**Components of ELK**

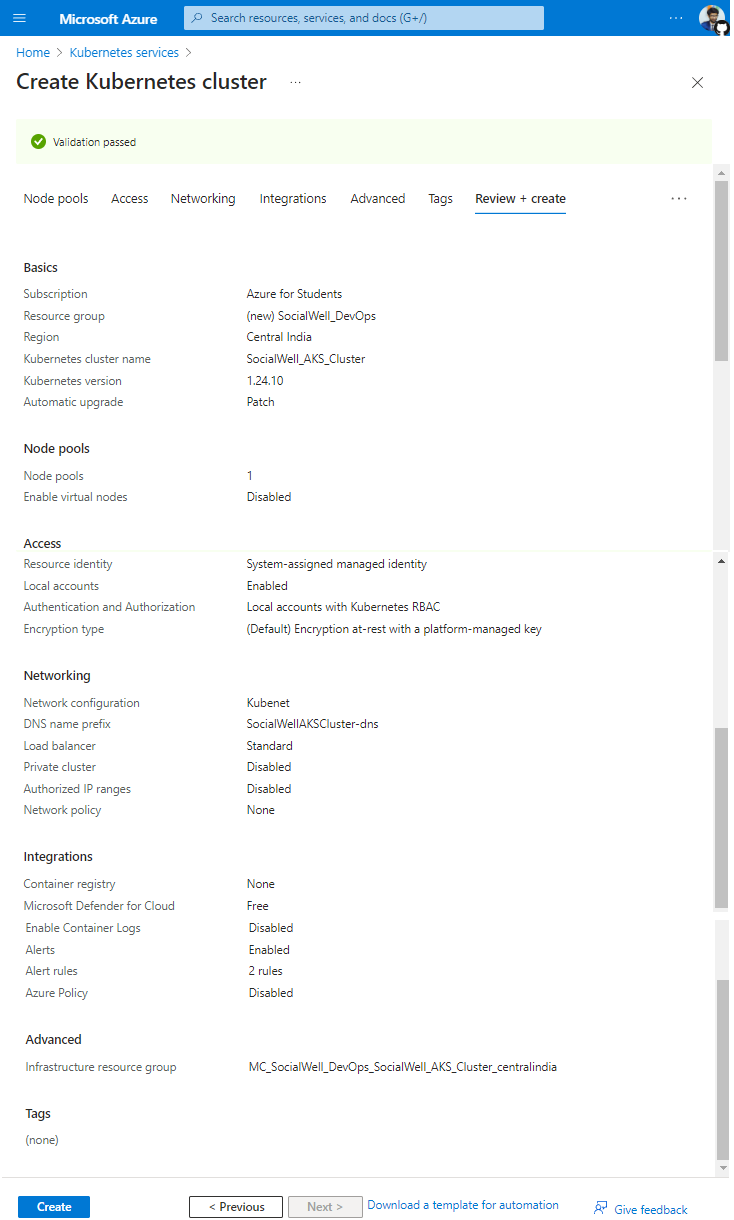
ELK stands for Elasticsearch, Logstash, and Kibana, and is a popular open-source software stack used for log management and analysis. Here are the main components of ELK:

1. **Elasticsearch**: It's a distributed search and analytics engine that stores and indexes data in real-time. It provides a RESTful API for querying and retrieving data from the Elasticsearch cluster.
2. **Logstash**: It's a data collection pipeline that ingests data from various sources, transforms it, and sends it to Elasticsearch for indexing. Logstash supports a wide range of input plugins, filter plugins, and output plugins to handle different types of data.
3. **Kibana**: It's a web-based user interface that allows users to visualize and analyse data stored in Elasticsearch. Kibana provides a variety of visualization options, including bar charts, line graphs, pie charts, and heat maps, among others.

**Resources and Configurations Required in AKS to Run ELK:**

AKS stands for Azure Kubernetes Service, which is a managed Kubernetes service offered by Microsoft Azure. Here's resources of ELK in AKS:

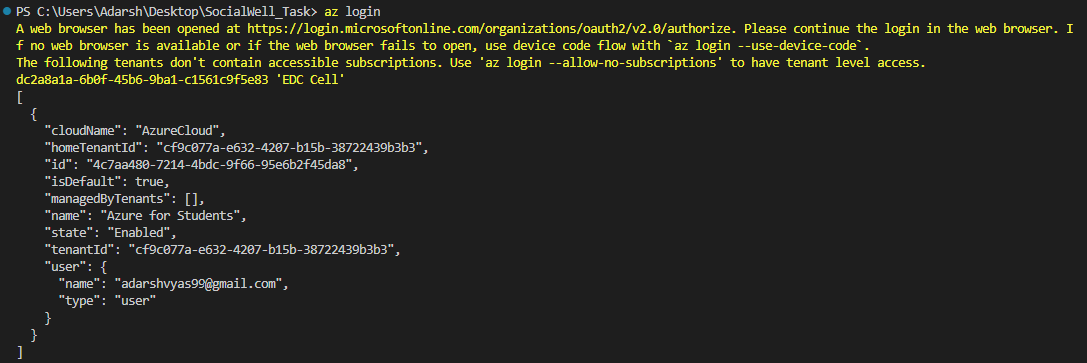
1. **AKS Cluster**: The first component is an AKS cluster that provides a managed Kubernetes environment for running ELK. Or at least two virtual machines are required for a minimum AKS cluster configuration, one for the Kubernetes control plane (Master) and one for the worker nodes. The size and number of VMs required depends on the number of nodes, the size of data to be processed, and the number of concurrent users.
2. **Elasticsearch Deployment**: The second component is the Elasticsearch deployment, which consists of multiple Elasticsearch nodes that are distributed across different Kubernetes nodes. Each Elasticsearch node has its own persistent volume for storing data.
3. **Logstash Deployment**: The third component is the Logstash deployment, which consists of multiple Logstash nodes that are distributed across different Kubernetes nodes. Each Logstash node is configured to read data from a specific input source, transform it, and send it to Elasticsearch.
4. **Kibana Deployment**: The fourth component is the Kibana deployment, which consists of a single Kibana node that provides a web-based user interface for visualizing and analyzing data stored in Elasticsearch.
5. **Ingress Controller**: The fifth component is an Ingress Controller that allows external users to access Kibana through a single IP address. The Ingress Controller is responsible for routing incoming requests to the appropriate Kibana pod.
6. **Load Balancer**: The sixth component is a Load Balancer that distributes incoming traffic across multiple Elasticsearch and Logstash nodes to ensure high availability and scalability.
7. **Config Maps and Secrets**: The seventh component is a set of Config Maps and Secrets that store configuration data and sensitive information, such as passwords and API keys, used by ELK components. ConfigMaps can be used to store Logstash and Kibana configuration files. Secrets are used to store sensitive information such as passwords and encryption keys. In ELK, secrets can be used to store Elasticsearch passwords and TLS certificates.
8. **Storage**: For data storage, a Persistent Volume Claim (PVC) is required for each Elasticsearch node. The size of PVC depends on the size of data to be stored and processed.



