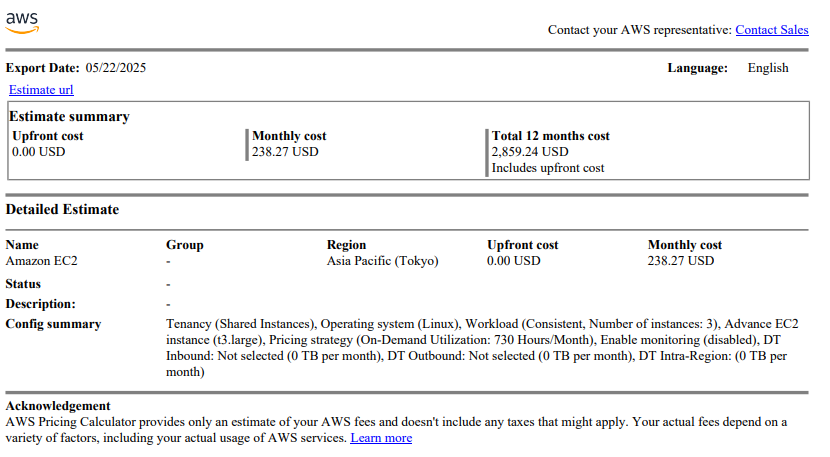
1. Generate an estimate from AWS to run ten Ubuntu machines with specs of two cores and 8 GB of RAM with an EBS volume of 8GB on each instance.
   1. Generate the estimate for an entire month.

The AWS Pricing Calculator is a free tool that helps estimate the cost of AWS services based on your specific configuration and usage needs.

System Specifications:

* Operating System: Ubuntu (Linux)
* Instance Type: t3.large (2 vCPUs, 8 GB RAM)
* Region: Asia Pacific (Tokyo)
* EBS Volume: 8 GB General Purpose SSD (gp3) per instance
* Pricing Model: On-Demand Instances
* Duration: 1 month (730 hours assumed by AWS)

For an entire month, t3.large instance of AWS EC2 was used, and the estimate is as follows:

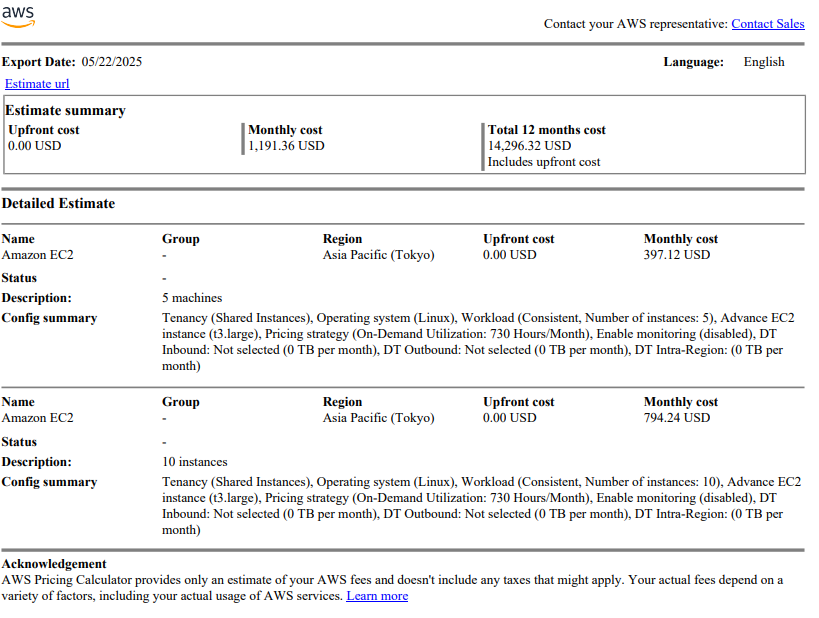


The estimate URL is as [AWS Estimate](https://calculator.aws/#/estimate). The monthly cost for this estimate was

238.27 USD per month.

* 1. Generate an estimate for an average workload of 5 machines and a burst workload of 10 instances twice a week each month.

For this estimate also t3.large instance family was used, and the estimate is as follows:



The estimate URL is as [AWS Estimate](https://calculator.aws/#/estimate). The monthly cost for this estimate was

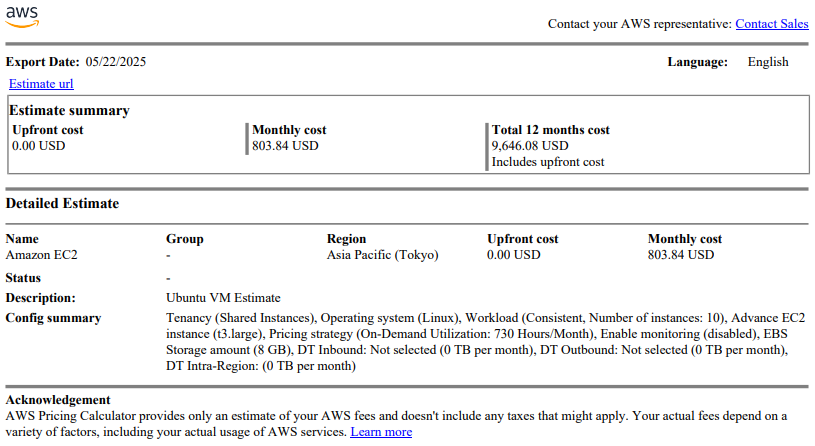
1191.36 USD.

Conclusion:

* Running 10 Ubuntu EC2 instances continuously for a month would cost approximately $238.27, assuming on-demand pricing.
* For a workload with 5 instances running full-time and 5 burst instances running twice a week, the monthly estimate is approximately $1191.36.

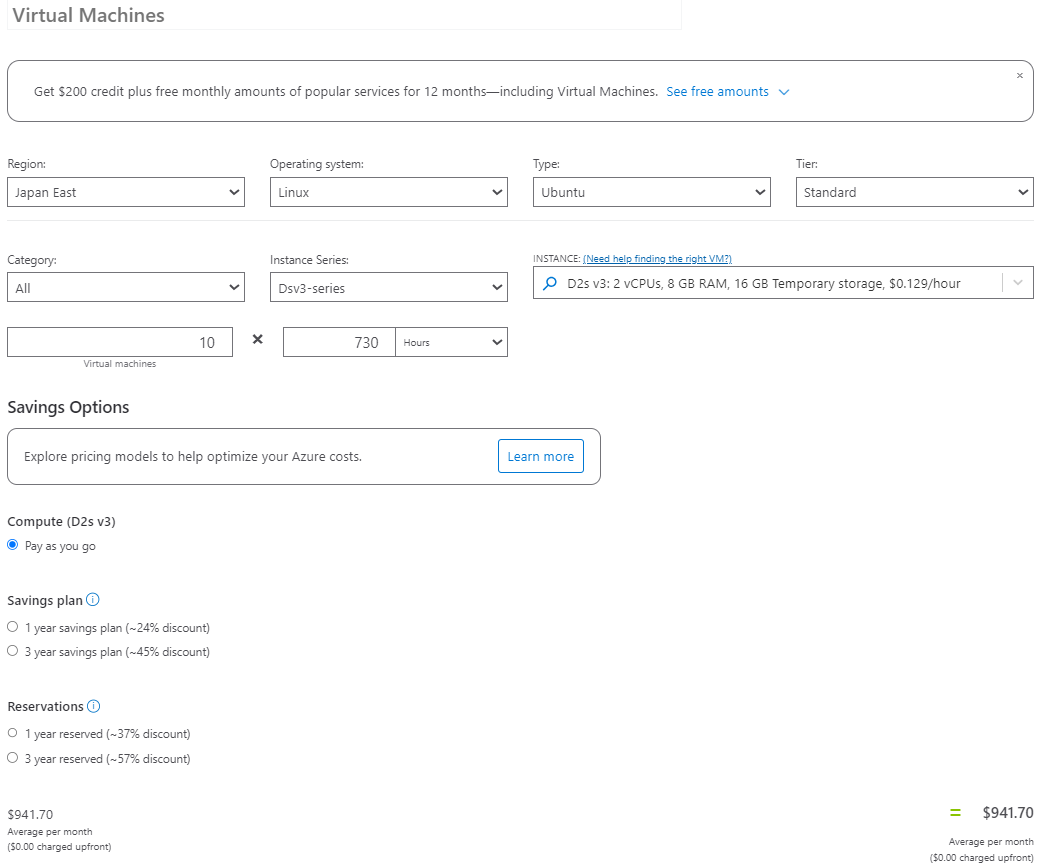
1. Generate an estimate from Azure to run ten Ubuntu machines with specs of two cores and 8 GB of RAM with a managed disk of 8 GB on each instance for a month. Compare the pricing of AWS and Azure for a month and suggest the best cloud option for this workload.

For this estimate, first we look at AWS Estimate for t3.large instance with 8 GB EBS attached. For this estimate the total cost was of 803.84 USD per month.



On the Azure side, we look at D2s v3 instance with (2 vCPUs, 8 GB Ram) 8GB Standard SSD attached. For this estimate the monthly cost was of 941.70 USD.

