Lab Orchestrator

Marco Schlicht Mohamed El Jemai

June 4, 2021

Abstract

Explanation of tools that we are using.

Contents

1 Tools

The Lab Orchestrator application uses different tools that may be explained before the installation of the application. This chapter will give you an introduction into the tools that are used and required in this project, as well as an explanation about Kubernetes that is needed to understand how the Lab Orchestrator application is working on the inside.

1.1 Make

Make is used to resolve dependencies during a build process. In this project make is used to have some shortcuts for complex build commands. For example there is a make command to generate the documentation: `make docs`.

1.2 Generating The Documentation

The documentation is written in markdown and converted to a pdf using pandoc. To generate the documentation pandoc and latex are used. Install `pandoc`, `pandoc-citeproc` and a latex environment. [1]

For the replacement of variables there is a lua script installed, so you need to install lua too. [2]

There is a make command to generate the docs: `make docs`.

1.2.1 Commands:

- `$ sudo apt install pandoc pandoc-citeproc make`
- `$ make docs`

1.3 Terminal Tools

1.3.1 nohup

If a terminal is closed (for example if you logout), a HUP signal is send to all programs that are running in the terminal. [3] `nohup` is a command that executes a program, with intercepting the HUP signal. That results into the program doesn't exit when you logout. The output of the program is redirected into the file `nohup.out` `nohup` can be used with `&` to start a program in background that continues to run after logout. [4]

1.4 Kubernetes

Kubernetes is an open source container orchestration platform. With Kubernetes it's possible to automate deployments and easily scale containers. It has many features that make it useful for the project. Some of them are explained here. [5]

1.4.1 Control Plane

The control plane controls the Kubernetes cluster. It also has an API that can be used with kubectl or REST calls to deploy stuff. [6]

1.4.2 Custom Resource

In Kubernetes it's possible to extend the Kubernetes API with so called custom resources (CR). A custom resource definition (CRD) defines the CR. [7]

1.4.3 Kubernetes Objects

Kubernetes Objects have specs and a status. The `spec` is the desired state the object should have. `status` is the current state of the object. You have to set the `spec` when you create an object.

Kubernetes objects are often described in yaml files. The required fields for Kubernetes objects are: [8] - `apiVersion`: Which version of the Kubernetes API you are using to create this object - `kind`: What kind of object you want to create - `metadata`: Helps to uniquely identify the object - `spec`: The desired state of the object

1.4.4 Pods

A pod is a group of one or more containers that are deployed to a single node. The containers in a pod share an ip address and a hostname.

1.4.5 Deployment

Deployments define the applications life cycle, for example which images to use, the number of pods and how to update them. [9]

1.4.6 Services

Services allows that service requests are automatically redirected to the correct pod. Services gets their own IP addresses that is used by the service proxy.

Services also allow to add multiple ports to one service. When using multiple ports, you must give all of them a name. For example you can add a port for http and another port for https.

Example of a Service:

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

This service has the name `my-service` and listens on the port 80. It forwards the requests to the pods with the selector `app=MyApp` on the port 9376.

There is also the ability to publish services. To make use of this, the `ServiceType` must be changed. The default `ServiceType` is `ClusterIP`, which exposes the service on a cluster-internal IP, that makes this service only reachable from within the cluster. One other service type is `ExternalName`, that creates a CNAME record for this service. Other Types are `NodePort` and `LoadBalancer`. [10]

You should create a service before its corresponding deployments or pods. When Kubernetes starts a container, it provides environment variables pointing to all the services which were running when the container was started. These environment variables has the naming schema `servicename_SERVICE_HOST` and `servicename_SERVICE_PORT`, so for example if your service name is foo: [11]

```
FOO_SERVICE_HOST=<the host the Service is running on>
FOO_SERVICE_PORT=<the port the Service is running on>
```

You can also use Ingress to publish services.

1.4.7 Ingress

An ingress allows you to publish services. It acts as entrypoint for a cluster and allows to expose multiple services under the same IP address. [10]

With ingresses it's possible to route traffic from outside of the cluster into services within the cluster. It also provides externally-reachable URLs, load balancing and SSL termination.

Ingresses are made to expose http and https and no other ports. So exposing other than http or https should use services with a service type `NodePort` or `LoadBalancer`.

Ingresses allows to match specific hosts only and you can include multiple services in an ingress by separating them with a path in the URL. [12]

