Lab 2.2:   
Kubernetes – Tasks

The goal of this laboratory is to learn the fundamentals of the container management technology Kubernetes. In this laboratory you are going to deploy the full case study on a Kubernetes cluster which you install on your own. (Check the installation document for instructions on how to setup the cluster).

Your group will get a set of virtual machines which they will use to setup the cluster. You will find the IPs, user, and password in a file in your group on Ilias, please connect to the VMs using VPN or via FB4STUD network. If you don’t get your machines, or have some issues with accessing them, please reach out to the teaching team.

Overview:

[Student Group Information 1](#_heading=h.30j0zll)

[Tasks 2](#_heading=h.1fob9te)

[1.1. VM Configuration 2](#_heading=h.3znysh7)

[Configuration & Basics 2](#_heading=h.2et92p0)

[2. Enhance your workflow & Basic Tools 3](#_heading=h.tyjcwt)

[3. Networking Modes 3](#_heading=h.3dy6vkm)

[4. Case Study 4](#_heading=h.1t3h5sf)

[5. Reflective Tasks 5](#_heading=h.4d34og8)

[Conclusion 7](#_heading=h.2s8eyo1)

[Outlook 7](#_heading=h.17dp8vu)

# Student Group Information

Enter your student group information in the following table and provide your group number.   
(This number will be important for later labs as well, please remember it!)

**Machine-IP’s**:

**GroupNumber (e.g. Group01):** Group\_\_

|  |  |
| --- | --- |
| **Member Name** | **Matriculation Number** |
| **Nimisha Vilayatarani** | **7219293** |
| **Eraj Rizvi** | **7219193** |
| **Chowdhury Abida Anum Era** | **7219089** |

# Tasks

This section contains the questions you have to answer. Those answers are required to pass the lab acceptance session. Please make sure that you first install VirtualBox and set-up your VM (document “Lab 1.1 VirtualBox - Installation Instructions") before you tackle these tasks.

## 1.1. VM Configuration

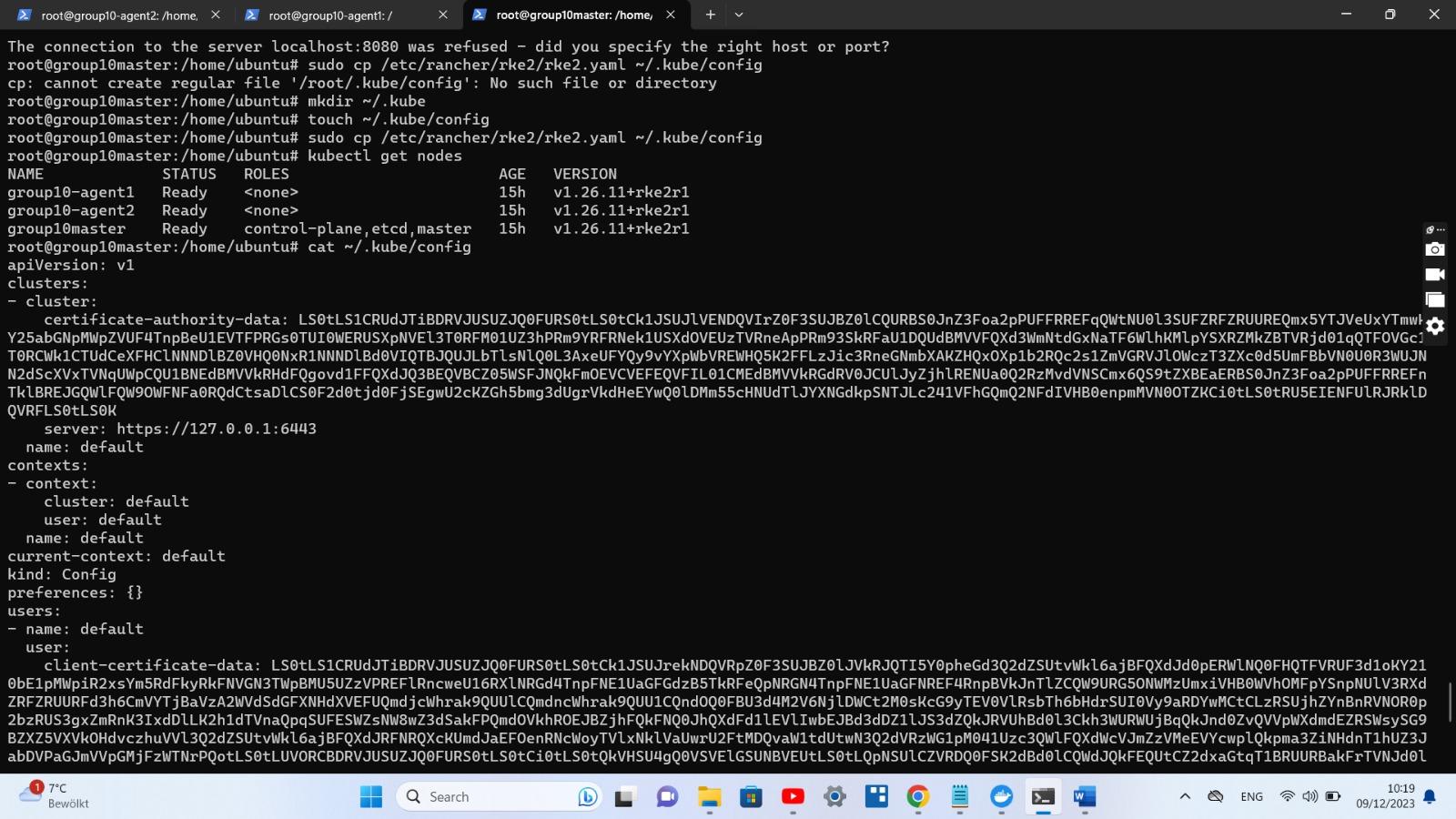
In this chapter you are going to get used to the basics of Kubernetes.

### Configuration & Basics

1.1. Explain the contents of the Kubernetes-Configuration-File on your local pc

The Kubernetes configuration file, often referred to as kubeconfig, is a YAML file that contains information about how to connect to a Kubernetes cluster. This file is used by the kubectl command-line tool to communicate with the cluster's API server. Here are the key elements and sections typically found in a Kubernetes configuration file:

* apiVersion:
  + Specifies the version of the Kubernetes API to use. It's usually set to v1.
* kind:
  + Specifies the type of resource. In a Kubernetes configuration file, it is usually set to Config.
* preferences:
  + Optional. It specifies preferences for specific behaviors.
* clusters:
  + Contains a list of clusters that can be accessed. Each cluster has a name, a server (the address of the Kubernetes API server), and an optional certificate-authority field (path to a certificate file for the server's SSL certificate).
* users:
  + Contains a list of users and their authentication credentials. Users can have a client-certificate and client-key for client certificate authentication, or username and password for basic authentication.
* contexts:
  + Contains a list of contexts. A context combines a cluster, user, and namespace into a single context. This is useful when working with multiple clusters or switching between different users or namespaces.
* current-context:
  + Specifies the default context that kubectl should use when no other context is specified.
* ApiVersion:
  + Specifies the version of the Kubernetes API to use. It's usually set to v1.
* kind:
  + Specifies the type of resource. In a Kubernetes configuration file, it is usually set to Config.
* preferences:
  + Optional. It specifies preferences for specific behaviors.
* clusters:
  + Contains a list of clusters that can be accessed. Each cluster has a name, a server (the address of the Kubernetes API server), and an optional certificate-authority field (path to a certificate file for the server's SSL certificate).
* users:
  + Contains a list of users and their authentication credentials. Users can have a client-certificate and client-key for client certificate authentication, or username and password for basic authentication.
* contexts:
  + Contains a list of contexts. A context combines a cluster, user, and namespace into a single context. This is useful when working with multiple clusters or switching between different users or namespaces.
* current-context:
  + Specifies the default context that kubectl should use when no other context is specified.



1.2. Which concepts, that you already know from docker, are also present in Kubernetes?

Kubernetes builds upon containerization concepts introduced by Docker but extends and orchestrates containerized applications at a higher level. Here are some concepts present in both Docker and Kubernetes:

1. **Container:**
   * **Docker:** A lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
   * **Kubernetes:** Uses container runtimes like Docker to encapsulate applications into containers. Containers in Kubernetes are the basic unit of deployment.
2. **Image:**
   * **Docker:** A snapshot of a file system and the parameters needed to run a container.
   * **Kubernetes:** Applications run in containers, and these containers are created from container images. Container images are stored in a container registry and pulled by Kubernetes when needed.
3. **Pod:**
   * **Docker:** A single container running in an isolated environment.
   * **Kubernetes:** The smallest deployable unit in Kubernetes. It can contain one or more containers that share the same network namespace and storage. Pods are scheduled onto nodes in a Kubernetes cluster.
4. **Service:**
   * **Docker:** Docker Compose can create services, which define how containers behave in production.
   * **Kubernetes:** A Kubernetes Service is an abstraction that defines a logical set of Pods and a policy by which to access them. It provides a stable IP address and DNS name, enabling other services to discover and communicate with it.
5. **Volume:**
   * **Docker:** Docker Volumes are used to persist data outside the container's lifecycle.
   * **Kubernetes:** Kubernetes Volumes are used to persist data across Pod restarts. They can be backed by various storage solutions and enable data sharing between containers in the same Pod.
6. **Networking:**
   * **Docker:** Docker provides its networking model for communication between containers.
   * **Kubernetes:** Kubernetes manages networking between Pods, allowing them to communicate with each other. It provides a flat network for containers across nodes.
7. **Environment Variables:**
   * **Docker:** Environment variables can be set at runtime to configure container behavior.
   * **Kubernetes:** Environment variables can be set in Pod specifications, and they are injected into containers at runtime.
8. **Health Checks:**
   * **Docker:** Health checks can be defined in a Dockerfile to determine the container's health.
   * **Kubernetes:** Kubernetes supports liveness and readiness probes to check the health of containers. These probes can be configured in the Pod specification.

While Kubernetes inherits container concepts from Docker, it introduces additional abstractions and features to manage and orchestrate containerized applications at scale. Kubernetes is designed to automate the deployment, scaling, and management of containerized applications in a clustered environment.

1.3. Which methods are there to create resources in Kubernetes?   
(Explain the difference and list some pro’s and con’s for each)

In Kubernetes, there are multiple methods to create and manage resources, each with its own advantages and disadvantages. Here are some common methods:

1. **Imperative Commands:**
   * **Description:** This method involves using imperative **kubectl** commands to directly create or modify resources.
   * **Pros:**
     1. Quick and easy for simple tasks.
     2. Suitable for one-off or debugging scenarios.
   * **Cons:**
     1. Not declarative, making it harder to track changes.
     2. Commands might become complex for more sophisticated setups.
   * **Use Case:**
     1. Quick testing or debugging.
2. **Declarative Configuration with kubectl apply:**
   * **Description:** This method involves creating or updating resources using YAML or JSON configuration files and applying them with **kubectl apply**.
   * **Example:**
   * **Pros:**
     1. Declarative approach allows tracking and versioning of configurations.
     2. Suitable for configuration management tools and automation.
   * **Cons:**
     1. May require a more complex setup for managing configurations.
   * **Use Case:**
     1. Configuration management, automation, and version control.
3. **Helm:**
   * **Description:** Helm is a package manager for Kubernetes that simplifies the deployment and management of Kubernetes applications.
   * **Example:**
   * **Pros:**
     1. Templating and packaging for managing complex applications.
     2. Versioning and rollback support.
   * **Cons:**
     1. Requires additional setup and understanding.
     2. Might be overkill for simple applications.
   * **Use Case:**
     1. Managing and deploying complex applications with dependencies.
4. **Kustomize:**
   * **Description:** Kustomize is a built-in tool in **kubectl** that allows customization of Kubernetes manifests.
   * **Example:**

bashCopy code

kubectl apply -k ./myapp

* + **Pros:**
    1. Declarative approach with configuration customization.
    2. Integration with **kubectl** makes it easy to use.
  + **Cons:**
    1. Slightly steeper learning curve.
    2. May not cover all use cases.
  + **Use Case:**
    1. Customizing base configurations for different environments.

1. **Operators:**
   * **Description:** Kubernetes Operators are a method of packaging, deploying, and managing applications using native Kubernetes tools.
   * **Example:** Operators are typically defined by custom resources (CRs) and controllers.
   * **Pros:**
     1. Enables application-specific lifecycle management.
     2. Automates complex operational tasks.
   * **Cons:**
     1. Requires development and understanding of custom controllers.
     2. More overhead for simple applications.
   * **Use Case:**
     1. Managing and automating complex, stateful applications.
2. **APIs and Custom Controllers:**
   * **Description:** Writing custom controllers using the Kubernetes API and the client libraries.
   * **Example:** Developing a custom controller in a programming language like Go using the Kubernetes client libraries.
   * **Pros:**
     1. Full control and flexibility in managing resources.
     2. Enables automation of specific tasks.
   * **Cons:**
     1. Requires programming skills and development effort.
     2. Higher learning curve compared to other methods.
   * **Use Case:**
     1. Building custom, domain-specific controllers for specific needs.

1.4. What is the equivalent to a docker container in Kubernetes?

The equivalent to a Docker container in Kubernetes is a **Pod**.

A Pod is the smallest deployable unit in Kubernetes and represents a single instance of a running process in a cluster. A Pod can encapsulate one or more containers, which are co-located and share the same network namespace, allowing them to communicate with each other using **localhost**. Containers within the same Pod also share the same storage volume.

