Examensarbete

“Exploring Lightweight Kubernetes Clusters: Installation, Trends, and Implications”

Namn	Josip Jovanovic, Javad Ein Moghassemi
Utbildning	Devops Developer
Uppdragsgivare	EC Utbildning, Fabled AB
Handledare/ examinator	Shahla Yagoutkar, Viktor Svensson
Datum	16 May 2023

Table of Content:

1. Abstract:

This study investigates the setup and maintenance of lightweight Kubernetes (K8s) clusters that have monitoring and CI/CD pipelines capabilities. Alternatively, the study is a limited survey of the K8s ecosystem, focusing on programs such as K3D (a lightweight K8s distributor), Grafana and Prometheus (for monitoring features), and Tekton (for CI/CD pipelines that are native to K8s). A secondary goal of this study is to facilitate new trainees with a tutorial that includes the installation process of this K8s bundle, and provide a brief description about each of its components. This report is written by two DevOps students at EC Utbildning who have combined their education with their LIA experience in order to equip individuals with the necessary knowledge and skills to leverage K8s effectively.

2. Introduction

2.1 Background:

Kubernetes (K8s), is the first graduate project (made available as a fully open source software) of the Cloud Native Computing Foundation (CNCF), and it has been majorly adopted to help manage swarms of containers within the IT industry since its initial release in September 2014 (Kubernetes, 2023). In its *State of Enterprise Open Source 2022* report, Redhat details that 70 percent of the 1,296 companies participating in its survey use K8s (Enterprise, 2023). Another report published the same year by Datadog titled *9 Insights on Real-World Container Usage*, describes that “nearly half of all organizations using containers run K8s to deploy and manage some of those containers" (CNCF, 2022). It must also be noted that the widespread and robust adaptation of K8s is partly due to the variety of K8s flavors that are offered by different distributors.

Some distributors offer basic K8s clusters (what is better known as “vanilla” K8s), whereas cloud computing platforms, such as Azure and AWS, offer clusters that are managed internally. There are also distributors that offer lightweight clusters with less dependencies. According to CNCF’s annual survey, in 2021, 30 percent of the participants used *Minikube* and 22 percent used *K3s*, which are both lightweight K8s distributors (CNCF, 2021).

To underline the significance of lightweight clusters, it is imperative to consider various scenarios where they offer a superior solution for computing needs. These scenarios include situations where access to abundant hardware power is limited, cost optimization is a priority, greater control over troubleshooting the clusters is desired, specific data storage locations are required, or the flexibility to relocate data to different physical locations is essential. In such cases, relying solely on cloud-based services may not be the optimal choice. Alternatively, the utilization of customized lightweight clusters, equipped with a curated selection of open-source software, proves to be highly advantageous compared to alternative options.

Moreover, according to CNCF’s surveys, training new K8s users is a major obstacle within the industry. Comprehensive tutorials play a pivotal role in enhancing the learning process. While numerous online tutorials are available that primarily focus on individual programs within the Kubernetes ecosystem, the availability of tutorials that investigate a collection of programs is relatively limited. This lack of such comprehensive resources has served as inspiration for producing this report.

2.2 Purpose

The objective of this study is to provide a comprehensive guide on establishing a lightweight and high-performing Kubernetes cluster with a complete feature set for CI/CD pipelines and observability. The installation process will also be documented to create a tutorial that can be used for training purposes. In addition to the cluster, other major phases of this report include deploying an application with a database to the cluster, and using K8s' exclusive package manager known as Helm, for the deployments.

2.3 Problem Formulation

This study investigates a collection of open-source programs that can be integrated into K8s clusters. It provides comprehensive information about the installation procedures, along with a brief description of each component. Furthermore, the study addresses prevalent challenges that users may encounter when working with these technologies.

The following topics will be investigated in this research:

1) Provide a concise overview of a few essential features of K8s components and create a tutorial based on it.
2) Determine which tools and software within the Kubernetes ecosystem can facilitate the deployment of clusters with CI/CD pipelines and observability features, particularly in scenarios where minimizing hardware usage is a priority.
3) Create a set of instructions for installing and troubleshooting the selected K8s bundle.

2.4 Limitations

This study acknowledges certain limitations concerning the depth of coverage regarding K8s' architecture. Due to the extensive number of components involved in K8s, it is deemed satisfactory to provide a fundamental understanding at a basic level, rather than delving into intricate details. The primary objective is to demonstrate the capabilities of the examined software to new students, offering a beginner-level perspective.

Additionally, it should be noted that the installation of prerequisites for the project will not be addressed within the scope of this study. While the installation of prerequisites is essential for the successful implementation of the project, it is beyond the immediate focus of this research.

3. Method

The primary objective of this study is to examine a specific K8s bundle that demonstrates minimal hardware resource consumption. The outcome of this investigation will result in the development of a replicable model that is accessible for first-time users of Kubernetes. In order to ensure clarity and enhance readability, this research relies extensively on referencing articles, surveys, and official documentation provided by vendors. Additionally, insights derived from an interview conducted with the authors' LIA supervisor, Robin Morero, have been incorporated into the text.

There are other tools and softwares that need to be pre installed when following this tutorial. Notably, the prerequisites for the project include WSL 2 (Windows Subsystem for Linux 2), Docker Desktop, and access to a Linux CLI. The programs in the K8s bundle will be installed in this order: K3D, Grafana, Tekton, Helm, and MongoDB

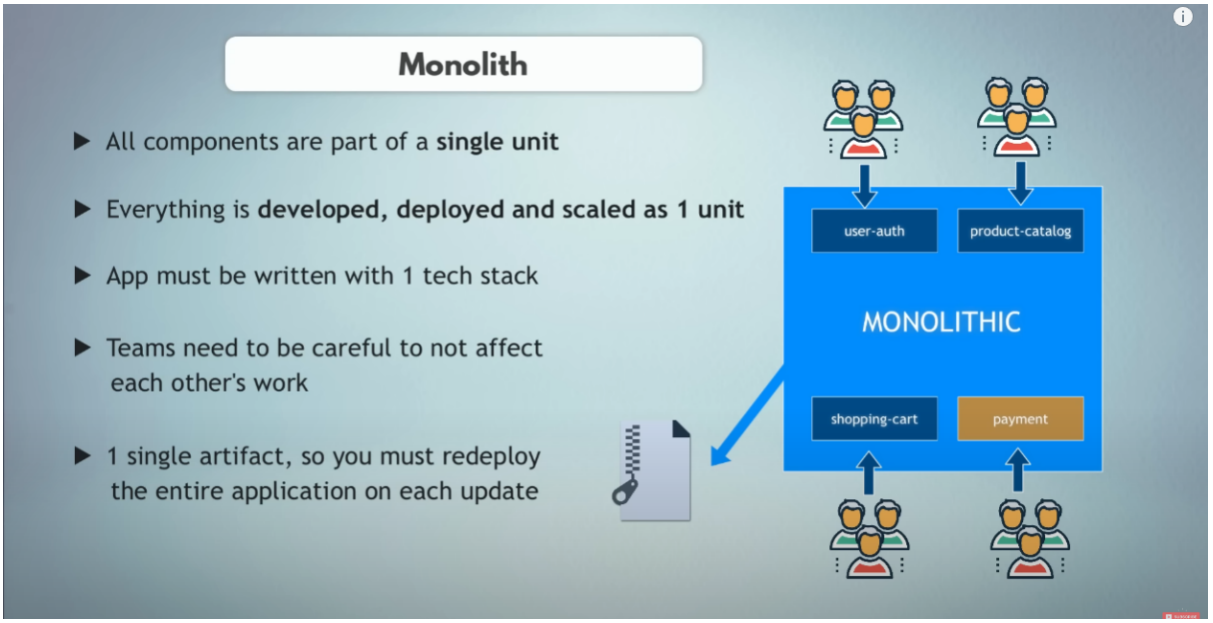
4. Theory

The softwares that are utilized in this study will be explained in the order provided below:

1) Monolithic Architecture VS. Microservices (a brief history)
2) K3D
3) Monitoring
4) CI/CD
5) Helm
6) MongoDB

Monolith Architecture:

The main characteristic of a Monolithic architecture is that all the different components of a software artifact are considered as a single unit or “stack”. By building software products with a single code-base, and limiting the deployment configurations to a single file, this type of architecture allows for easier development, debugging, and deployment. However, this approach also has some shortcomings. For example, if developers need to make small updates or changes to an artifact, they ought to interact with the entire source code of that artifact, which in turn makes small updates more time consuming and more costly. Furthermore, monolithic software is harder to scale.

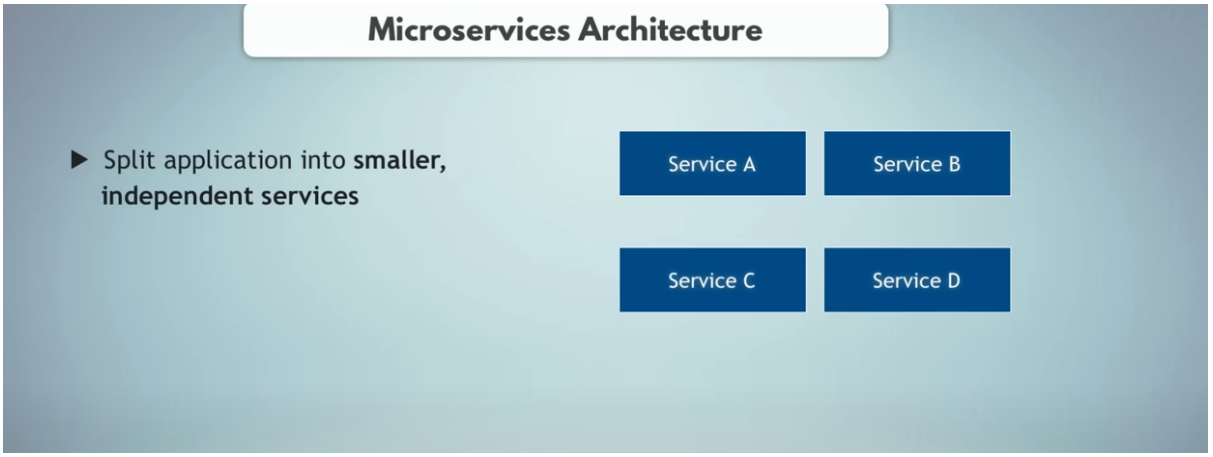


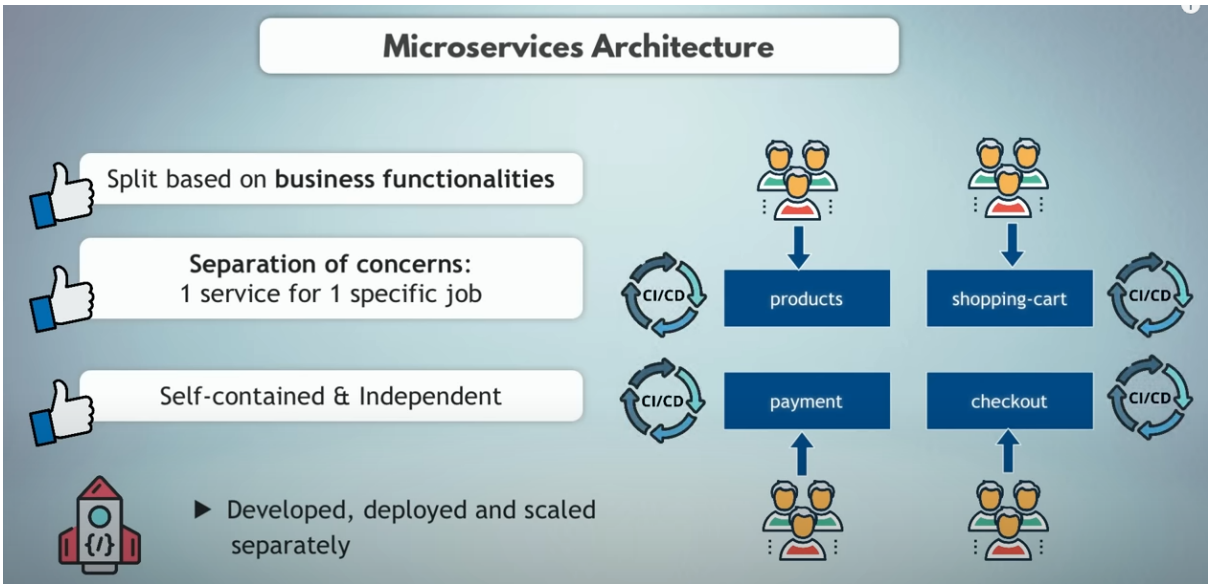
(TechWorld with Nana. 2022)

Microservices Architecture:

The Microservice architecture divides the different components of an artifact into smaller, independent services, giving rise to micro applications. Since microservices are self-contained and independent, it means that they can be built and deployed separately .

“Monoliths are not always negative. For a simple prototype, or a single team, it may be more effective (Robin Morero, 2023)”





(TechWorld with Nana. 2022)

K3D:

"K3D is a lightweight wrapper to run K3S (Rancher Lab's minimal Kubernetes distribution) in docker"(K3D, 2022).

K3S is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances. We will be using K3D for our cluster, for its multi-node capabilities.

Monitoring (Prometheus and Grafana):

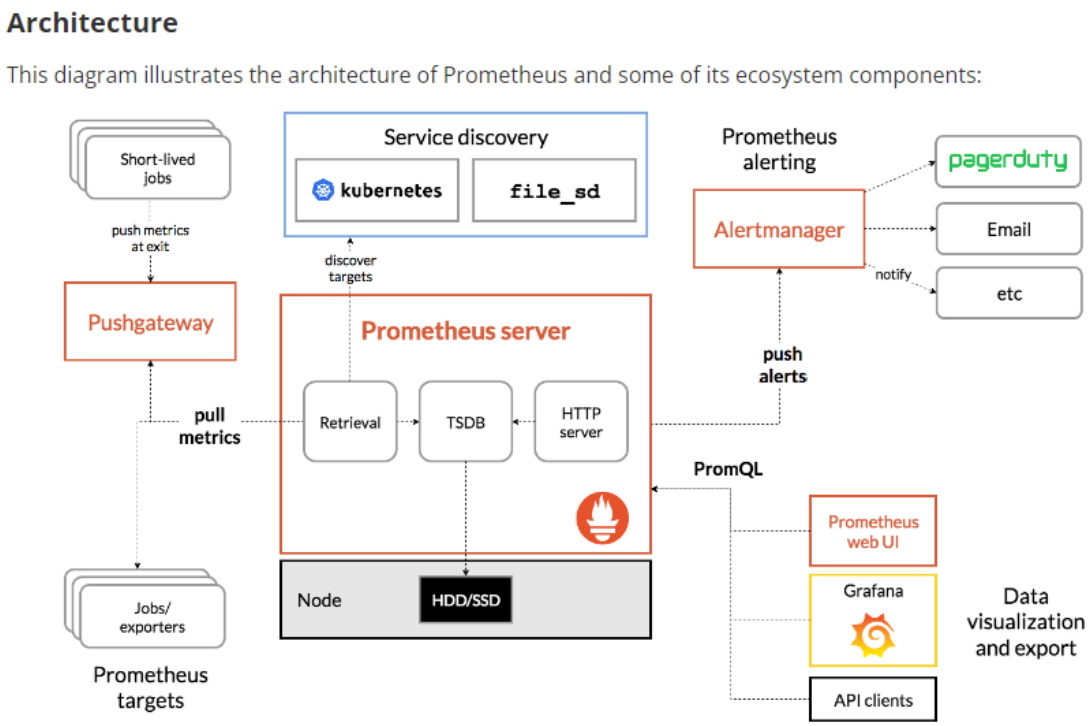
Prometheus

“Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company. Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels”(Prometheus, 2023).

Prometheus uses a language called PromQL for its queries.

This is very useful for our clusters well-being, this way we can monitor our pods and services.

We will install it to our cluster using helm charts.



Grafana

“Grafana is an open source interactive data-visualization platform, developed by Grafana Labs, which allows users to see their data via charts and graphs that are unified into one dashboard (or multiple dashboards!) for easier interpretation and understanding” (Grafana 2022).

Grafana goes hand in hand with Prometheus. Grafana can be connected to Prometheus by adding Prometheus as a data source in Grafana. This allows Grafana to retrieve metrics data from Prometheus and display it in dashboards.We will also install grafana through a helm chart.