* **Region**: Japan East
* **Operating System**: **Ubuntu**
* **Tier**: Standard
* **Instance**: Select **D2s v3** (2 vCPUs, 8 GB RAM)
* **Number of Instances**: Set to **10**
* **Billing Option**: Leave as **Pay-as-you-go** (for On-Demand)
* **Hours per month**: Enter **730 hours** (typical monthly usage)
* **Managed Disks**:
* Add 1 **Premium SSD** disk of **8 GB** per VM
* Choose **8 GB P1** or use custom disk size if available



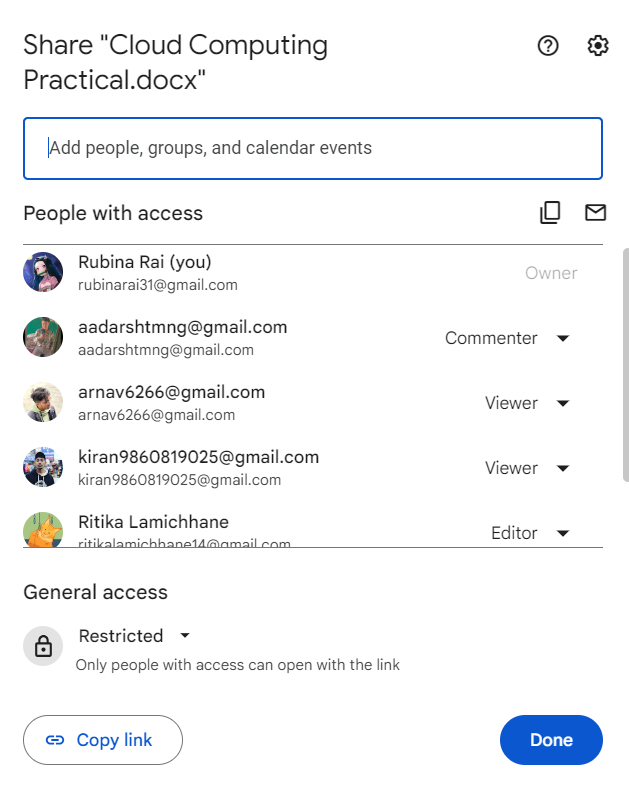
Comparison:

| **Feature** | **Azure (D2s v3)** | **AWS (t3.large)** |
| --- | --- | --- |
|  |  |  |
| vCPUs | 2 | 2 |
|  |  |  |
| RAM | 8 GB | 8 GB |
|  |  |  |
| Storage | 8 GB Premium SSD | 8 GB gp2SSD |
|  |  |  |
| Monthly Hours | 730 | 730 |
|  |  |  |
| Instances | 10 | 10 |
|  |  |  |
| Estimated Cost | ~$941.70/month | ~$803.84/month |

By comparison, I would suggest AWS EC2 as the best option for given requirements.

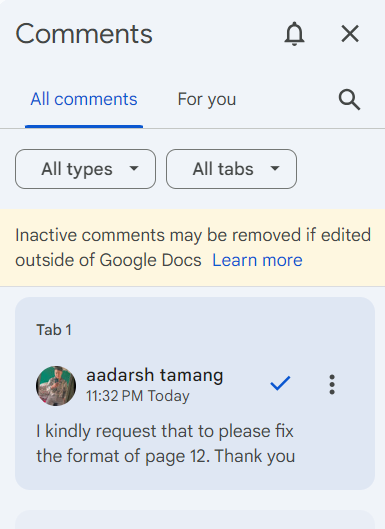
1. Create a Google doc and collaborate with your friends, assigning your friends editor and viewer roles. Include screenshots of working together on a document and setting various privilege levels.

For this, I have created a Google Doc titled “Doc Example for Cloud Computing Practical”, and created a word file. Adding on, I provided access to people as follows:

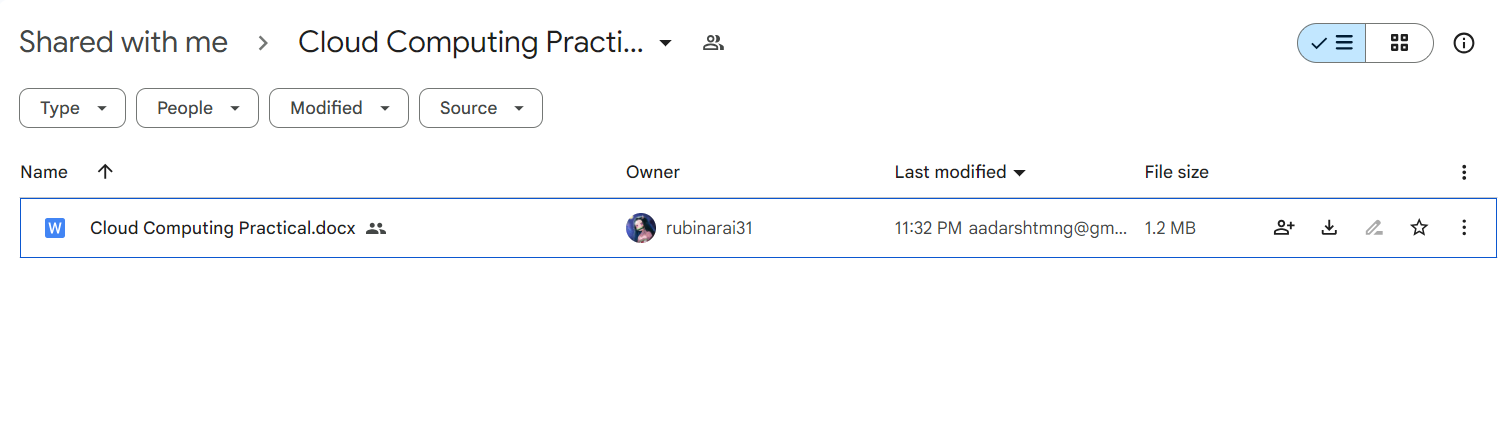


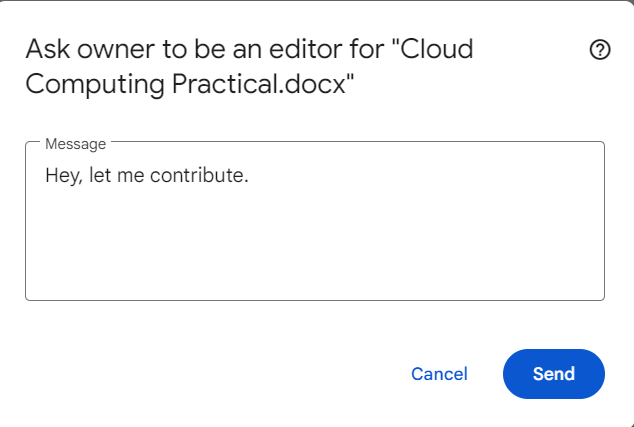
Here only the owners and editor can access the document and edit the contents here. For the commenter, the only access they have is to comment on the document.

Comment from the commenter:

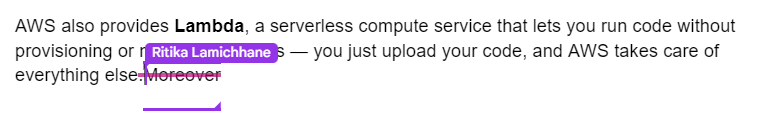


From the viewer’s pov:

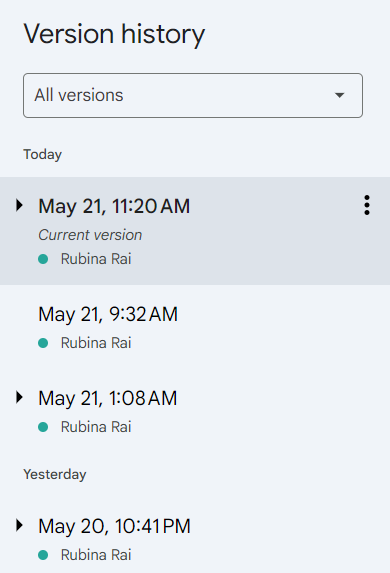




The editor on the other hand can edit the text itself which is visible as:



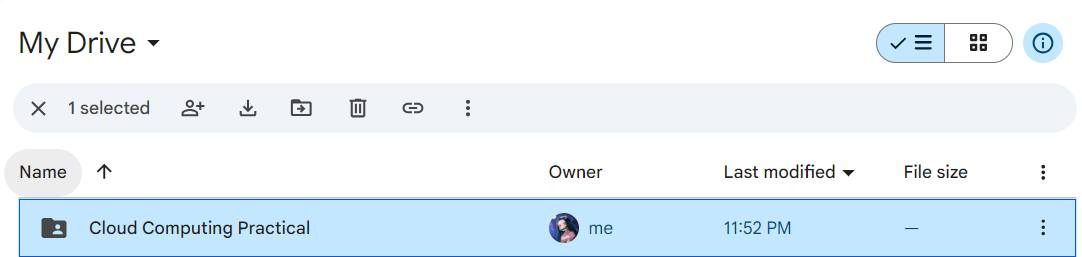
We can also see the version history to see which user edited which portion of the doc and what edit they performed as:



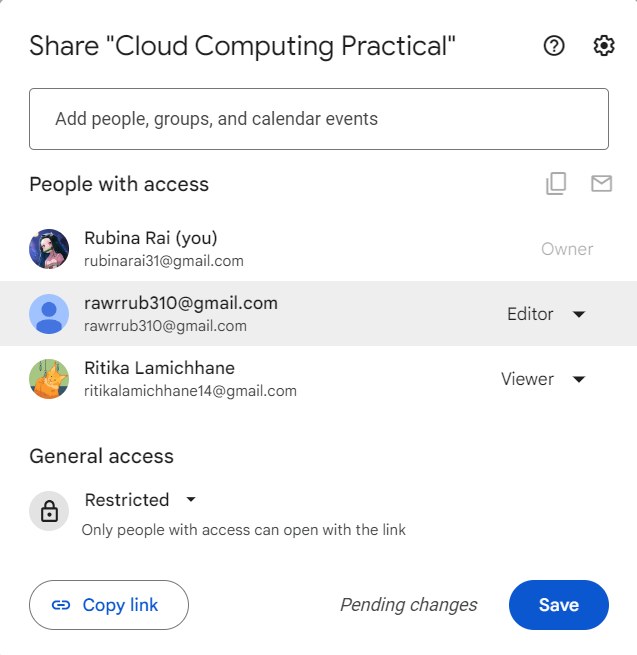
Thus, in this way we can use Google Docs as a SAAS platform.

