Cloud Operations Report

General Info

Chosen topics:

Major

Terraform - IaC

Minor

Operator pattern
Autoscaling with Keda

Interpretation of the general assignment:
To study each topic up to the point that it's purpose, main functionality and features are understood and tested on a Kubernetes setup.

Every topic will consist of:

- Description of the technology
- Testing Goals
- Implementation of the testing goals + their Results
- Encountered issues

Setup

For Minor topics

1. I created a Virtual Machine using KVM with Ubuntu 2024 server.
2. On the Ubuntu server I configured Kubernetes node with https://k3s.io/

```
curl -sfL https://get.k3s.io | sh -
# Check for Ready node, takes ~30 seconds
sudo k3s kubectl get node
```

3. In order to use it I exported the kube config to my host and installed kubectl https://kubernetes.io/docs/reference/kubectl/
4. I also installed and used Lens IDE https://lenshq.io/ which became my main source of interactions with kubernetes

5. Finally, installed helm on Ubuntu Server https://helm.sh/

For the major topic

1. I created an Ubuntu server Virtual machine with installed:
   - kubectl,
   - terraform https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
   - talosctl https://docs.siderolabs.com/talos/v1.8/getting-started/talosctl
2. I set up a Proxmox server instance on my second host with a wired connection to the home lab's network.

Topics

Autoscaling with Keda

Description

As I understood from all the resources, KEDA (Kubernetes Event-Driven Autoscaling) - is a additional component that extends Kubernetes scaling capabilities, mainly, the Horizontal Pod Autoscaler (HPA).

The limits of the HPA are:

- HPA cannot scale the number of pods of a deployment to 0 and the other way (from 0 to many).
- HPA supports only basic scaling metrics like memory and cpu usage

Keda component allows:

- Scale to/from 0 number of pods
- Scale based on huge number of metrics (by establishing dozens of different scalers) Keda can be considered an "advisor" for HPA with a small addition that it covers Zero scaling (which HPA cannot cover by design)

How they work together

- KEDA polls the external system
- KEDA exposes a metric (e.g., `queueLength = 120`)
- HPA reads that metric
- HPA computes desired replicas based on the HPA object created by KEDA when you establish a scaled object
- Kubernetes scales the Deployment

Keda does all that by establishing:

- Scalers - components that know how to connect,read,extract metrics from different systems (Like Prometheus scaler extracts metrics from Prometheus server)
- ScaledObject, ScaledJob - a wrapper for HPA declaration that maps Scaler Metrics with a scaling logic
- Agent - a special component responsible for Scaling to Zero

These are the main components I am planning to interact with (there are others but they are not the "Core" of the Keda)

Goals

- Install Keda
- Implement Scaled Object with CronScaler
- Implement Scaled Object with Workload Scaler + Test how scaled objects can interact together
- Implement Scaled Job with Prometheus Scaler

Implementation

1. Install keda with

```
kubectl apply --server-side -f
https://github.com/kedacore/keda/releases/download/v2.18.3/keda-2.18.3.yaml
```