Tekton:

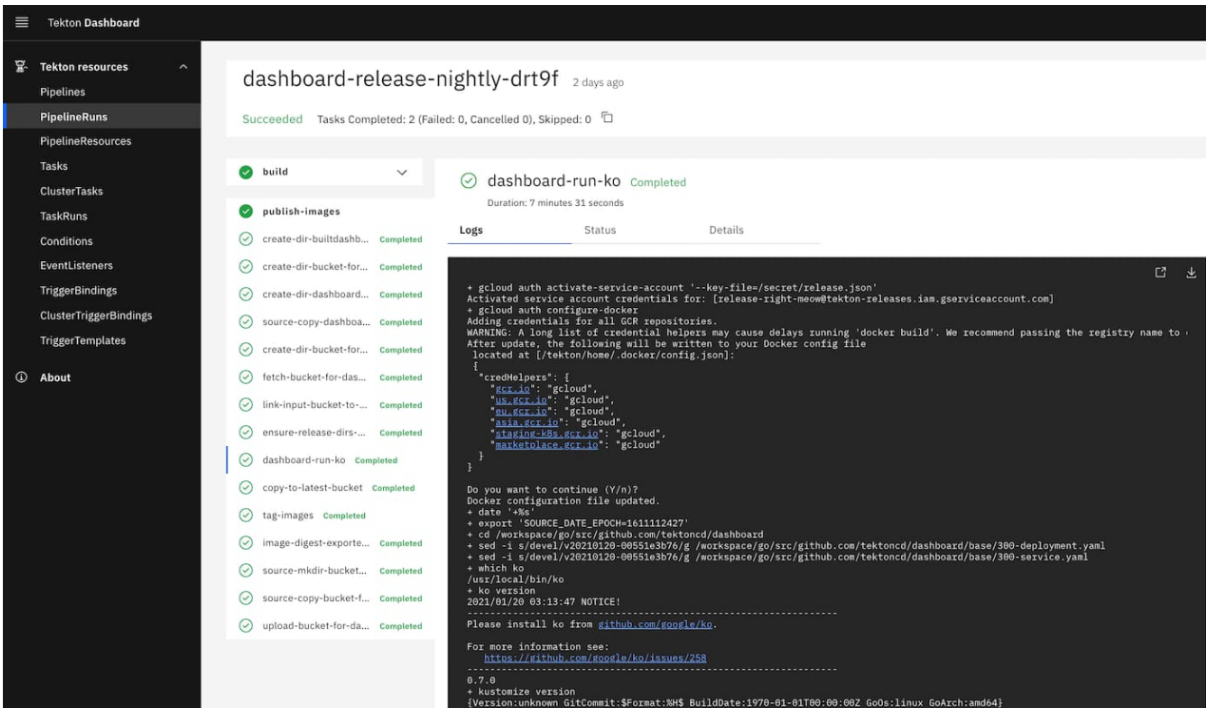
“ Tekton is a cloud-native solution for building CI/CD pipelines. It consists of Tekton Pipelines, which provides the building blocks, and of supporting components, such as Tekton CLI and Tekton Catalog, that make Tekton a complete ecosystem” (Tekton, 2018).

We decided that we are trying out Tekton to see if this matches what Robin says.

In our practical part we will create a simple pipeline using tekton that gets the information from a github repo and deploys the pipe.

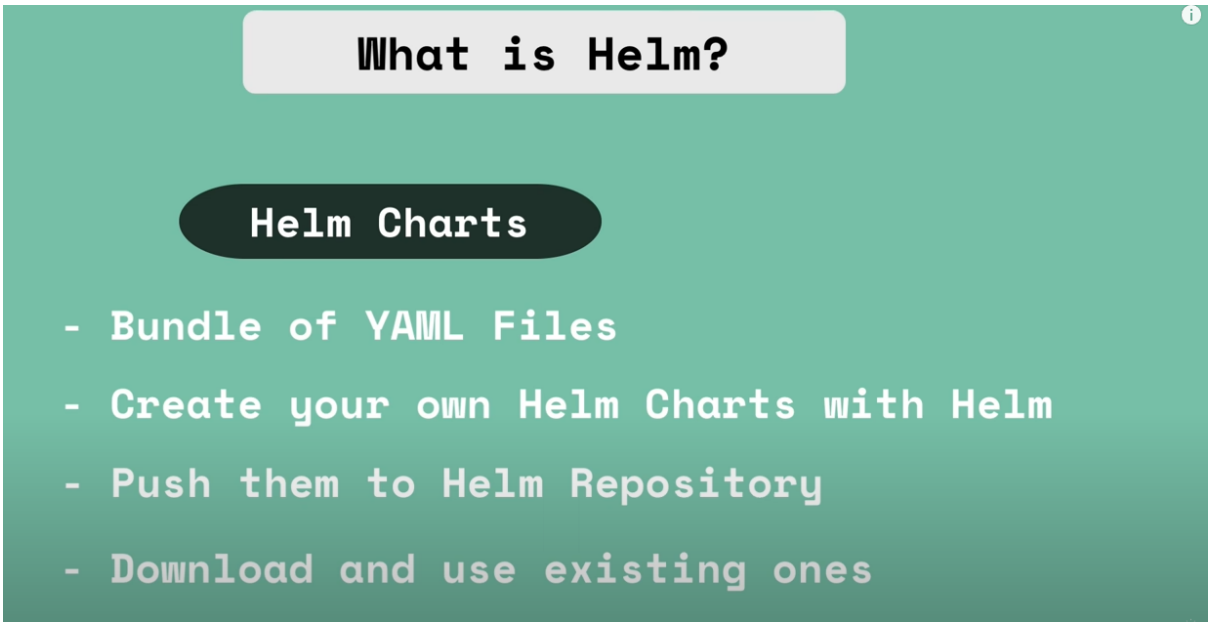
We will also install Tekton Dashboard for its GUI capabilities.

We will install it to our cluster using helm charts.



Helm:

Helm is a package manager for K8s and it can be used to “define, install, and upgrade K8s applications” (Helm). This package manager reduces redundancy through its “Helm Charts” feature. This feature allows to bundle all the yaml files that are required by K8s for deployment (Secret, Configmap, Deployment.yaml, etc.) into a single chart. By utilizing “Helm Charts”, developers can avoid applying yaml files one by one.



MongoDB

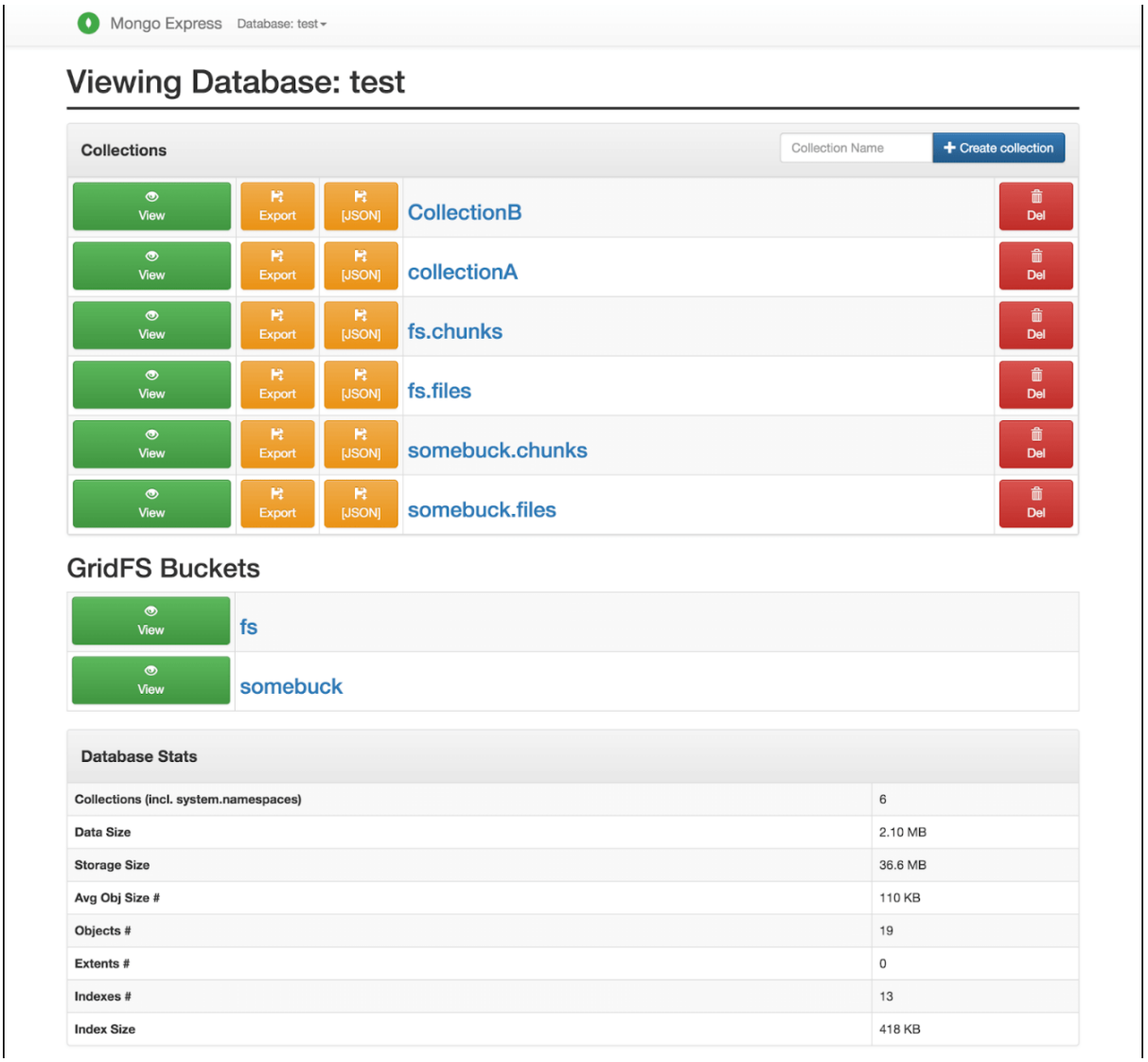
MongoDB is a popular open-source document-oriented NoSQL database. It uses a flexible and dynamic schema which allows for easy and efficient data modeling, storage, and retrieval. Rather than using tables and rows as in traditional relational databases, MongoDB stores data as JSON-like documents with dynamic schemas, allowing for more flexibility and scalability.

MongoDB is also known for its high performance, scalability, and availability. It can handle large amounts of data, making it well-suited for applications that require high levels of scalability and performance. Additionally, MongoDB provides built-in replication and sharding, which enables data to be distributed across multiple servers for improved availability and scalability.

Other benefits of using MongoDB include its support for a variety of programming languages, ease of use, and rich querying capabilities. It is also highly flexible and can be used for a variety of use cases, including web and mobile applications, content management systems, and analytics platforms.

MongoDB Express

MongoDB Express is a web-based administrative interface for MongoDB. It allows users to manage MongoDB databases, collections, and documents using a graphical user interface rather than the command-line interface provided by the MongoDB shell. MongoDB Express is often used as a companion tool to MongoDB, providing a simple and intuitive way for developers and administrators to interact with their MongoDB data. It is an open-source project and can be downloaded and installed for free. We will install both with helm.



5. Results

5.1 Interview

We interviewed Robin Morero on 10 of May, 2023. He is our LIA supervisor at Fabled and we got an interesting interview from him. Robin was talking about his company, how he sees his company moving forward, and based on what criteria they choose to work with an open source software.

Javad: How does Fabled choose which software to use from the K8s ecosystem?
Robin:

“It should be easy to work with, easy to maintain and it should work well in the cluster. That's the 3 main components i think! It should also be easy to upgrade and a good community and fairly stable. We should also be able to replace components when we need to. Keeping it lightweight supports all of that in a way that more heavy software does not! ”

Josip: Why did you decide to use Tekton, Grafana, K3D, Database?

Robin:

Tekton

“Fabled is using Tekton for its lightweight, flexible capabilities, complex pipelines and they also wanted something that runs well natively in k8s. And at the moment Tekton was our best option, and runs fairly well today! ”

Grafana

“ ElasticSearch would be an alternative but Grafana had the possibility of keeping all of its components open-source and it's so lightweight. That's a big deal for us, i mean ElasticSearch has Java involved, OpenSearch is the same, uses a lot more resources and takes a lot more maintenance. Grafana stack is low maintenance, you don't have to do a lot with it! ”

K3S

“Instead of going to Vanilla K8s, we wanted K3S because it's low on resources. That's the main rivalry. It's not a big deal really it's only 5 percent, but since we have a lot of small customers we want a minimal possible server to run it at because it's cheaper. It makes a difference! So it's a matter of cost effectiveness. ”

Database

“There are a lot of Postgres everywhere, and we tried a different flavor of postgres, we tried Cockroach, Yugabyte, we tried different operators for postgres for it to run in k8s. So far none of the solutions has been ideal, we like postgres but we don't like how it manages in k8s. Yugabyte had a problem of being very slow with starting up creating tables, that's what bothered us with that one. Cockroach is missing some features that were missing from postgres. ”

Josip: “ Do you think mongoDB is a good viable choice database?

Robin: ”MongoDB is a safe bet I think! It's been used in a lot of places, fairly easy to manage, it's a nice developer experience if you worked with document databases before. It depends on the use case! The reason we are using postgres is that the standard MYSQL has been missing some features we have been using and it has been scaling worse. Postgres has become the standard I think in terms of Relational DB mess. ”

Javad: “ What do you believe are the future trends for these features? ”

Robin:

“ What we are seeing is a couple of trends, the first one is to manage the complexity, which means you set a separate platform team that will manage that. You use vendors to manage the complexity, you must keep adding stuff into k8s cluster to handle it as well, to get line of costs to keep track of what it is costing you, you get carbon footprint, you get security visibility, compliance visibility and so on.

The other approach is to minimize complexity, that means you make it as lightweight as you can. You reduce the maintenance work you have to do, and you make it very easy for developers to do whatever they really need to do in the cluster. And our approach has always been to minimize the complexity, it works better, because when adding things to the cluster the complexity grows. So we try to avoid adding it when not necessary. I Mean no one is going to really complain to you about your k8s

cluster costing too much if it's not really costing a lot. So if you keep it lightweight you're not gonna get complaints in the beginning.

Josip: “ If you wouldn't choose k8s today in your structure, what would you use today? ”

Robin: “Nomad is still an option for k8s today. Not that common but it's still there. K8s dominates the market today! ”

Javad: “ Will Grafana and Tekton be relevant in the near future?”

Robin: “Grafana is growing in popularity right now and people are adopting it and will be relevant in the near future.

Tekton might be replaced by other options because it can get complex, let's say you want callbacks from git repo to have a webhook, that could get complicated. ”

5.2 The Installation Tutorial

Installing Tekton

Do the following steps in this order.

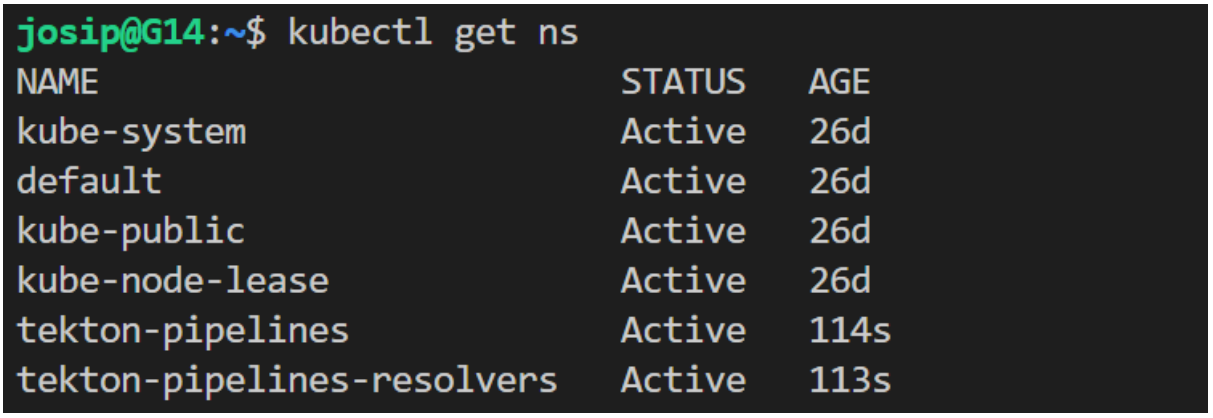
Go to this git repo: https://github.com/JOJO840/Tekton-pipe-demo

then do a: git clone https://github.com/JOJO840/Tekton-pipe-demo.git

or with the ssh method: git clone git@github.com:JOJO840/Tekton-pipe-demo.git

Run the bash file ./tekton-cmds.sh, this command will install Tekton pipelines and dashboard and also get the updates from apt library.

If you run this cmd ´kubectl get ns´ you will now see a new namespace where tekton-pipelines lives.