1. Create a folder on Google Drive. Share it with your friends with various access levels like editor and viewer. Include screenshots of assigning multiple access levels.

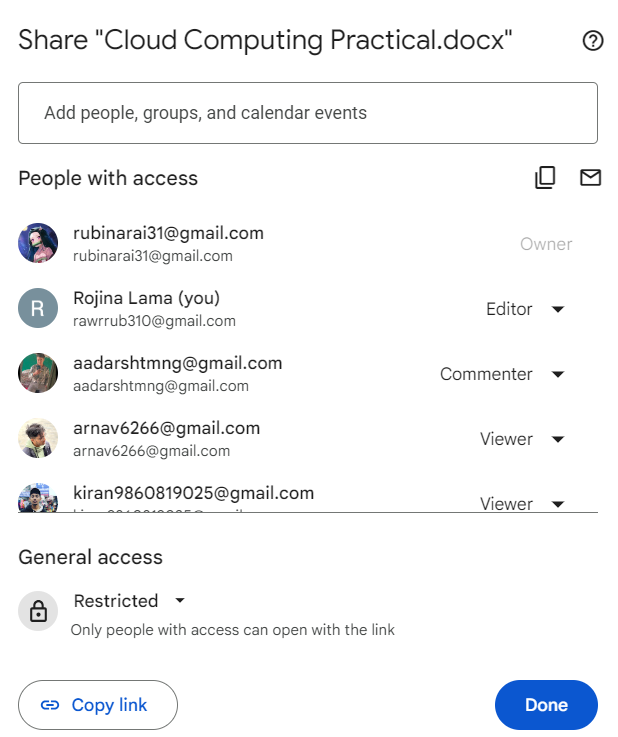
First, a folder is created on Google Drive as:



The, shared the folder as editor to one user and viewer to other as:

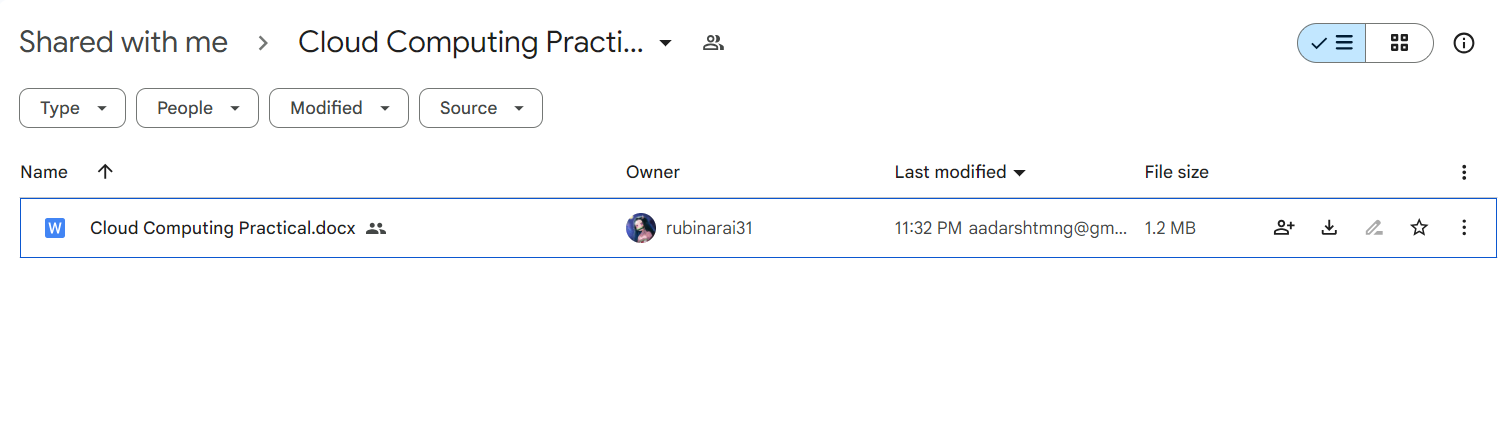


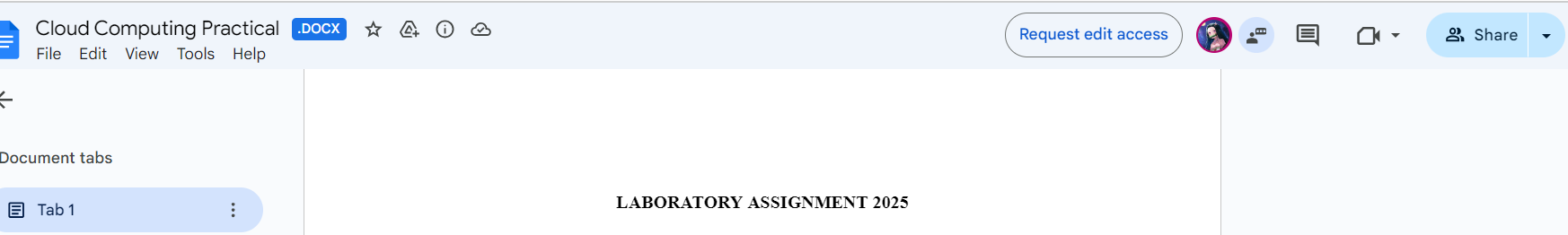
From the editor perspective:



The editor can see who the folder is shared to and add, edit, and delete the contents of the folder.

From the viewer’s perspective, they can just see and view the contents of the folder as:





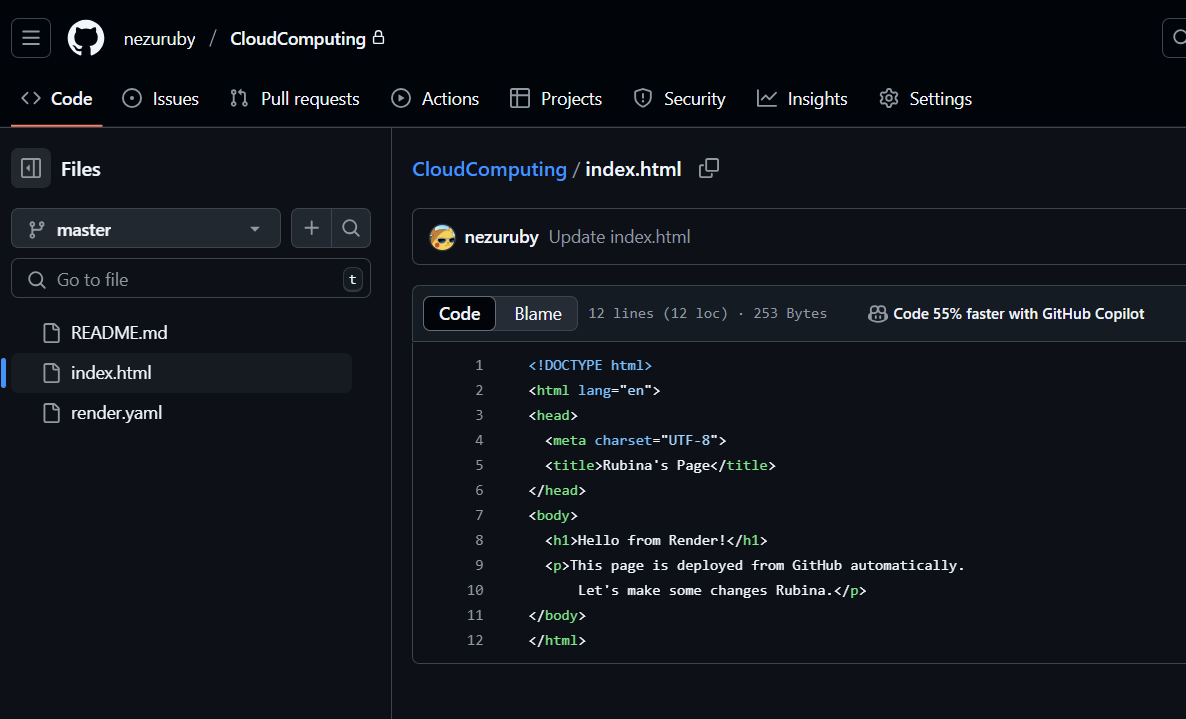
1. Create a code repository on GitHub. Create a simple project, including an HTML page. Use render.io to deploy the code automatically from GitHub to render.io.

Render is a PaaS platform assisting with all scaffolding to deploy code.