Result - installed successfully

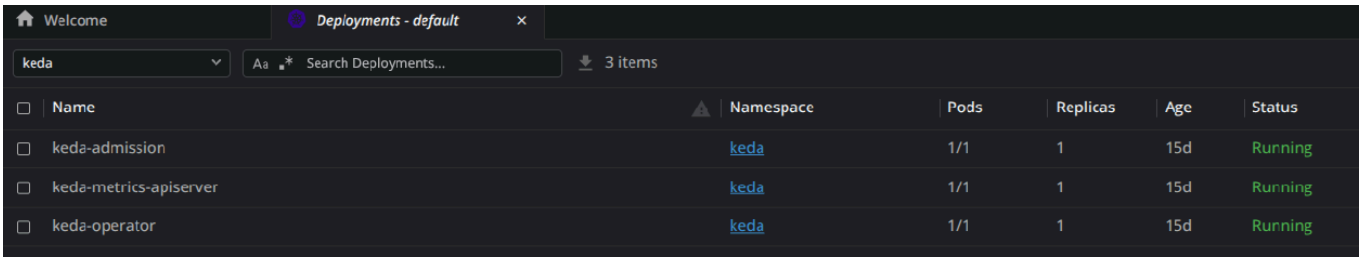


Figure 1 - Installed Keda deployments in Keda namespace

2. Scaled Object with CronScaler

- Created a namespace for testing https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/namespace.yaml
- Created a basic nginx deployment https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/nginx.yaml

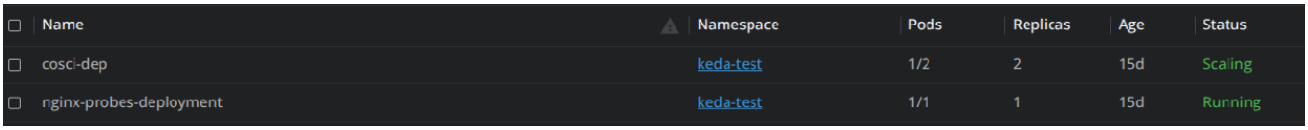
- Created a Scaled object with Cron Scaler: https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/nginx_scaler.yaml
  From 21.00 till 24.00 it scales the number of replicas to 2
  from 00.00 till 6.00 it scales the number of replicas to 1
- Result - it worked as described immediately.

3. Scaled Object with Workload Scaler

- Created a basic coscicorp deployment https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/coscicorp.yaml
- Created a Scaled object with Workload Scaler: https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/cosci_scaler.yaml

It scales based on the number of instances of Nginx pods. The number of coscicorp pods is always as twice as much the number of the nginx pods.

- Result - when I scale the nginx replicas up - within a few minutes the corresponding number of "cosci" is created



Name	Namespace	Pods	Replicas	Age	Status
cosci-dep	keda-test	1/2	2	15d	Scaling
nginx-probes-deployment	keda-test	1/1	1	15d	Running

Figure 2 - The process of autoscaling of "cosci" replicas

!!! But when I scale down the number of nginx replicas - the "cosci" will scale down only after 5+ minutes. That was confusing to discover. It is due to special HPA mechanism -the HPA intentionally waits a certain amount of time before scaling the app down.
You can see scaling logs example here:
https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/scaler_info.txt

4. Interaction of two scalers on 1 resource

- In addition to Coscicorp Workload Scaler I added a very similar to nginx cron scaler https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/nginx_scaler.yaml (just a different name for deployment)

The logic of the HPA is simple - it evaluates all the scalers and chooses the Highest number of required replicas. So if the cron scaler says we need 1 replica, but workload scaler says that we need 4 - the HPA will scale deployment to 4 replicas.

5. Implement Scaled Job with Prometheus Scaler

- Set up a random basic prometheus server as a resource on Kubernetes https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/prometheus.yaml
- Created service for it https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/prometheus_service.yaml

In the beginning it was not clear what jobs are mainly for so I created them as Alerting systems

- The first one is: https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/prometheus_scaled_job.yaml
  When there are too much replicas of nginx - it creates alerting jobs
- The second one: https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/Keda/prometheus_scaled_job_2.yaml
  Was planned to be as a notifier that number of replicas has changed for nginx deployment
- Results:
  The jobs are technically doing what I wanted them to do. But it is obvious that they are designed for a bit different functionality. You cannot make them be created only once a certain event occur, they are more like a loop - as long as the condition is met they keep being created and executed. Nevertheless they work and it is clear how they do that.