To see the every pod in that namespace type in this cmd

´kubectl get all -n tekton-pipelines´

```
josip@G14:~$ k get all -n tekton-pipelines
NAME                                              READY   STATUS    RESTARTS   AGE
pod/tekton-pipelines-controller-85767d6dc8-p6gmj  1/1     Running   0          45m
pod/tekton-pipelines-webhook-54cc86bf59-dqsbk     1/1     Running   0          45m
pod/tekton-dashboard-7487756644-tdwjb             1/1     Running   0          15m

NAME                                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                        AGE
service/tekton-pipelines-controller   ClusterIP   10.43.34.196   <none>        9090/TCP,8008/TCP,8080/TCP     45m
service/tekton-pipelines-webhook      ClusterIP   10.43.178.73   <none>        9090/TCP,8008/TCP,443/TCP,8080/TCP   45m
service/tekton-dashboard              ClusterIP   10.43.77.139   <none>        9097/TCP                       20m

NAME                                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/tekton-pipelines-controller  1/1     1            1           45m
deployment.apps/tekton-pipelines-webhook     1/1     1            1           45m
deployment.apps/tekton-dashboard             1/1     1            1           20m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/tekton-pipelines-controller-85767d6dc8  1         1         1       45m
replicaset.apps/tekton-pipelines-webhook-54cc86bf59     1         1         1       45m
replicaset.apps/tekton-dashboard-7487756644             1         1         1       15m
replicaset.apps/tekton-dashboard-6694db567c             0         0         0       20m

NAME                                                                    REFERENCE                                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/tekton-pipelines-webhook   Deployment/tekton-pipelines-webhook   4%/100%   1         5         1          45m
```

To connect to the Tekton dashboard svc write in this cmd:

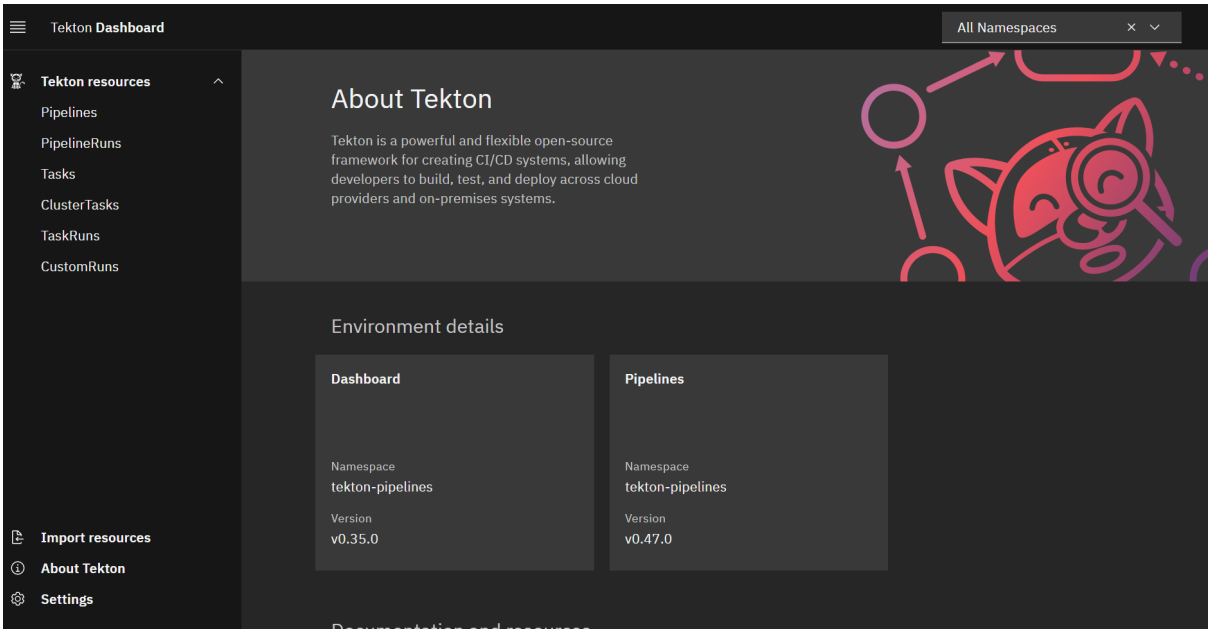
kubectl port-forward -n <namespace> svc/<svc.name> port number

in our case:

kubectl port-forward -n tekton-pipelines svc/tekton-dashboard 9097

```
josip@G14:~/examensProject/Tekton-pipe-demo$ kubectl port-forward -n tekton-pipelines svc/tekton-dashboard 9097
Forwarding from 127.0.0.1:9097 -> 9097
Forwarding from [::1]:9097 -> 9097
Handling connection for 9097
Handling connection for 9097
Handling connection for 9097
Handling connection for 9097
Handling connection for 9097
```

type in localhost:9097 in your browser, we use chrome. This will redirect you to the Tekton dashboard.

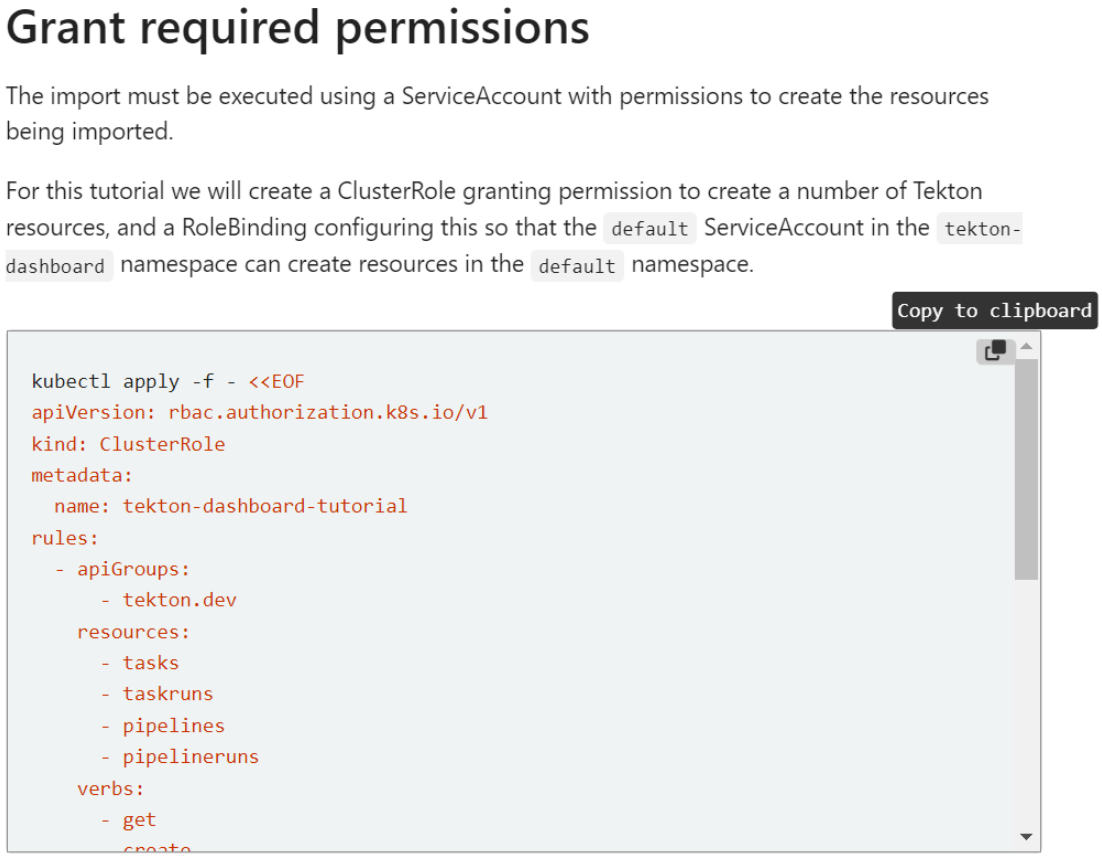


Tekton Import Resources:

To make a pipeline in Tekton we will use the 2 files in the github repo pipeline.yaml and tasks.yaml. https://github.com/JOJO840/Tekton-pipe-demo/tree/main

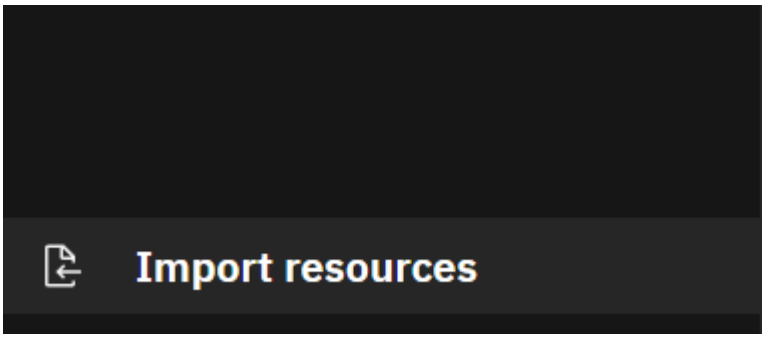
Steps for pipeline:

1. For Tekton to import and create resources to your cluster it needs permission to do so. We can get it at Tektons site at https://tekton.dev/docs/dashboard/tutorial/ and use the copy to clipboard icon and paste in the commands in your cli.

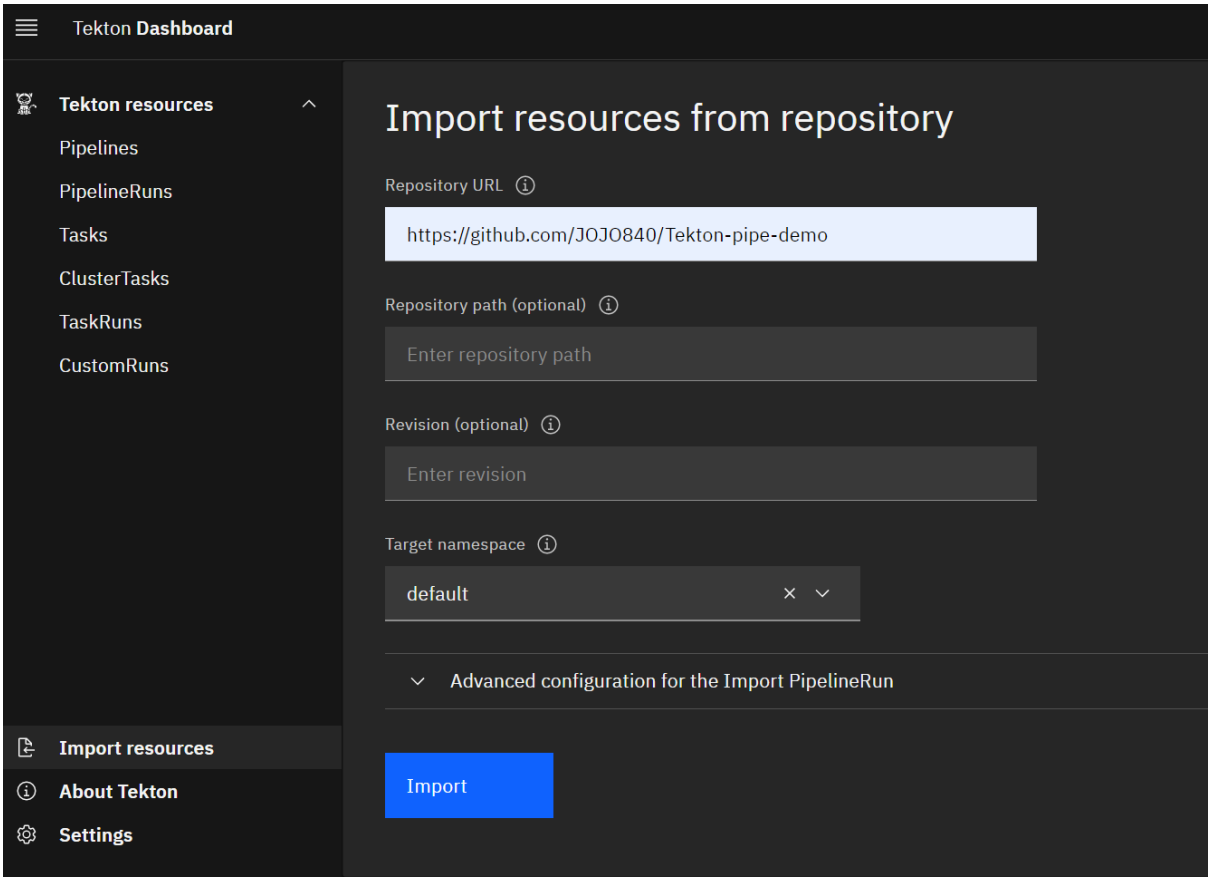


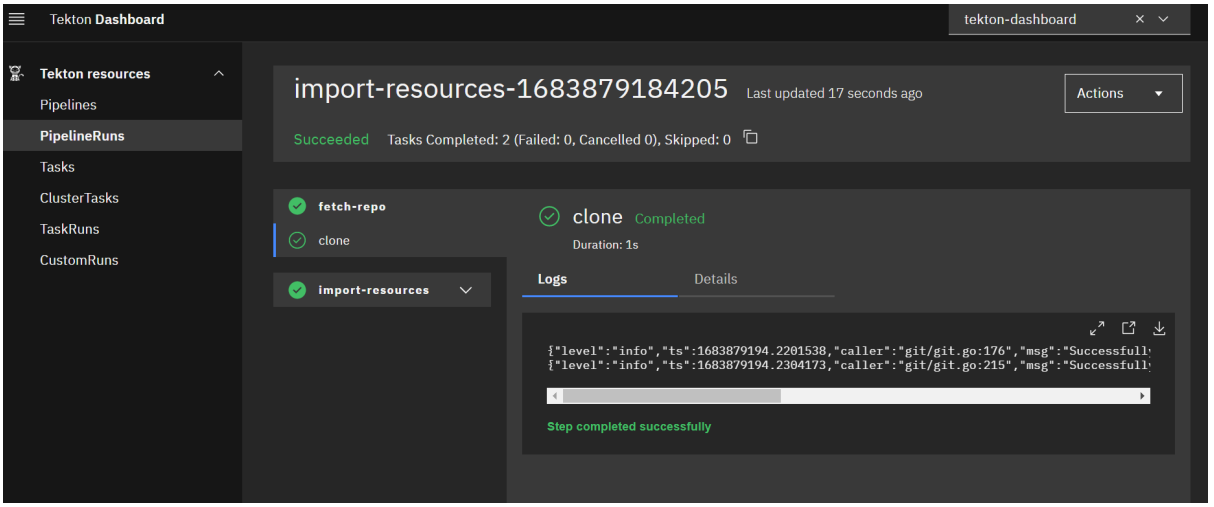
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

