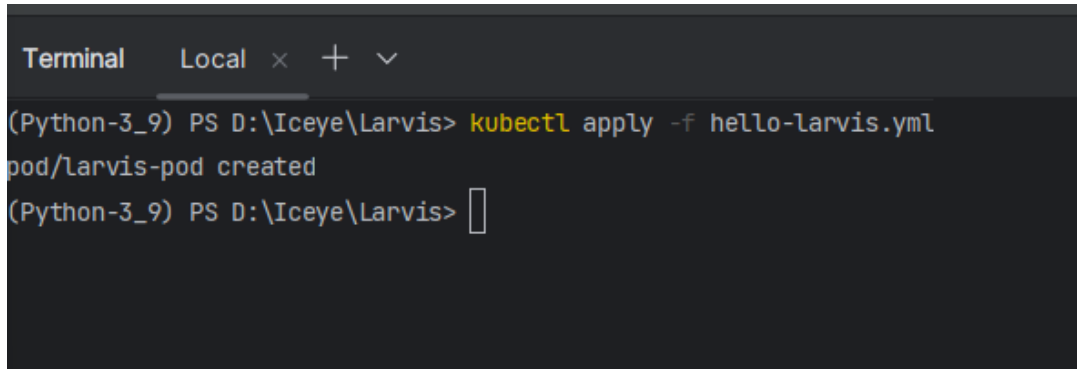



Larvis in Kubernetes.

After verifying that the Larvis bootstrap was correctly running in a Docker container, the next step was creating the Kubernetes cluster using a YAML file *hello-larvis.yml*.



```
Terminal Local x + v
(Python-3_9) PS D:\Iceye\Larvis> kubectl apply -f hello-larvis.yml
pod/larvis-pod created
(Python-3_9) PS D:\Iceye\Larvis> 
```

The pod creation was successful, and the image was pulled correctly from the repository at <https://hub.docker.com/> with the image name yonshar/Jarvis. However, the container failed to create, and hence the process restarted. In the pod Events, I get < "Back-off restarting failed container larvis in pod larvis_default(92a379ae-9b12-4e97-aa62-30819f433b71)">.

	larvis.1796b0dd3e5168dd	BackOff	Back-off restarting failed container larvis in pod larvis_default(92a379ae-9b12-4e97-aa62-30819f433b71)	kubelet minikube
	larvis.1796b0dc33cbc902	Pulling	Pulling image "yonshar/larvis:latest"	kubelet minikube
	larvis.1796b0dc83097576	Started	Started container larvis	kubelet minikube

I spent considerable time debugging the issue and investigating why the pod had reached this state. However, I faced a problem that caused the container to roll back and restart. Unfortunately, I am running out of time to complete my assessment, and I have made the difficult decision to stop the implementation of Kubernetes larvis.

I found several reasons for this error to happen when the Pod keeps crashing:

- An error happens when deploying the software.
- General system misconfiguration
- Incorrect assigned managed identity on your Pod
- Incorrect configuration of container or Pod parameters

- Lack of memory resources
- Locked database since other Pods are currently using it.
- References to binaries or scripts that can't be found in the container.
- Setup issues with the init-container
- Two or more containers use the same port, which doesn't work if they're from the same Pod.

I took a methodic way to debug this issue:

- I deployed an image with only an Ubuntu operating system, and then I tried the same with a smaller Ubuntu image, such as Alpine flavour, but I still encountered the same issue.
- Although I made some changes to the system configuration while trying to troubleshoot the issue, I deleted the entire Docker environment and recreated all the steps, but the problem persisted.
- I made sure that the Pod was created in the default namespace and was the only one running in the cluster.
- While I am not familiar with the settings for the init-container, I think it is less likely to be the cause of the error.

Based on my observation, two possible reasons for the issue are either the Minikube daemon is running out of allowed memory, or some other service is using port 8080. I tested creating a random pod on port 8080, and it was successfully created. Therefore, I concluded that it is a memory-like issue in my system running Windows 11, which is quite frustrating.

Working Steps to create a minikube cluster

Here are the steps to install and start a Minikube Kubernetes cluster:

1. Install minikube using Chocolatey (<https://chocolatey.org/install>)

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> choco install minikube
Chocolatey v2.2.2
Installing the following packages:
minikube
By installing, you accept licenses for the packages.
Progress: Downloading Minikube 1.32.0... 100%

Minikube v1.32.0 [Approved]
Minikube package files install completed. Performing other installation steps.
  ShimGen has successfully created a shim for minikube.exe
  The install of Minikube was successful.
  Software installed to 'C:\ProgramData\chocolatey\lib\Minikube'

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\WINDOWS\system32>
```

2. Start minikube with default configuration using the docker virtual machine.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> minikube start
* minikube v1.32.0 on Microsoft Windows 11 Pro 10.0.22631.2506 Build 22631.2506
* Using the docker driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Updating the running docker "minikube" container ...
* Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
! Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 2.9962491s
* Restarting the docker service may improve performance.
  - Using image docker.io/kubernetes/metrics-scraper:v1.0.8
  - Using image docker.io/kubernetes/dashboard:v2.7.0
  - Using image registry.k8s.io/metrics-server/metrics-server:v0.6.4
* Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server






* Enabled addons: storage-provisioner, default-storageclass, dashboard, metrics-server
* Done! kubectrl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\WINDOWS\system32> minikube addons enable metrics-server
* metrics-server is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/metrics-server/metrics-server:v0.6.4
* The 'metrics-server' addon is enabled
PS C:\WINDOWS\system32>
```