I have used Node count range of 1 because I am using Azure student subscription. Autoscaling may fail above 1 nodes due to quota restrictions (4 cores remaining).

**Configure AKS**: Set up a new AKS cluster using the Azure CLI by running the following commands:

# az login



# az account set --subscription <subscription\_id>



Next step is to create resource group and AKS cluster which can be done either by CLI or GUI:

1. By using CLI

Create a resource group to hold the AKS cluster:

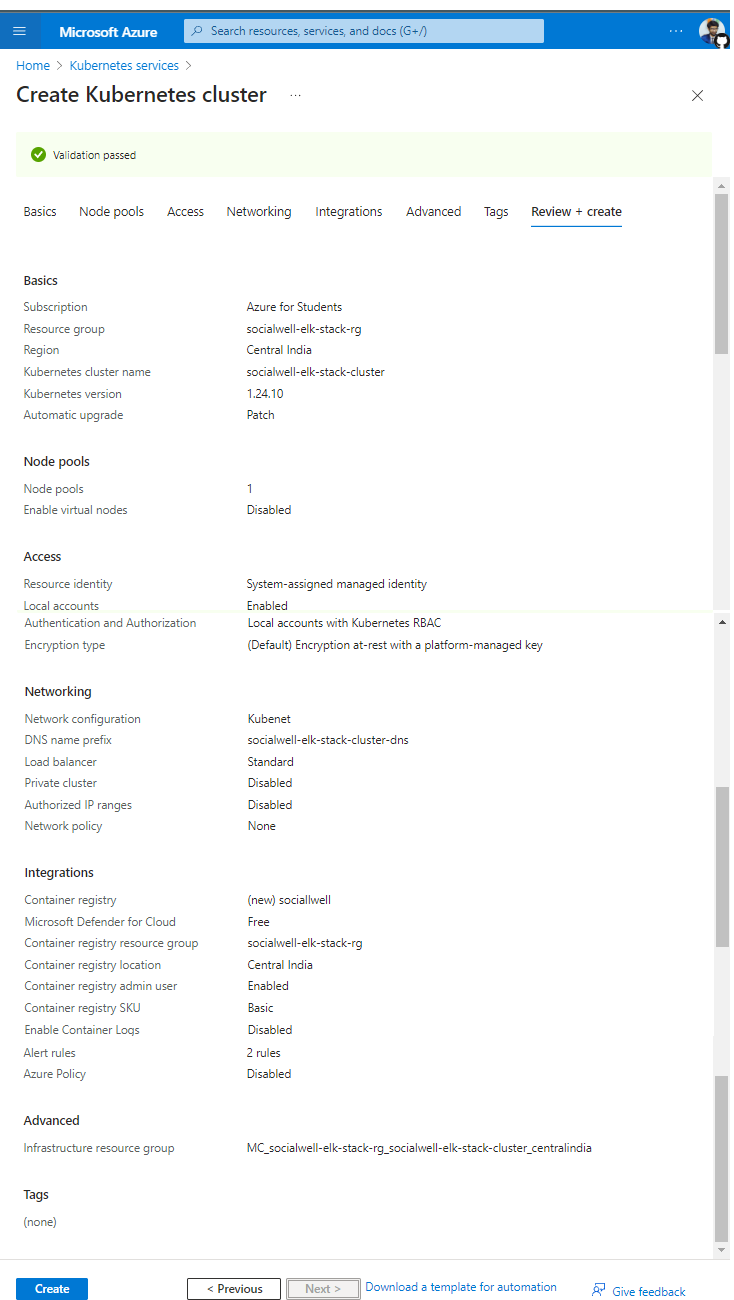
# az group create --name socialwell-elk-stack-rg --location centralindia

Create an AKS cluster with the default settings:

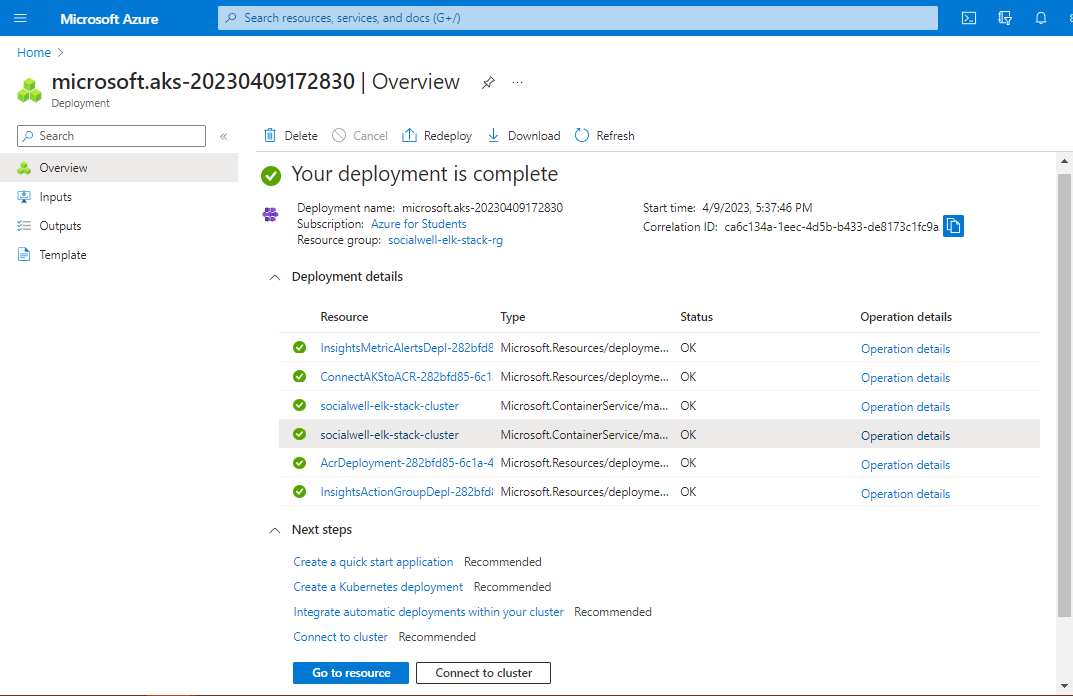
# az aks create --name socialwell-elk-stack-cluster --resource-group socialwell-elk-stack-rg --node-count 1 --generate-ssh-keys

OR

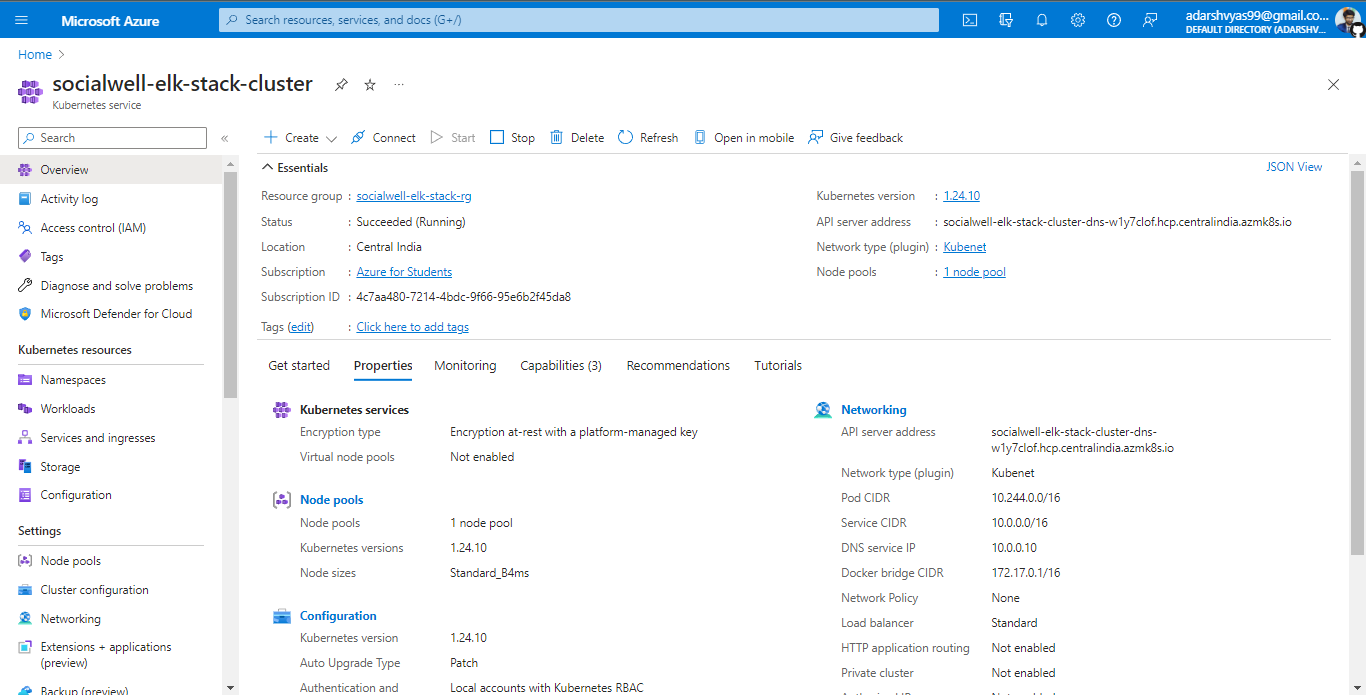
1. By using GUI



Once we click on create button we then soon able to see the deployment progressed and succussed.

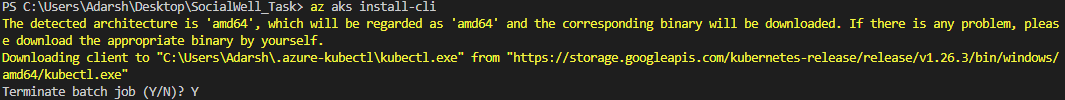


We can now see the details of cluster.



Now since the cluster is created, we need to connect it with our local development machine to deploy and control.

I have kubectl client already installed in my machine to communicate with control plane as I use it for minikube. But azure also provides it to install it if we don’t have it already installed.