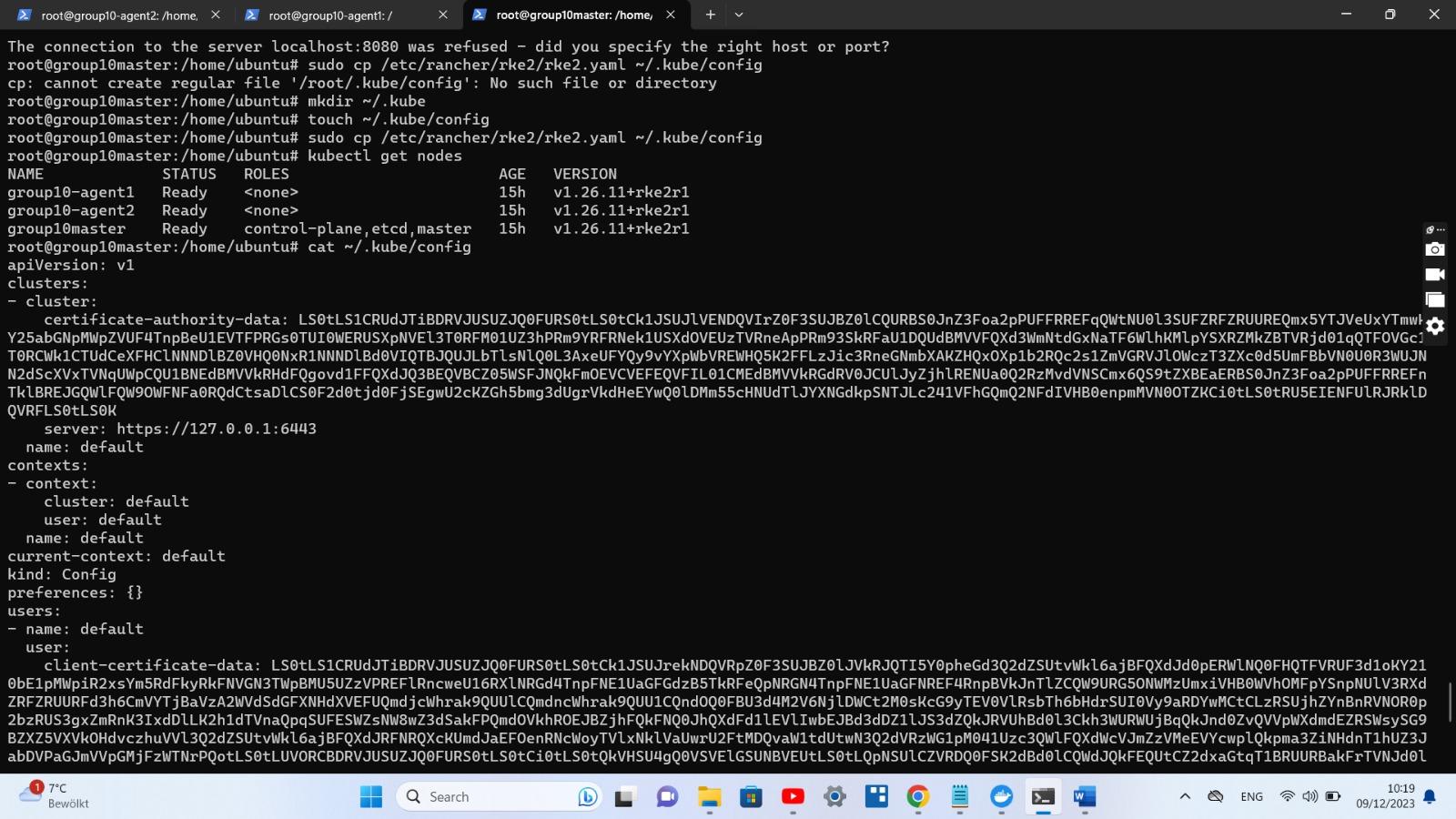
While Docker containers are standalone and typically represent a single application or service, a Kubernetes Pod may contain multiple containers that work together as a single unit. This enables applications with tightly coupled components to be deployed and managed together.

1.5. How can you connect to your cluster? (Also think of the config-file containing more than one cluster)

To connect to your Kubernetes cluster, you typically use the kubectl command-line tool. Here's how you can configure and use kubectl to connect to your cluster:

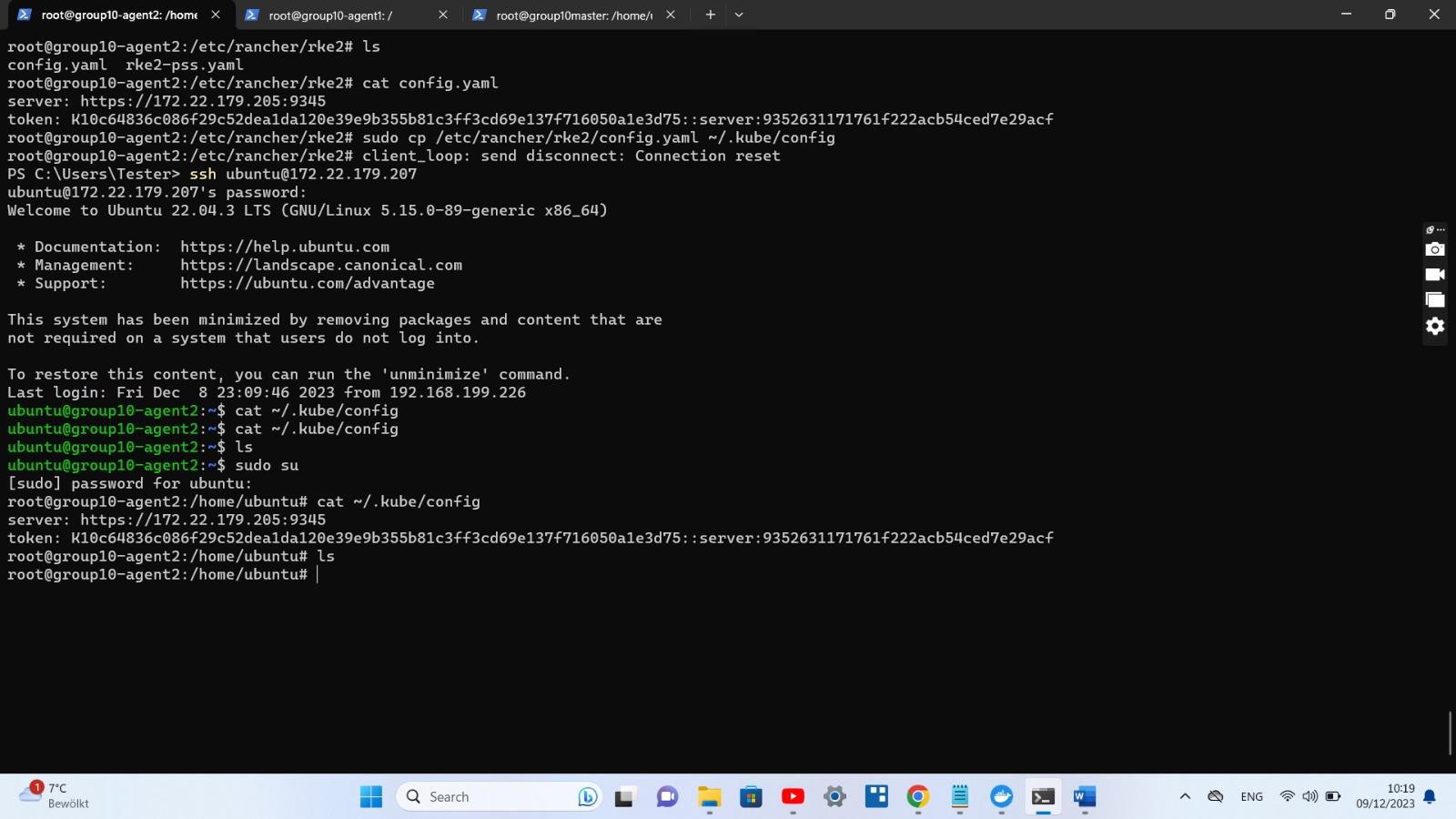
Master:

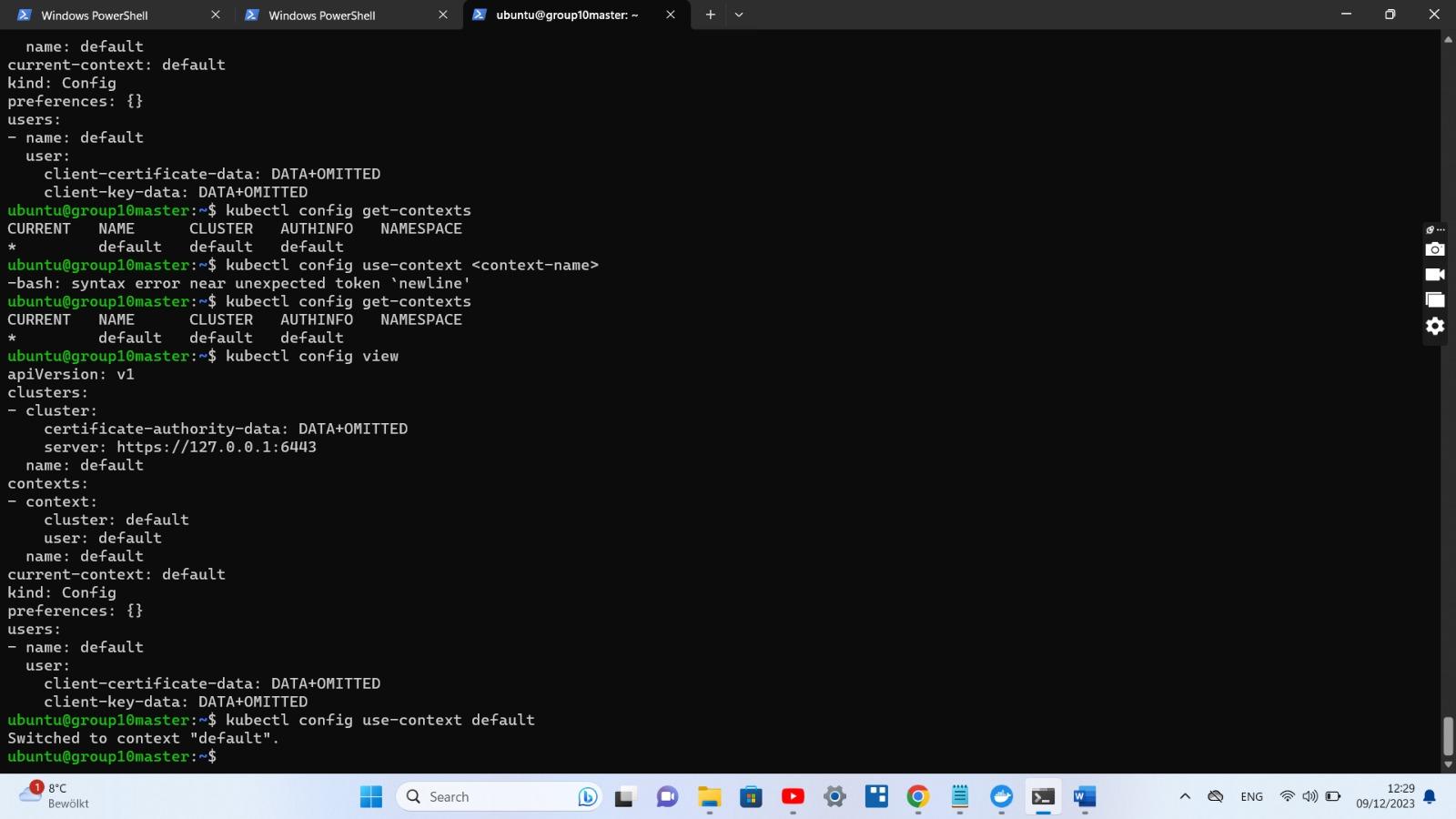
1. go to the ~/.kube/config
2. Also get the token from node-token file so that we can add it on the agents to connect with the cluster.

