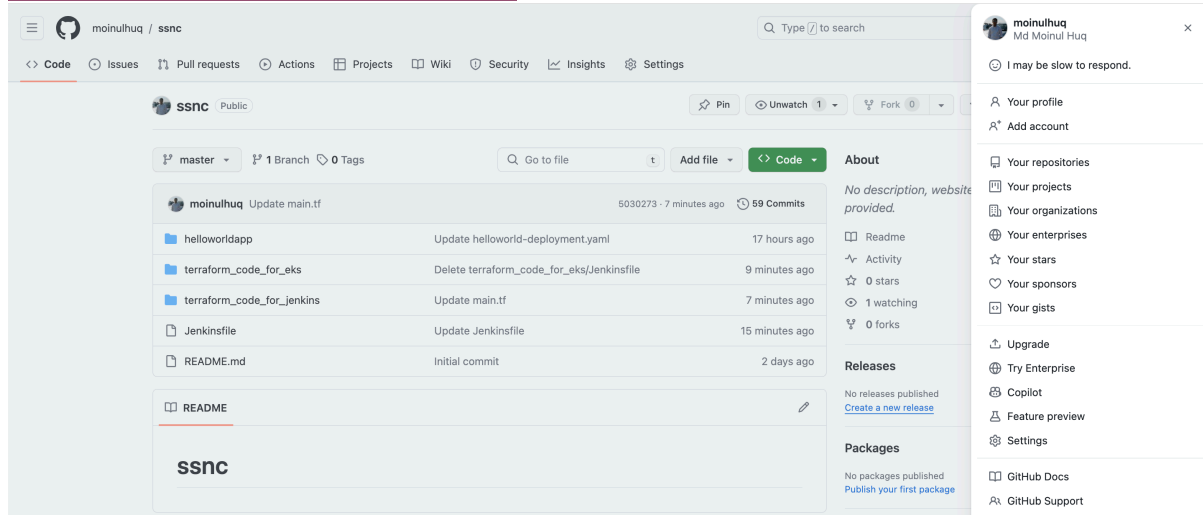


SSNC PROJECT

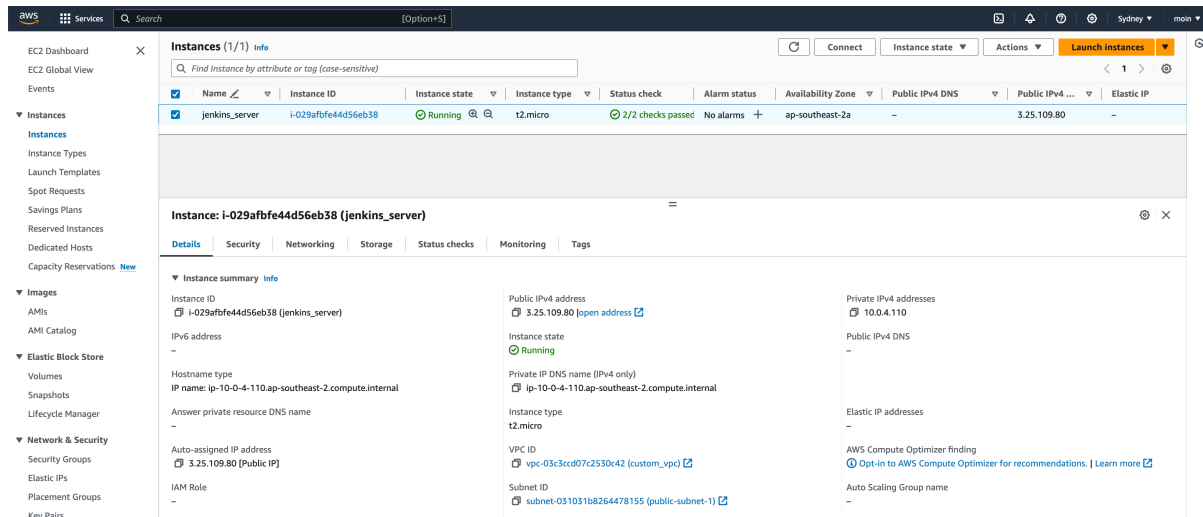
Step01: Create Jenkins server using “terraform_code_for_jenkins” from this GitHub link