**Configure kubectl**: Install and configure kubectl to manage your AKS cluster by running the following command:  
# az aks install-cli  


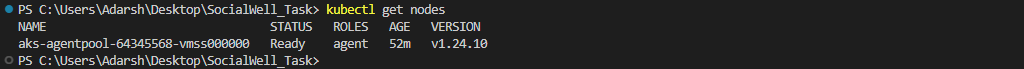
**Connect to the AKS cluster:**

# az aks get-credentials --resource-group socialwell-elk-stack-rg --name socialwell-elk-stack-cluster



**Verify that you can connect to the AKS cluster:**

# kubectl get nodes



**Deploying the ELK Stack**

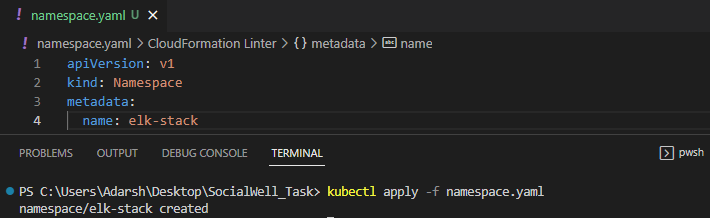
**Create Kubernetes manifests for ELK stack**

Next, we need to create Kubernetes manifests for the ELK stack components, which include Elasticsearch, Logstash, and Kibana.

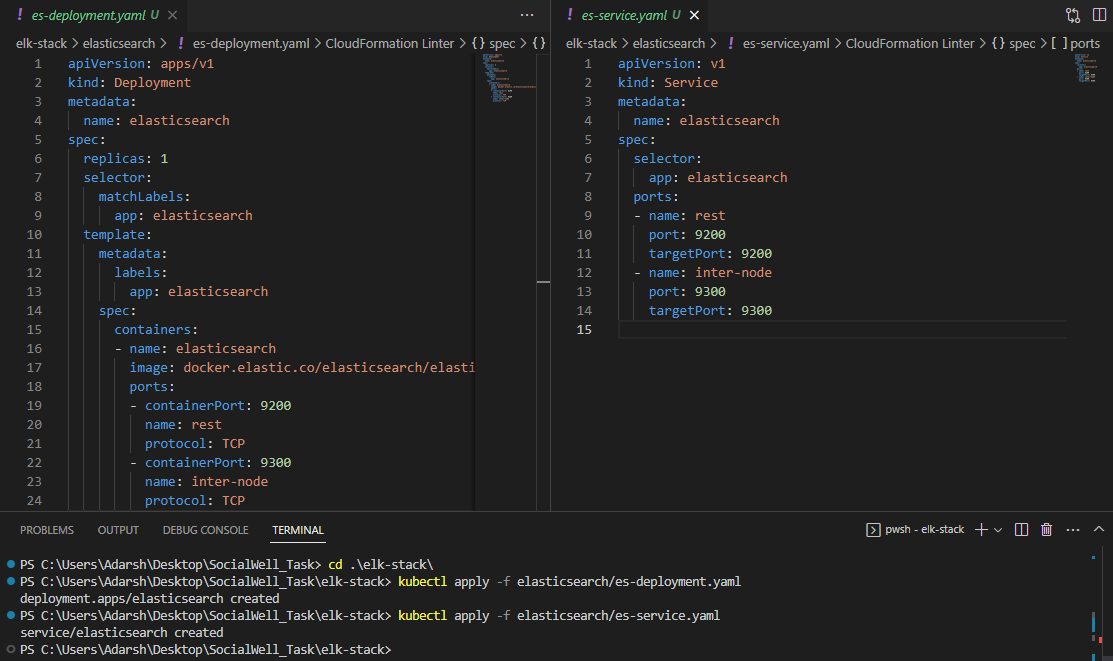
**Create a namespace**

To isolate the ELK stack from other applications, we can create a separate namespace for it. To create a namespace, create a file named "namespace.yaml" with the following contents and then, apply the manifest using the following command:

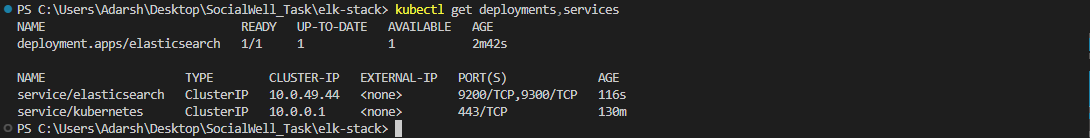
# kubectl apply -f namespace.yml



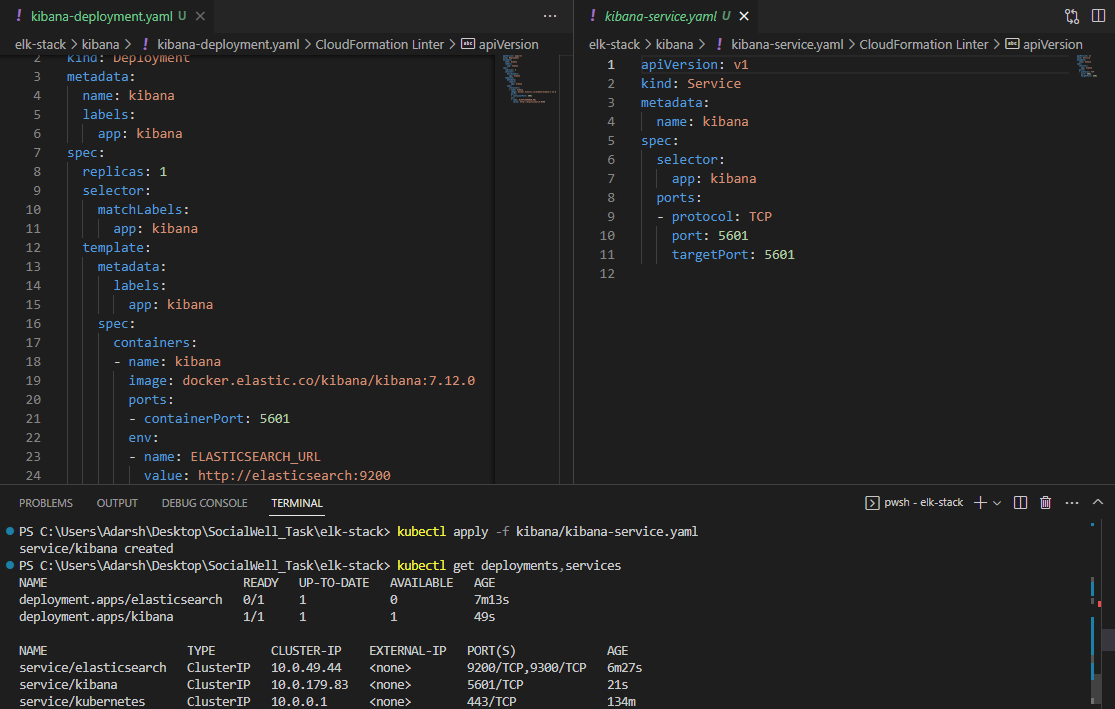
**Create Elasticsearch Deployment and Service**