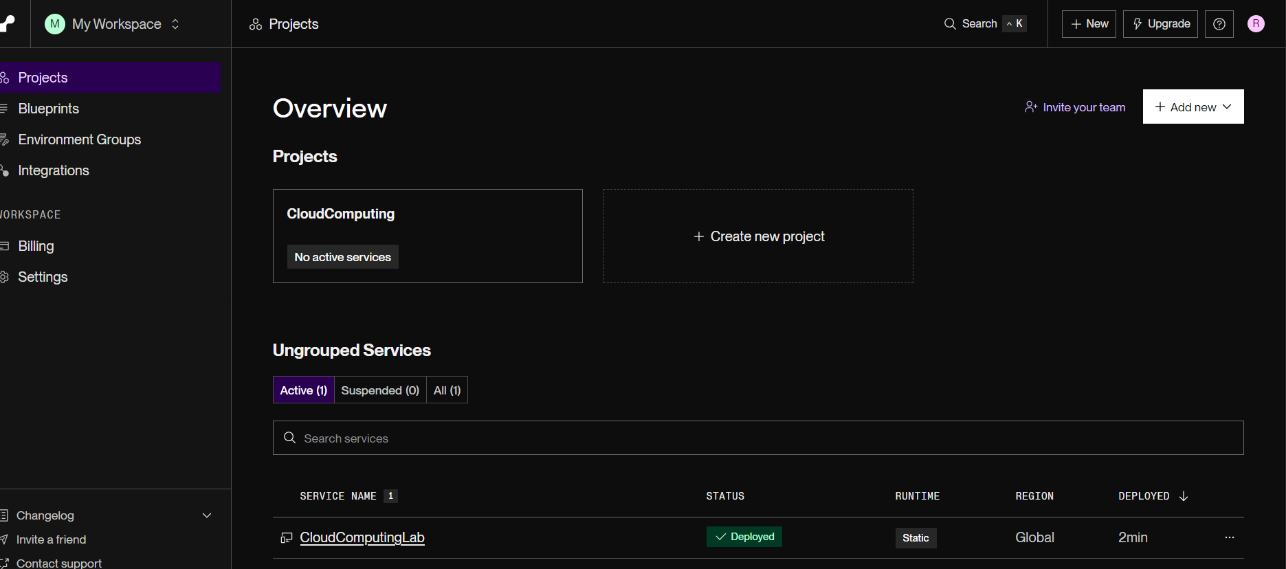
To perform this task, we first need to create a simple repository on GitHub and add an HTML file containing the simple static page we want to display.

I created a simple private repository named “Cloudcomputing”, and added an

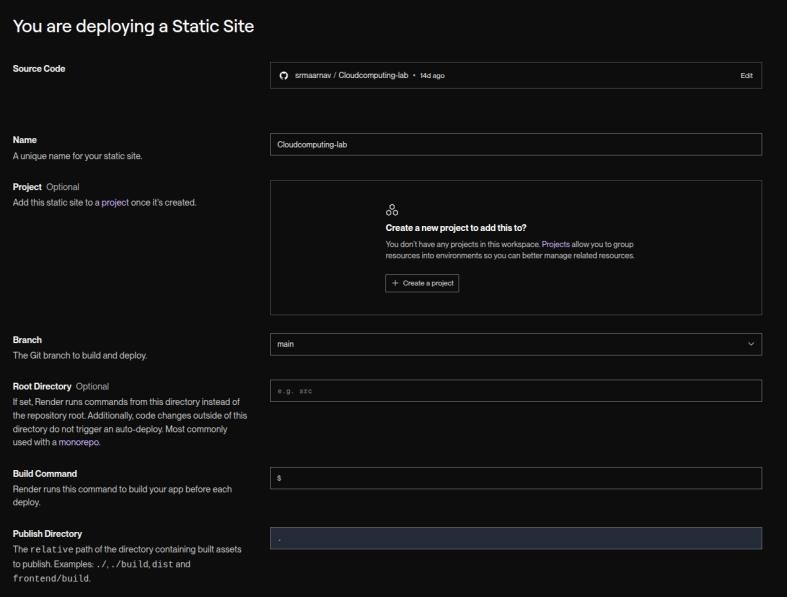
index.html as follows:



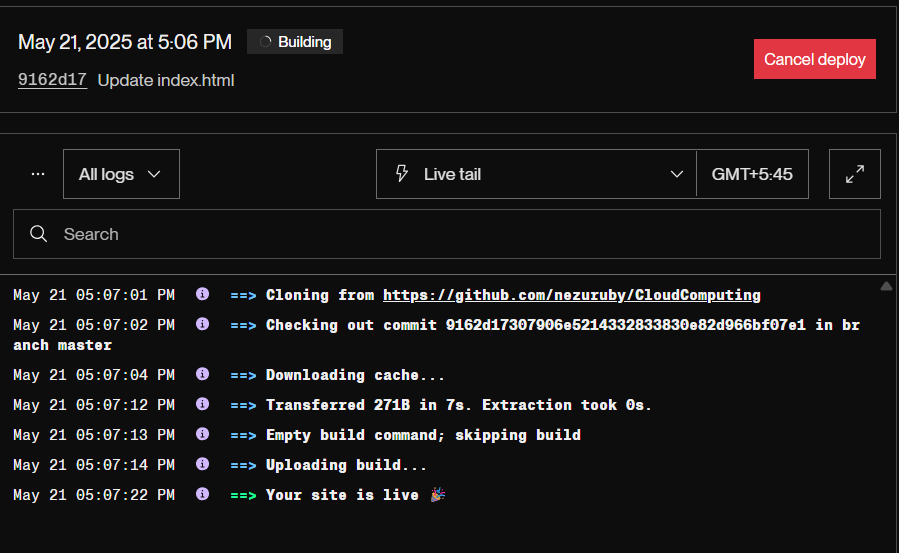
Then, I created a render account with my GitHub account and logged into render, and we can see the dashboard as:



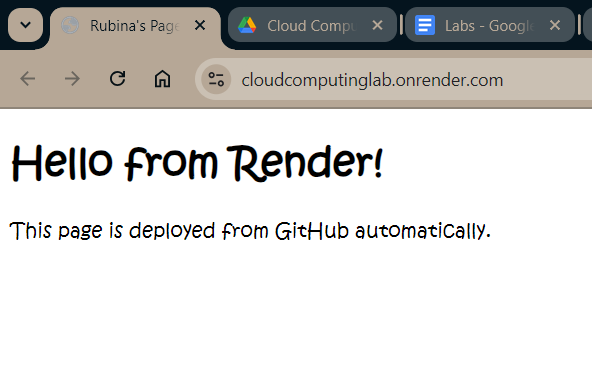
Then, we select Add New and the Static site and then choose our GitHub repository as Source code.



Then we click on deploy static site, which starts the deployment process as:

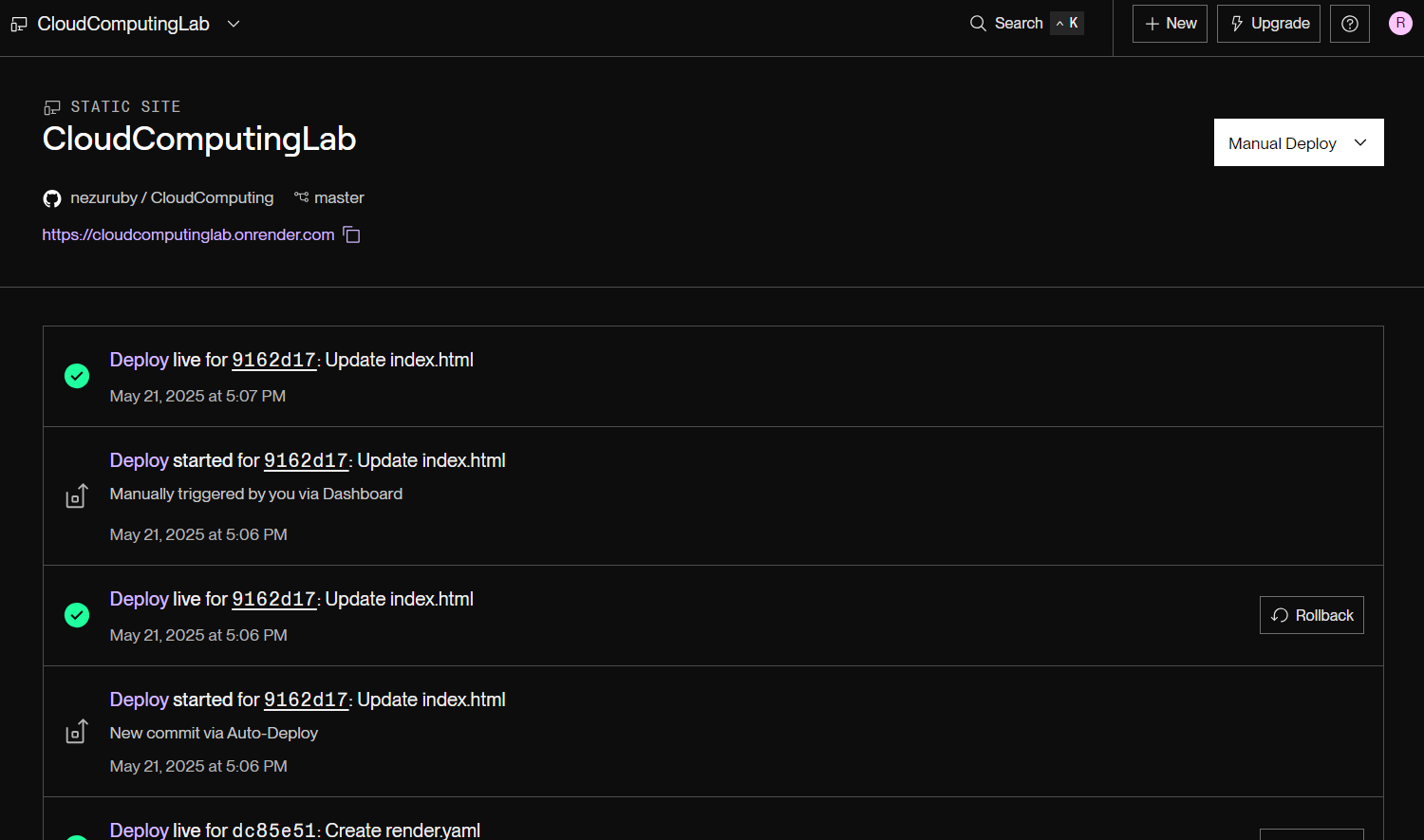


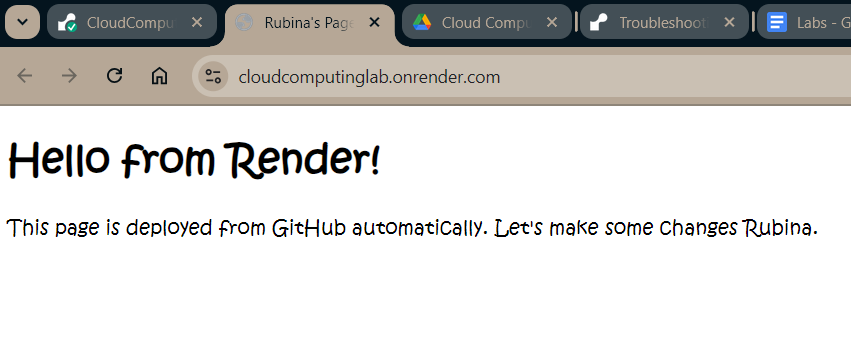
After it says site is live, it provides an URL which we can visit to find our static web page as:



Now, on every push to the main branch on the repository the website is redeployed with the new changes.



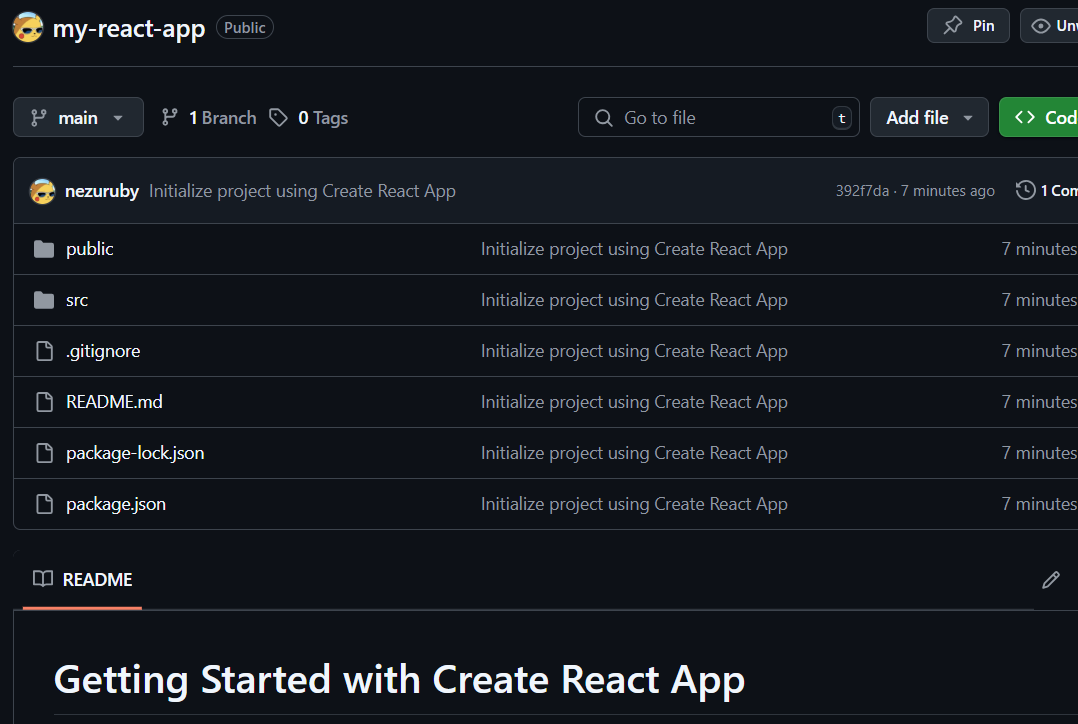




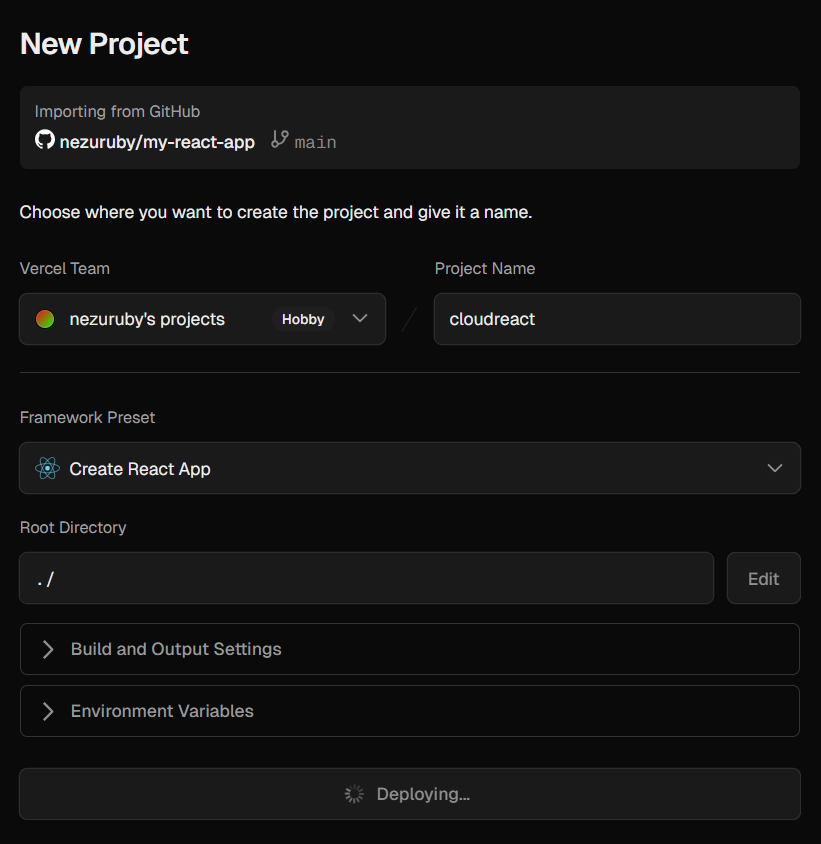
1. Create a simple react based project in GitHub. Deploy this project to Vercel.

Vercel is also a cloud platform for front-end frameworks and static sites, optimized for fast deployment and seamless integration with tools like Next.js.

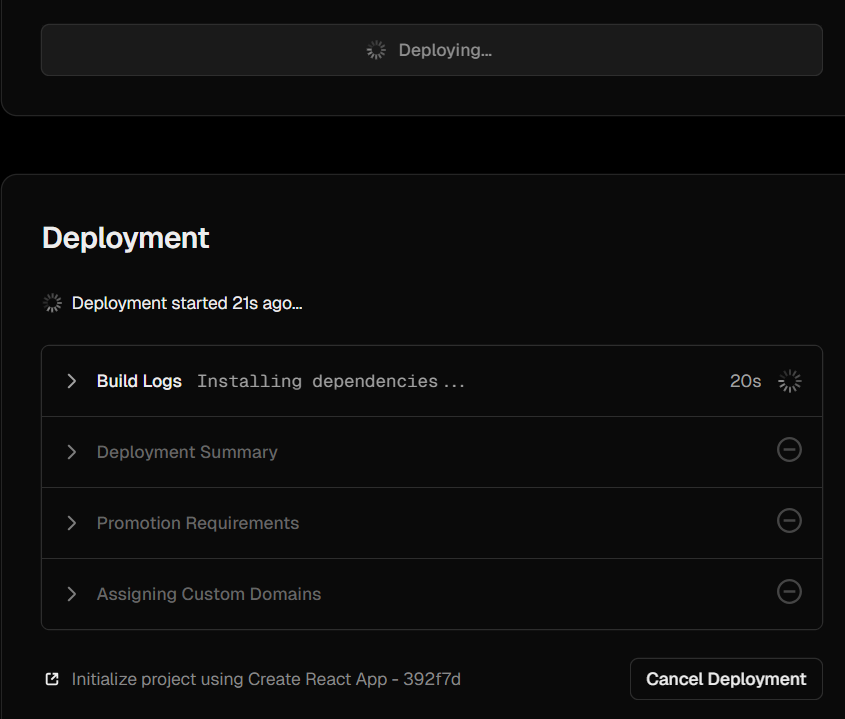
Here, similarly a GitHub repo was created named CloudLabs, and then a simple react app was created as:

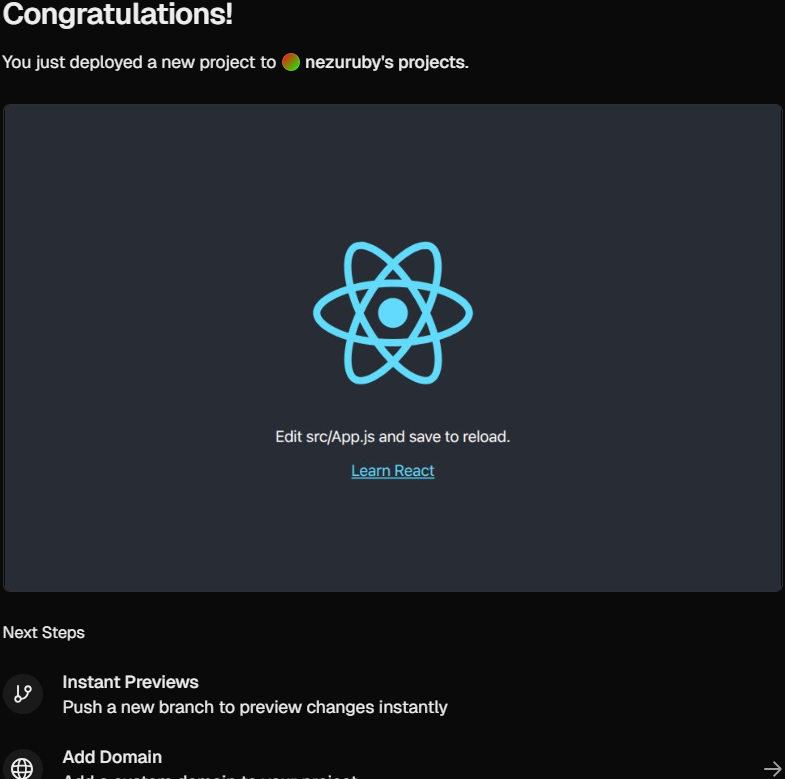


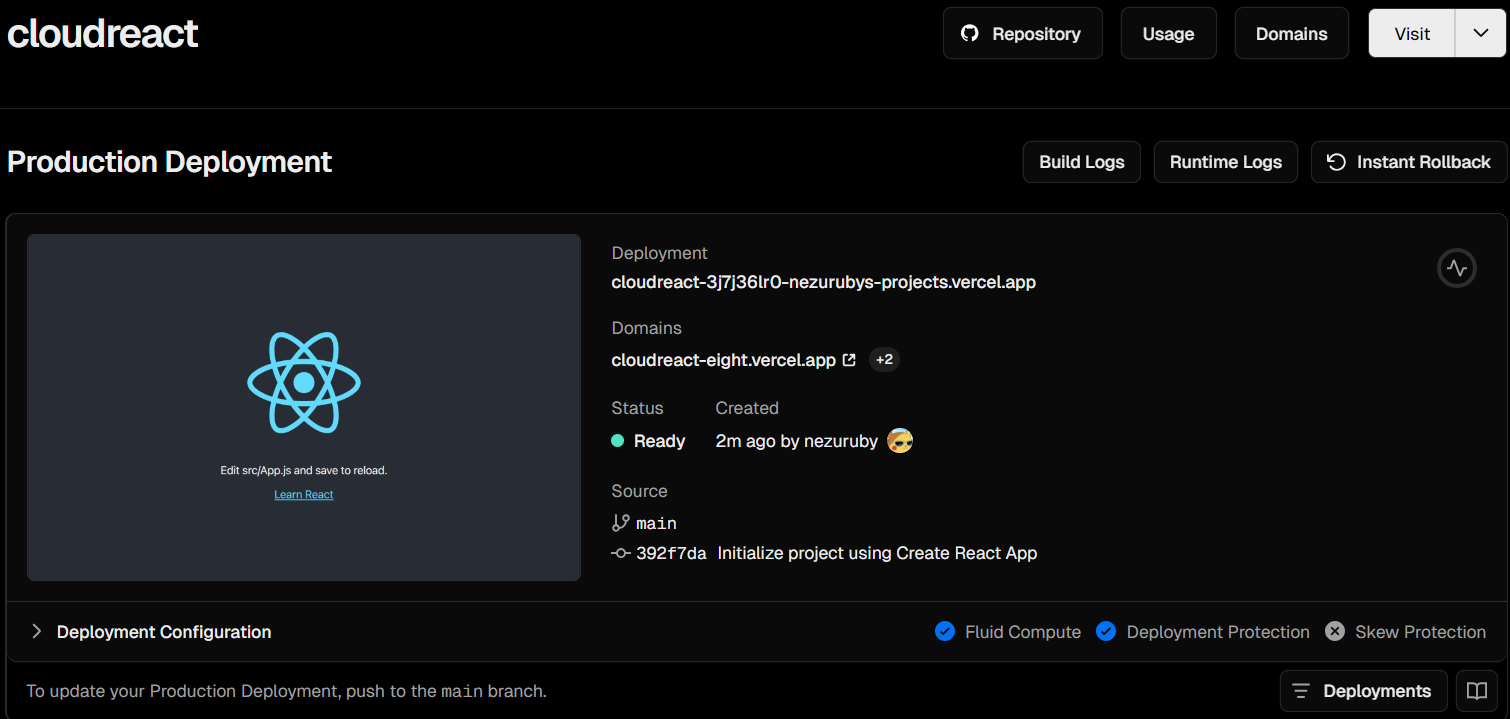
Then, on vercel, I selected the repository from GitHub as:



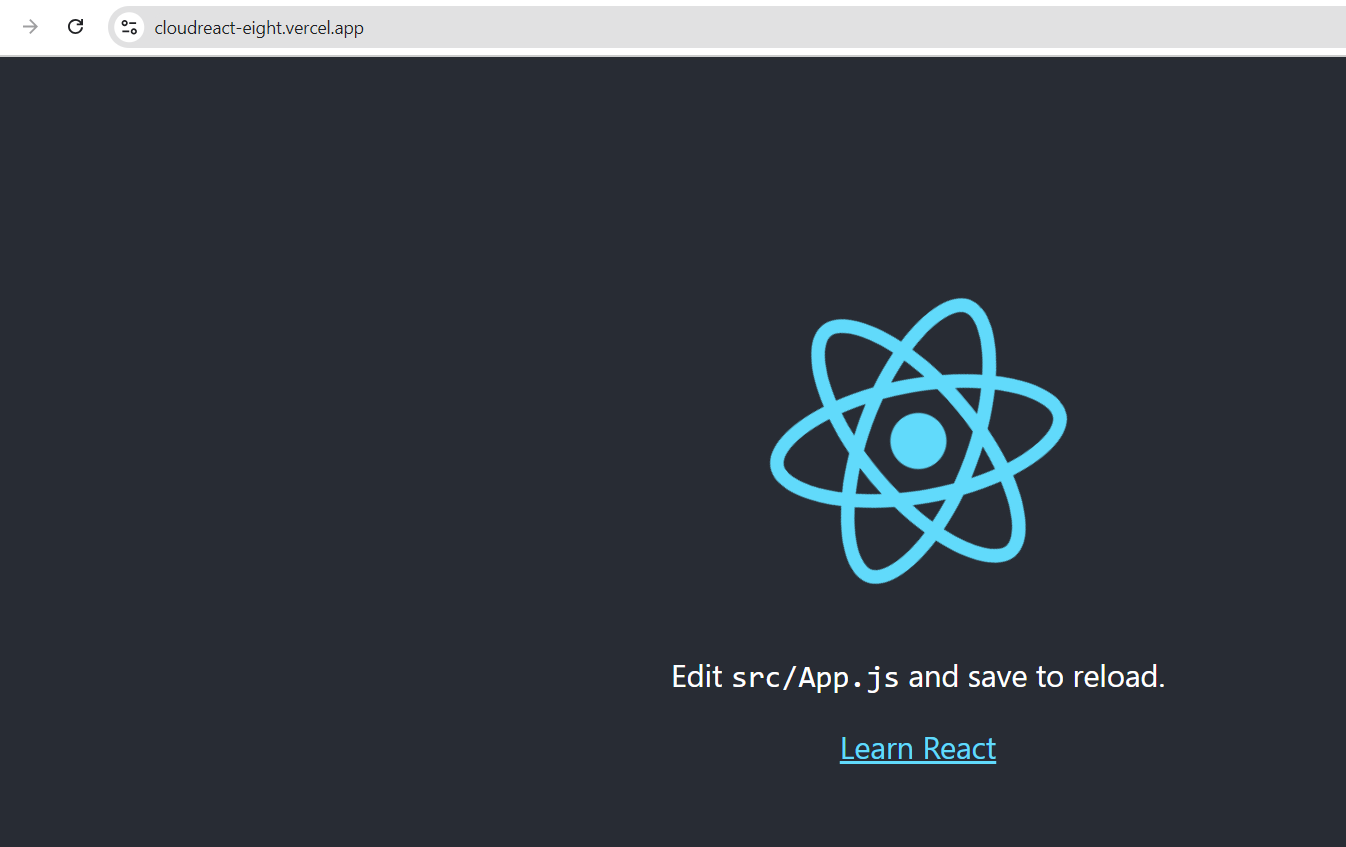
And left the default configuration, but chose Vite as the Framework Preset, then the application is deployed as:



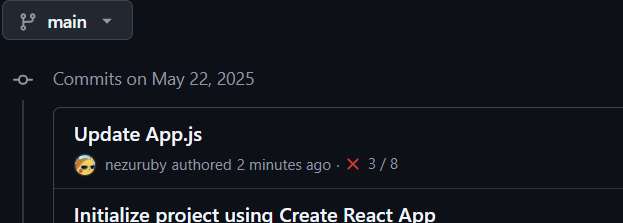


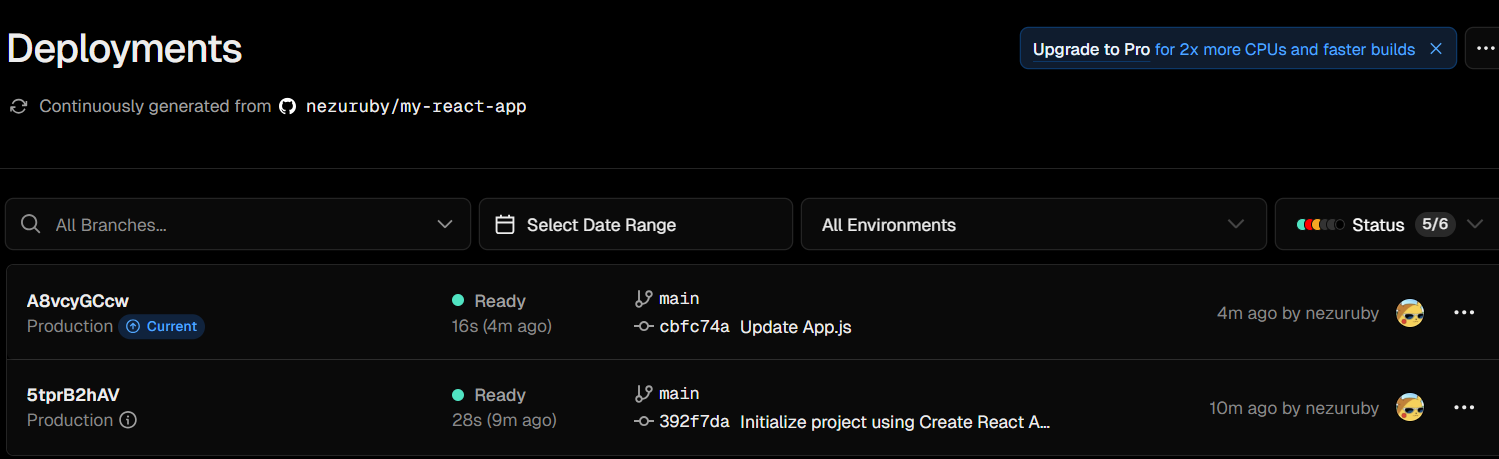


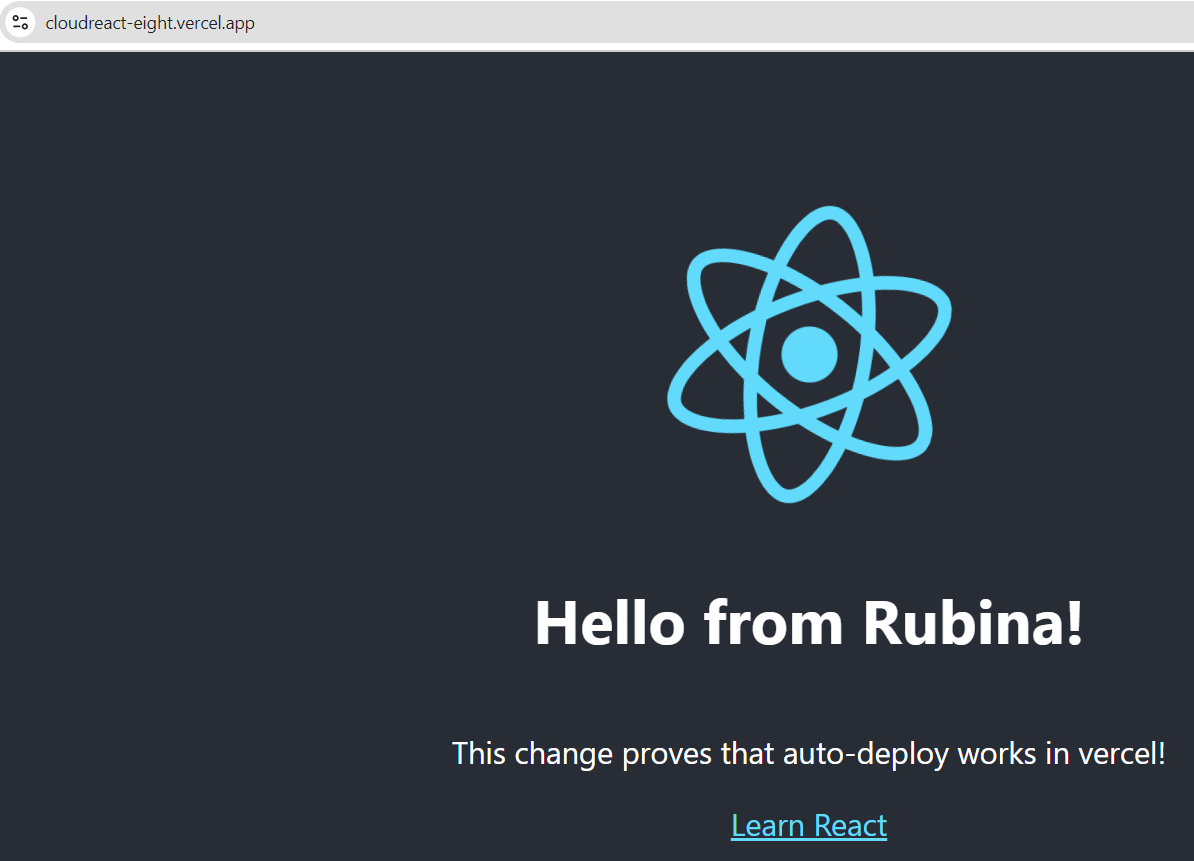
And on visiting it, we can see:



After we push a new code and deploy it again, we can see the following, and the site is also updated accordingly.



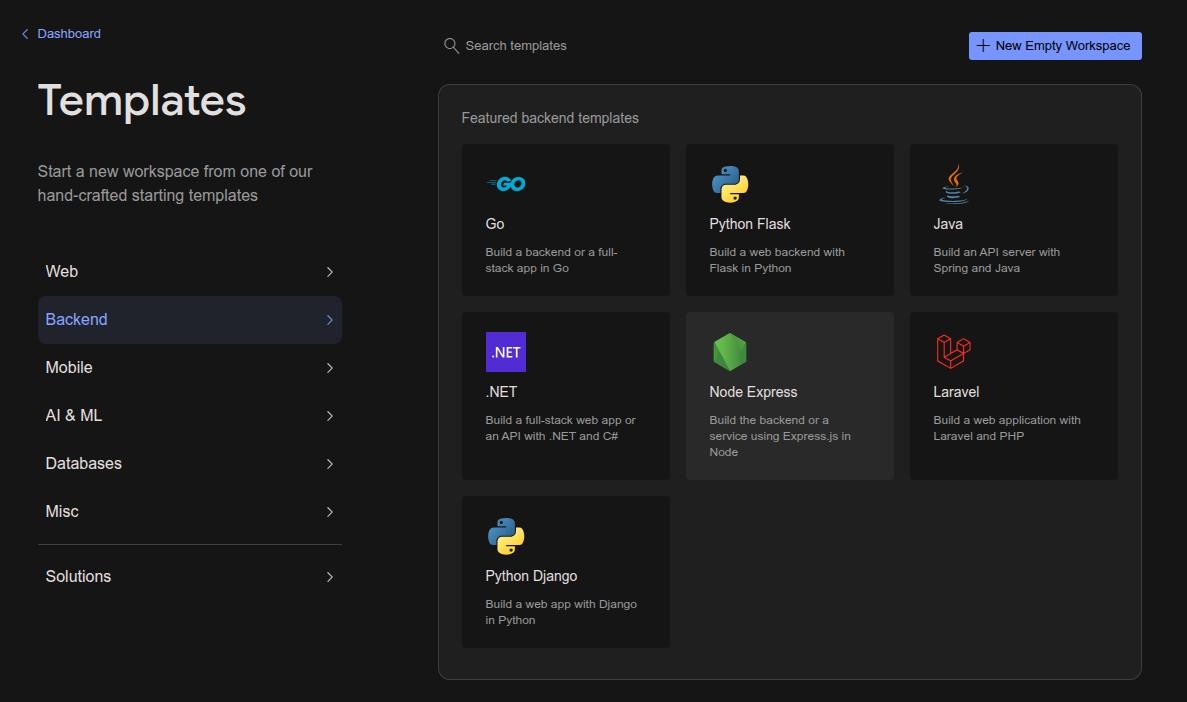




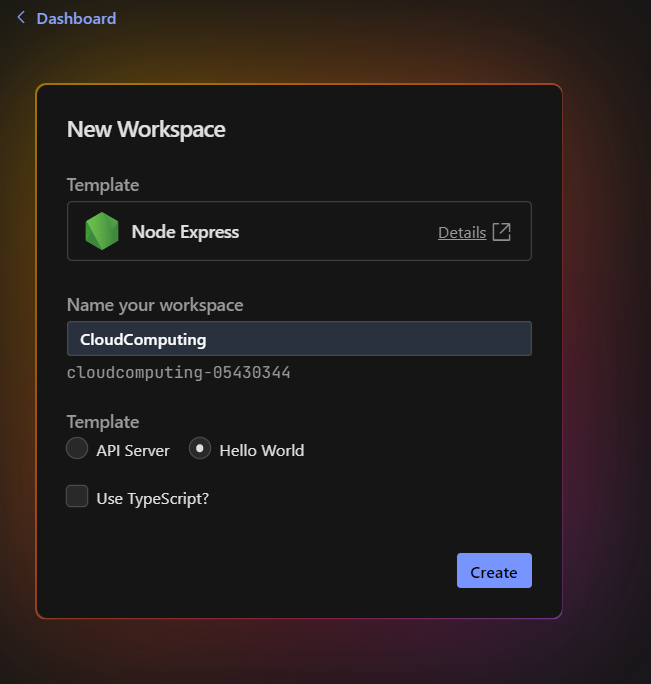
1. Using firebase studio create a sample node js project. Taking help from the AI, write a simple test case for the project and run the test case and the project.