<https://github.com/moinulhuq/ssnc.git>



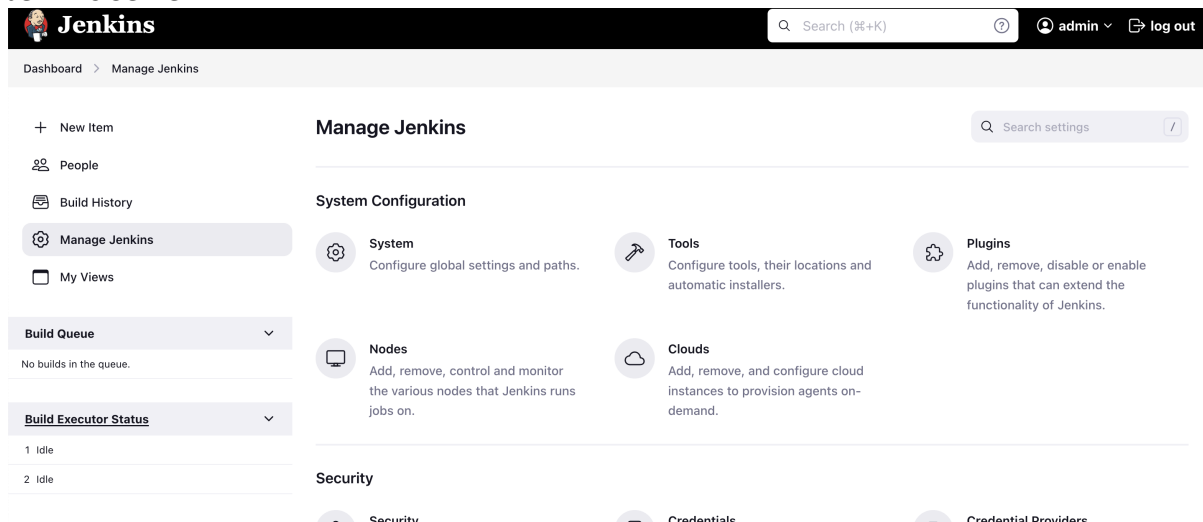
The screenshot shows the GitHub repository page for 'ssnc' by user 'moinulhuq'. The repository is public and has 59 commits. The commit history shows updates to 'main.tf' and 'Jenkinsfile'. The README section is visible, showing the repository name 'ssnc'.

EC2 - Aws



The screenshot shows the AWS Management Console for the 'jenkins_server' instance. The instance is running and has a public IP address of 3.25.109.80. The instance details are expanded, showing the instance ID, state, type, and various addresses.