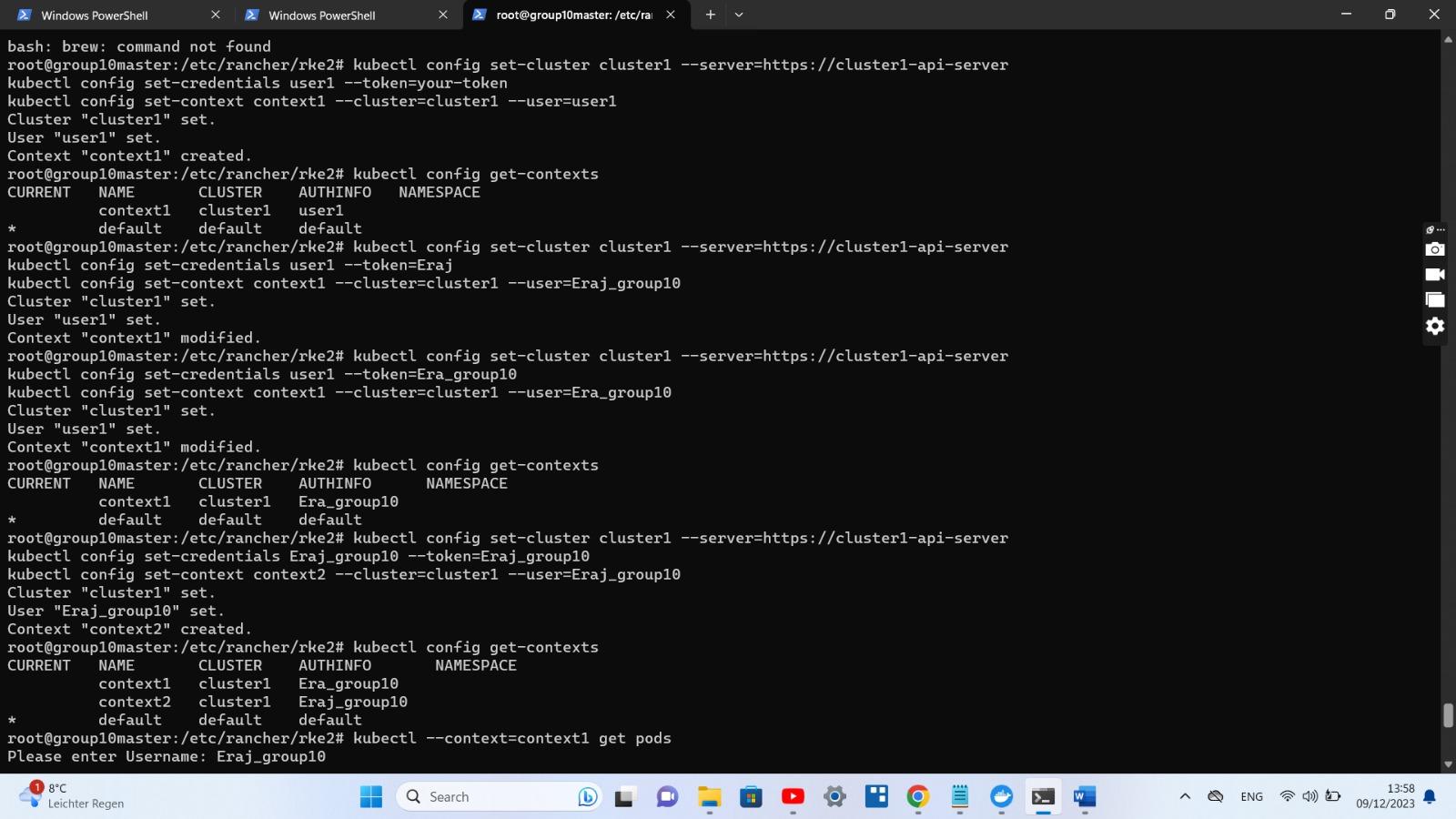


Agents:

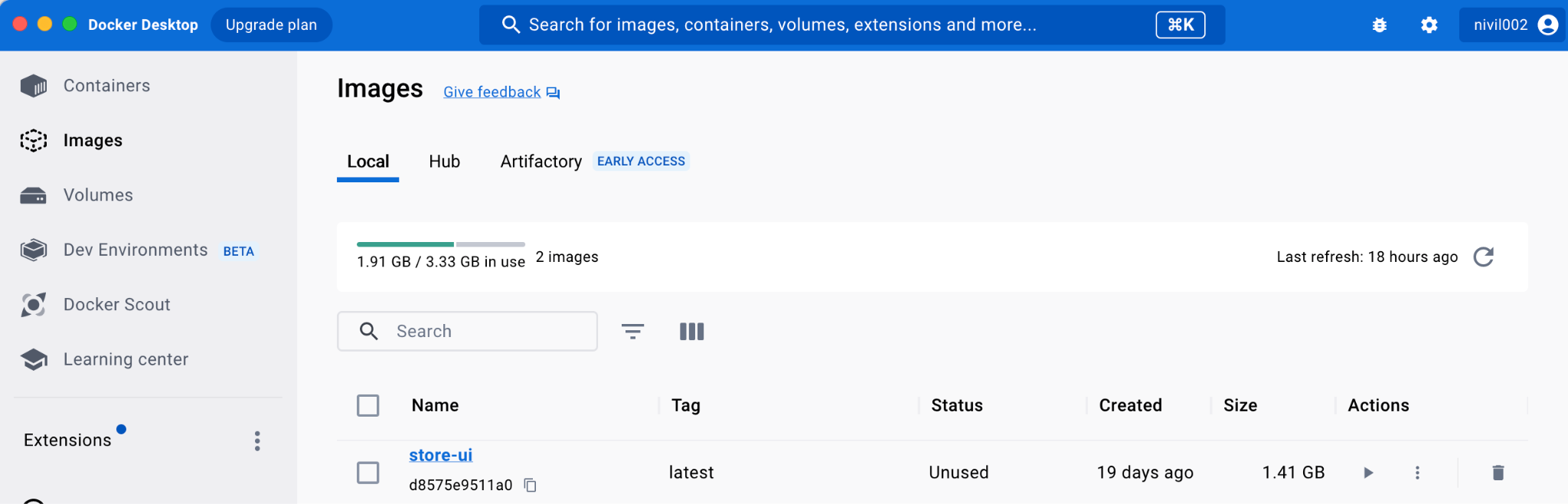
1. In agent, we have created the config file in /etc/rancher/rke2/de
2. Added the **server and token** in this agent’s config file

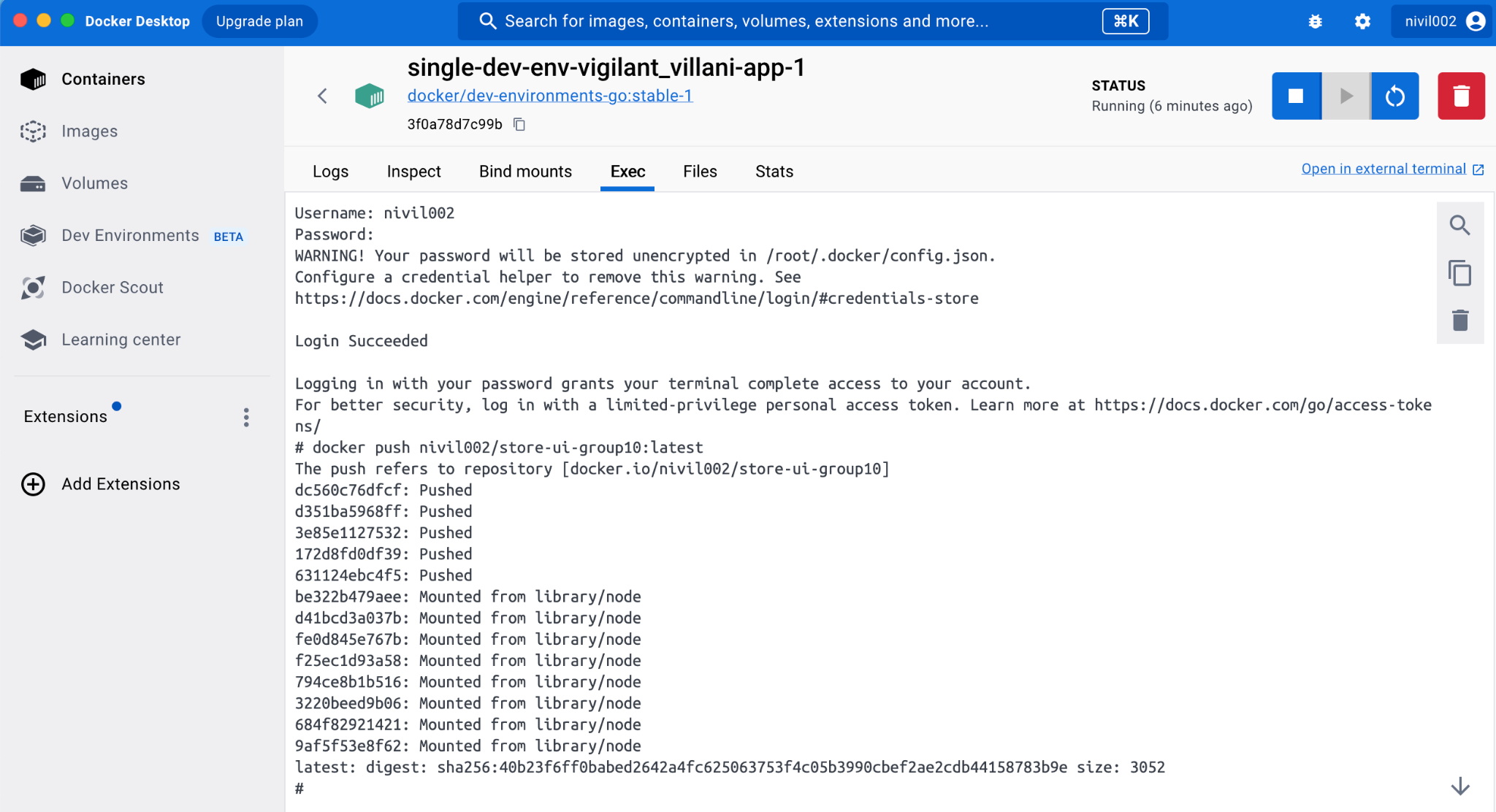


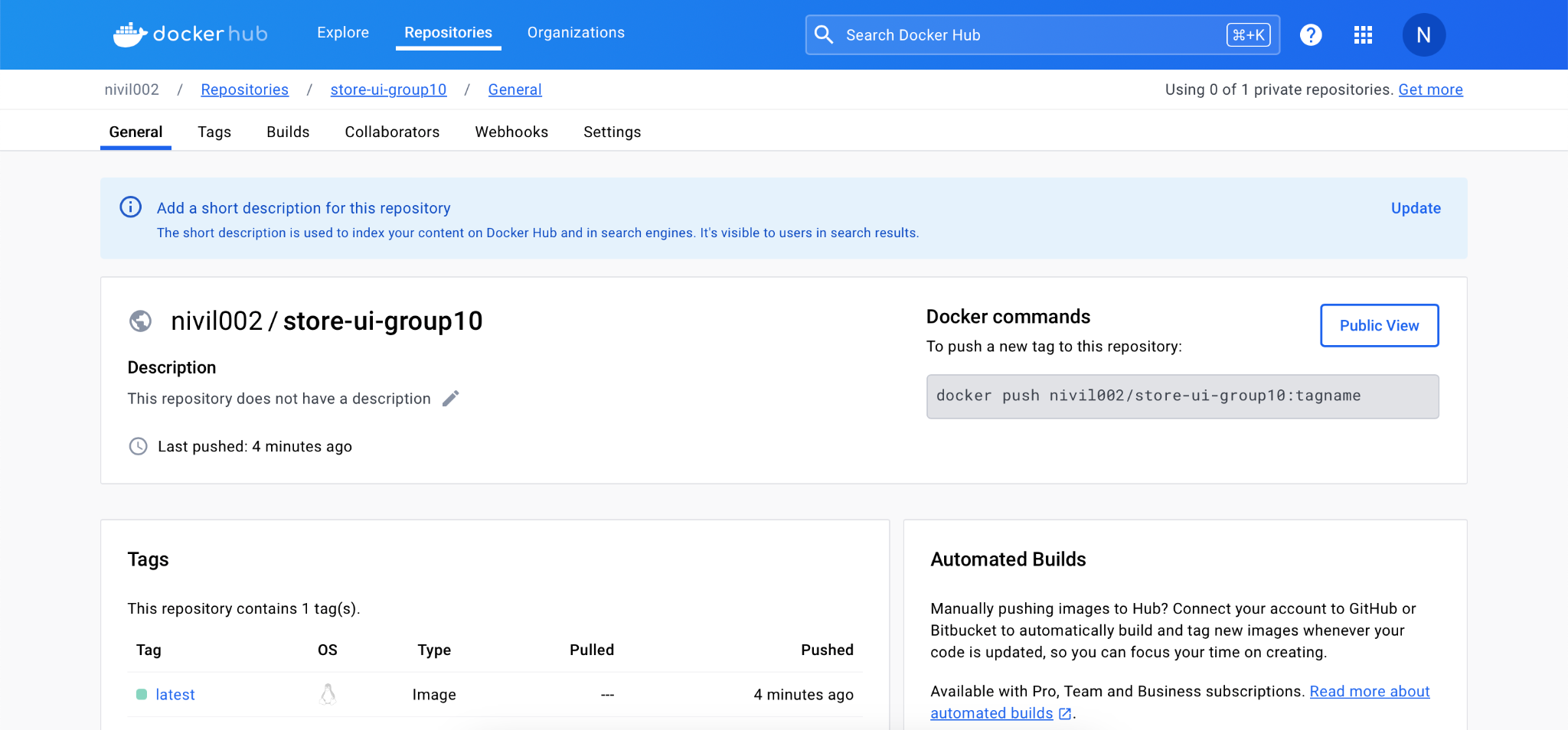




1.6. How can you enable your cluster to pull images from the gitlab?   
(Don’t think of the usage now have a look on the capabilities to enable Kubernetes to pull from private registries)





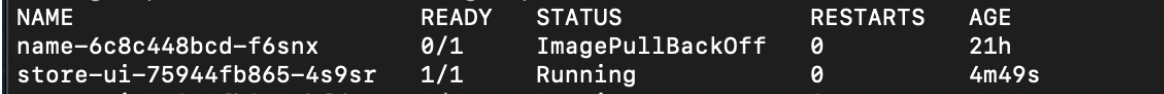


1. Created the store-ui-deployment.yaml and store-ui-service.yaml files.
2. I have apply the commands of:

kubectl apply -f store-ui-deployment.yaml

kubectl apply -f store-ui-service.yaml

1. kubectl get pods



### 2. Enhance your workflow & Basic Tools

2.1. Check the tool k9s and explore your cluster. What is the exploration teaching you about the rancher installation of the Kubernetes cluster?