Encountered issues

There were 2 main problems

1. It was very unclear what keda precisely is and how it works. Most likely because I am new to kubernetes and was not sure about its principles.
2. Working with Prometheus. Prometheus server was also very unclear to use considering how its' queries work. They usually are based on timing, like check if the condition was met in a span of the last 2 minutes. It is not a usual metric system like "The status of the system is - Stable". It is a metric system like "In the random(or not) moment of the last 2 minutes the status of the system was - Stable"

Operator Pattern

Description

As I understood from all the resources, Operator pattern is just a proper way to extend your Kubernetes functionality with new features. For example you have a backend pod, that requires

having a postgresql instance with a proper volume each time it is created. You can make such functionality for kubernetes by making a Custom Resource Definition(CRD) - backend resource. Than you can add a special controller pod that has functionality to work with this CRD - it will be responsible for creating all necessary resources within kubernetes for your backend, while you can just declare "I want a backend", instead of "I want a backend, I want posgtresql, I want a volume". It is quite common in programming to extend or declare new functionality and this is what operator pattern is about.

Basically operator pattern implementation can be called anything that declares a CRD and a controller that interacts with it.

1. When can it be used?
   I already gave a usage example with a backend pod. If you have some complex system interactions in order to reduce repetitive tasks while working with them you can wrap them into operators.
2. Comparison with helm
   I believe operators are quite a different thing than Helm. Helm is a package manager and is able to execute premade templates. The key here is that it is stateless. Helm can actually install an operator.
   In contrast, operator is an extension of Kubernetes, that usually constantly works, is stateful, allows to modify and interact with the systems it is responsile for.
3. Drawbacks
   The drawbacks of the operator pattern is quite common for programmers - the risk of the overengineering and the increasing complexity of the system. You should be very aware of what you are adjusting and adding to the basic kubernetes functionality in order to keep the system maintanable and usable. It is not easy to debug the new controllers, track their hidden interactions, resource usage, etc.

Goals

- Install a Keda Operator
- Install Prometheus stack operator
- Verify Prometheus stack operator working

Implementation

1. Install Keda Operator

Why keda is an implementation of operator pattern?

- Keda declares new CRD's like ScaledObject

- Keda declares controllers, that work with the keda CRD's and connect them with Kubernetes

In the previous Topic I installed a keda operator and worked with it. Hence, it is done.

2. Install Prometheus stack operator

https://prometheus-operator.dev/docs/getting-started/introduction/
It worked out with git setup
https://github.com/prometheus-operator/kube-prometheus.git

Results - Working prometheus ecosystem on monitoring namespace

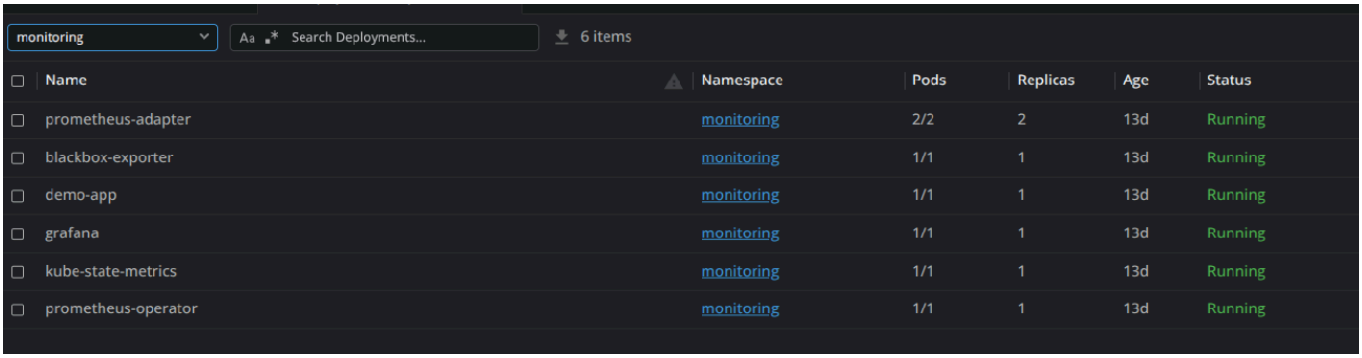


Figure 3 - The prometheus controllers

3. Verify Prometheus stack operator working

I followed the testing sequence described there:
https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/operator-pattern/prometheus-test.md

and recieved the final result:

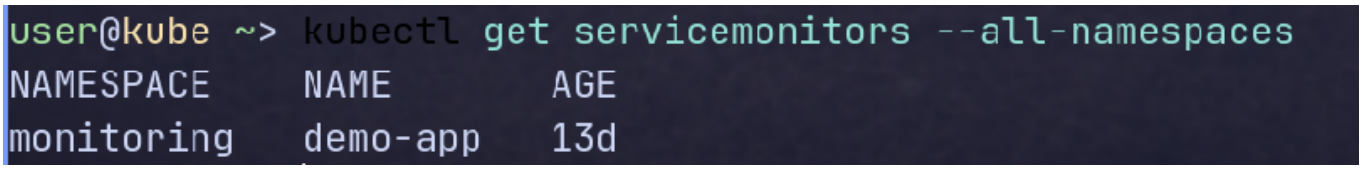


Figure 4 - Created service monitor

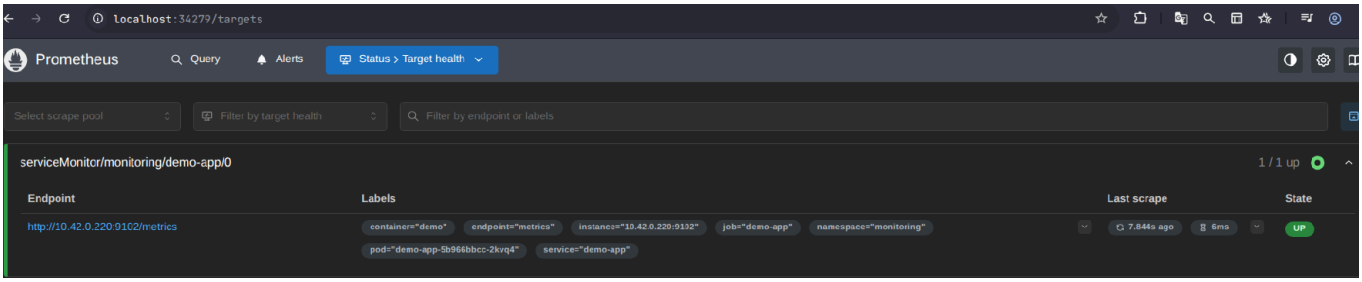


Figure 5 - Verifying it on the web browser

Encountered issues

There was only 1 problem - installing Prometheus operator stack. It did not work via helm for many attempts and worked after many attempts via git repository of kube-prometheus https://github.com/prometheus-operator/kube-prometheus.git

Terraform - IaC

Description

To create a fully automatic deployment of talos OS's kubernetes server on Proxmox platform with Terraform I need to know what are these technologies.

- Talos OS - a special operating system with only 1 task - hosting Kubernetes. It does not support ssh connection and a lot of other interactions with the OS. You use it via a special command tool - TalosCTL.
- Proxmox - a popular virtualization platform that is also an operating system.
- Terraform - Infrastructure as Code (IaC) concept implementation - It is a tool that allows you to manage different (platform agnostic) infrastructures with config files rather than physically. The main innovation in terraform is that it not only allows to configure something on the service, but it also monitors the state, assuring that the service reflects your desire at any time you want it. It also supports dynamic changes to your services. It has dozens provides for different infrastructure. In this lab I will use Proxmox and Talos providers.

With understanding of the technologies I can proceed to implementation

Goals

- Create terraform modules
- Apply terraform modules to infrastructure
- Discuss how I can now manage the infrastructure without recreating anything

Implementation

1. Create terraform modules

It took me a while and many versions of scripts to figure out how to write terraform modules. I will show here only the final version that does what is needed.