3. Launch the minikube dashboard

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> minikube dashboard
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
* Opening http://127.0.0.1:62209/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboar
```

Namespaces		
Name	Labels	Phase
 kubernetes-dashboard	<div>addonmanager.kubernetes.io/mode: Reconcile</div> <div>kubernetes.io/metadata.name: kubernetes-dashb oard</div> <div>kubernetes.io/minikube-addons: dashboard</div>	Active
 default	<div>kubernetes.io/metadata.name: default</div>	Active
 kube-node-lease	<div>kubernetes.io/metadata.name: kube-node-lease</div>	Active
 kube-public	<div>kubernetes.io/metadata.name: kube-public</div>	Active
 kube-system	<div>kubernetes.io/metadata.name: kube-system</div>	Active

```
PS C:\WINDOWS\system32> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

4. Kubectl is config to run with minikube

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://kubernetes.docker.internal:6443
    name: docker-desktop
- cluster:
    certificate-authority: C:\Users\yonat\.minikube\ca.crt
    extensions:
    - extension:
        last-update: Sat, 11 Nov 2023 22:41:12 GMT
        provider: minikube.sigs.k8s.io
        version: v1.32.0
      name: cluster_info
    server: https://127.0.0.1:57234
    name: minikube
contexts:
- context:
    cluster: docker-desktop
    user: docker-desktop
    name: docker-desktop
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Sat, 11 Nov 2023 22:41:12 GMT
        provider: minikube.sigs.k8s.io
        version: v1.32.0
      name: context_info
    namespace: default
    user: minikube
    name: minikube
current-context: minikube
```

Steps to Create a Pod

It can be two ways:

First running Kubectl run: `kubectl run larvis-demo --image yonshar/larvis:latest`

```
Terminal - Larvis
Terminal Local x
(Python-3_9) PS D:\Iceye\Larvis> kubectl run larvis-demo --image yonshar/larvis:latest
pod/larvis-demo created
(Python-3_9) PS D:\Iceye\Larvis> 
```

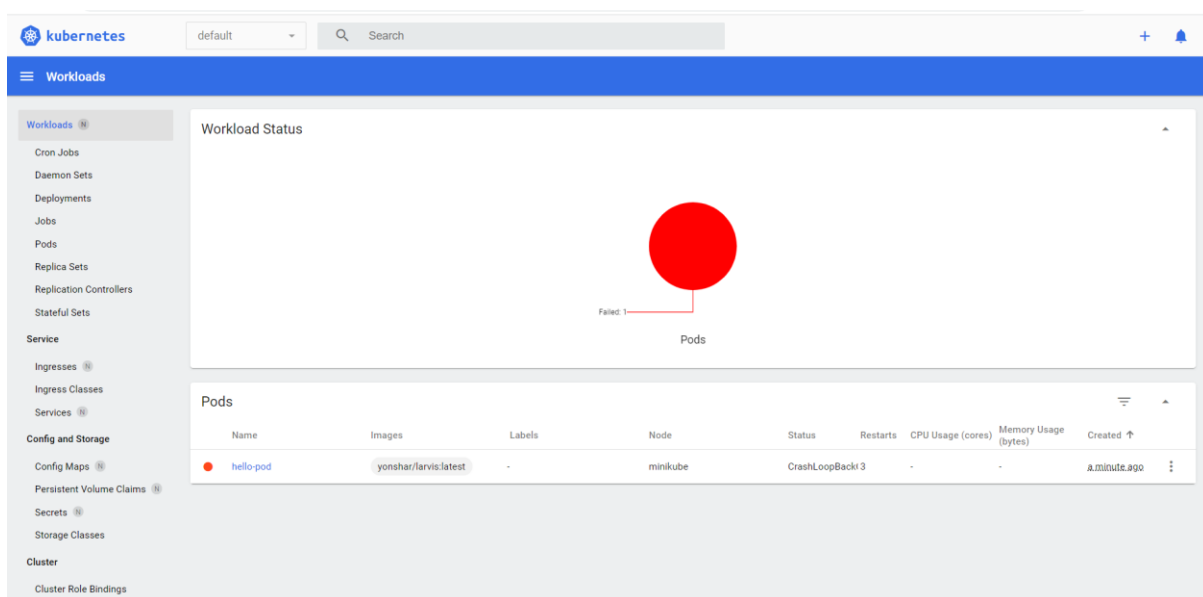
Second, run a yaml file using `kubectl apply -f <yaml file name>` included in the assessment repository.

```
Terminal - Larvis

Terminal Local x + v

(Python-3_9) PS D:\Iceye\Larvis> kubectl apply -f hello-larvis.yml
pod/hello-pod created
(Python-3_9) PS D:\Iceye\Larvis> 
```

In the minikube dashboard, display the failed Pod



Next Steps

I am devastated that I cannot demonstrate my skills thoroughly. Instead, I am trying to show my integrity and way of thinking honestly and truthfully in front of you, which could be my future team.

The system should be implemented with a replica set, service, and ingress to make Larvis available. This would allow a Python API to query the larvis binary and change the port where it listens for requests.