josip@G14:~$ kubectl apply -f - <<EOF
> apiVersion: rbac.authorization.k8s.io/v1
> kind: ClusterRole
> metadata:
>   name: tekton-dashboard-tutorial
> rules:
>   - apiGroups:
>       - tekton.dev
>     resources:
>       - tasks
>       - taskruns
>       - pipelines
>       - pipelineruns
>     verbs:
>       - get
>       - create
>       - update
>       - patch
> ---
> apiVersion: rbac.authorization.k8s.io/v1
> kind: RoleBinding
> metadata:
>   name: tekton-dashboard-tutorial
>   namespace: default
> roleRef:
>   apiGroup: rbac.authorization.k8s.io
>   kind: ClusterRole
>   name: tekton-dashboard-tutorial
> subjects:
>   - kind: ServiceAccount
>     name: default
>     namespace: tekton-dashboard
> EOF
clusterrole.rbac.authorization.k8s.io/tekton-dashboard-tutorial created
rolebinding.rbac.authorization.k8s.io/tekton-dashboard-tutorial created
josip@G14:~$
```

2. Now in the Tekton dashboard Click on Import Resources

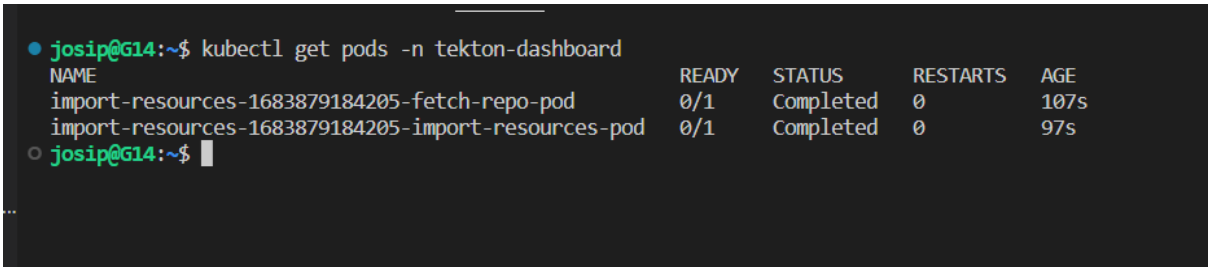


3. Put in the git repo and namespace to default like the picture below and click on import.



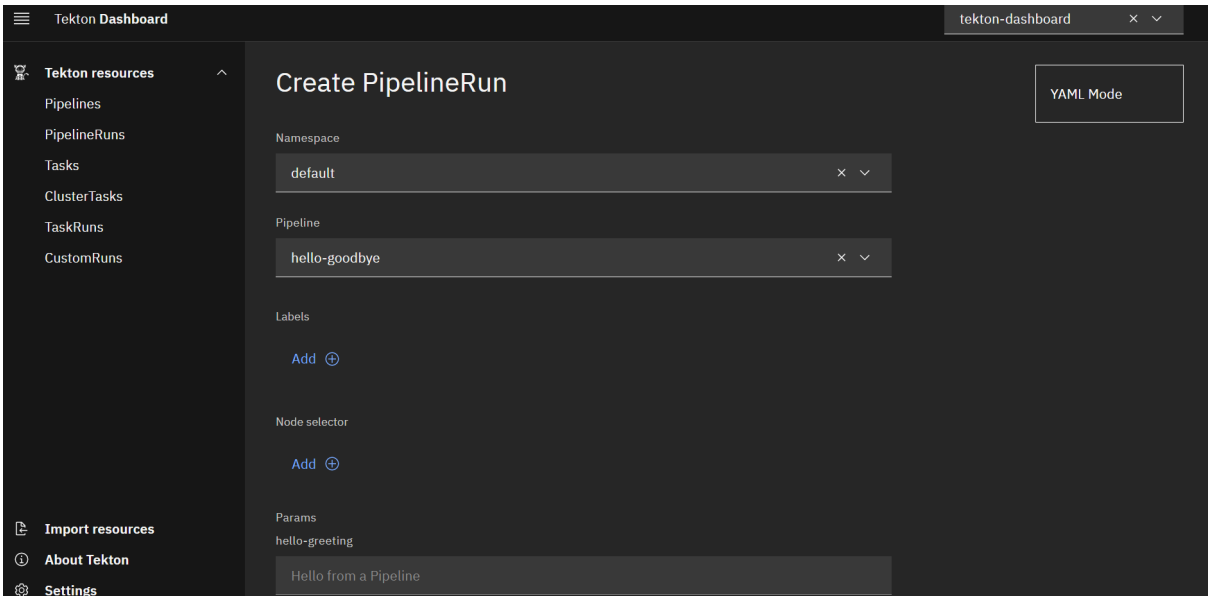


you should have two pods up and running in your cluster like this:

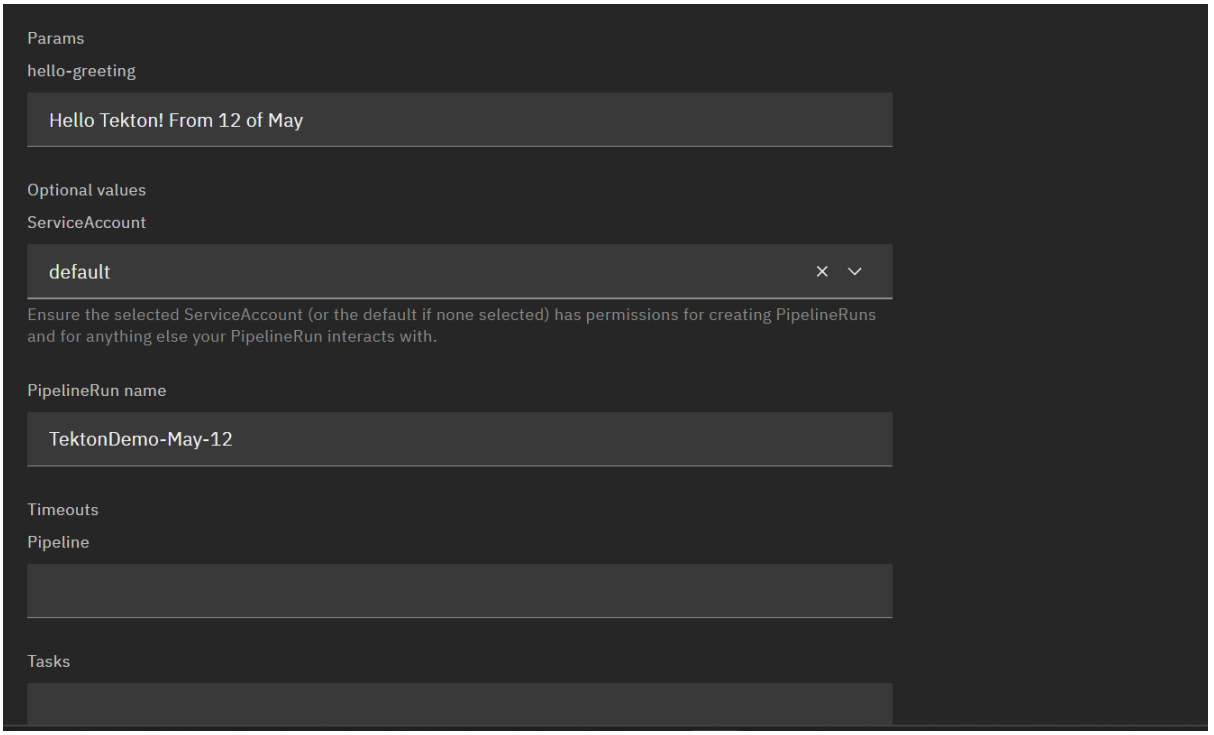


Create a PipelineRun

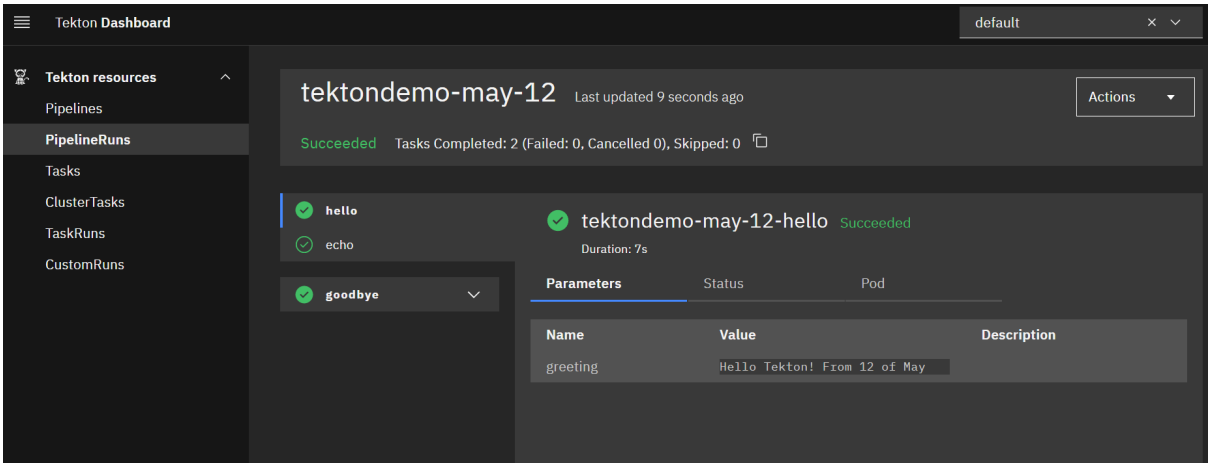
1. Next let's create a PipelineRun using all of the resources we imported



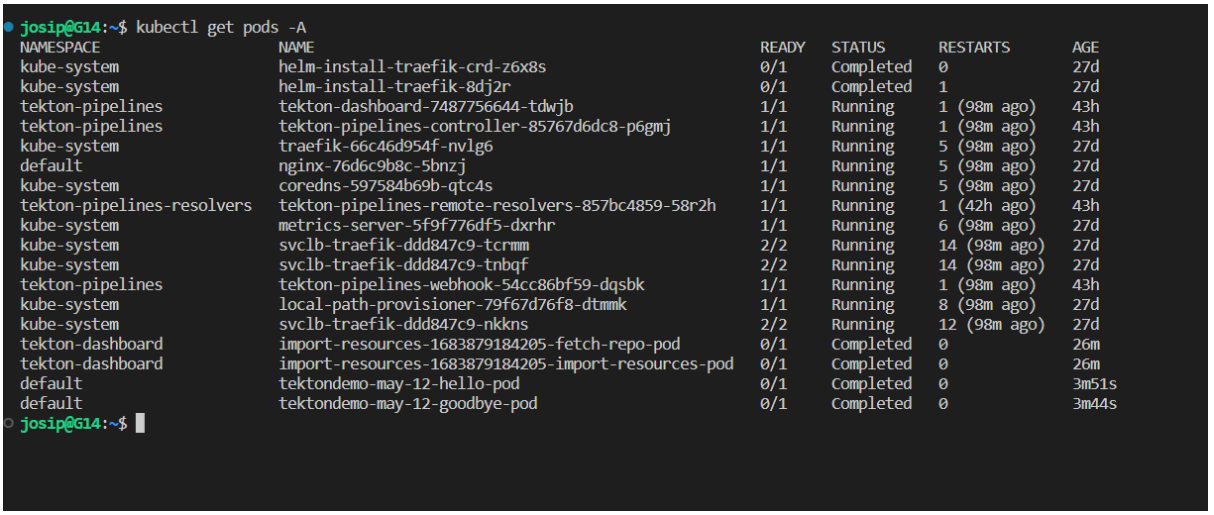
2. Here you can also choose paragraphs for your pipeline and name it. Then click blue button "Create"



3. Now the pipeline should be green and you will also see some more pods in your cluster.



4. type in kubectl get pods -A, you will see two new pods in the default namespace now.



Installing Grafana, Prometheus and Loki

1. Firstly we need to add the repos to Helm for the installation. Run these commands:
2. helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
3. helm repo add grafana https://grafana.github.io/helm-charts
4. helm repo list

```
josip@G14:~$ helm repo list
NAME                    URL
bitnami                 https://charts.bitnami.com/bitnami
nginx-stable            https://helm.nginx.com/stable
ingress-nginx           https://kubernetes.github.io/ingress-nginx
prometheus-community    https://prometheus-community.github.io/helm-charts
grafana                 https://grafana.github.io/helm-charts
josip@G14:~$
0 ⚠ 0   k3d-k3s-default   default
```

this should be your output.

5. helm install prometheus prometheus-community/kube-prometheus-stack

```
josip@G14:~$ helm install prometheus prometheus-community/kube-prometheus-stack
NAME: prometheus
LAST DEPLOYED: Fri May 12 12:23:12 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using t
he Operator.
```

6. helm install loki grafana/loki-stack
7. helm list

```
josip@G14:~$ helm install loki grafana/loki-stack
NAME: loki
LAST DEPLOYED: Fri May 12 12:25:25 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
The Loki stack has been deployed to your cluster. Loki can now be added as a datasource in Grafana.

See http://docs.grafana.org/features/datasources/loki/ for more detail.
josip@G14:~$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERS
ION
loki            default         1               2023-05-12 12:25:25.754099674 +0200 CEST        deployed        loki-stack-2.9.10               v2.6.1

prometheus      default         1               2023-05-12 12:23:12.866842416 +0200 CEST        deployed        kube-prometheus-stack-45.27.2   v0.65.1
```

You should have a cluster like this in the default namespace

```
josip@G14:~$ kubectl get pods
NAME                                                     READY   STATUS      RESTARTS        AGE
nginx-76d6c9b8c-5bnzj                                    1/1     Running     5 (3h25m ago)   27d
tektondemo-may-12-hello-pod                              0/1     Completed   0               111m
tektondemo-may-12-goodbye-pod                            0/1     Completed   0               111m
prometheus-prometheus-node-exporter-hhtvg                1/1     Running     0               3m42s
prometheus-prometheus-node-exporter-b78jn                1/1     Running     0               3m42s
prometheus-prometheus-node-exporter-4q2wf                1/1     Running     0               3m42s
prometheus-kube-state-metrics-858496487b-nxvpn           1/1     Running     0               3m42s
prometheus-kube-prometheus-operator-5c4bd7f4cc-ghn52     1/1     Running     0               3m42s
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running     0               3m32s
prometheus-grafana-5f57fbddd9-bfnnc                      3/3     Running     0               3m42s
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running     0               3m32s
loki-promtail-665vv                                      1/1     Running     0               102s
loki-promtail-g6w89                                      1/1     Running     0               102s
loki-promtail-8vdks                                      1/1     Running     0               102s
loki-0                                                   1/1     Running     0               102s
josip@G14:~$
```

with Prometheus, grafana and loki installed.

8. Next part is to connect to Grafana and insert the databases, we start with the command: kubectl get svc -n default

```
josip@G14:~$ kubectl get svc -n default
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
kubernetes                                ClusterIP   10.43.0.1       <none>        443/TCP                      27d
nginx                                     ClusterIP   10.43.89.42     <none>        80/TCP                       27d
prometheus-kube-prometheus-operator       ClusterIP   10.43.165.67    <none>        443/TCP                      31m
prometheus-kube-state-metrics             ClusterIP   10.43.68.63     <none>        8080/TCP                     31m
prometheus-grafana                        ClusterIP   10.43.22.216    <none>        80/TCP                       31m
prometheus-prometheus-node-exporter       ClusterIP   10.43.108.20    <none>        9100/TCP                     31m
prometheus-kube-prometheus-prometheus     ClusterIP   10.43.130.169   <none>        9090/TCP                     31m
prometheus-kube-prometheus-alertmanager   ClusterIP   10.43.131.38    <none>        9093/TCP                     31m
alertmanager-operated                     ClusterIP   None            <none>        9093/TCP,9094/TCP,9094/UDP   31m
prometheus-operated                       ClusterIP   None            <none>        9090/TCP                     31m
loki-memberlist                           ClusterIP   None            <none>        7946/TCP                     29m
loki-headless                             ClusterIP   None            <none>        3100/TCP                     29m
loki                                      ClusterIP   10.43.62.169    <none>        3100/TCP                     29m
josip@G14:~$
```

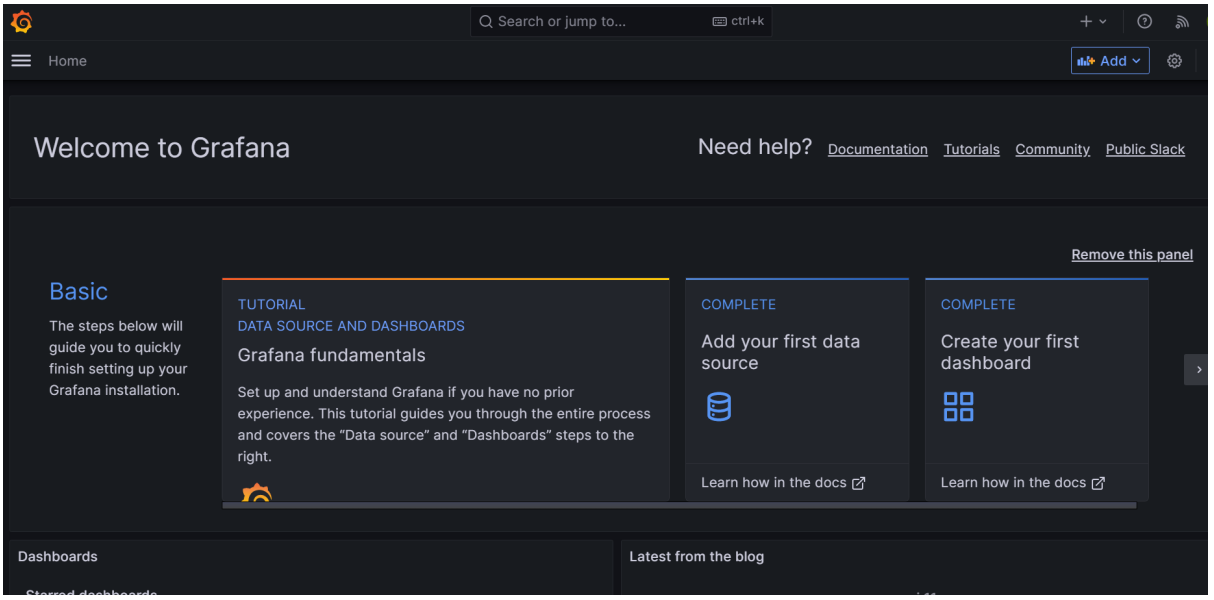
We need to connect to the svc of grafana which is called prometheus-grafana.

9. To access the grafana GUI we need the secret password from kubectl get secret, run this command: kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}' | base64 --decode