- Declare providers for proxmox and talos in "providers.tf" https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/providers.tf
- Declare TalosOS file to download in Proxmox and use within terraform in "files.tf". https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/files.tf

!!! Important that I download a special TalosOS with cloud-init and qemu-agent support so that I can insert and extract initial configuration into/from the installed OS with Terraform.

- Declare proxmox connection for terraform in "main.tf"
  https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/main.tf
- Declare dynamic variables used in configs in "variables.tf"
  https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/variables.tf
- Declare talos OS controlplane patch to make it "single-node"
  https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/files/cp-scheduling.yaml
- Declare talos OS cluster configuration in "cluster.tf"
  https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/cluster.tf
  What it does:
  Creates secrets → generates Talos configs → applies them and additional patches to a Talos VM → bootstraps Kubernetes → waits for health → outputs talosconfig & kubeconfig
  !!! Note that the "execution" part of the config depends on "proxmox_virtual_environment_vm.talos_single_node" - which would be a virtual machine. Therefore this config wont do anything actual until there is such a vm created.
- Declare virtual machine to create in "virtual_machines.tf"
  https://github.com/Cloud-Operations-UCLL/individual-assignment-Rostislav-Dusnii/blob/main/terraform/working_version/virtual_machines.tf

1. Apply terraform modules

Now we can simply copy the declared terraform module into my Ubuntu server with terraform, go to the folder of the module and execute:

- Load terraform providers

```
terraform init
```

- Verify the changes we will apply

```
terraform plan
```

- Apply the changes

```
terraform apply
```

- After application we would be able to see a running cluster on the Proxmox server