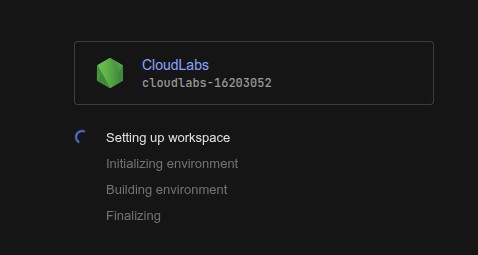
Firebase is a comprehensive app development platform by Google that provides backend services like authentication, real-time databases, cloud storage, and hosting for building web and mobile applications.

First, we visit the firebase studio, click on New Workspace, and then choose Backend and NodeJs as template.



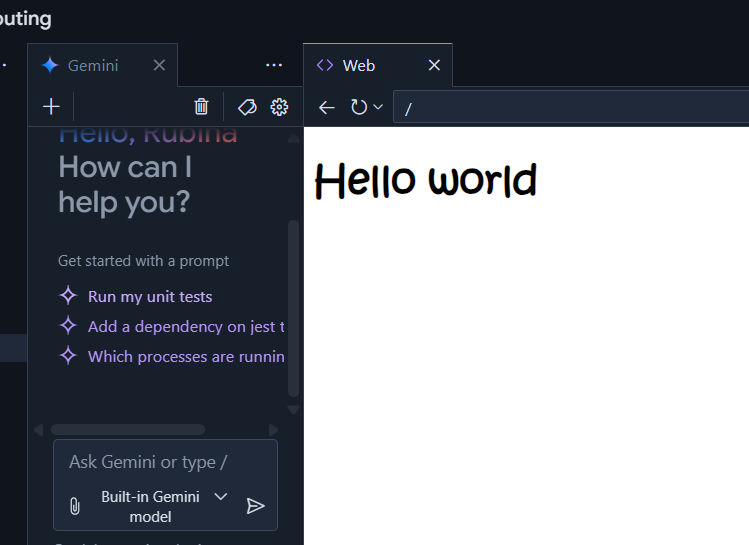
Then, we choose the name for workspace and the template and select create which sets up the project.





After project creation, we are presented with a web code editor for our use, where we can use Gemini API as coding assistant.

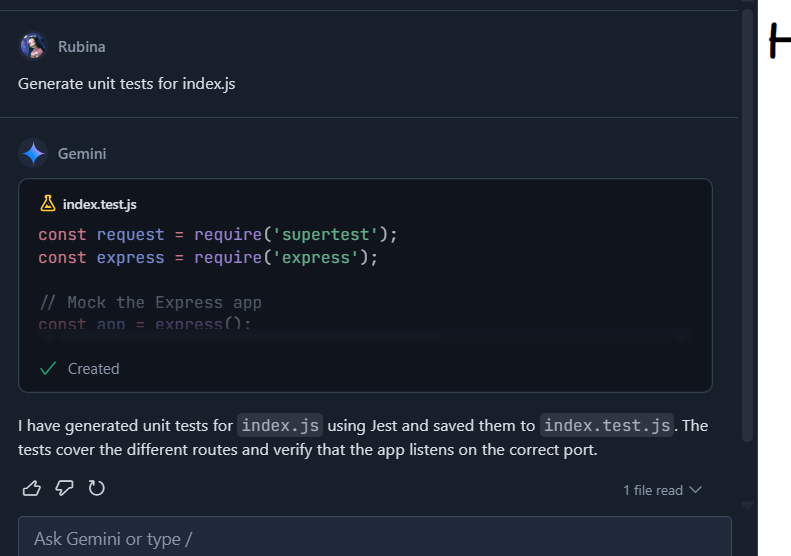
We can preview our Hello World NodeJs application from internal browser as:



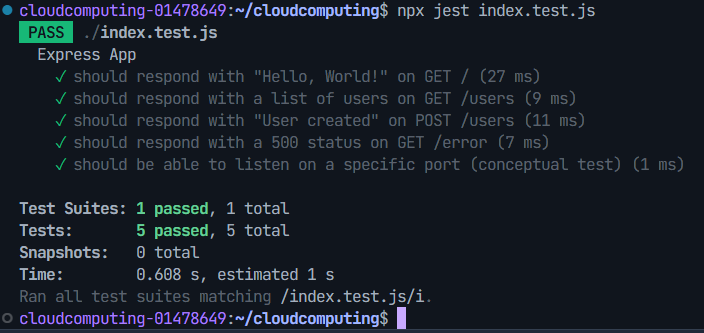
The, we find the defaut index.js file with a simple api route for ‘/’, and we can open

gemini assistant and ask it to create a simpleunit test for index.js, which it does and asks

for permission to write it into a file named index.test.js as:



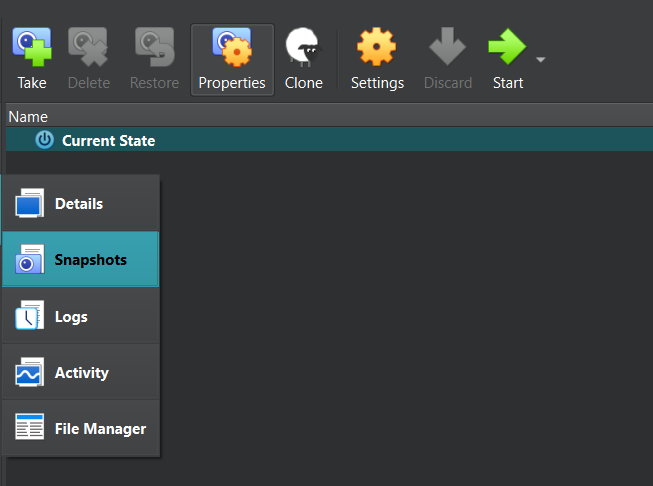
And on running the test, we can see:



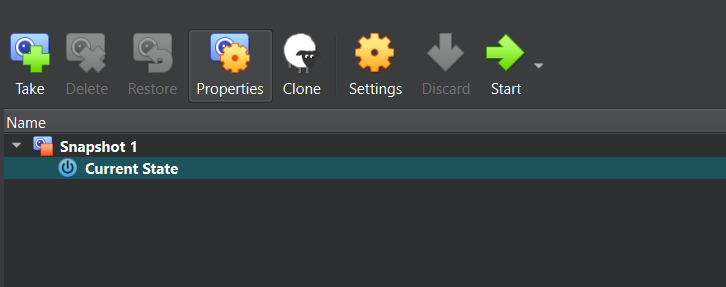
In this way, we can use Firebase Studio to completely code on the web with Gemini as assistant.

1. Create a snapshot for a virtual machine hosted in VirtualBox. Using rm-rf remove the files in the virtual machine. Reboot the virtual machine. Use the snapshot to restore the virtual machine.

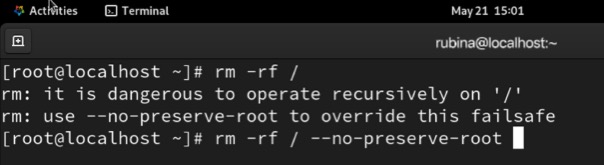
First, we create a virtual machine on VirtualBox, and then select the tools option, and go to the Snapshot menu.



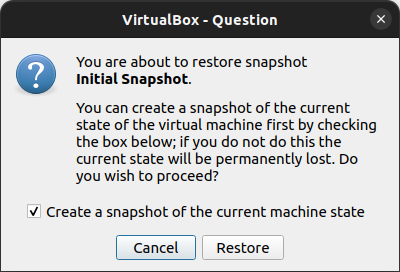
There, select the Take option to create the snapshot, name it and create a snapshot as:



Then, inside the VM, we perform the following:



After that, we shut down the machine, and the restore it using the snapshot we just took as:



Then, machine is restored to our first snapshot, and then the current state is saved as another snapshot. Then, we can find all the dirs removed after rm –rf / as well as:

**COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY**

Gangahity, Chabahil, Kathmandu, Nepal

(Affiliated to Tribhuvan University)

****

**LABORATORY REPORT OF**

**Introduction to Cloud Computing (CSC - 467)**

**Submitted by:** **Submitted to:**

**Name:** Rubina Rai **Instructor:** Mr. Roshan Khatri

**Roll No:** 124

**Semester:** Eighth

**Faculty:** Science and Technology

**Level:** Bachelor

**Program:** CSIT

**COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY**

Gangahity, Chabahil, Kathmandu, Nepal

(Affiliated to Tribhuvan University)

****

**LABORATORY REPORT OF**

**Advanced Database (CSC - 461)**

**Submitted by:** **Submitted to:**

**Name:** Rubina Rai **Instructor:** Mr. Pralhad Khanal

**Roll No:** 124

**Semester:** Eighth

**Faculty:** Science and Technology

**Level:** Bachelor

**Program:** CSIT