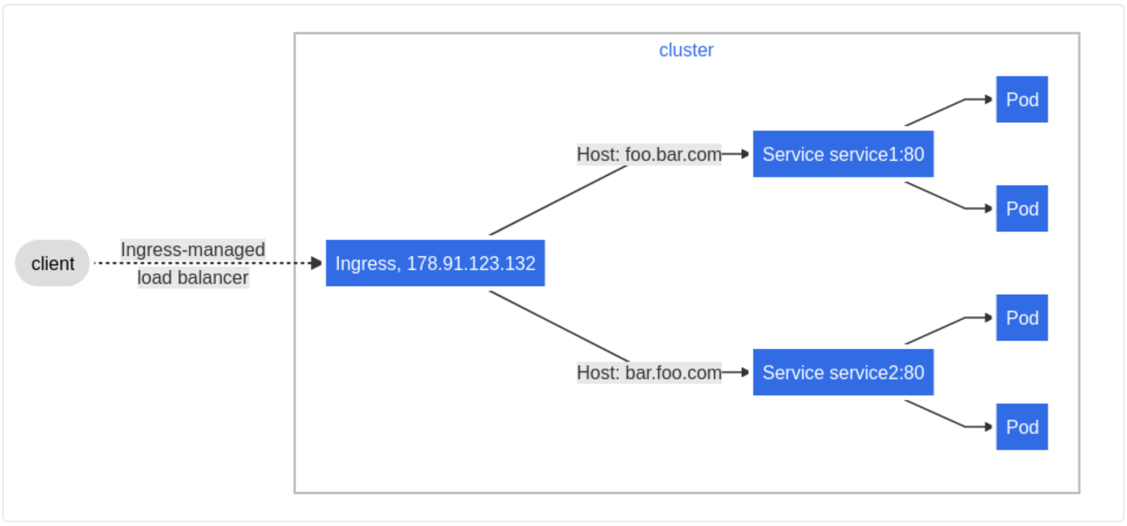

Figure 1: How Ingress interacts with Services and Pods [12]

Example:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

To use ingresses you need to have an ingress controller.

1.4.8 Ingress Controllers

Ingress controllers are responsible for fulfilling the ingress.

Examples of ingress controllers are: ingress-nginx[1] and Traefik Kubernetes Ingress provider[2].

1.4.9 Namespaces

Namespaces allows you to run multiple virtual clusters backed by the same physical cluster. They can be used when many users across multiple teams or projects use the same cluster.

Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. Namespaces are also a way to divide cluster resources between multiple users.

Namespaces may be useful to separate the networks of individual users.

[1]https://kubernetes.github.io/ingress-nginx/deploy/
[2]https://doc.traefik.io/traefik/providers/kubernetes-ingress/

1.4.10 Network Policies

With Network Policies it is possible to control the traffic flow at the ip address or port level. It allows you to specify how a pod is allowed to communicate with various network entities over the network. This can be useful to separate the networks of individual users. [13]

1.4.11 Config Maps and Secrets

A ConfigMap is an an API object to store configuration in key-value pairs. They can be used in pods as environment variables, command-line arguments or as a configuration file. [14]

Secrets does the same, but for sensitive information. They are by default unencrypted base64-encoded and can be retrieved as plain text by anyone with api access. But it's possible to enable encryption and RBAC (role based access control) rules. [15]

1.5 Kubernetes Tools

1.5.1 kubectl

`kubectl` is a command line tool that lets you control Kubernetes clusters. It can be used to deploy applications, inspect and manage cluster resources and view logs. [16]

1.5.2 kind and minikube

`kind` is used to deploy a local Kubernetes cluster in docker.

`minikube` is used to deploy a local Kubernetes cluster that only runs one node.

Both tools are used to get started with Kubernetes, to try out stuff and for daily development. To run Kubernetes in production you should install other solutions or use cloud infrastructure. [16]

In this project we use minikube for development.

1.5.3 KubeVirt

KubeVirt enables Kubernetes to use virtual machines instead of containers.

2 Installation

2.1 Prerequisites

2.1.1 Kubernetes Development Installation

To run Lab Orchestrator you need an instance of Kubernetes. If you want to use VMs instead of containers you additionally need to install KubeVirt.

For development we use minikube. To install minikube install docker or some other driver and follow this guide[3]. There are multiple options for the driver and we use the Docker driver.

After the installation you should be able to start minikube with the command `minikube start` and get access to the cluster with `kubectl get po -A`. The command `minikube dashboard` starts a dashboard, where you can inspect your cluster on a local website. If you like you can start it with this command in the background: `nohup minikube dashboard >/dev/null 2>/dev/null &`, but then it's only possible to stop the dashboard by stopping minikube with `minikube stop`.

2.1.2 Kubernetes Productive Installation

2.1.3 KubeVirt Installation

2.2 Lab Orchestrator Installation

To install the program, execute make install.

[3]https://minikube.sigs.k8s.io/docs/start/

List of Figures

3 Bibliography

1. Wissenschaftliche texte schreiben mit markdown und pandoc. Retrieved June 1, 2021 from https://vijual.de/2019/03/11/artikel-mit-markdown-und-pandoc-schreiben/

2. Replacing placeholders with their metadata value. Retrieved June 1, 2021 from https://pandoc.org/lua-filters.html#replacing-placeholders-with-their-metadata-value

3. Ubuntuusers signale. Retrieved June 1, 2021 from https://wiki.ubuntuusers.de/Signale/

4. Ubuntuusers nohup. Retrieved June 1, 2021 from https://wiki.ubuntuusers.de/nohup/

5. What is kubernetes? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-kubernetes

6. Introduction to kubernetes architecture. Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/kubernetes-architecture

7. What is a kubernetes operator? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-operator

8. Understanding kubernetes objects | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

9. What is a kubernetes deployment? Retrieved June 2, 2021 from https://www.redhat.com/en/topics/containers/what-is-kubernetes-deployment

10. Service | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/service/

11. Configuration best practices | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/overview/

12. Ingress | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/ingress/

13. Network policies | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/services-networking/network-policies/

14. ConfigMaps | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/configmap/

15. Secrets | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/concepts/configuration/secret/

16. Install tools | kubernetes. Retrieved June 3, 2021 from https://kubernetes.io/docs/tasks/tools/