Exploring your Kubernetes cluster using **k9s** can provide insights into various aspects of your cluster's configuration and status. The specific information you can gather depends on the features and functionality offered by **k9s**. Below are some general areas that exploring your cluster with **k9s** might teach you about the Rancher installation:

1. **Nodes and Workloads:**
   * View details about the nodes in your cluster, including their health, status, and resource usage.
   * Explore the workloads running on your cluster, such as Deployments, StatefulSets, and DaemonSets.
2. **Pods and Containers:**
   * Inspect individual pods, view their logs, and understand their current state.
   * Examine the containers within each pod and their resource utilization.
3. **Namespaces:**
   * Explore different namespaces to understand how workloads are organized and isolated.
4. **Services and Ingress:**
   * View details about services and ingress resources to understand how external access is managed.
5. **ConfigMaps and Secrets:**
   * Inspect ConfigMaps and Secrets to understand how configuration data and sensitive information are managed.
6. **Resource Quotas and Limits:**
   * Check resource quotas and limits set for namespaces to ensure resource isolation.
7. **Cluster Events:**
   * Monitor cluster events and get insights into changes and issues within the cluster.
8. **RBAC (Role-Based Access Control):**
   * Understand the RBAC configuration to ensure proper access controls are in place.
9. **Custom Resources:**
   * If you are using custom resources, **k9s** may provide insights into their status and configurations.
10. **Cluster Metrics:**

* Some tools, including **k9s**, may offer metrics and monitoring capabilities, allowing you to view the overall health of your cluster.

Keep in mind that the specific features and capabilities of **k9s** can vary based on the version you are using and the configuration of your Kubernetes cluster. The exploration with **k9s** is an interactive process, allowing you to navigate through different views and get real-time information about the state of your cluster.

It's also worth noting that Rancher itself provides a web-based UI for managing Kubernetes clusters, and the exploration with **k9s** complements the graphical interface by providing a command-line, terminal-based exploration option.

**You**

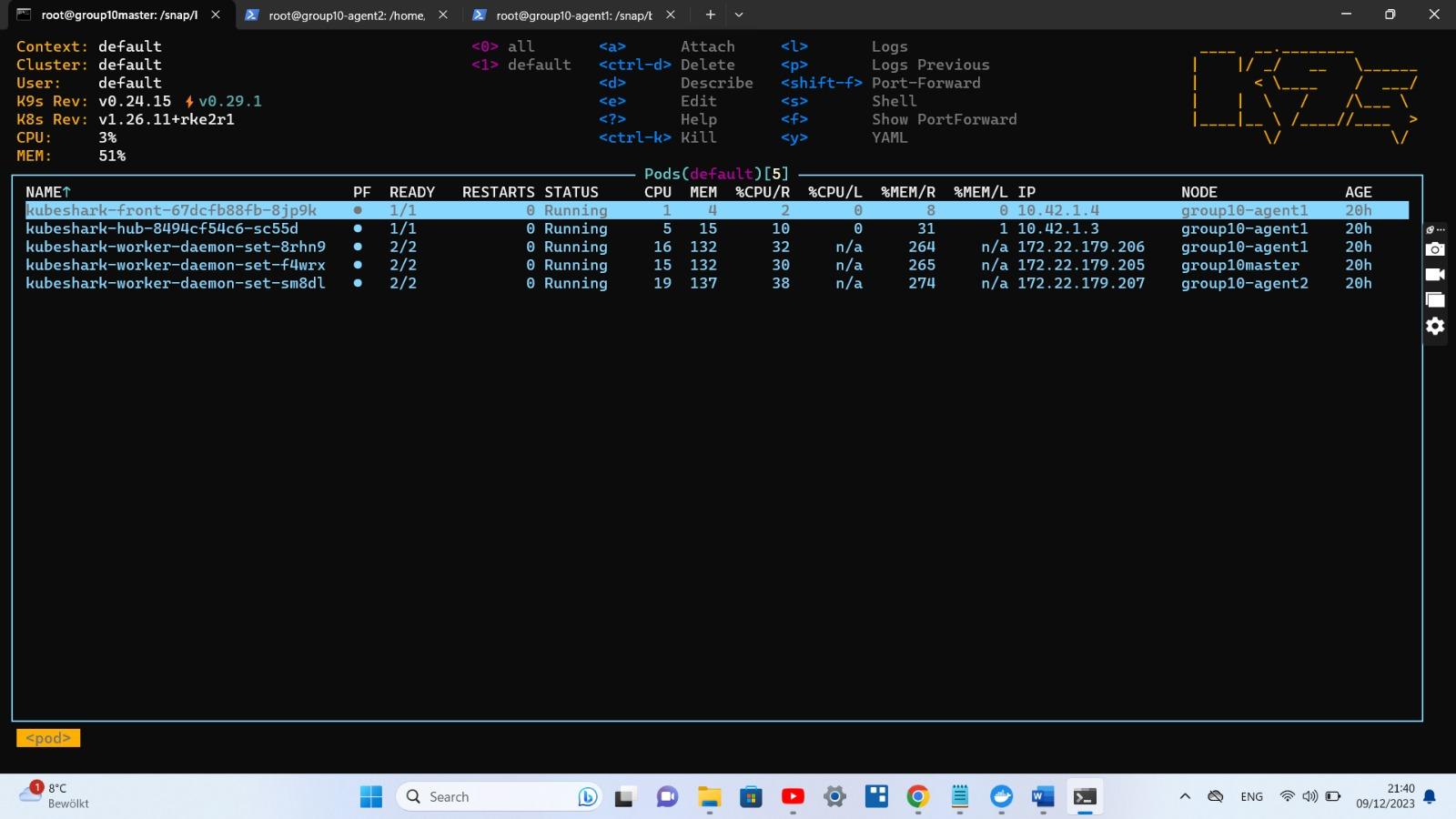
What is the exploration teaching you about the rancher installation of the Kubernetes cluster?

The exploration of a Kubernetes cluster using tools like **k9s** can provide insights into

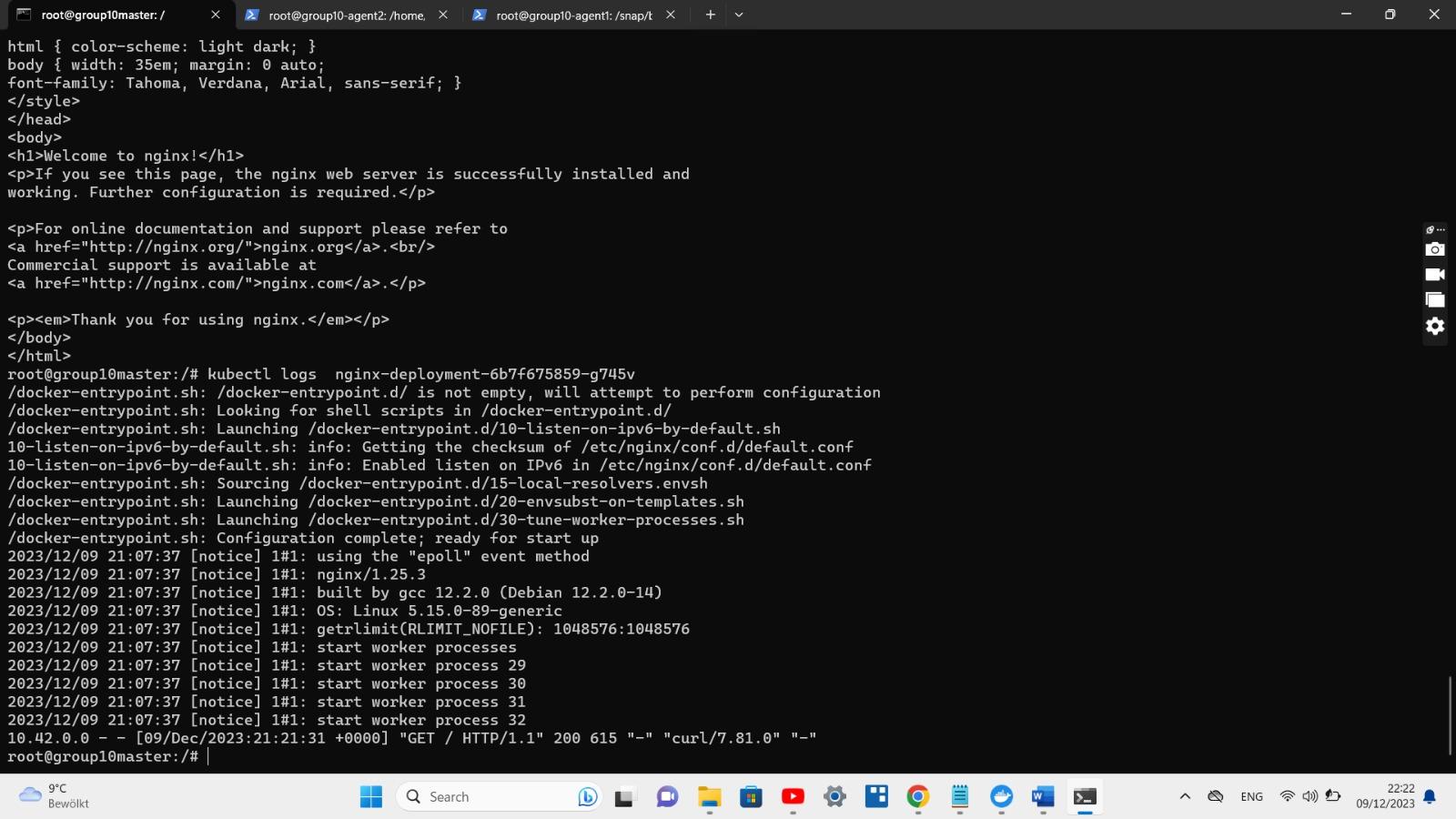
various aspects of the cluster, including its configuration, health, and resource utilization. Specifically for Rancher-managed clusters, the exploration might reveal:

1. **Rancher Workloads:**
   * Rancher typically uses custom resources to manage workloads. You might see Rancher-specific custom resources for clusters, projects, and other Rancher-managed entities.
2. **Rancher Agents and Pods:**
   * Investigate the status and health of Rancher agents and pods running on the nodes. This is crucial for ensuring Rancher's control over the cluster.
3. **Node Health:**
   * Check the health and resource usage of individual nodes to ensure they are performing well. Rancher relies on these nodes to deploy and manage workloads.
4. **Cluster Configuration:**
   * Explore the cluster configuration to understand any Rancher-specific configurations or settings applied to the cluster.
5. **Rancher System Pods:**
   * Inspect pods related to the Rancher system to ensure that critical components, such as the Rancher server, are running smoothly.
6. **Projects and Namespaces:**
   * Understand how Rancher organizes workloads into projects and namespaces. This can give you insights into the multi-tenancy aspects managed by Rancher.
7. **Monitoring and Logging:**
   * Depending on the configuration, you might use **k9s** to explore monitoring and logging resources set up by Rancher to gain visibility into the cluster's health and performance.
8. **Cluster and Project Events:**
   * Monitor events related to Rancher-managed clusters and projects to understand changes, issues, or activities within the Rancher environment.
9. **Rancher-specific Resources:**
   * Rancher may introduce its own custom resources or controllers to extend Kubernetes functionality. Use **k9s** to explore these resources and understand their roles.
10. **Integration with Rancher UI:**
    * The exploration with **k9s** complements the Rancher web-based UI. Understanding how the cluster looks from both perspectives can provide a more comprehensive view.

