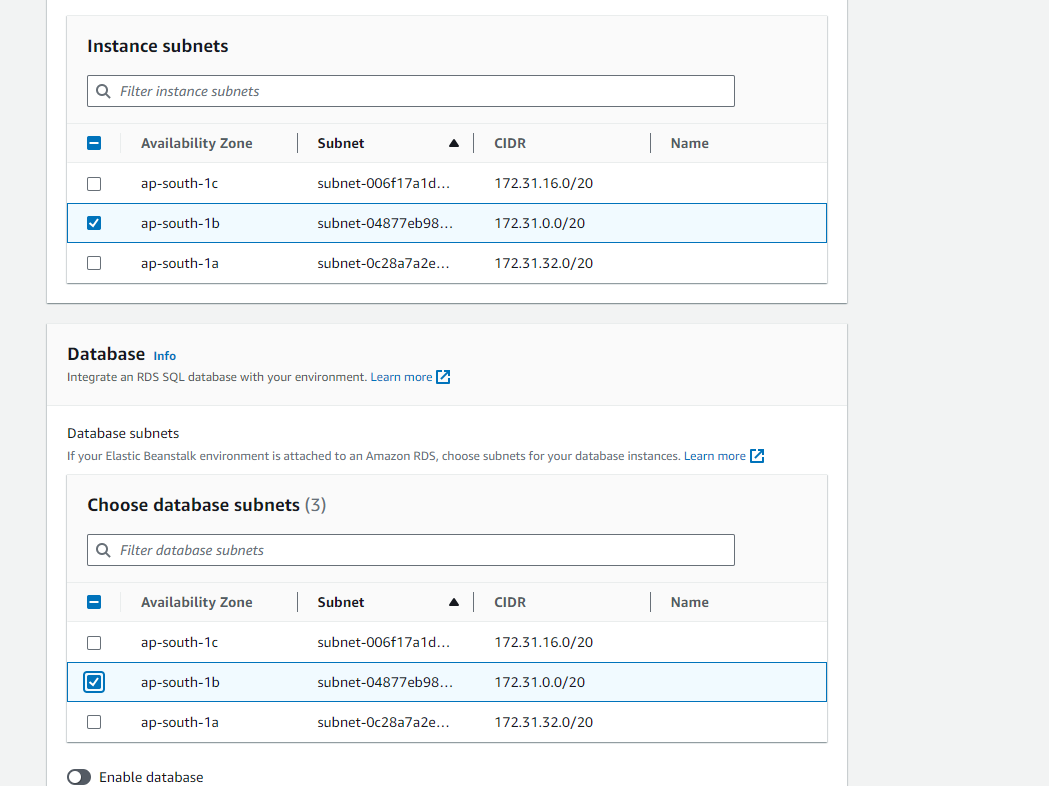


Assign the service roles and key pairs.