```
josip@G14:~$ kubectl get secret prometheus-grafana -o jsonpath='{.data.admin-password}' | base64 --decode
prom-operatorjosip@G14:~$
```

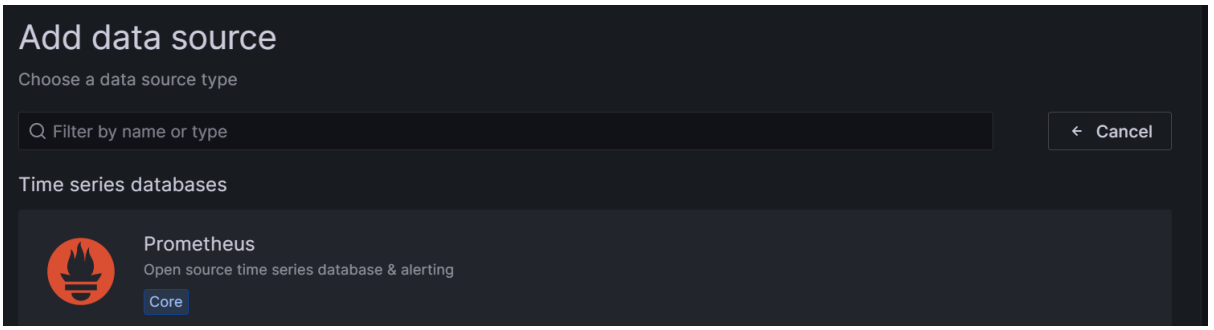
in our case its prom-operator. And the username is of course admin.

10. Now we port-forward into the svc locally with: kubectl port-forward svc/prometheus-grafana 8082:80. We selected 8082 here because it's an available port, and type in localhost:8082 in our browser.
11. Now we can add prometheus and loki as data sources.

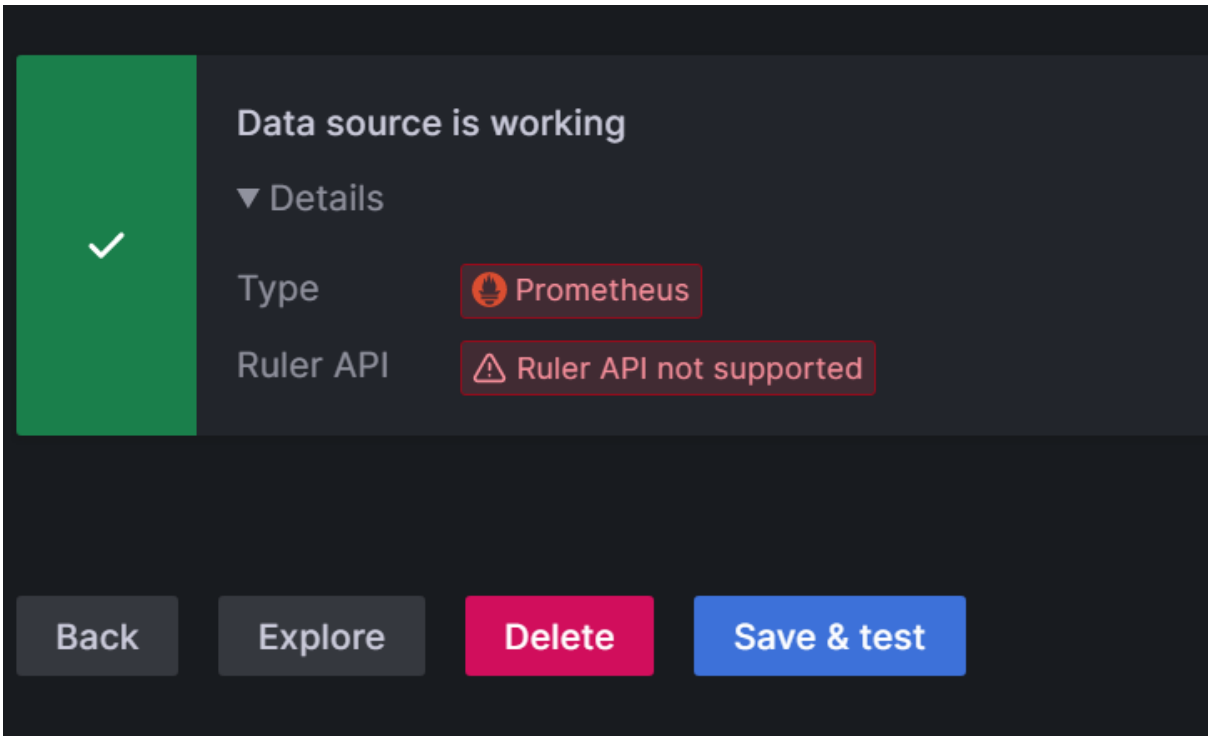


12. click on Add your first data source.

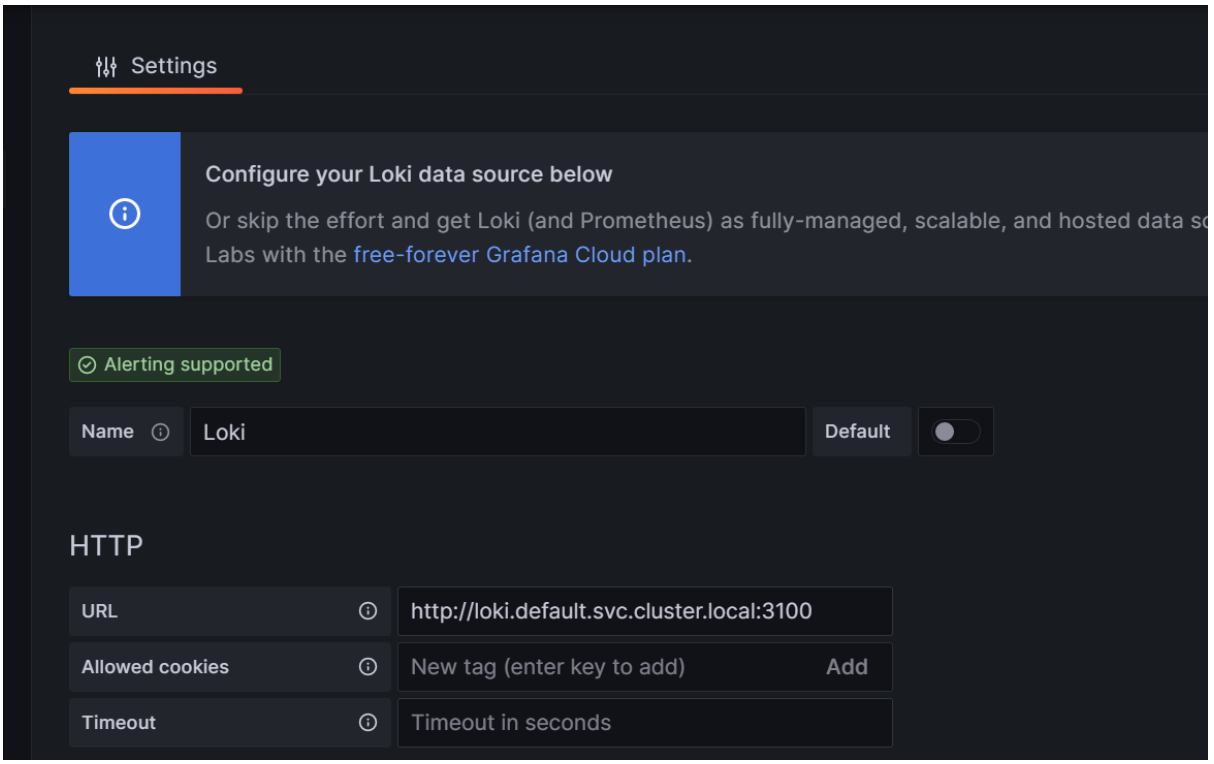
13. We select prometheus to begin with.



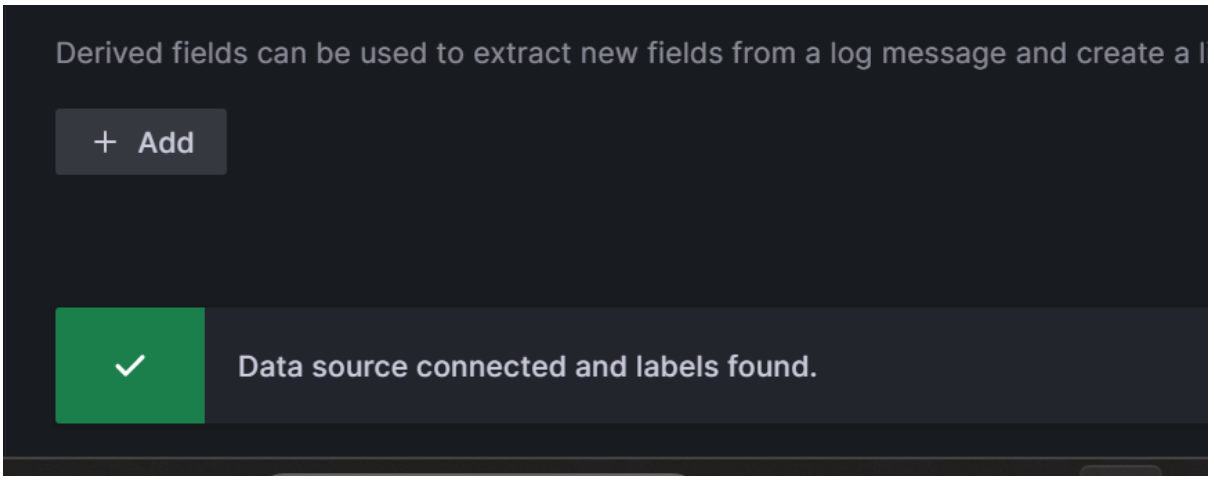
14. Here we select the url for prometheus. in this case we need the service name of prometheus,namespace and port: http://prometheus-kube-prometheus-prometheus.default.svc.cluster.local:9090 this is the url in our case. If it worked you should get a data source with a working message.



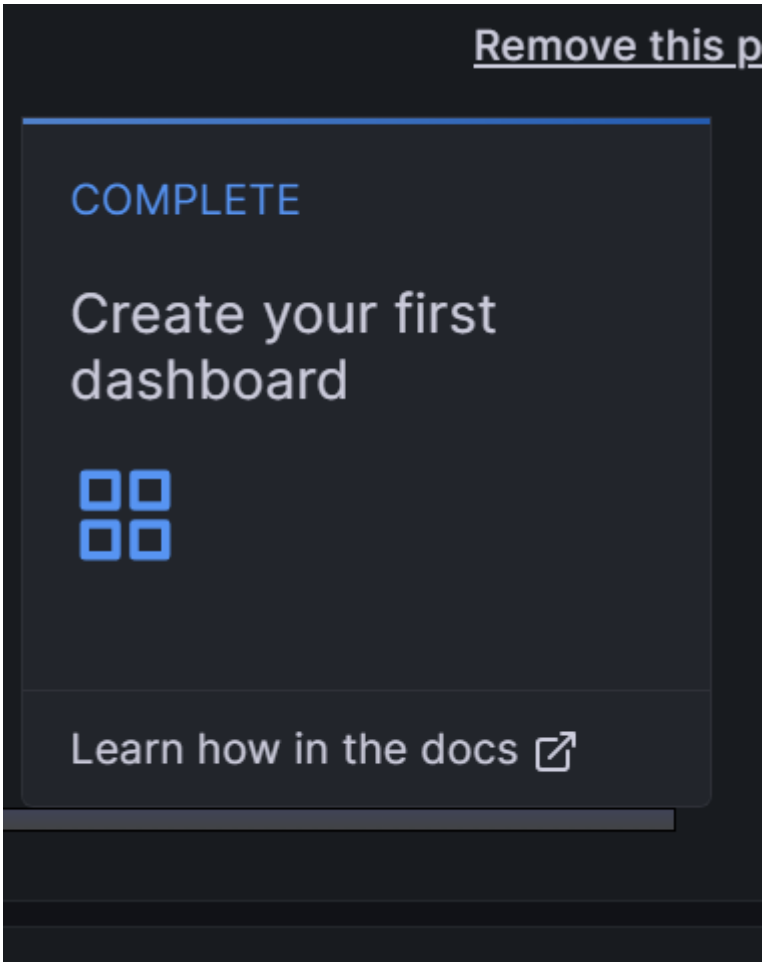
15. For the Loki datasource we almost do the same commands.



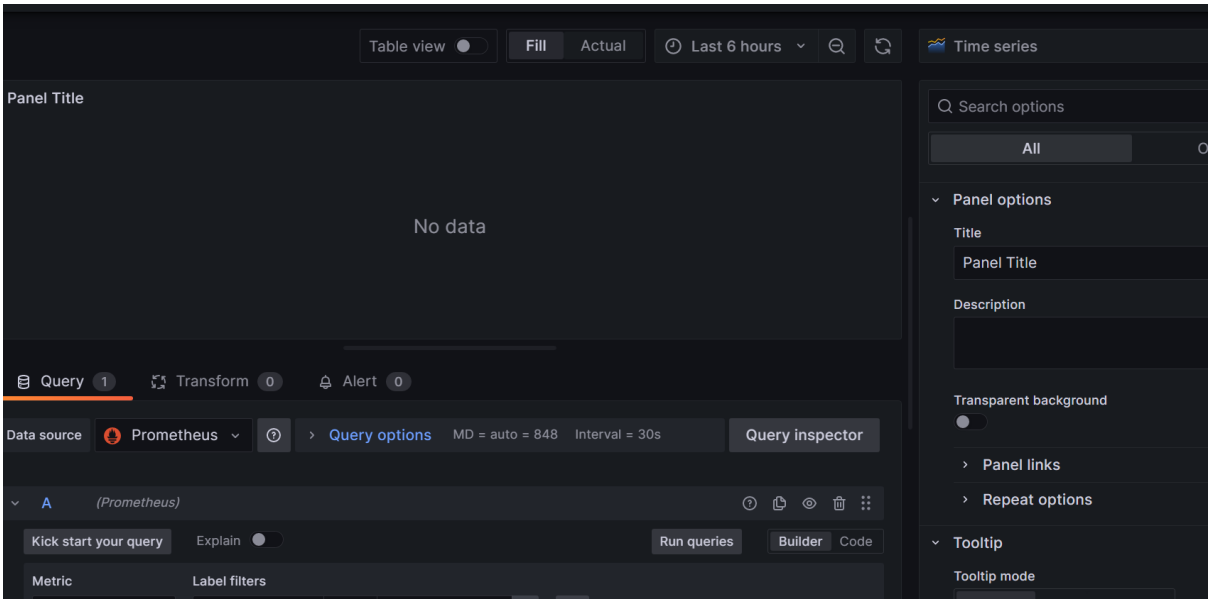
16. http://loki.default.svc.cluster.local:3100



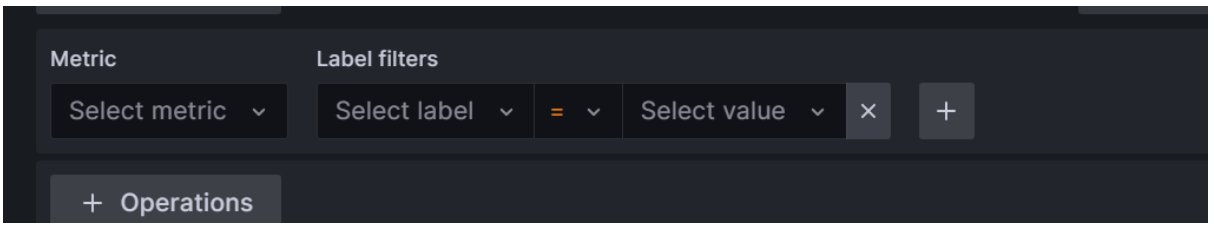
17. To create our first dashboard and to find some info about the cluster. Go to create your first dashboard



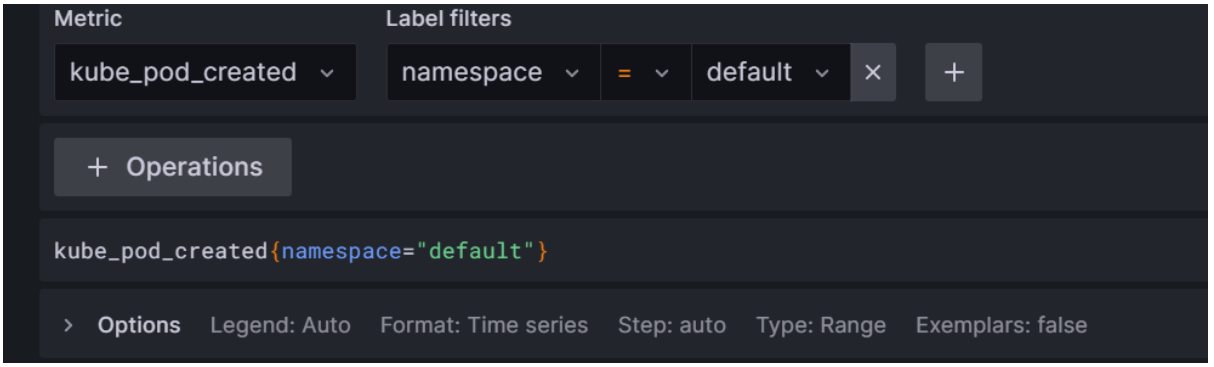
from here add visualization.

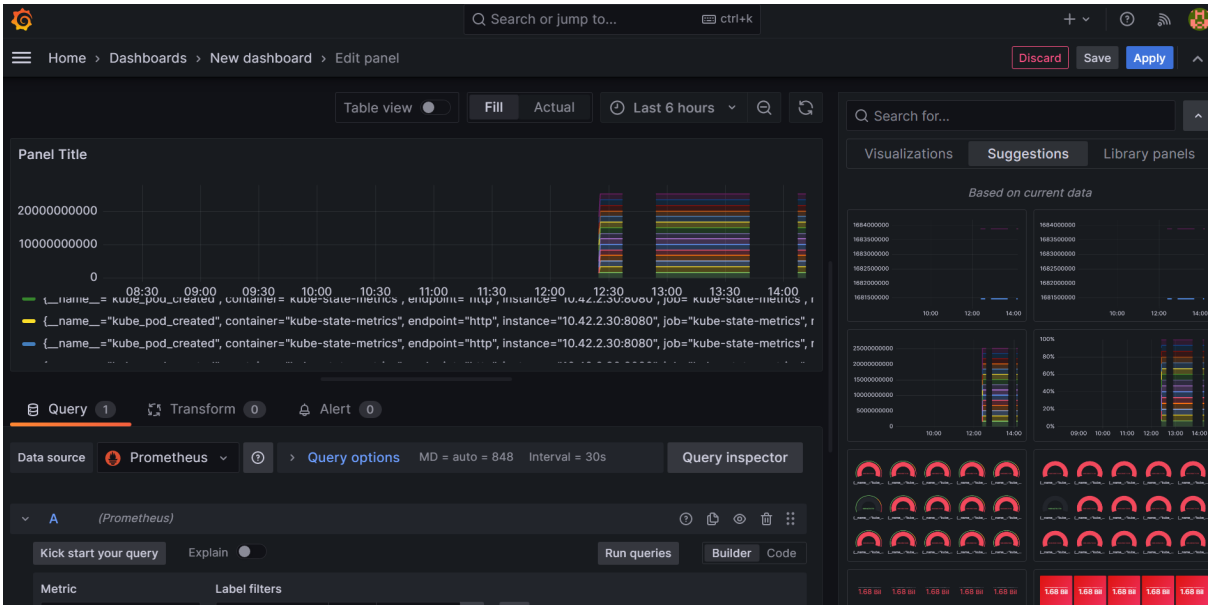


add Prometheus as a data source go to the metrics section and here you can enter in a specific query about your cluster.



or use the predefined ones.





select a dashboard and save and apply it as easy as that.

Installing MongoDB and MongoDB Express

For installation follow the steps:

1. git clone https://github.com/JOJO840/MongoDB-and-Express.git, it should be looking like this now.