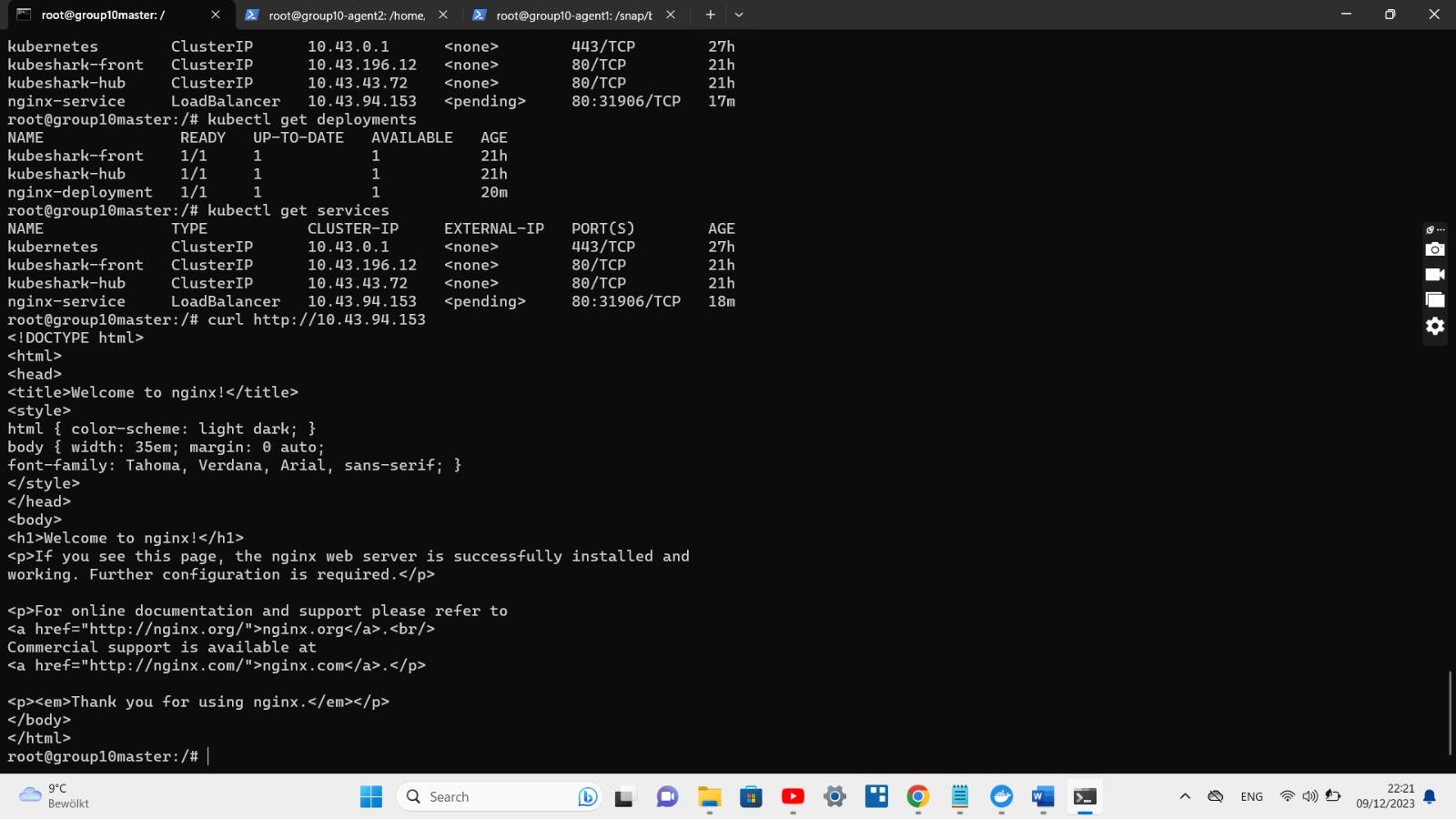
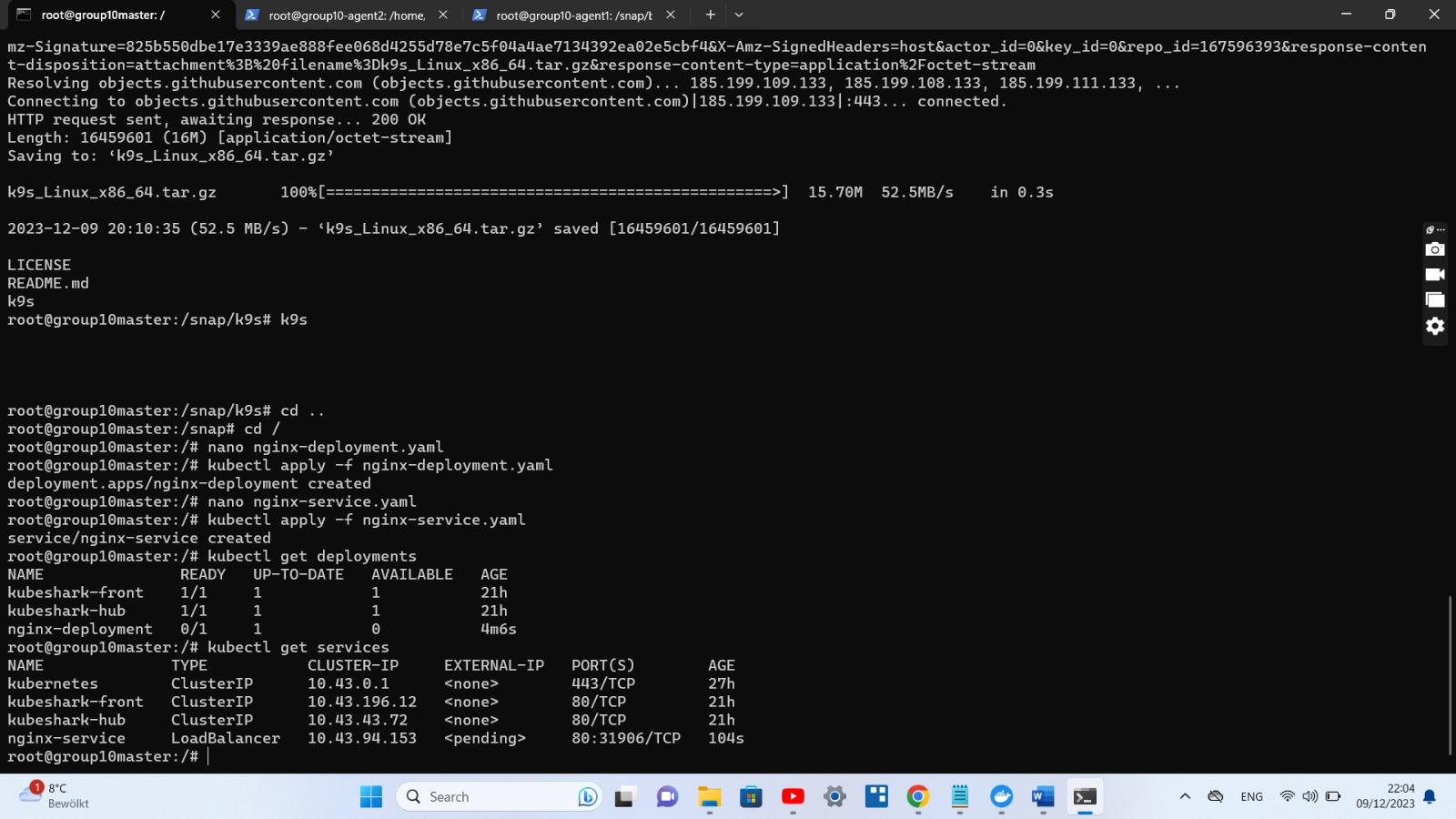
In Master:



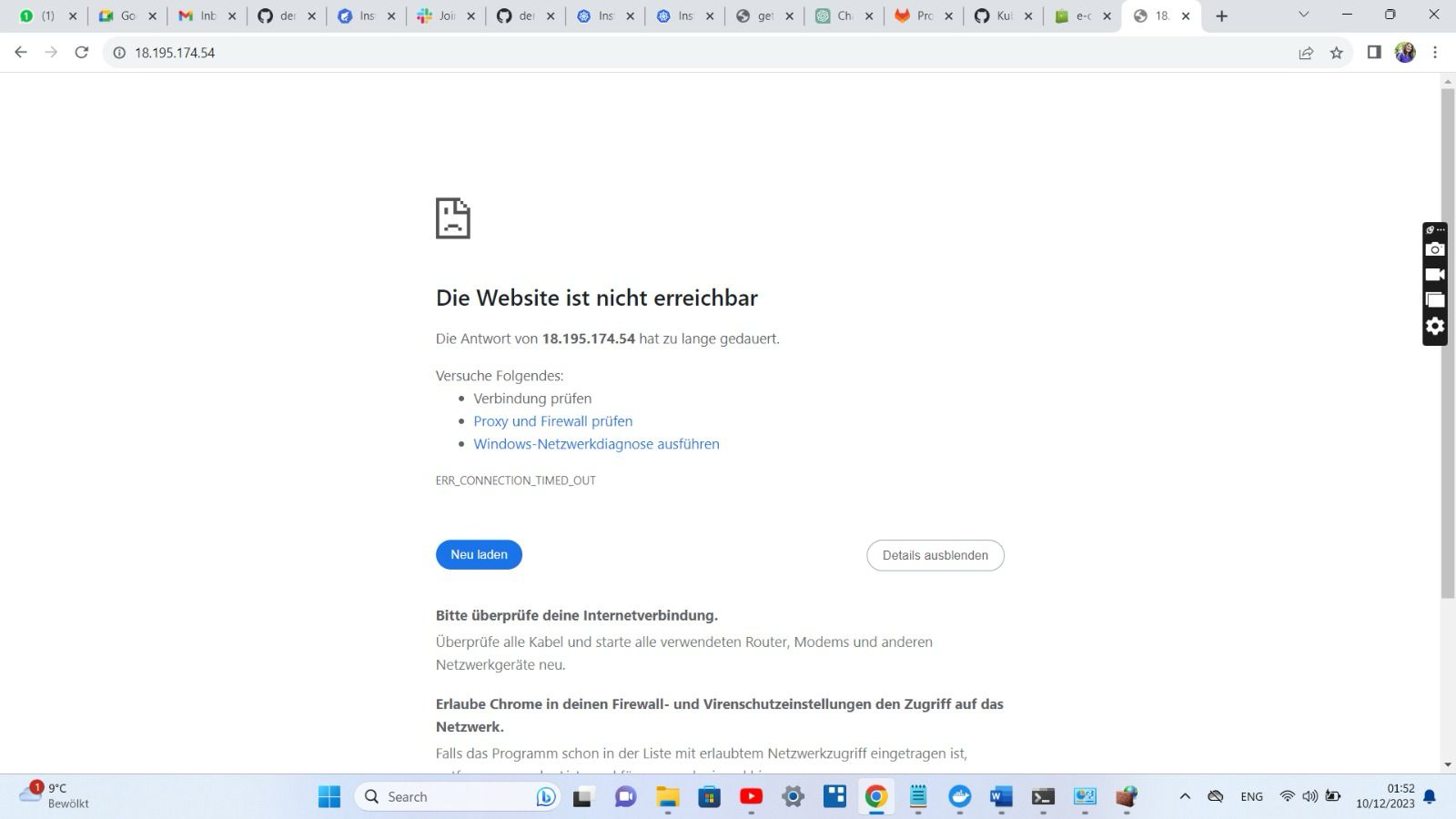
2.2. How can you create an instance of a nginx on your cluster?  
(Check the logs to see if its running, when finished)

**Logs:**   


**create an instance of a nginx on your cluster**

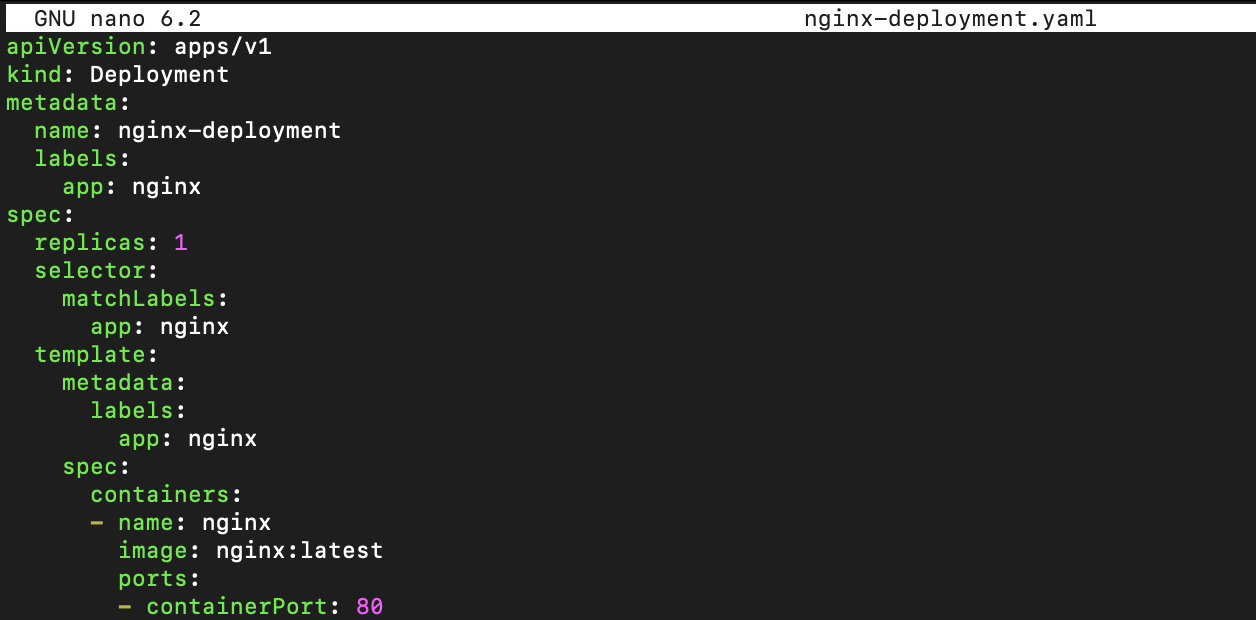


2.3. Is the nginx accessible from your browser?   
(Why is it/isn’t it accessible?)  
  
No it is not accessible from the browser.