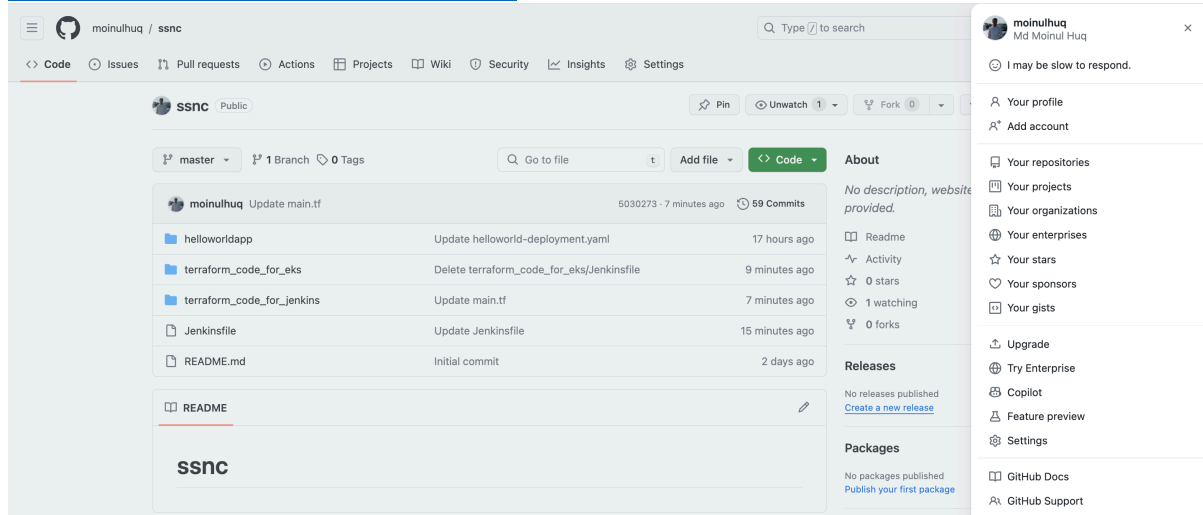
Jenkins Server



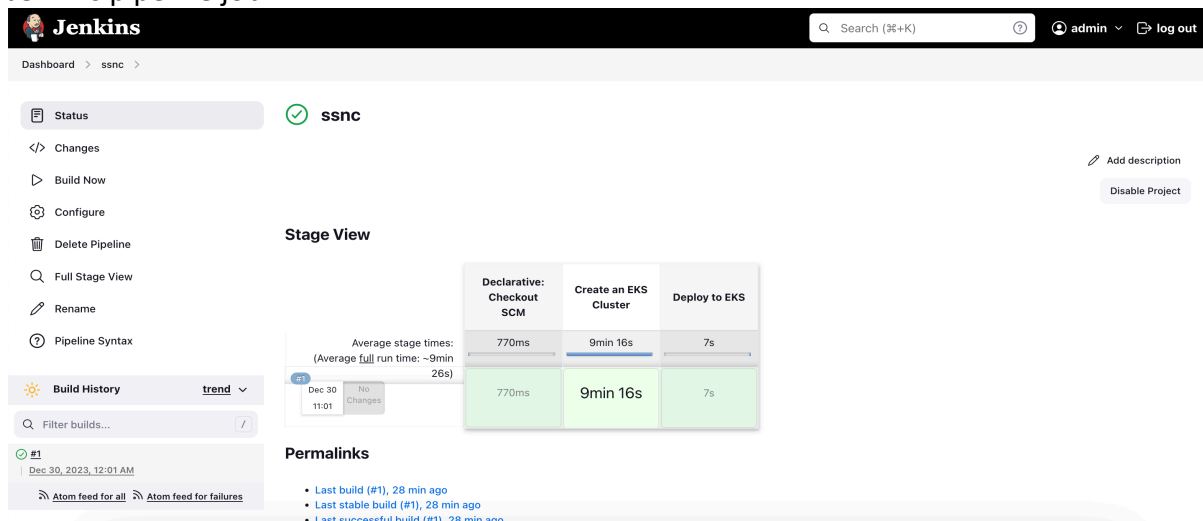
The screenshot shows the Jenkins web interface. The 'Manage Jenkins' page is displayed, showing system configuration, tools, plugins, nodes, and security settings.

Step02: Create one pipeline job under Jenkins and locate “Jenkinsfile”. Pipeline will checkout this “terraform_code_for_eks” to create EKS cluster along with application deployment.
GitHub link