```
josip@G14:~/examensProject/MongoDB-and-Express$ ls
README.md  ingress.yaml  k8s-commands.md  mongo-configmap.yaml  mongo-express.yaml  mongo-secret.yaml  mongo.yaml
josip@G14:~/examensProject/MongoDB-and-Express$
```

2. You need to apply these yaml files in order from the k8s-commands.md file that you have git cloned from the github repo

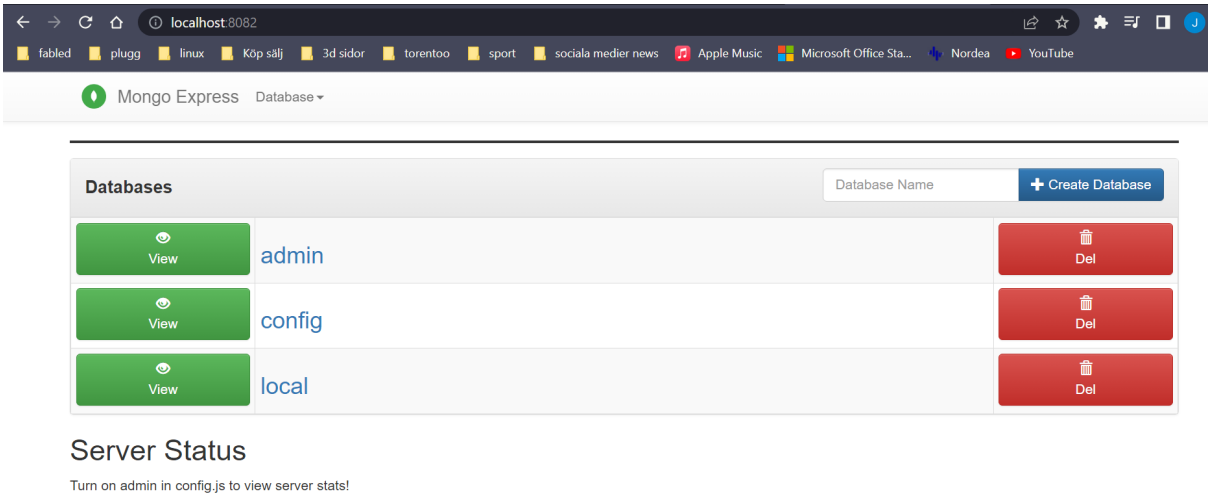
kubectl apply commands in order

```
kubectl apply -f mongo-secret.yaml
kubectl apply -f mongo.yaml
kubectl apply -f mongo-configmap.yaml
kubectl apply -f mongo-express.yaml
```

3. Your cluster should be looking like this now with the mongodb and express svcs up and running.