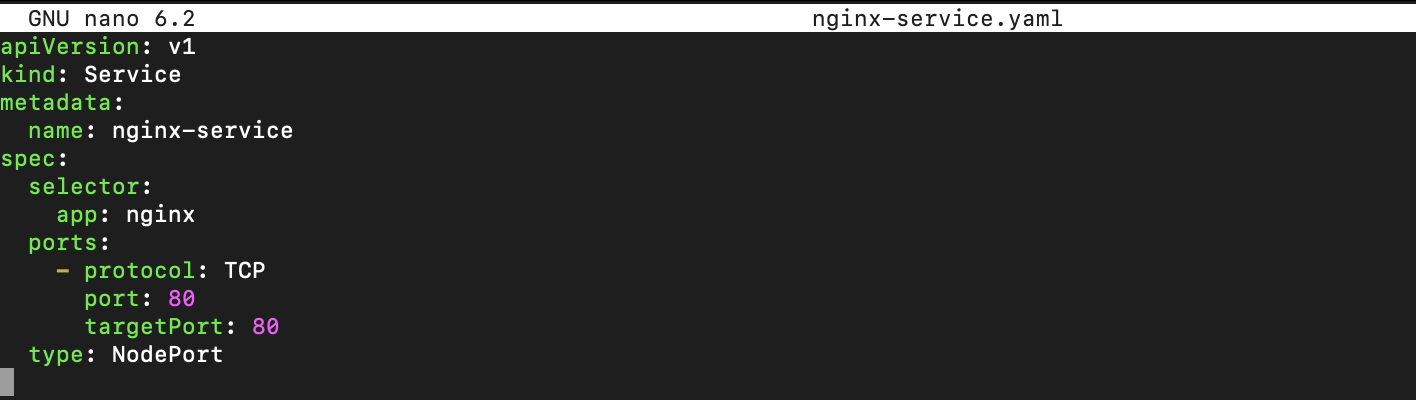


then to access it from browser we need to follow some steps:

we need to create development.yaml file



then we need a service.yaml file



then apply

kubectl apply -f nginx-deployment.yaml

kubectl apply -f nginx-service.yaml

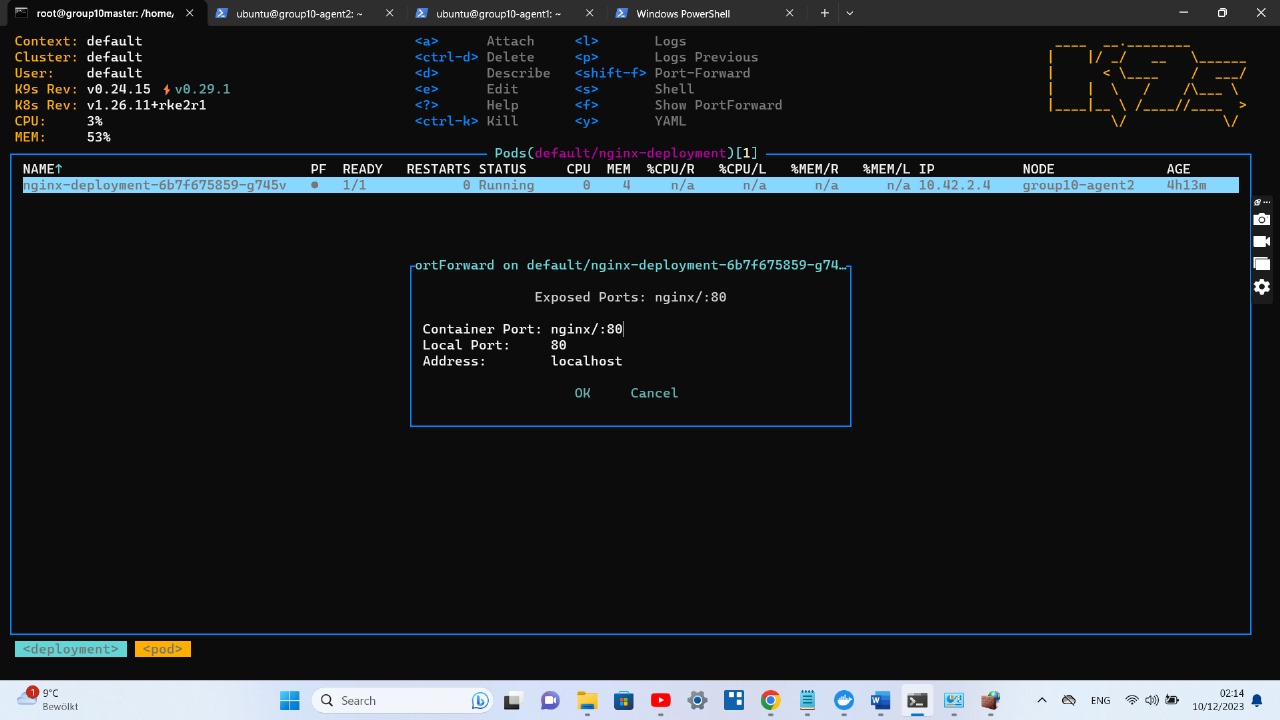


2.4. Explore the resources you created using k9s. What options do you see making the nginx available from your browser? (For now to test the nginx, you can use the k9s Port-Forwarding)

To make Nginx available from your browser for testing purposes using k9s port-forwarding, you can follow these steps:

* **Port-Forward Nginx:**
  + Open k9s and navigate to the Deployment or Pod managing Nginx.
  + Select the Nginx Pod or Deployment and press : to open a command prompt.
  + Type port-forward deployment/nginx-deployment 8080:80
* **Access Nginx from Browser:**
  + Open your web browser and navigate to http://localhost:8080.

Now, you should be able to access Nginx from your browser, and any requests to http://localhost:8080 will be forwarded to the Nginx Pod running in your Kubernetes cluster.



### 3. Networking Modes

3.1. What ways are there to expose the Nginx application to the outside world. What’s the difference between them. Name them and make clear difference.

In Kubernetes, there are several ways to expose applications to the outside world. The main methods include:

1. **NodePort:**
   * **Description:** A NodePort service exposes an application on a specific port of each node in the cluster. It allows external access to the service by reaching any node on that specific port.

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: NodePort

**Access URL:** **http://<node-external-ip>:<node-port>**

**2. LoadBalancer:**

* **Description:** A LoadBalancer service is used to expose applications externally through cloud provider load balancers. It automatically provisions a load balancer with an external IP and forwards traffic to the service.

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

**Access URL:** External IP assigned by the cloud provider.

**3. Ingress:**

* **Description:** Ingress is an API object that manages external access to services within a cluster. It allows you to define rules for routing HTTP and HTTPS traffic to services based on hostnames or paths.

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: nginx-ingress

spec:

rules:

- host: nginx.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: nginx-service

port:

number: 80

**Access URL:** **http://nginx.example.com**

**4. ClusterIP (Internal Access):**

* **Description:** A ClusterIP service exposes an application inside the cluster. It is accessible only within the cluster and is not directly accessible from outside.
* **Usage:**

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: ClusterIP

**Access URL:** **http://nginx-service:80**

**Key Differences:**

* **NodePort**:
  + Exposes the service on a static port on each node.
  + External access is through any node's IP and the specified NodePort.
  + Suitable for development and testing.
* **LoadBalancer**:
  + Exposes the service externally using a cloud provider's load balancer.
  + Provides a dedicated external IP.
  + Ideal for production environments with external traffic.
* **Ingress**:
  + Routes external traffic based on rules (hostnames or paths).
  + Enables more advanced traffic routing and SSL termination.
  + Suitable for managing multiple services with different URLs.
* **ClusterIP**:
  + Only accessible within the cluster.
  + Suitable for internal communication between services.
  + Not intended for external access.

3.2. How can you use service type NodePort to expose the Nginx application to the outside world and explain what challenges you faced? Add a screenshot of the application from your browser with URL.

Using a **NodePort** service type to expose an NGINX application involves a few steps.

**1.Create NGINX Deployment:** Create a Deployment to run NGINX pods

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

**Apply the configuration:**

kubectl apply -f nginx-deployment.yaml

**2.Create NodePort Service:** Create a **NodePort** service to expose the NGINX deployment:

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

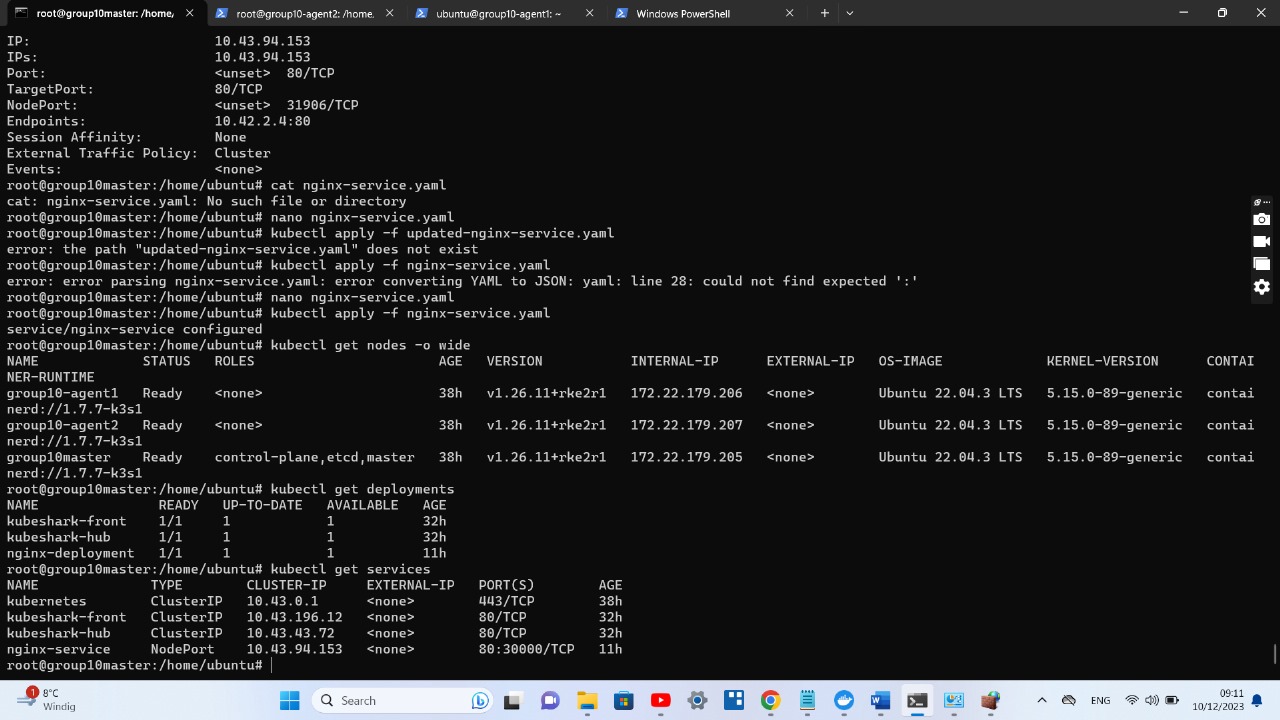
ports:

- protocol: TCP

port: 80

targetPort: 80

type: NodePort



**Apply the configuration**:

kubectl apply -f nginx-service.yaml

kubectl get nodes

echo http://$(kubectl get nodes -o jsonpath='{.items[0].status.addresses[0].address}'):$(kubectl get services nginx-service -o jsonpath='{.spec.ports[0].nodePort}')



3.3. How can you use cluster IP to expose the Nginx application? Is it still accessible from your browser?

Yes, we can access it from outside.

Steps Followed:

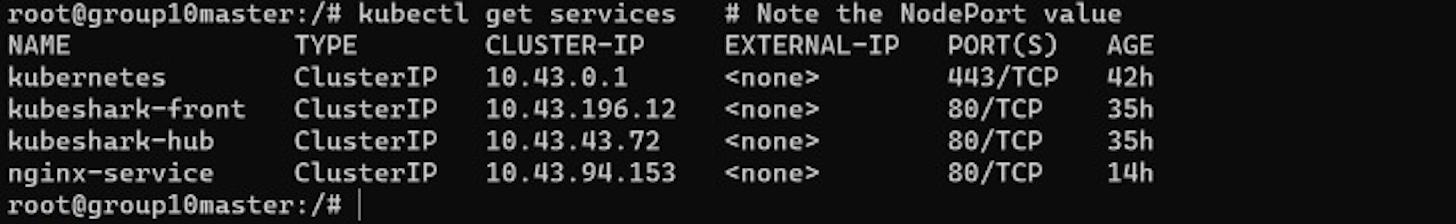
1. create deployment.yaml

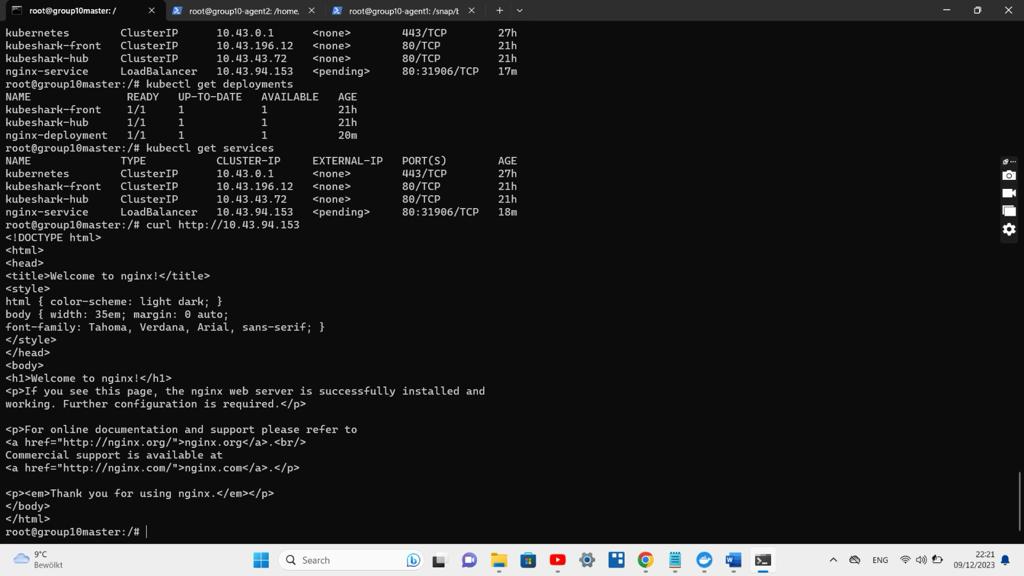


1. create service.yaml

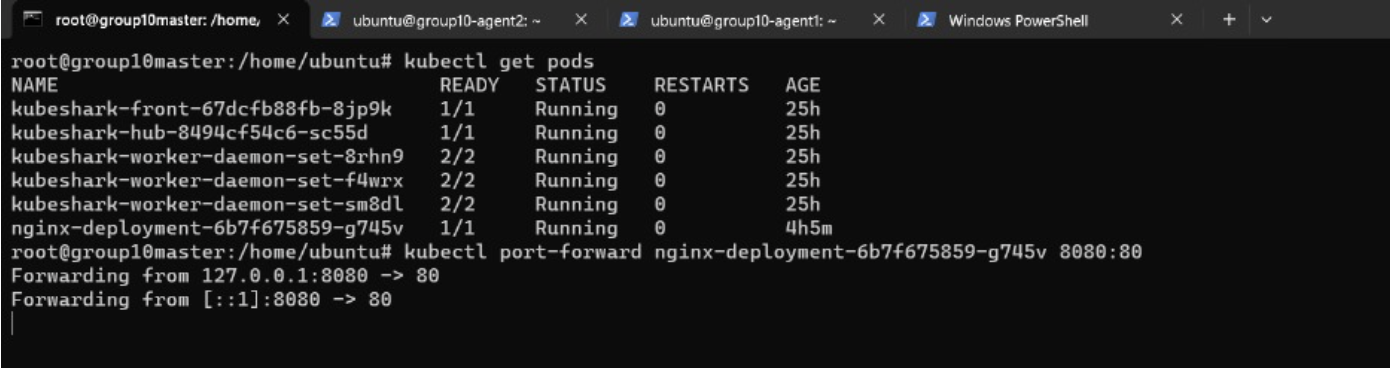


1. apply the deployment and service files
2. kubectl get services

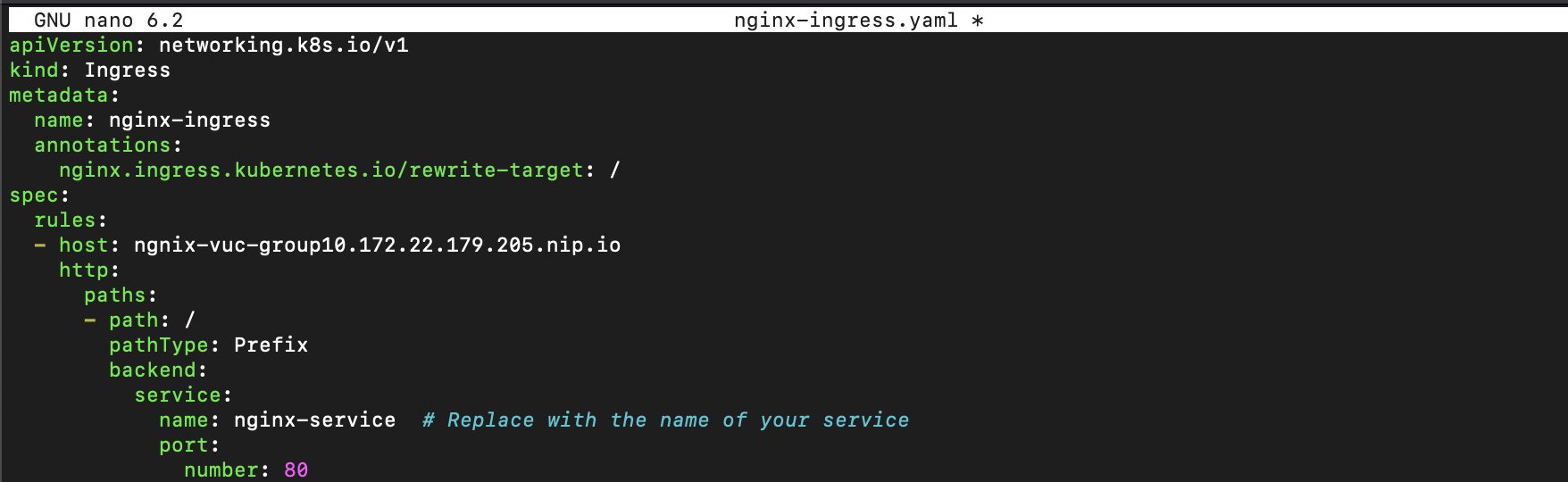


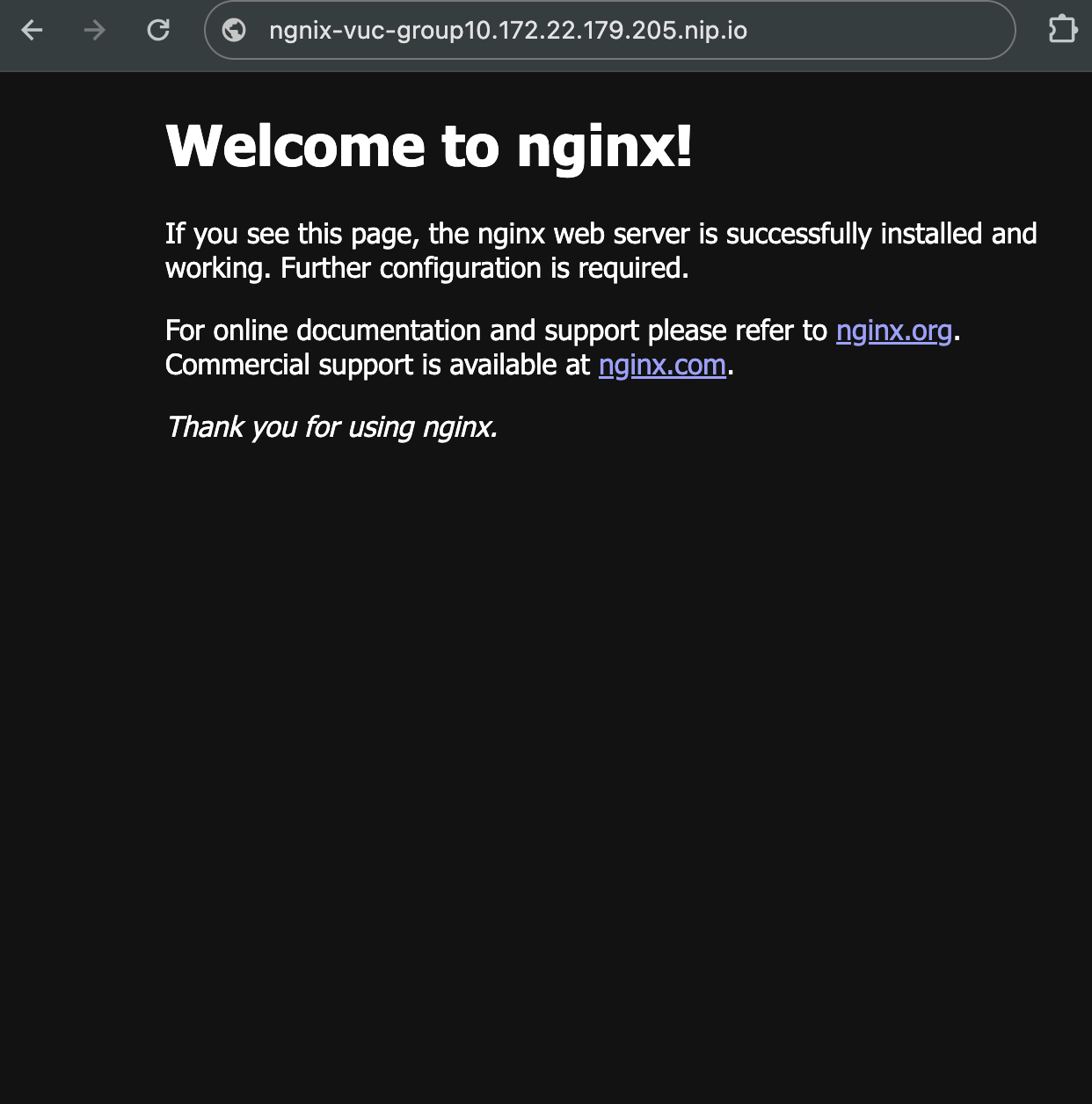


3.4. How can you try to access the service using the command kubectl run? What are the ways to make it accessible? Please add a screenshot with your results of the browser and kubectl pod.



3.5. Now, how can you use the Kubernetes service type Ingress to expose the application to the outside world? What are the challenges and what benefits do you see? Please add the screenshot of the browser running the Nginx application with the URL. (Hint1: ngnix-vuc-group-<name>.com should link to nginx appliation using Ingress and add the domain to the localhost file) (Hint2: Rke2 already have a running Ingress Controller)

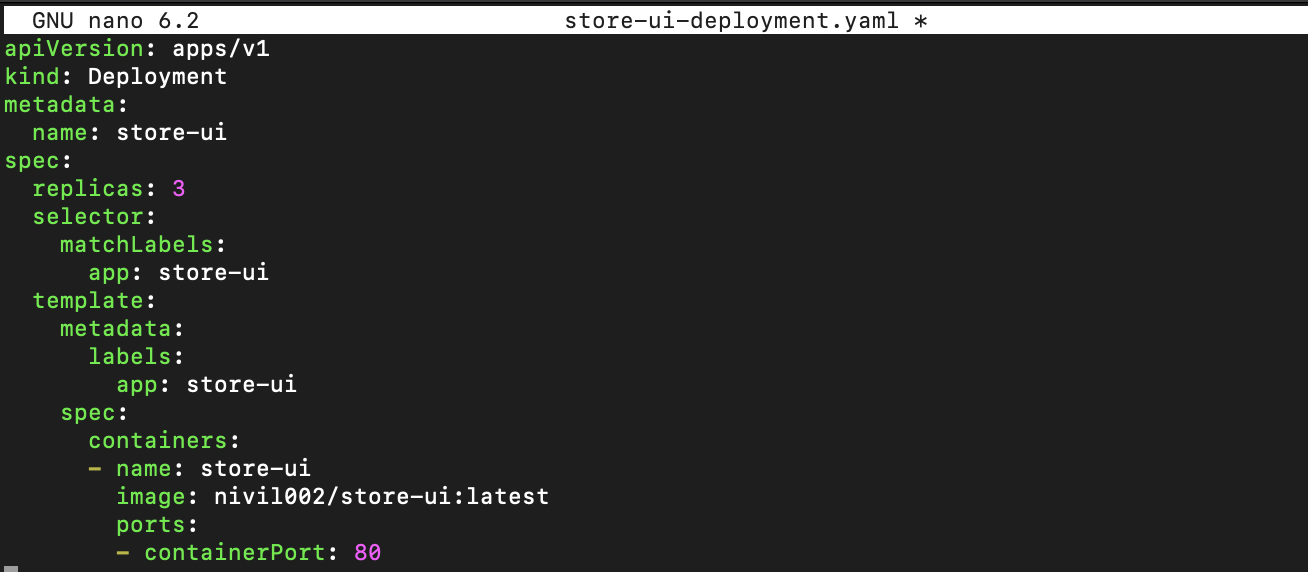




3.6. How does the communication between pods, services and external URL work. Explore the KubeShark tool for communication verification. Add the screenshot of the Kubeshark dashboard showing the request send to the service and to the ingress controller from the browser. Also add the screenshot of the Service Map for both scenarios (service and ingress). (Hint: use the filter field to filter out the requests)

## 4. Case Study

With the gathered knowledge you are going to deploy the whole application in your kubernetes cluster. Therefore you probably need to create a new version of the store ui image.

4.1. How can you create all the services from the case study on your kubernetes cluster (link to case study)? (How do the yaml files look like?) (Hint1: notice that we have 3 services that aren’t provided in gitlab, you will come across them while checking .env files of some microservices, how can you get them?)(Hint2: in total you should have 8 services running)  
  
  


I have apply the commands of:

kubectl apply -f store-ui-deployment.yaml

kubectl apply -f store-ui-service.yaml

we have run the:

**kubectl get pods**



store-ui-nodeport.yaml created:

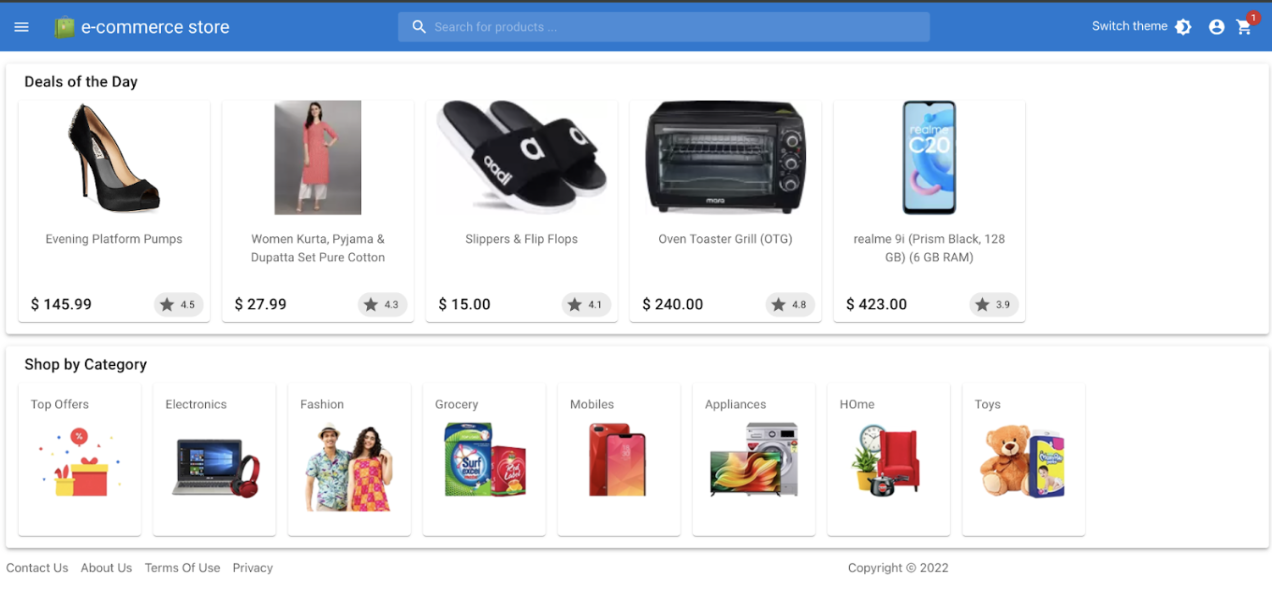


2) Apply the yaml file with this command:

**kubectl apply -f store-ui-nodeport.yaml**

3) Enter the <serverIP>:<NodePort> in browser:

[http://172.22.179.205:3000/](http://172.22.179.214:30599/)



4.2. What methods are used to configure the services? Is there any way to change the configuration without the need to touch the deployment of the service itself? (You probably want to “outsource” the configuration from the actual deployment)

Configuring services in Kubernetes may be done independently of the deployment process by using ConfigMaps, Secrets, and environment variables. Key-value pairs for general configuration are stored in ConfigMaps, and sensitive data is handled in Secrets. Environment variables offer an easy approach to set up programs. Independent configuration management is made possible via external configuration suppliers like Helm charts. For more complicated requirements, custom resources and dynamic configuration reloading are available; implementing GitOps principles entails version-controlling configurations for controlled updates without directly altering deployments. Flexibility and maintainability are improved when configuration and deployment are separated.