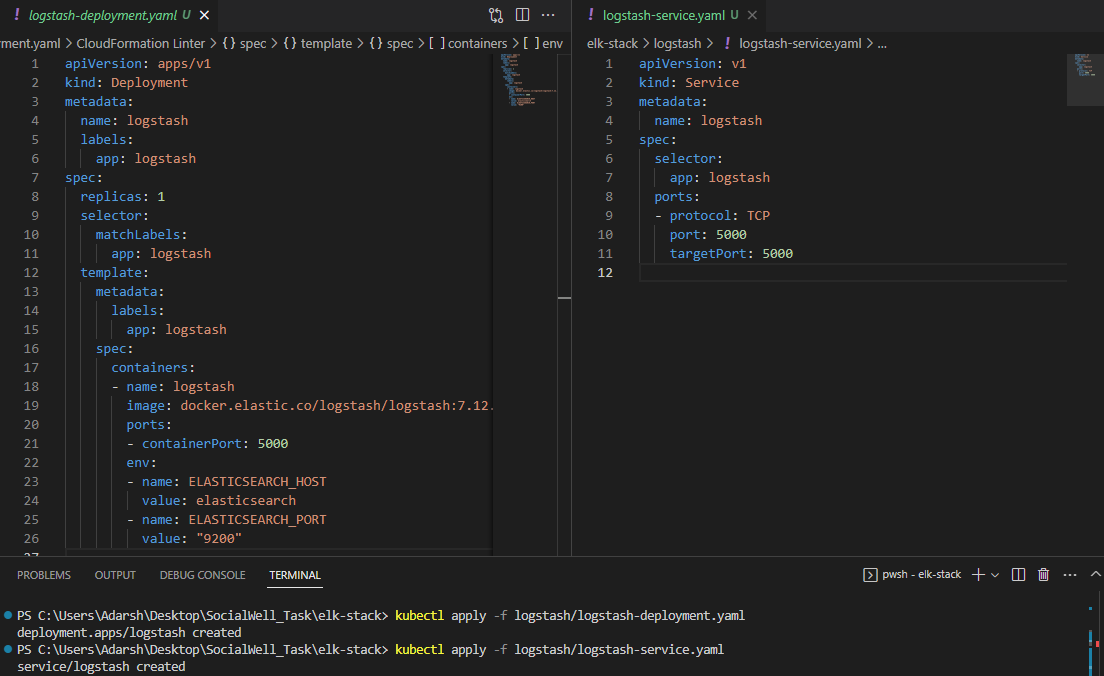
Now let’s check once that what we did was done successfully.



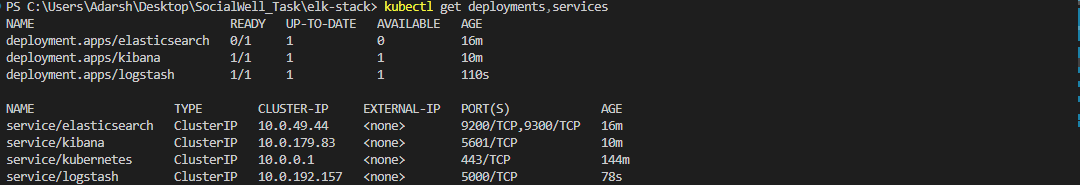
**Create Kibana Deployment and Service**