```
josip@G14:~/examensProject/MongoDB-and-Express$ kubectl get all | grep mongo
pod/mongodb-deployment-844789cd64-8qh6w                  1/1     Running   0             4m30s
pod/mongo-express-5bf4b56f47-njhsx                       1/1     Running   0             2m54s
service/mongodb-service                  ClusterIP      10.43.40.255    <none>                                  27017/TCP
          4m30s
service/mongo-express-service            LoadBalancer   10.43.97.227    172.20.0.3,172.20.0.4,172.20.0.5   8081:30000/TCP
          2m54s
deployment.apps/mongodb-deployment          1/1     1            1           4m30s
deployment.apps/mongo-express               1/1     1            1           2m54s
replicaset.apps/mongodb-deployment-844789cd64                1         1         1       4m30s
replicaset.apps/mongo-express-5bf4b56f47                     1         1         1       2m54s
```

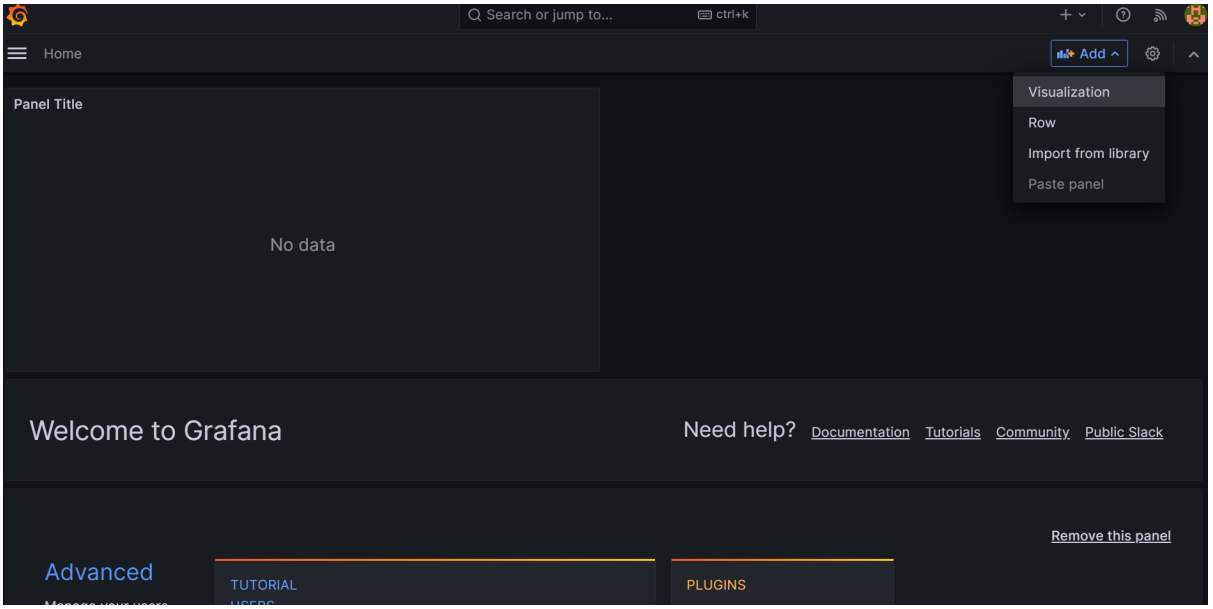
4. To connect to mongodb and express we port-forward into the svc of express, kubectl port-forward svc/mongo-express-service 8082:8081. In this case we connect to port 8082 as 8081 is taken by another app. Then type in localhost:8082 in your browser.



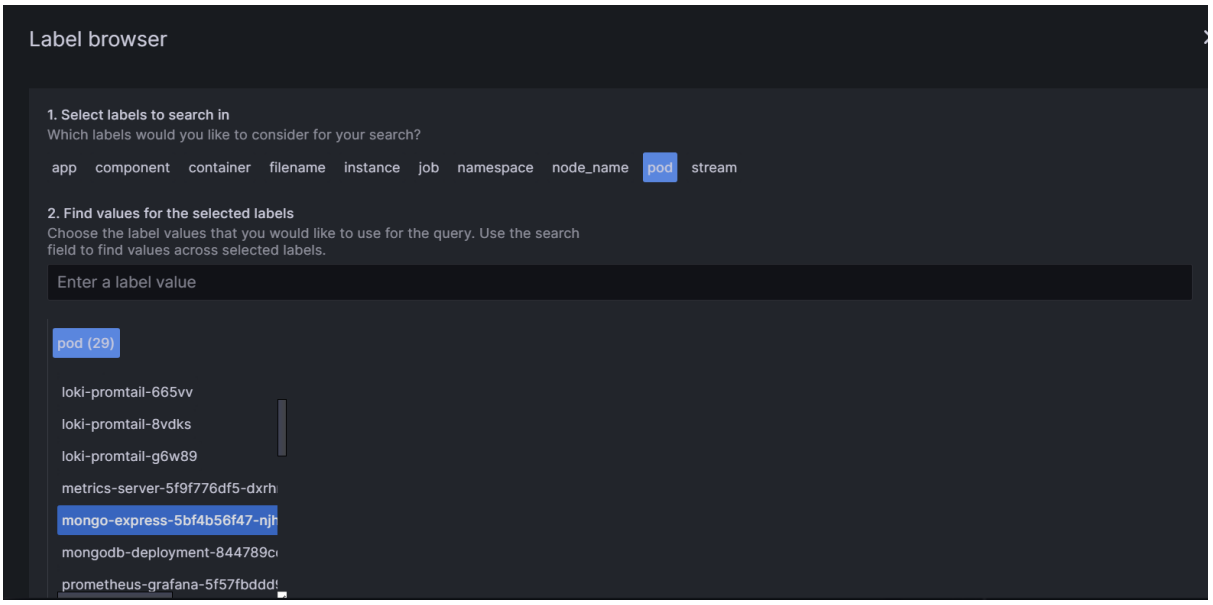
now your inside of mongodb and express used as a GUI.

Show logs from Mongodb to Grafana

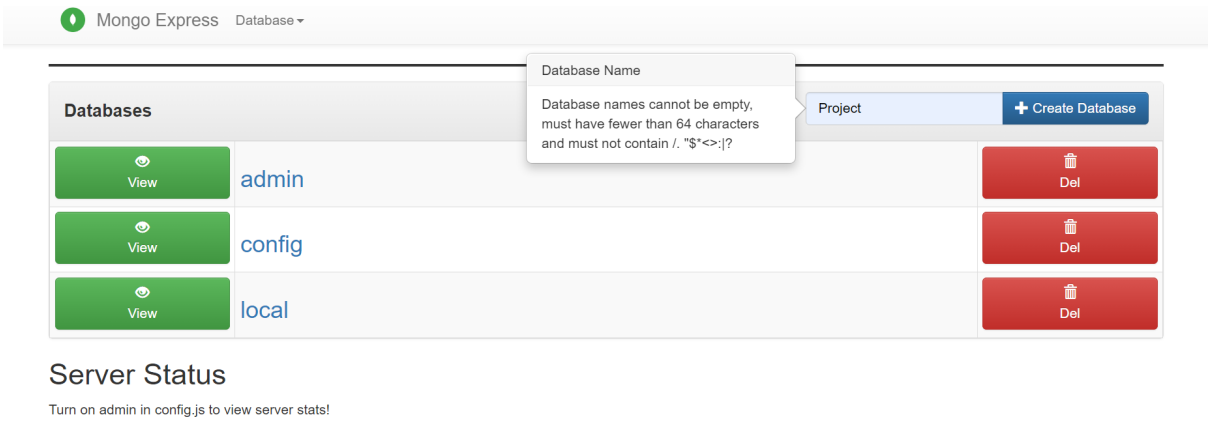
To show the logs from mongodb in a grafana dashboard we are going to use Loki queries. We start with making a dashboard and choose loki as a datasource. The first step is to go to the home page and press the little add + button, choose visualization.



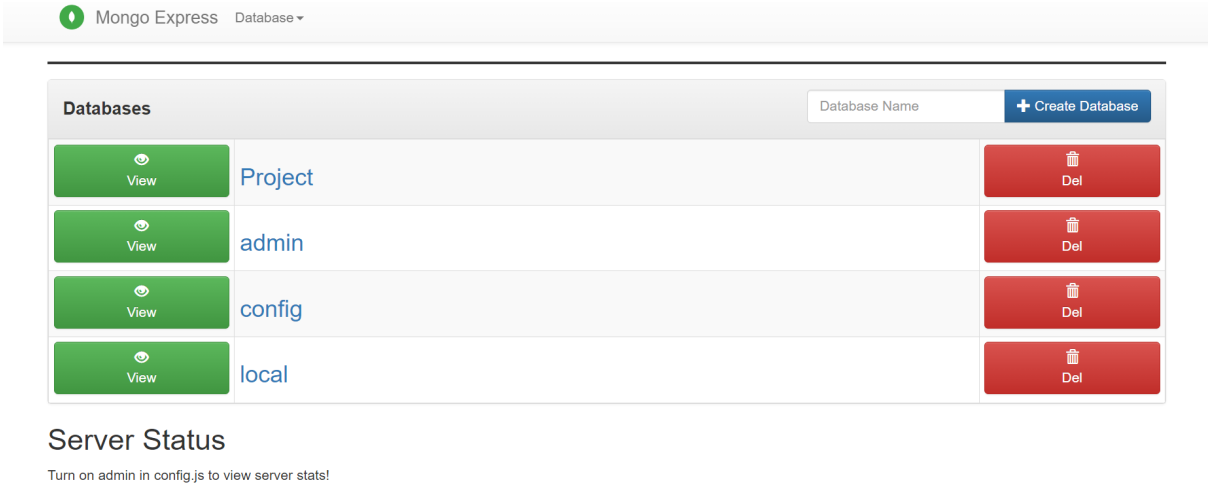
Here we can choose loki as a datasource and choose the label browser and choose pod and enter in the mongo-express pod from your cluster.



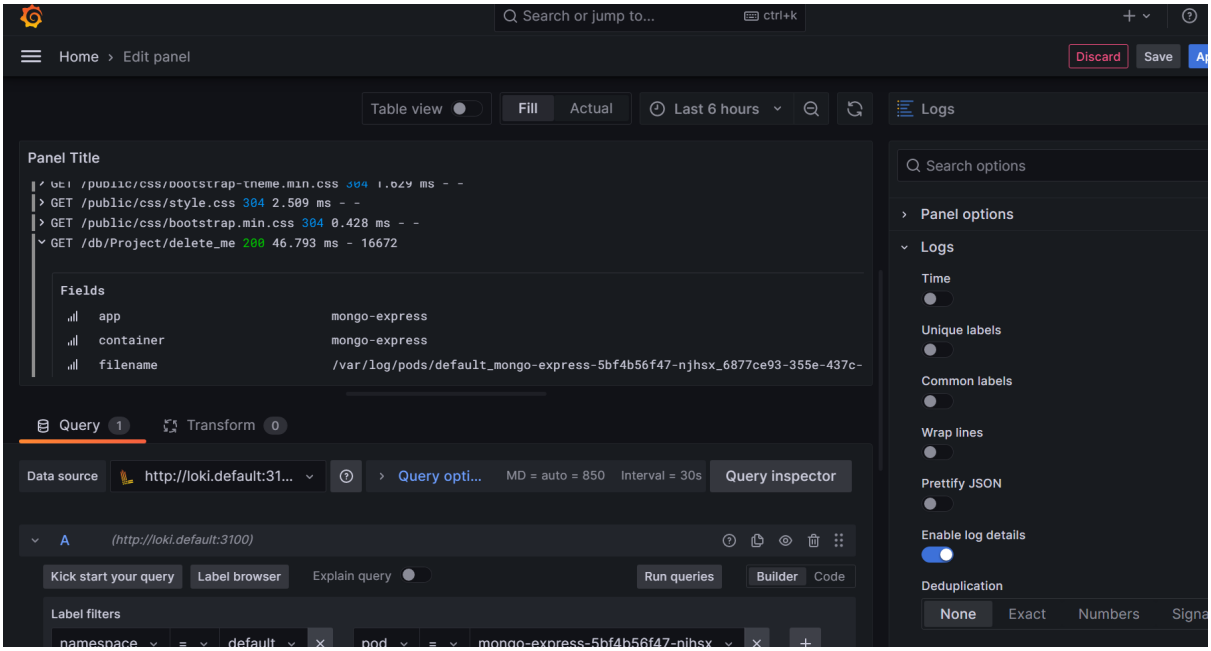
Next we are going into the mongodb-express service with port-forwarding, we are just creating a database here so we can show some logs in grafana.

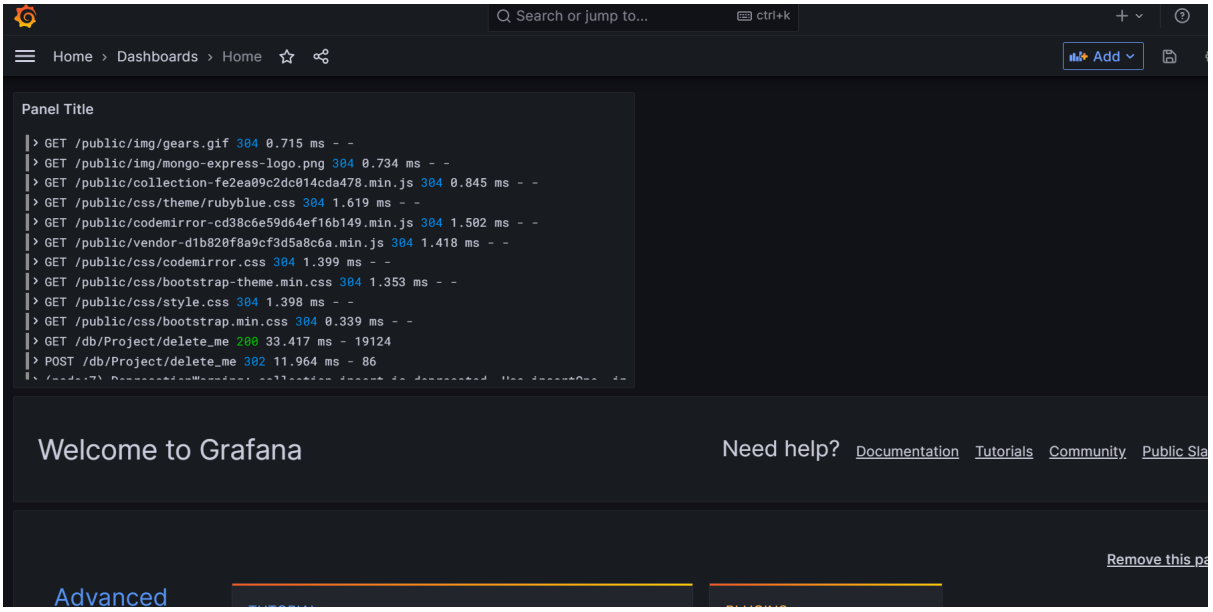


Enter in a name for our database and press the Create Database button.



Now we can see the logs for the database created in grafana. Then click on Save Dashboard. You will have a dashboard in the home section.




Search or jump to...
ctrl+k
Home › Dashboards › Home
Add
Panel Title
GET /public/img/gears.gif 304 0.715 ms - -
GET /public/img/mongo-express-logo.png 304 0.734 ms - -
GET /public/collection-fe2ea09c2dc014cda478.min.js 304 0.845 ms - -
GET /public/css/theme/rubyblue.css 304 1.619 ms - -
GET /public/codemirror-cd38c6e59d64ef16b149.min.js 304 1.502 ms - -
GET /public/vendor-d1b820f8a9cf3d5a8c6a.min.js 304 1.418 ms - -
GET /public/css/codemirror.css 304 1.399 ms - -
GET /public/css/bootstrap-theme.min.css 304 1.353 ms - -
GET /public/css/style.css 304 1.398 ms - -
GET /public/css/bootstrap.min.css 304 0.339 ms - -
GET /db/Project/delete_me 200 33.417 ms - 19124
POST /db/Project/delete_me 302 11.964 ms - 86
Welcome to Grafana
Need help?
Documentation
Tutorials
Community
Remove this pa
Advanced

6. Discussion:

The objective of this study was to utilize a bundle of programs from the K8s ecosystem and document the installation process for each component. The tutorial presented in the "Results " section successfully achieved this objective. The clear and functional installation instructions provided by the vendors, along with the readily available official documentation, facilitated straightforward procedures that can be easily replicated. This stable workflow can be credited to the strong community of developers that contribute to the K8s ecosystem regularly.

Additionally, a second tutorial was created to provide an overview of K8s architecture and its components. This tutorial is included in the "Attachment" section. Given the abundance of online resources available about K8s components, we deemed it valuable to include an overview to enhance the proficiency of future trainees.

Regarding the future trends within the industry, Tekton is an example of a tool that is likely to be replaced by other options addressing CI/CD pipelines. Emerging trends are enabling pipelines that can be easily relocated to other platforms. An example of this new approach is GitHub Actions, which, although not currently fully transferable between platforms, has gained popularity as a pipeline solution. When it comes to monitoring capabilities, Grafana remains the preferred choice, offering open-source, cost-effective, and user-friendly features. In terms of cluster setup, K3D stands out as one of the fastest options, offering multi-node capability unlike Minikube, which is limited to a single node

7. Conclusion

In conclusion, this study aimed to explore the setup and maintenance of lightweight Kubernetes (K8s) clusters with monitoring and CI/CD pipeline capabilities. The investigation focused on a specific K8s bundle comprising Tekton, Grafana, K3D, and MongoDB, which were selected based on their lightweight nature, ease of use, and compatibility with K8s.

Through interviews with Robin Morero, our LIA Supervisor at Fabled, valuable insights were gained regarding the software selection criteria and the rationale behind choosing these specific tools. The interviews revealed that Fabled prioritizes ease of maintenance, seamless integration, and the ability to upgrade and replace components. The lightweight nature of the chosen software was seen as advantageous, ensuring efficient resource utilization and cost-effectiveness.

The tutorial provided in this study offers step-by-step instructions for installing the K8s bundle and serves as a valuable resource for beginners seeking to leverage K8s effectively. It is important to note that the installation of prerequisites such as WSL 2, Docker Desktop, and access to a Linux CLI is essential but beyond the immediate scope of this research.

Furthermore, the study acknowledges certain limitations, including the level of coverage of K8s architecture and the focus on providing a fundamental understanding rather than going into specific details. The primary goal of this research is to equip individuals with the necessary knowledge and skills to utilize K8s effectively, particularly for lightweight and cost-effective cluster management.

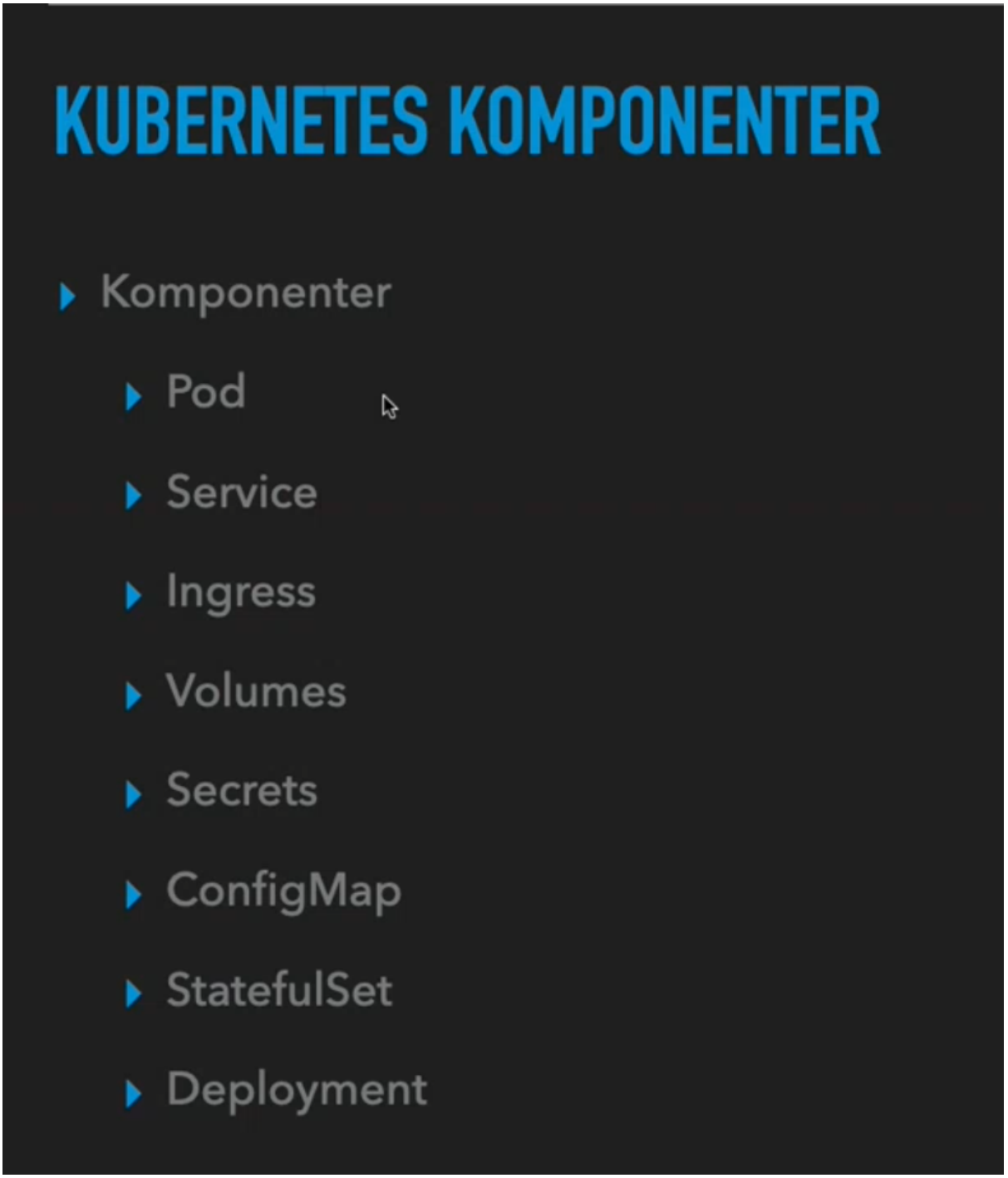
The insights gained from the interviews highlight the importance of managing complexity within K8s clusters. While one approach involves setting up a separate platform team and utilizing vendors for comprehensive management, the preferred approach at Fabled is to minimize complexity by keeping the cluster lightweight and using open source components. This approach reduces maintenance work and provides developers with a seamless experience, aligning with Fabled's emphasis on cost-effectiveness and ease of use.

In summary, this study has provided insights into the setup and maintenance of lightweight K8s clusters with monitoring and CI/CD pipeline capabilities. By exploring the specific K8s bundle, interviews with industry professionals, and the tutorial for installation, this research equips individuals with practical knowledge and resources for harnessing the power of K8s effectively in their respective projects.

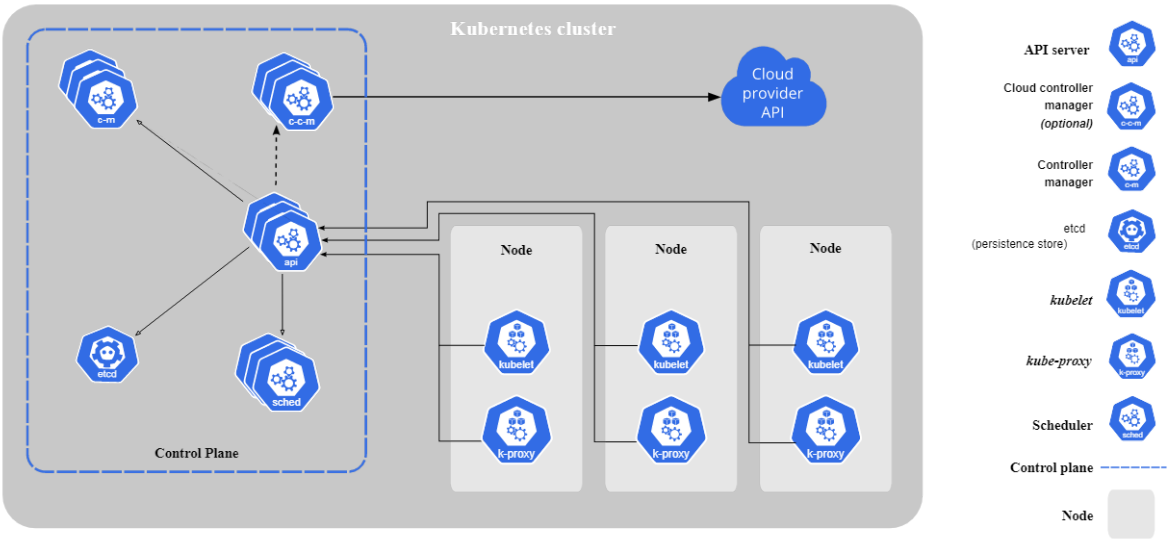
8. Attachments:

Kubernetes: An Overview

k8s Components



A picture of an Cluster

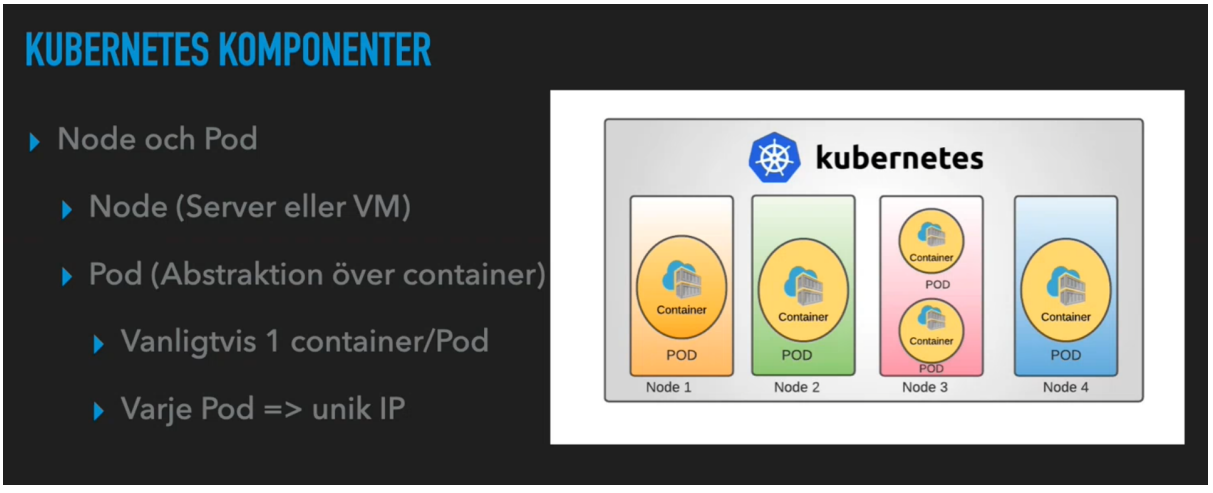


When you deploy Kubernetes, you get a cluster.

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

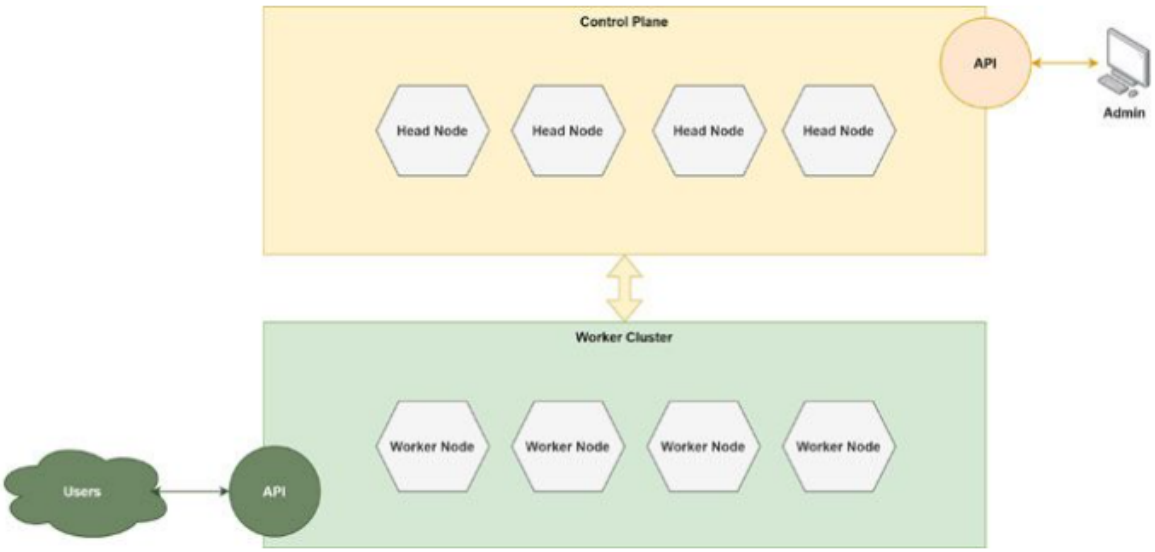
Nodes and Pods:



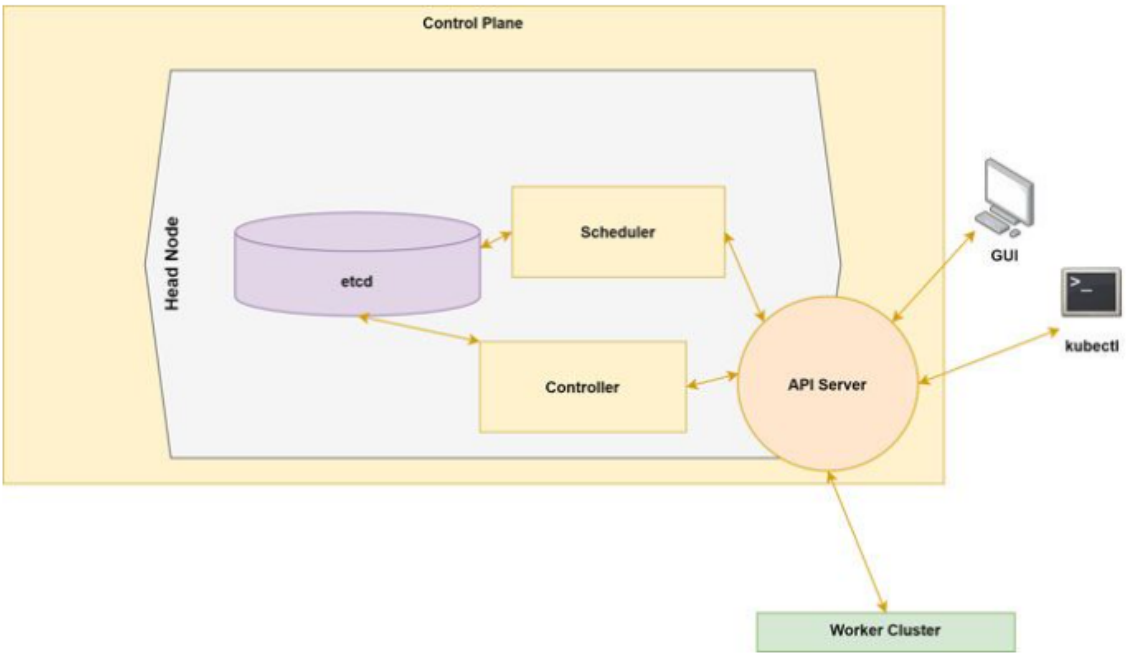
Master Nodes (head nodes):

A Node is a **VM** or a "**computer**".

Inside of the worker nodes here is where the containerized apps live, and in the master nodes are the tools to handle that cluster. See the figures.



A Master Node in detail:



etcd (et:si:di)

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for that data.

You can find in-depth information about etcd in the official documentation.

Scheduler

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

Api server

The core of Kubernetes' control plane is the API server. The API server exposes an HTTP API that lets end users, different parts of your cluster, and external components communicate with one another.

The Kubernetes API lets you query and manipulate the state of API objects in Kubernetes (for example: Pods, Namespaces, ConfigMaps, and Events).

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

Apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

Controller-Manager

Control plane component that runs controller processes.

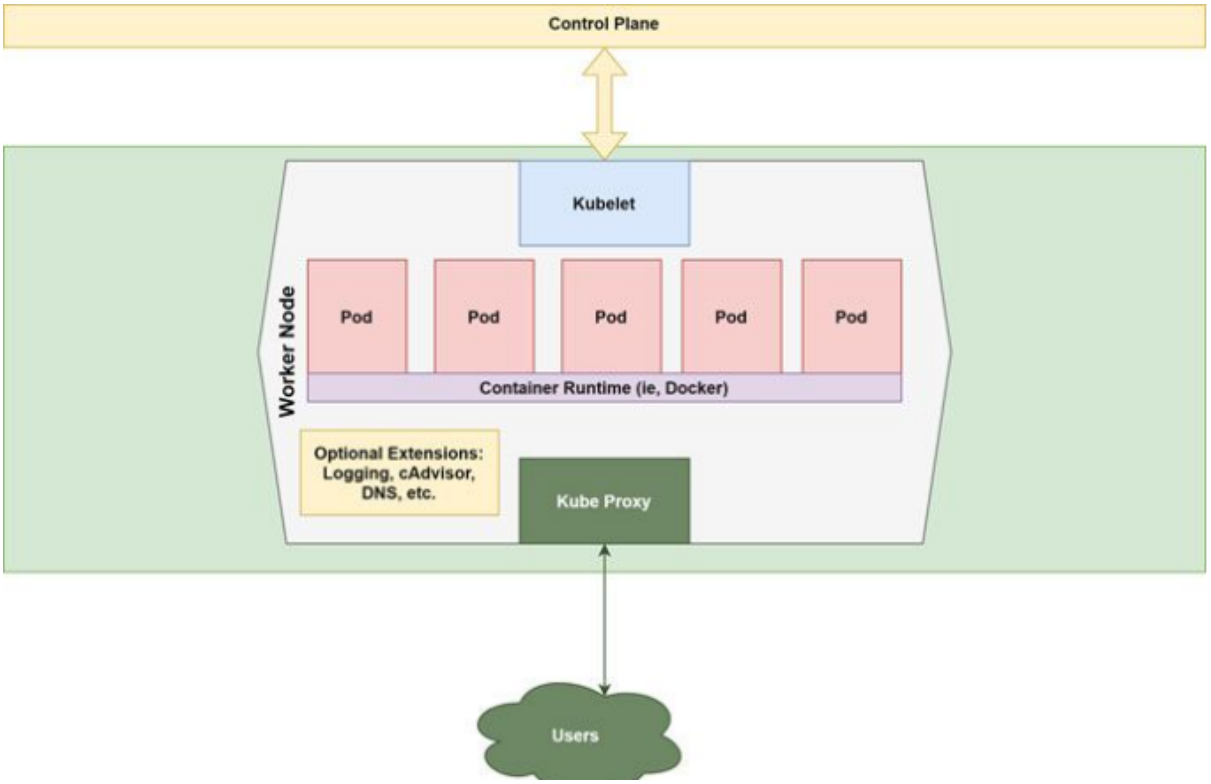
The controller is the component that ensures that the cluster remains in the configured state and returns it to that state if it begins to drift. The controller acts as a kind of thermostat that sets a desired state and then strives to maintain it.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

- Node controller: Responsible for noticing and responding when nodes go down.
- Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
- EndpointSlice controller: Populates EndpointSlice objects (to provide a link between Services and Pods).
- ServiceAccount controller: Create default ServiceAccounts for new namespaces.

Worker Nodes:
The Components inside of a worker node consist of a “Kubelet”, “Kubeproxy”, “Container Runtime” (like Docker for instance) and Pods. Also There are optional Extensions.
See the figure.



Kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

Kube-proxy

Kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

Container runtime

The container runtime is the software that is responsible for running containers.

Kubernetes supports container runtimes such as Containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

Pods

Each pod has a unique IP address, also a pod is an abstraction over a container. Pods are discrete units of work on the node. They are at the replication level. Pods are abstractions that wrap one or more containerized applications.
Pods provide the ability to logically group and isolate containers that run together, while enabling communication between pods on the same machine. The relationship between containers and pods is controlled by Kubernetes deployment descriptors.
Read more on pods on kubernetes doc:
https://kubernetes.io/docs/concepts/workloads/pods/

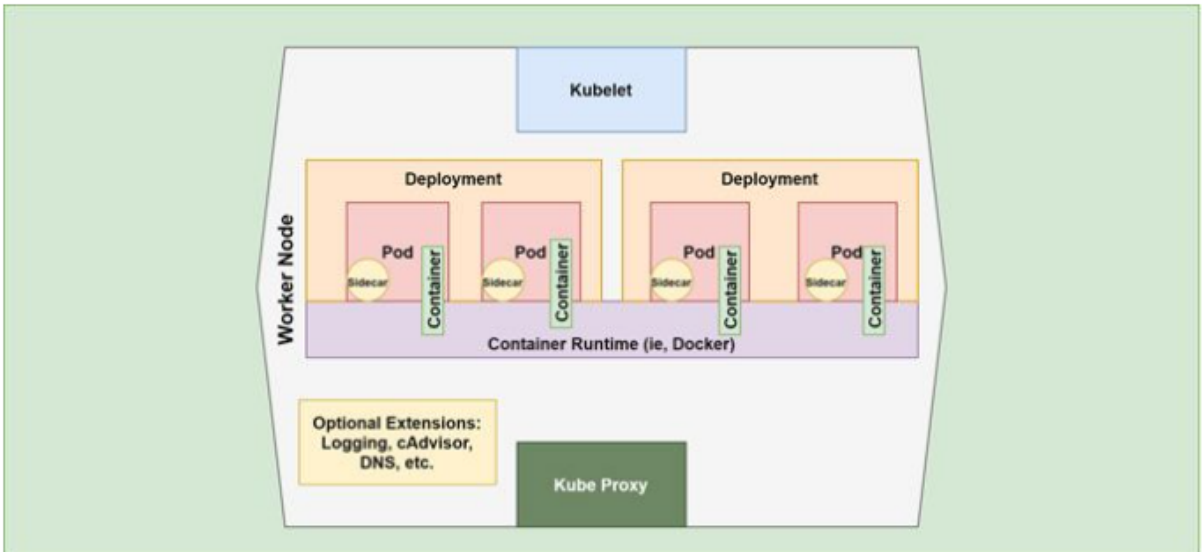
Deployments and Replica Sets

Typically, pods are configured and deployed as parts of ReplicaSets. A ReplicaSet defines the desired execution properties of the pod and makes Kubernetes work to maintain that state.

ReplicaSets are usually defined by a Deployment, which specifies both the parameters of the current ReplicaSet and the strategy to be used (that is, if pods are to be updated or newly created) when managing the cluster.

Sidecar (Add-ons)
At the pod level, extra functions can be added through add-ons called sidecars. The sidecars handle tasks such as logging at the pod level and collecting statistics.

A Pod in Detail:

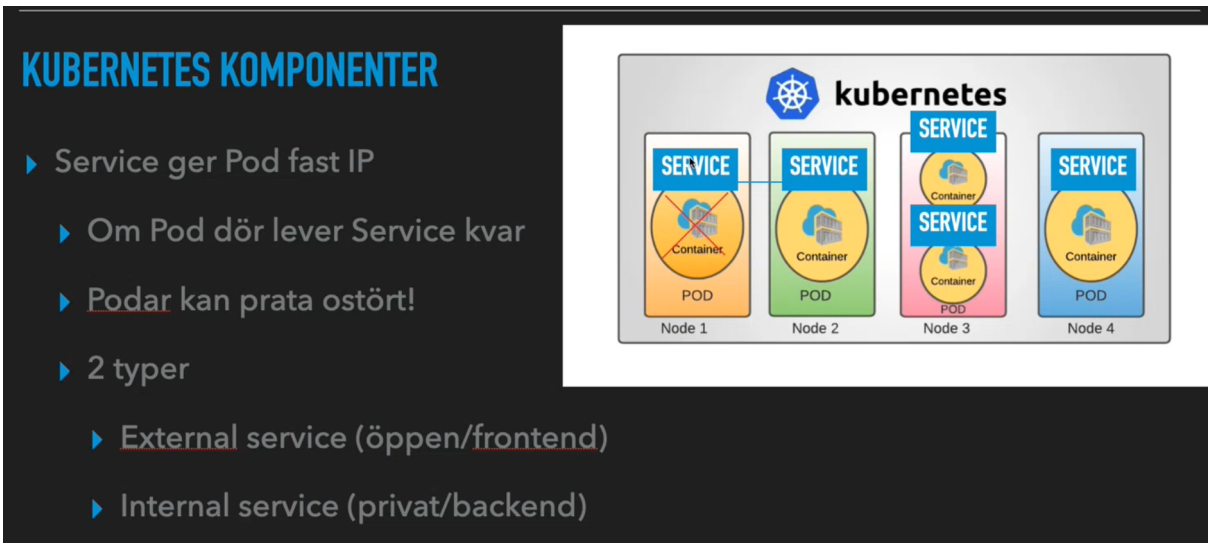


Service

In Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster.

A key aim of Services in Kubernetes is that you don't need to modify your existing application to use an unfamiliar service discovery mechanism. You can run code in Pods, whether this is a code designed for a cloud-native world, or an older app you've containerized. You use a Service to make that set of Pods available on the network so that clients can interact with it.

If you use a Deployment to run your app, that Deployment can create and destroy Pods dynamically. From one moment to the next, you don't know how many of those Pods are working and healthy; you might not even know what those healthy Pods are named. Kubernetes Pods are created and destroyed to match the desired state of your cluster. Pods are ephemeral resources (you should not expect that an individual Pod is reliable and durable).



Each Pod gets its own IP address (Kubernetes expects network plugins to ensure this). For a given Deployment in your cluster, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later.

This leads to a problem: if some set of Pods (call them "backends") provides functionality to other Pods (call them "frontends") inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?

https://kubernetes.io/docs/concepts/services-networking/service/

Defining a Service

A Service in Kubernetes is an object (the same way that a Pod or a ConfigMap is an object).

You can create, view or modify Service definitions using the Kubernetes API. Usually you use a tool such as `kubectl` to make those API calls for you.

For example, suppose you have a set of Pods that each listen on TCP port 9376 and are labeled as `app.kubernetes.io/name=MyApp`. You can define a Service to publish that TCP listener:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Applying this manifest creates a new Service named "my-service", which targets TCP port 9376 on any Pod with the `app.kubernetes.io/name: MyApp` label.

Kubernetes assigns this Service an IP address (the *cluster IP*), that is used by the virtual IP address mechanism. For more details on that mechanism, read Virtual IPs and Service Proxies.

Port definitions

Port definitions in Pods have names, and you can reference these names in the `targetPort` attribute of a Service. For example, we can bind the `targetPort` of the Service to the Pod port in the following way:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy
spec:
  containers:
  - name: nginx
    image: nginx:stable
    ports:
      - containerPort: 80
        name: http-web-svc

---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app.kubernetes.io/name: proxy
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: http-web-svc
```