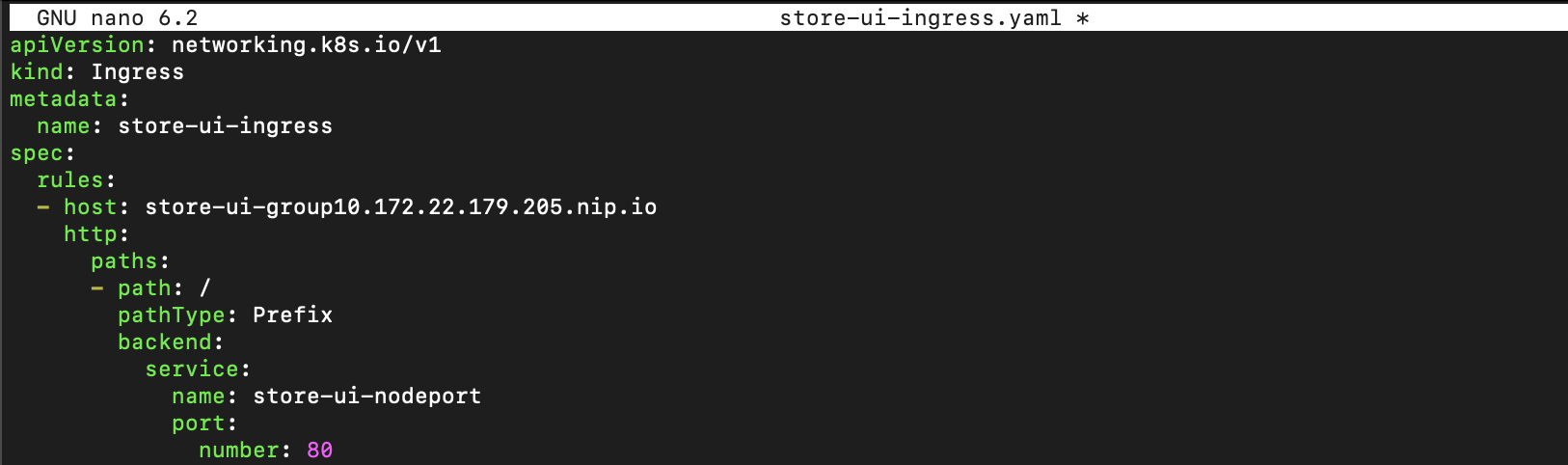
4.3. Some of the services from the case study you created in Task 4.1 are databases, how can we make them available to the ecosystem of the store? (Please attach a screen shot of the response of the end points!) (Hint: the store-ui isn’t fully functional as it wasn’t fully completed, so to check the functionality of some the micro-services in the ecosystem of the store you need to run some API requests in your browser)

A list of these API requests can be seen at the end of the document.

4.4. Configure your cluster to make the services of the case study available, use the following naming convention: <service>.<group>.com

Provide a screenshot with the store UI and the dev-tools showing the network tab with the data being loaded.  
(You should be able to show the full application during the acceptance session)

1. created the store-ui-ingress.yaml file

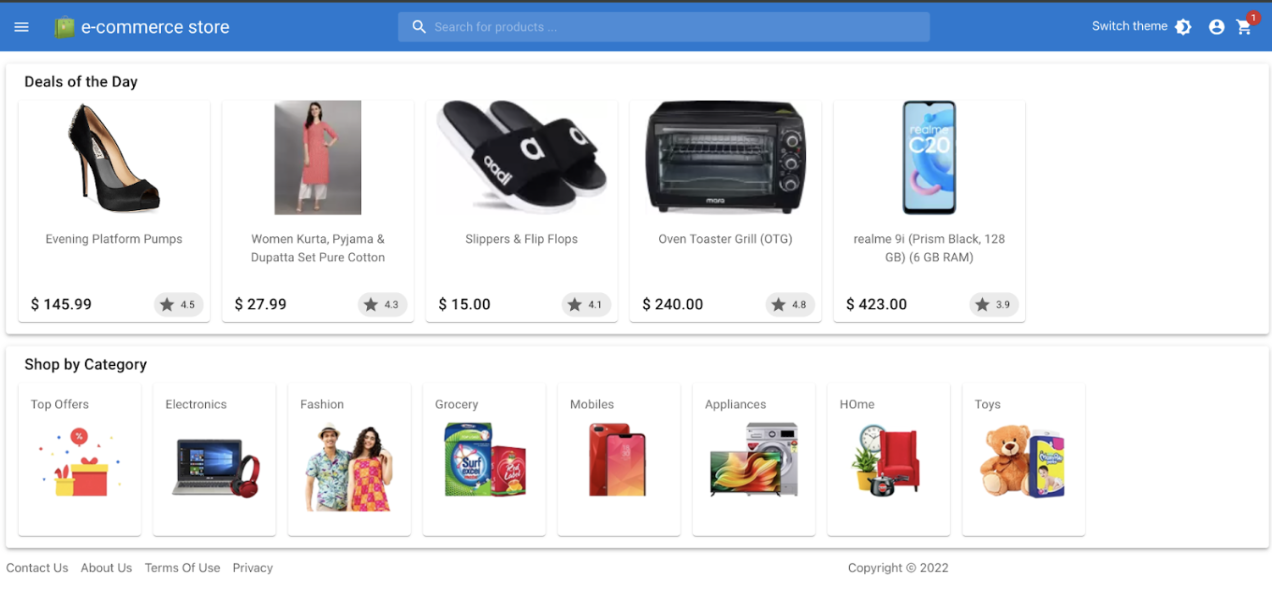


2) Apply the yaml file with this command:

**kubectl apply -f store-ui-ingress.yaml**

3) enter the host in browser

[http://store-ui-group10.172.22.179.205.nip.io/](http://store-ui-group06.172.22.179.214.nip.io/)



## 5. Reflective Tasks

Based on the knowledge you gathered, there are some reflective questions to be answered. (Always keep in mind the case study while answering these questions)

5.1. Do you think the way the solution has been created in this laboratory is efficient and worth it?

**For Development**:

1. Kubernetes offers efficient scalability choices that enable developers to easily modify resources in response to demand, which is particularly helpful in the development and testing stages.

2. Kubernetes facilitates cooperation and guarantees a uniform deployment environment. Developers can work on different parts of the application with confidence knowing that the deployment environment won't change.

3. Kubernetes improves resource efficiency through the effective administration of containerized applications. By allowing developers to define resource requirements for every component, resource allocation efficiency is increased overall.

**To be run as a server**

1. Reliability and high availability are two of Kubernetes' best qualities. Enhanced uptime is a result of features like automated scaling and pod replication.
2. In Kubernetes, load balancing and horizontal scaling are used to optimize performance. Monitoring tools make it easier to find and fix possible performance bottlenecks.
3. A variety of tools for troubleshooting and monitoring are offered by Kubernetes. Putting strong logging and monitoring systems in place helps find and fix problems quickly.

**To rent them to customers:**

1. Scaling to accommodate changing client demands is an area where Kubernetes thrives. Efficient management of growing traffic and workloads is ensured by proper configuration.
2. Strong security features like network policies and pod security policies are built into Kubernetes. Maintaining security and isolation between customer instances requires proper configuration.
3. Through efficient scaling and resource optimization, Kubernetes enhances cost-effectiveness. Sustained cost-effectiveness depends on ongoing optimization and monitoring initiatives.

5.2. Is the used approach scalable? (vertically and horizontally)  
  
The Kubernetes-deploy solution's scalability depends on both vertical and horizontal dimensions. Vertical scalability, which refers to resource growth within individual nodes, depends on how well the program can use more resources. In the meantime, efficient workload distribution and load balancing techniques are required for horizontal scalability, which is attained by expanding the Kubernetes cluster with additional nodes. The system's scalability is guaranteed via a carefully calibrated mix of vertical and horizontal scaling, as well as proactive monitoring for optimization. The evaluation ought to take into account the application's capacity to manage amplified demand, the effectiveness of resource allocation, and the cluster's ease of dynamically adjusting to fluctuating workloads, thereby establishing the foundation for future expansion.

5.3. How can you migrate your solution to another machine/cluster?

Use tools like `kubectl’ to export your Kubernetes resources to YAML files so you may move your Kubernetes solution to another computer or cluster.

1. Make that the new cluster's container images are accessible, update cluster-specific configurations, and handle secrets safely.
2. Use `kubectl apply` to deploy the exported resources on the new cluster and perform a comprehensive functional test of the application.
3. Modify the DNS, Ingress, and monitoring setups; have a plan for rolling back changes; and update the documentation to reflect the modifications.

This all-encompassing strategy guarantees a seamless transfer and upholds your application's dependability in the new setting.

5.4. Compare the knowledge gathered during this lab with the knowledge and experiences from AWS. Which one do you prefer and why? Do you see scenarios for a usage of both?

For scalable infrastructure and a wide range of managed services that integrate seamlessly, AWS is the best cloud computing platform. Kubernetes is the preferred option if platform independence, application mobility, and container management are among top concerns. Some businesses use a hybrid strategy, combining Kubernetes for containerized workloads with AWS services for certain requirements. In the end, our choice should be in line with our application's design, our need for scalability, and our preferences for containerization and managed services.

5.5. How could you detect failures in running pods quickly to prevent outages?Think of a way to replace a failed pod with a new one. Does Kubernetes offer any assistance for this scenario?

Using health checks like readiness and liveness probes in Kubernetes helps identify issues in running pods rapidly and prevent outages. Based on the findings of these probes, which assess a pod's capacity to handle traffic, traffic is immediately restarted or redirected. Replica Sets and Deployments are examples of controllers that maintain the required number of replicas by automatically replacing failed pods. The number of replicas is dynamically adjusted by auto-scaling technologies, and rolling deployments provide seamless updates with little to no downtime. Pod Operators automate operational activities, whereas Pod Disruption Budgets control disturbances during updates. Putting monitoring and logging systems in place improves proactive failure detection even more, which makes the system more robust and self-healing.

## API Endpoint List:

Please keep in mind that the IPs and ports here are just some examples, and shouldn’t necessarly be the ones you need to use, so change them accordingly. Also POST/PUT requests typically can’t be performed from the address bar of your browser, so please check another way to perform these types of requests.

|  |  |  |  |
| --- | --- | --- | --- |
| Cart | GET | http://172.22.145.110:8080 | Returning info on the api |
| GET | http://172.22.145.110:8080/cart | Get all carts |
| GET | http://172.22.145.110:8080/cart/{customerId} | Get the cart for one customer |
| POST | http://172.22.145.110:8080/cart | Create a new cart entry |
| Products | GET | http://172.22.145.110:5000/deals | Returning the deals |
| GET | http://172.22.145.110:5000/products/sku/:id | Returning the product with the id |
| Search | GET | http://172.22.145.110:4000/test | Just to test whether the search is working, should give an error |
| Users | GET | http://172.22.145.110:9090/users | Returning all Users |
| GET | http://172.22.145.110:9090/users/{user\_id} | Returning a specific user |
| PUT | http://172.22.145.110:9090/users/{user\_id} | Updating a specific user |
| POST | http://172.22.145.110:9090/users | Create a new user |

# Conclusion

This lab should teach you the basics of containerized virtualization in kubernetes. You should be able to install a cluster and deploy an application to the cluster.  
There are further topcis that weren’t focused in the laboratory yet but feel free to investigate further on:

* Storage Providers (e.g. from a SAN)
* More advanced Load Balancing technologies
* Multi-Pod-Deployments
* Difference between Deployments & Stateful-Sets

# Outlook

With Kubernetes you know know a solution to manage container-based applications in an professional IT-environment. However, you still need to set-up the Kubenetes environment and possibly the underlying host machines. Going forward you will look at so called serverless solutions in cloud service offerings, building and using functions as the basis for microservices.