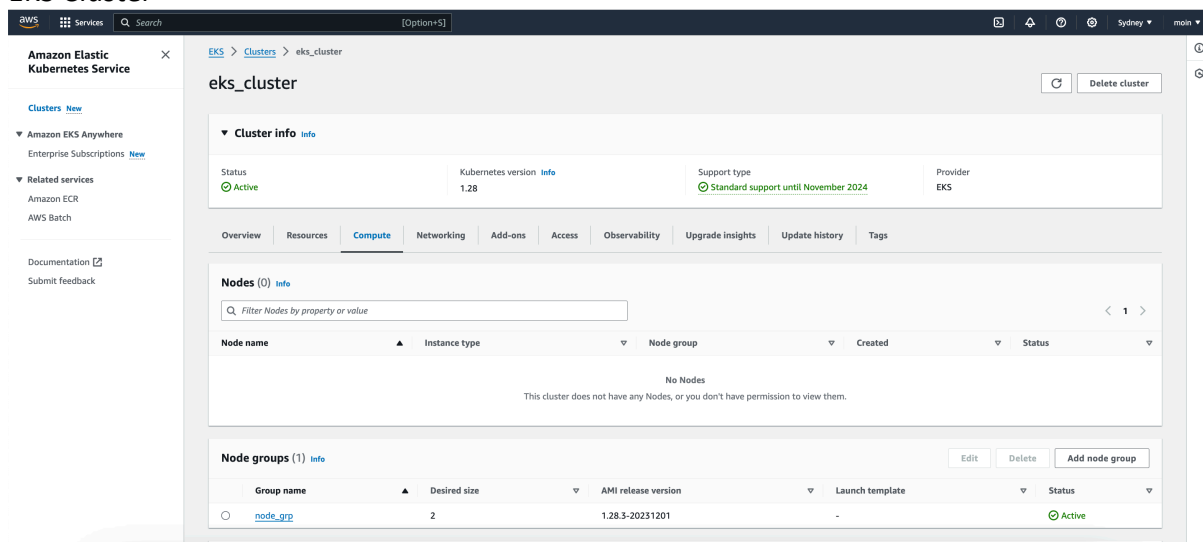
<https://github.com/moinulhuq/ssnc.git>



Jenkins pipeline job



EKS Cluster



Load balancer

The screenshot shows the AWS Management Console for a Classic Load Balancer. The left sidebar contains navigation links for EC2 Dashboard, Global View, Events, Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, and Network Interfaces. The main content area displays the details for the load balancer `ac0e83cc82f8d4dcdcae859b92c48148c`. The details include: Load balancer type (Classic), Status (2 of 2 instances in service), VPC (vpc-03c3cc07c2530c42), Date created (December 30, 2023, 11:10 (UTC+11:00)), Scheme (Internet-facing), Hosted zone (Z1GM3OXH4ZPM65), and Availability Zones (ap-southeast-2b and ap-southeast-2a). A notification banner indicates that the Classic Load Balancer can be migrated to an Application Load Balancer or Network Load Balancer. Below the details, there are tabs for Listeners, Network mapping, Security, Health checks, Target instances, Monitoring, Attributes, and Tags. The Listeners tab is currently selected, showing a single listener.

Hello world application up and running

The screenshot shows a web browser window with the address bar displaying `ac0e83cc82f8d4dcdcae859b92c48148c-290380726.ap-southeast-2.elb.amazonaws.com`. The browser shows a single tab titled 'Hello world!'. The address bar also shows the domain `ac0e83cc82f8d4dcdcae859b92c48148c-290380726.ap-southeast-2.elb.amazonaws.com`. The browser's taskbar at the bottom shows several open applications, including '4x Car Door Sill Plat...', '4x Door Sill Protect...', 'Getting Started', 'Ayush Sharma', and several Terraform-related applications.



Hello world!

My hostname is helloworld-556885b499-sr9dd

Links found

HELLOWORLD listening in 80 available at tcp://172.20.209.200:80

KUBERNETES listening in 443 available at tcp://172.20.0.1:443

CloudWatch for monitoring

The screenshot shows the AWS CloudWatch console for the Container Insights dashboard. The left sidebar contains navigation links for Dashboards, Favorites and recents, Alarms, Logs, Metrics, X-Ray traces, Events, and Application Signals. The main content area displays the 'Clusters state summary (1)' for the `eks_cluster`. The summary shows the cluster's utilization (99% CPU, 53% Memory) and alarm states per resource type (Cluster, Node, Namespace, Service, Workload, Pod, Container). The right sidebar contains the 'Performance and status summary' for the last 1 minute, showing Clusters CPU % (Utilization 96, Reserved 75) and Clusters Memory % (Utilization 41, Reserved 34). The bottom section shows the 'Control plane summary' for the last 3 hours, displaying metrics for Max API server requests, Average API server requests latency, Total number of stored objects, and Average admission controller latency.

The design of your app and other AWS services to be consumed (or would like to consume if not available on the free tier)

I intend to employ virtual machines with a robust configuration, such as the T2.LARGE instance type, which possesses significant capabilities to expedite the processing.

If this was to be implemented in a production environment, which other services would you have preferred to include

In a production environment, I would optimize the virtual machine configuration to maximize processing speed and deploy efficient monitoring tools like Prometheus or Grafana to enhance observability. As an illustration, for a Jenkins server, I would utilize a Jenkins Master and Jenkins Slave with the T2.LARGE instance type to ensure faster processing.

Your decisions on securing your service

In a DevOps context, secure services by integrating security into CI/CD pipelines, practicing secure coding in infrastructure as code, implementing access controls, scanning containers for vulnerabilities, managing secrets securely, enabling robust logging, patching systems regularly, planning for incident response, enforcing network security, and fostering a security-aware culture through education and collaboration.

Testing your microservice

Test microservices in Kubernetes by implementing unit tests for individual services, integration tests for service interactions, and end-to-end tests for overall functionality.

What types of observabilities would be useful for a productions service like this one.

For a productionized service, essential observabilities include logging for detailed records, metrics for performance monitoring, tracing for request tracking, and alerting for proactive issue detection. Continuous monitoring of infrastructure, deployment pipelines, and security is crucial for maintaining reliability and ensuring rapid incident response in a DevOps environment.

A list of nice haves that you didn't implement because of time or cost constraints.

A list of nice haves that can't implement because of time or cost constraints

- 1) Serverless adoption for scalability.
- 2) Comprehensive automated testing expansion.
- 3) Full disaster recovery setup.
- 4) Advanced monitoring and analytics tools.
- 5) Infrastructure cost optimization.
- 6) Fine-grained IAM policies.

What are the limitations and opportunities to consider when scaling your solution as the usage of your web-application grows.

Limitations: Scaling introduces complexity, potential higher costs, skill set challenges, tooling limitations, and security concerns.

Opportunities: Improved availability, performance optimization, resource efficiency, automated scalability, enhanced collaboration, innovation, flexibility, global reach, continuous improvement, and support for business growth.