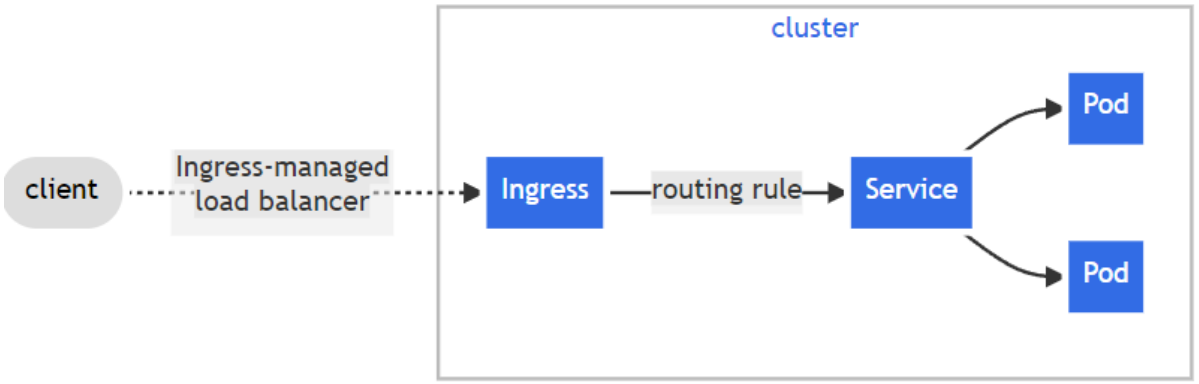
Ingress

What is Ingress?

An API object that manages external access to the services in a cluster, typically HTTP.

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Here is a simple example where an Ingress sends all its traffic to one Service:



An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name-based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic.

An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.

Read More on Ingress Here:
https://kubernetes.io/docs/concepts/services-networking/ingress/

Volumes

In Kubernetes, a volume can be thought of as a directory which is accessible to the containers in a pod. We have different types of volumes in Kubernetes and the type defines how the volume is created and its content.

The concept of volume was present with Docker, however the only issue was that the volume was very much limited to a particular pod. As soon as the life of a pod ended, the volume was also lost.

On the other hand, the volumes that are created through Kubernetes is not limited to any container. It supports any or all the containers deployed inside the pod of Kubernetes. A key advantage of Kubernetes volume is, it supports different kind of storage wherein the pod can use multiple of them at the same time.

https://www.tutorialspoint.com/kubernetes/kubernetes_volumes.htm

On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem is the loss of files when a container crashes. The kubelet restarts the container but with a clean state. A second problem occurs when sharing files between containers running together in a `Pod`. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested.

https://kubernetes.io/docs/concepts/storage/volumes/

Configmap

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

Use a ConfigMap for setting configuration data separately from application code.

For example, imagine that you are developing an application that you can run on your own computer (for development) and in the cloud (to handle real traffic). You write the code to look in an environment variable named `DATABASE_HOST`. Locally, you set that variable to `localhost`. In the cloud, you set it to refer to a Kubernetes Service that exposes the database component to your cluster. This lets you fetch a container image running in the cloud and debug the exact same code locally if needed.

A ConfigMap is not designed to hold large chunks of data. The data stored in a ConfigMap cannot exceed 1 MiB. If you need to store settings that are larger than this limit, you may want to consider mounting a volume or use a separate database or file service.

ConfigMap object

A ConfigMap is an API object that lets you store configuration for other objects to use. Unlike most Kubernetes objects that have a `spec`, a ConfigMap has `data` and `binaryData` fields. These fields accept key-value pairs as their values. Both the `data` field and the `binaryData` are optional. The `data` field is designed to contain UTF-8 strings while the `binaryData` field is designed to contain binary data as base64-encoded strings.

The name of a ConfigMap must be a valid DNS subdomain name.

Each key under the `data` or the `binaryData` field must consist of alphanumeric characters, `-`, `_` or `.`. The keys stored in `data` must not overlap with the keys in the `binaryData` field.

Starting from v1.19, you can add an `immutable` field to a ConfigMap definition to create an immutable ConfigMap.

ConfigMaps and Pods

You can write a Pod `spec` that refers to a ConfigMap and configures the container(s) in that Pod based on the data in the ConfigMap. The Pod and the ConfigMap must be in the same namespace.

Note: The `spec` of a static Pod cannot refer to a ConfigMap or any other API objects.

Here's an example ConfigMap that has some keys with single values, and other keys where the value looks like a fragment of a configuration format.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

There are four different ways that you can use a ConfigMap to configure a container inside a Pod:

1. Inside a container command and args
2. Environment variables for a container
3. Add a file in read-only volume, for the application to read
4. Write code to run inside the Pod that uses the Kubernetes API to read a ConfigMap

These different methods lend themselves to different ways of modeling the data being consumed. For the first three methods, the kubelet uses the data from the ConfigMap when it launches container(s) for a Pod.

The fourth method means you have to write code to read the ConfigMap and its data. However, because you're using the Kubernetes API directly, your application can subscribe to get updates whenever the ConfigMap changes, and react when that happens. By accessing the Kubernetes API directly, this technique also lets you access a ConfigMap in a different namespace.

Read more on Configmap here:

https://kubernetes.io/docs/concepts/configuration/configmap/

9. Reference List:

Interview:

Robin Morero. 2023. Fabled

Internet:

Casey, K. (2022) The Enterprisers Project “Kubernetes by the numbers, in 2022: 11 stats to see”
https://enterprisersproject.com/article/2022/10/kubernetes-statistics-2022
(Accessed: 8 May 2023)

CNCF. (2022) “CNCF Annual Survey 2022”.
https://www.cncf.io/reports/cncf-annual-survey-2022/
(Accessed: 9 May 2023)

CNCF. (2022) “CNCF Annual Survey 2021”.
https://www.cncf.io/wp-content/uploads/2022/02/CNCF-Annual-Survey-2021.pdf
(Accessed: 9 May 2023)

Grafana. 2022. “What is Grafana”.
https://www.redhat.com/en/topics/data-services/what-is-grafana
(Accessed: 2 May 2023)

K3D. 2022 “What is k3d”. https://k3d.io/v5.4.9/
(Accessed: 1 May2023)

TechWorld with Nana. 2022, “Microservices explained - the What, Why and How?”.
https://www.youtube.com/watch?v=rv4LlmLmVWk&ab_channel=TechWorldwithNana
(Accessed: 1 May2023)

TechWorld with Nana 2020 “Complete Application Deployment using Kubernetes Components”.
https://www.youtube.com/watch?v=EQNO_kM96Mo&ab_channel=TechWorldwithNana
(Accessed: 1 May 2023)

Tekton. 2023. “Overview”.
https://tekton.dev/docs/concepts/overview/ (Accessed: 1 May 2023)