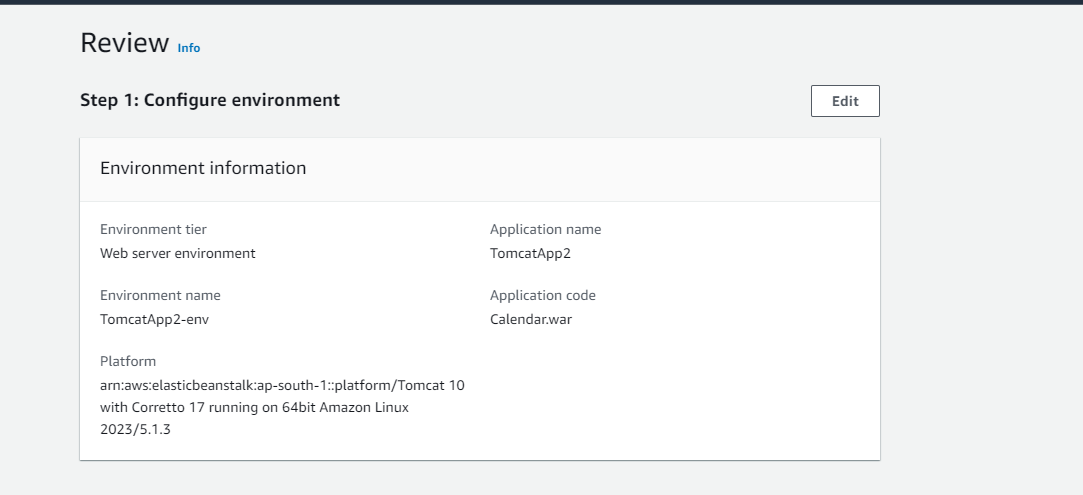


Choose the VPC and subnets

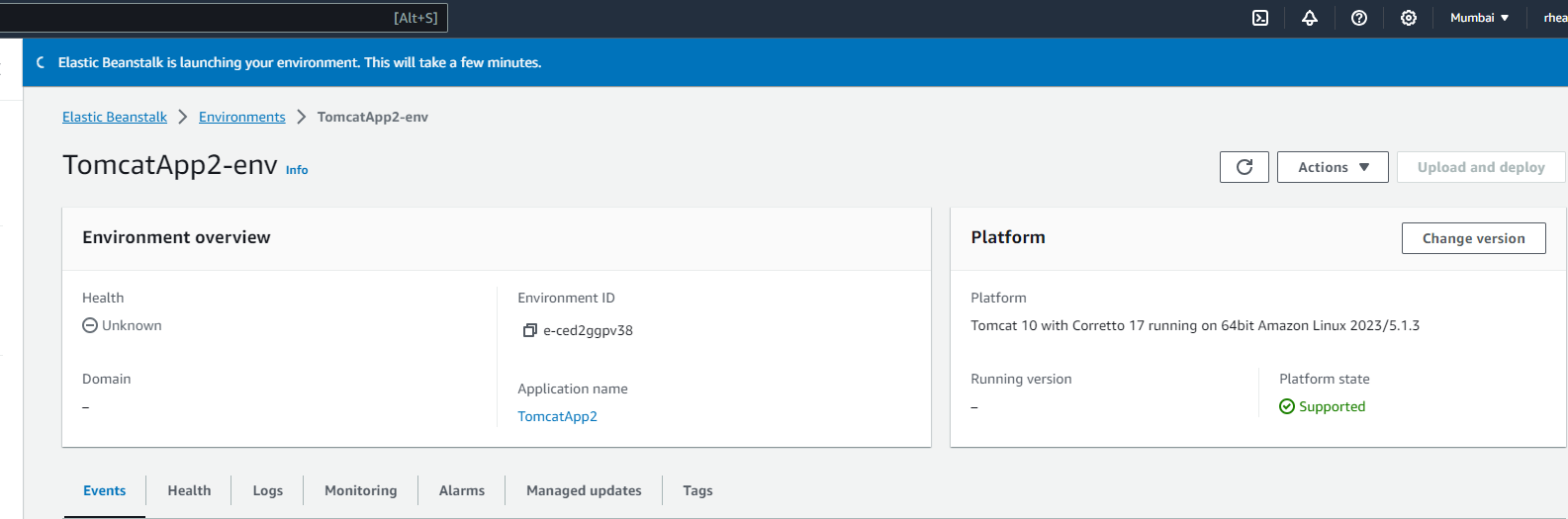


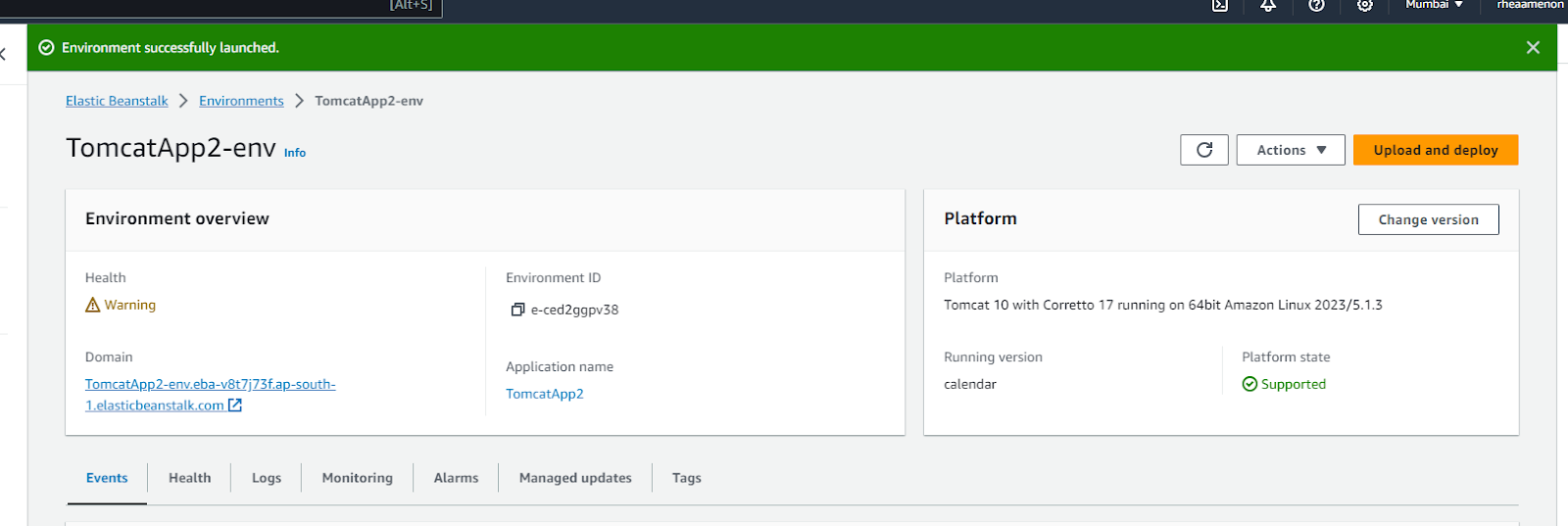


Leave all other settings as default



After the environment is launched, click on it and open the domain link.





The application in the WAR file will now execute.