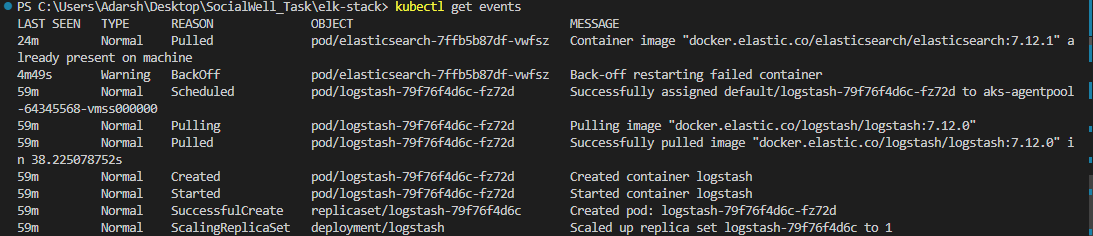


**Create Logstash Deployment and Service**

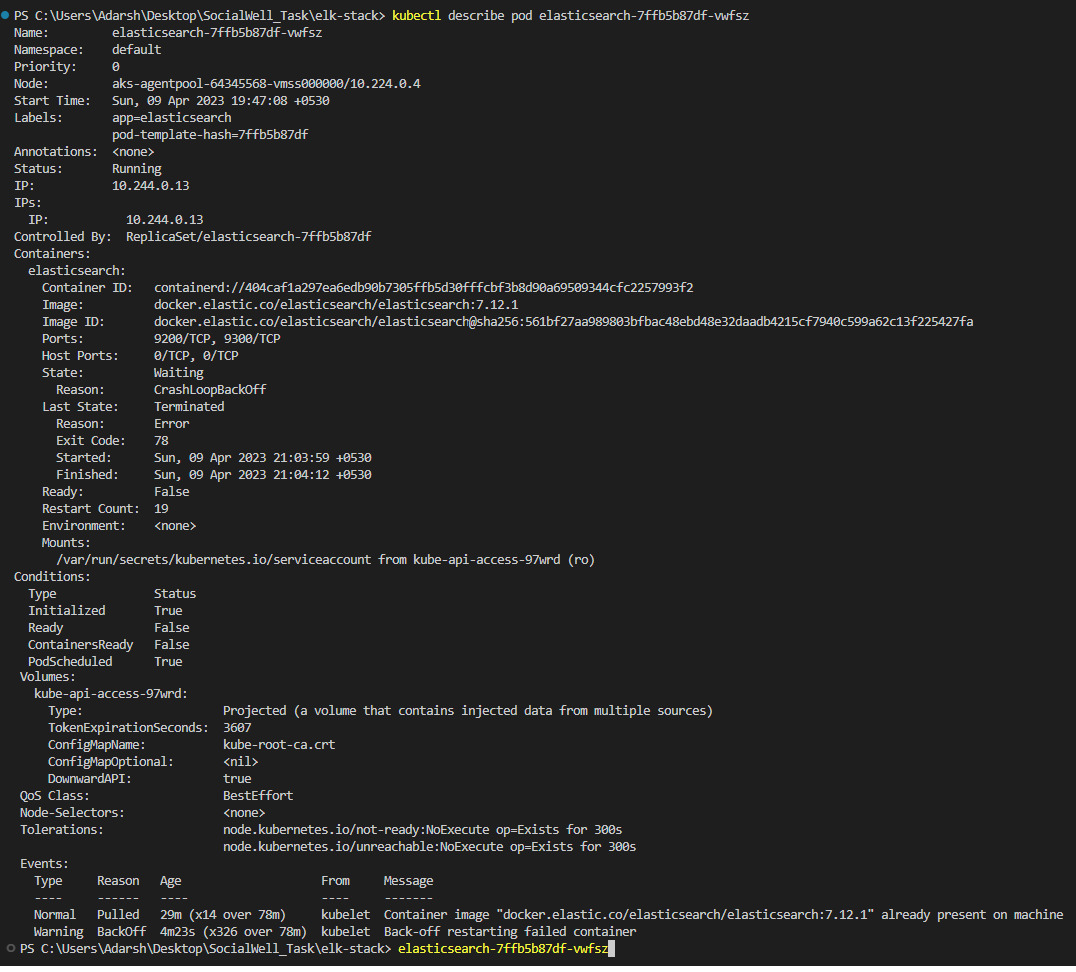
****

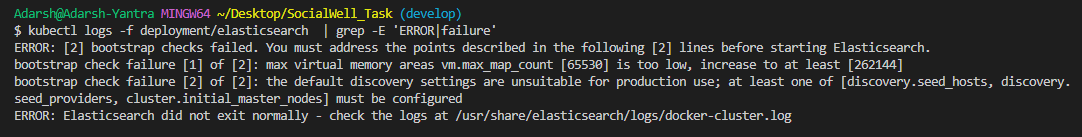
Logstash is deployed but there is some challenge with Elasticsearch pod as it is not ready even after several restarts. Let’s troubleshoot it.