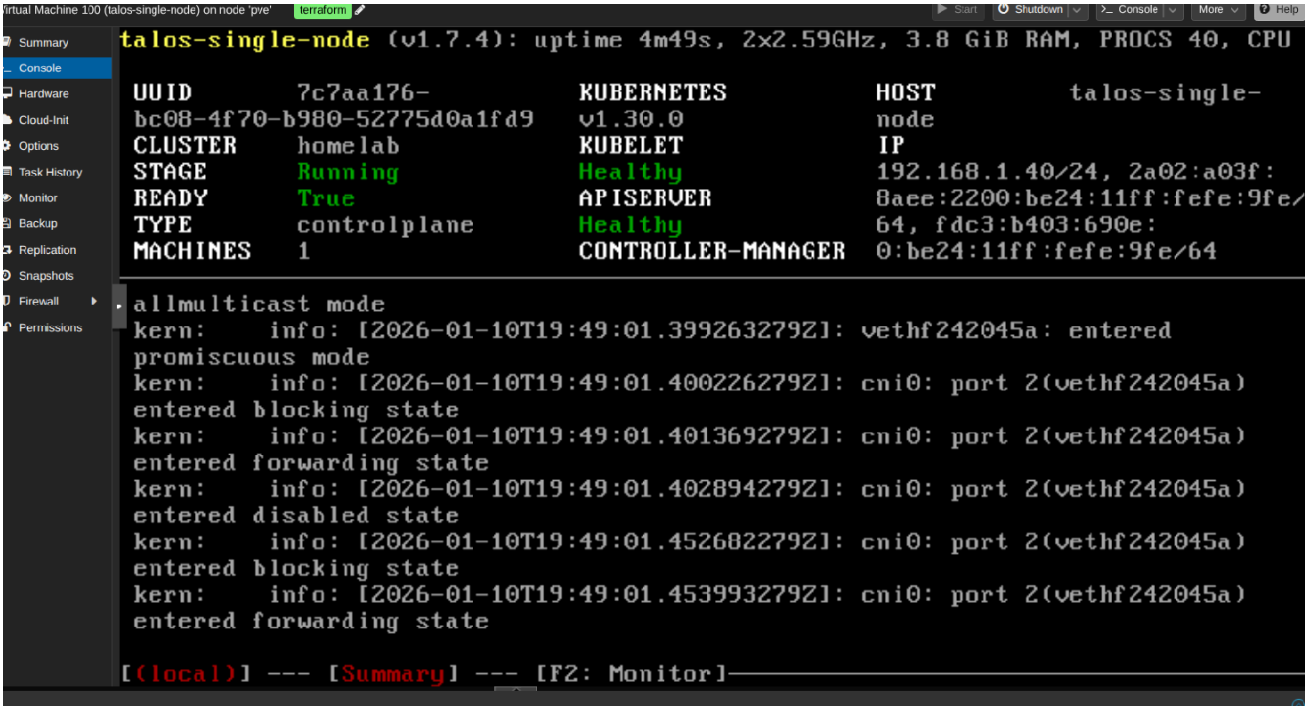


Figure 6 - Verifying the Talos OS virtual machine ready and running in Proxmox
Note that it says "MACHINES 1"

- We can also export kubeconfig and talosconfig via terraform outputs

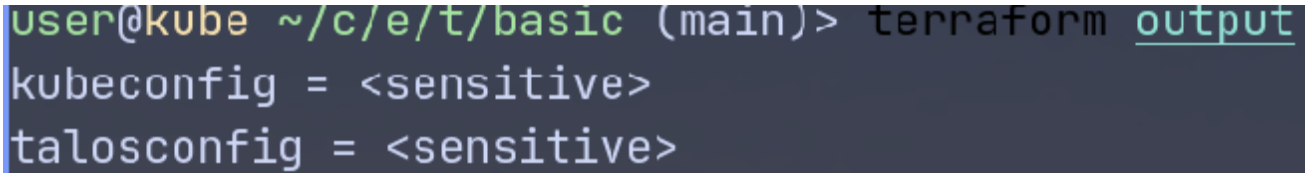


Figure 7 - Verifying that the desired outputs were exported

- After exporting kubeconfig into Lens we can see that we can manage the Kubernetes server and it is working fine