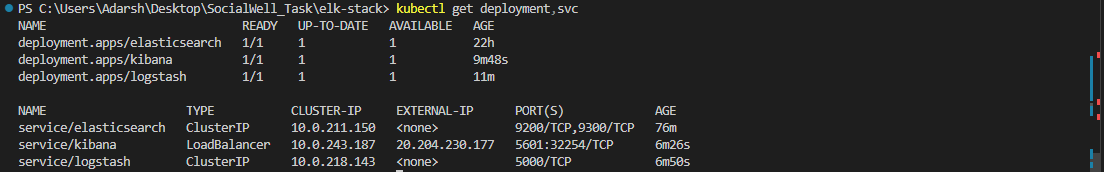




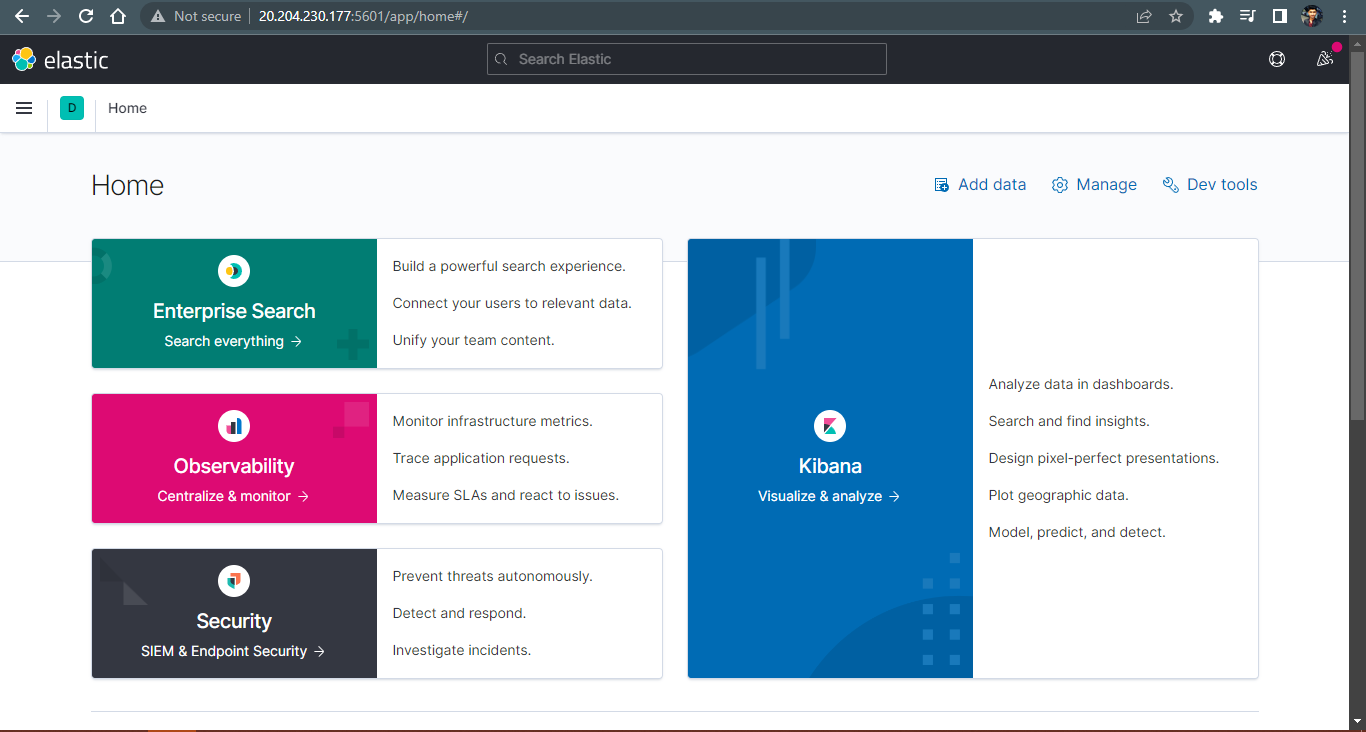
As I can see from deployment log of elasticsearch, this error is because of insufficient virtual memory. We can manually go into the container can set the proper limit there but it will not work later when the new container was launched. Therefore I have to define resources in my manifest file and relaunch it.

Also, now I will relaunch the resources in elk-stack namespace that I had created before.





Great, we can now access kibana dashboard from the service External IP.



Create a ConfigMap for the Elasticsearch configuration:

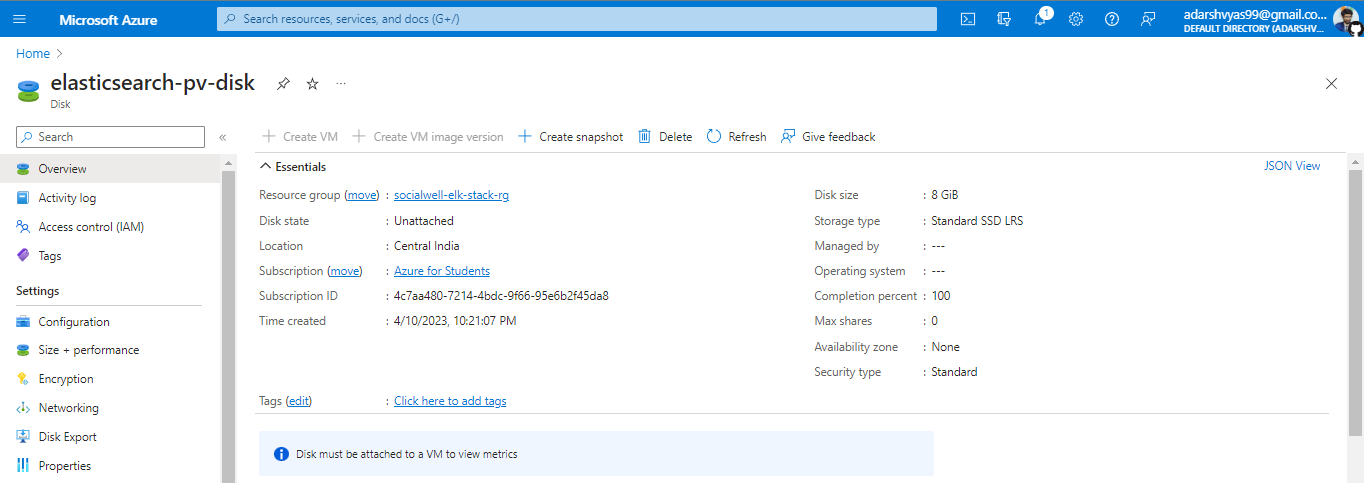


Create a Secret for the Elasticsearch password:

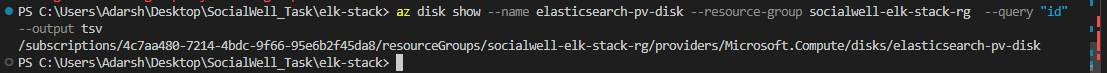


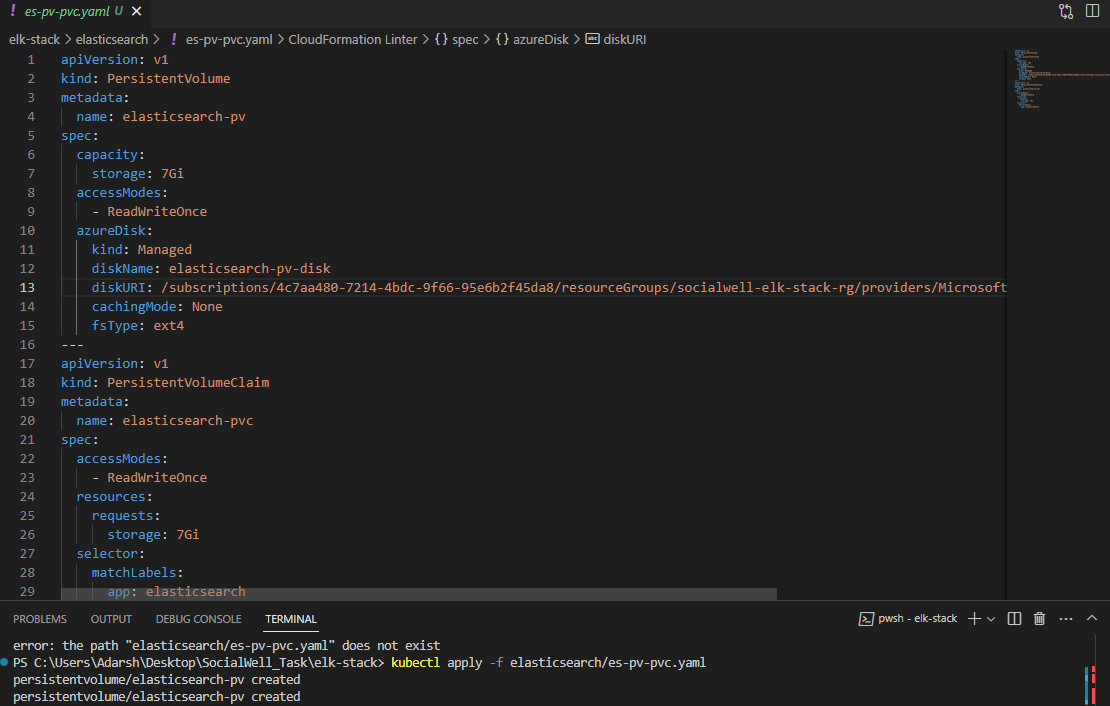
Create a Persistent Volume and Persistent Volume Claim for Elasticsearch:

For this task, I had created 8 GiB disk and copied its resource id from properties that we are going to use for provisioning persistent volume.

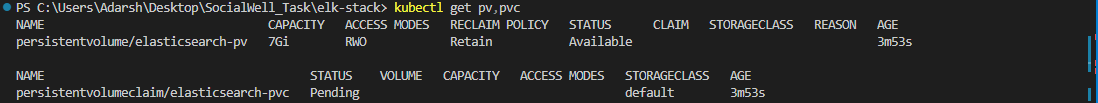


This is another way to get resource id or URI of disk.



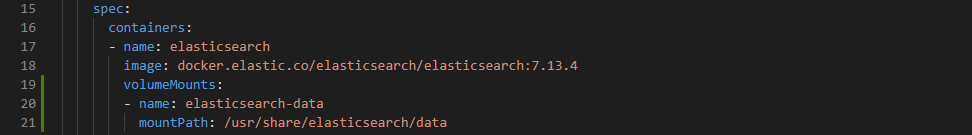


Okay so pv and pvc resources are created but pvc is showing the pending state.



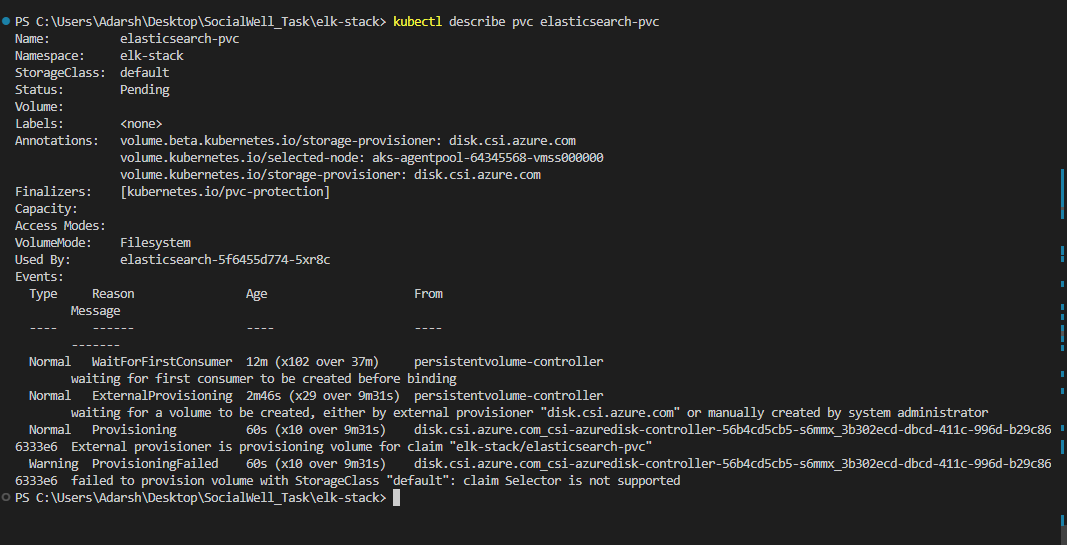
This is a normal behavior of Kubernetes, as it ensures that the storage resources are not unnecessarily provisioned until they are actually needed. Once a Pod requests the PersistentVolumeClaim, the control plane will bind the claim to an available PersistentVolume and mount it to the Pod.

Obviously, I need to update elasticsearch manifest to do PersistentVolumeClaim.

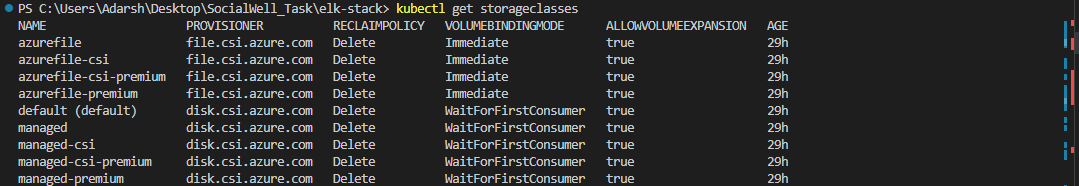




Even after updating, I am getting same error. So lets describe it once to see events.



Okay this is because I haven’t provided storage class in PVC configuration and by default it is using default sc which is getting mismatched while provisioning.



Opss, I don’t have azure-disk external provisioner driver installed. So I have applyed the below yaml file to create azure-disk storage class resource. And also added theStorageClassName in PVC manifest.