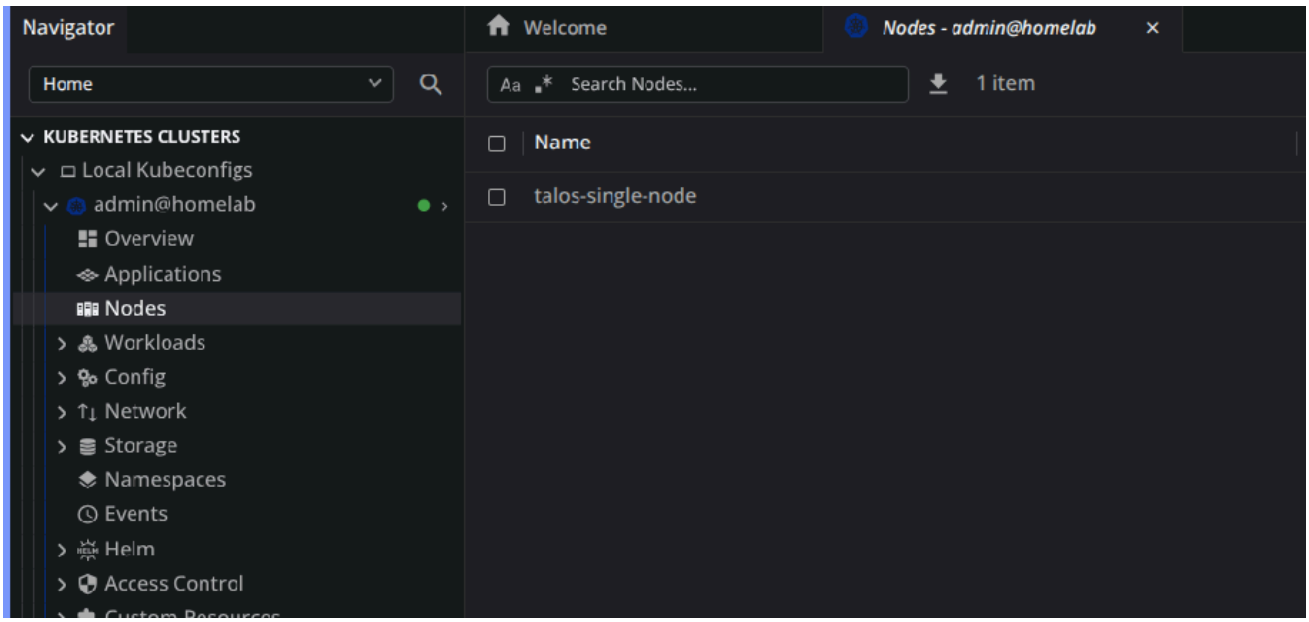


Figure 8 - Verifying the cluster via Lens and kubeconfig

- I created "test-nginx" pod to verify that the single-node cluster setup is working

```
user@kube ~/terraform> kubectl get pods -A
NAMESPACE     NAME                                          READY   STATUS    RESTARTS        AGE
default       test-nginx                                    1/1     Running   0               20s
kube-system   coredns-64b67fc8fd-5mvk8                      1/1     Running   0               5m9s
kube-system   coredns-64b67fc8fd-qpsgk                      1/1     Running   0               5m9s
kube-system   kube-apiserver-talos-single-node              1/1     Running   0               3m37s
kube-system   kube-controller-manager-talos-single-node     1/1     Running   1 (5m40s ago)   3m57s
kube-system   kube-flannel-bjh7l                            1/1     Running   0               5m5s
kube-system   kube-proxy-f7k7t                              1/1     Running   0               5m5s
kube-system   kube-scheduler-talos-single-node              1/1     Running   1 (5m39s ago)   3m56s
```

Figure 9 - Verifying that single-node cluster can run the new pods

3. Discuss how I can now manage the infrastructure without recreating anything

All of the technologies applied support dynamic changes that are well documented:

- If you want to change anything about the Virtual Machine itself - change the "virtual_machines.tf" and issue "terraform apply" - terraform will be able to automatically shutdown the machine, apply the changes and start the machine again
- If you want to change anything about the Talos Operating System itself you have 2 options:
  Use TalosCTL patching https://docs.siderolabs.com/talos/v1.9/configure-your-talos-cluster/system-configuration/patching
  Or change "cluster.tf" to let the Terraform do the job for you since it also supports patching. https://registry.terraform.io/providers/siderolabs/talos/latest/docs/resources/machine_configuration_apply
- If you want to change anything within the cluster - you can use Kubectl and other regular tools like Lens.

Encountered issues

1. It was confusing to discover that basic Talos OS does not support SSH or anything else but TalosCTL. Very unusual experience. Basically I always need 1 additional host to manage Talos OS
2. Firstly I tried to host Proxmox as a VM on my main host. This created a lot of different issues like the problem with bridging the wlan network interface, the problem that Proxmox is lagging and erroring while being a VM and way more. This was literally the main issue of the topic. When I decided to simply host proxmox on another Host with physical ethernet cable - it magically started working without any problems at all.
3. There are many Proxmox terraform providers. Some of them are outdated. It adds some complexity and challenges to figure out how to connect and use them. Gladly, at least Talos provider had a lot of useful documentation with examples.
   https://github.com/siderolabs/contrib/tree/main/examples/terraform/basic

Conclusion

Finally, all the requirements of the topics were met and I gained a lot of valuable experience discovering the Kubernetes magic. As usual, when you know what to do - the job takes minutes, but when you do not - it may take days or weeks. It is always funny to know. This course by far is the most satisfying in the semester - does not overwhelm you and gives you gold knowledge that you can choose. Simple and efficient. Thank you for your attention and support.