Advanced Concepts

Special Pod Cases. Autoscaling and Scheduling

Daemon Sets and Jobs. Ingress

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg/

Have a Question?

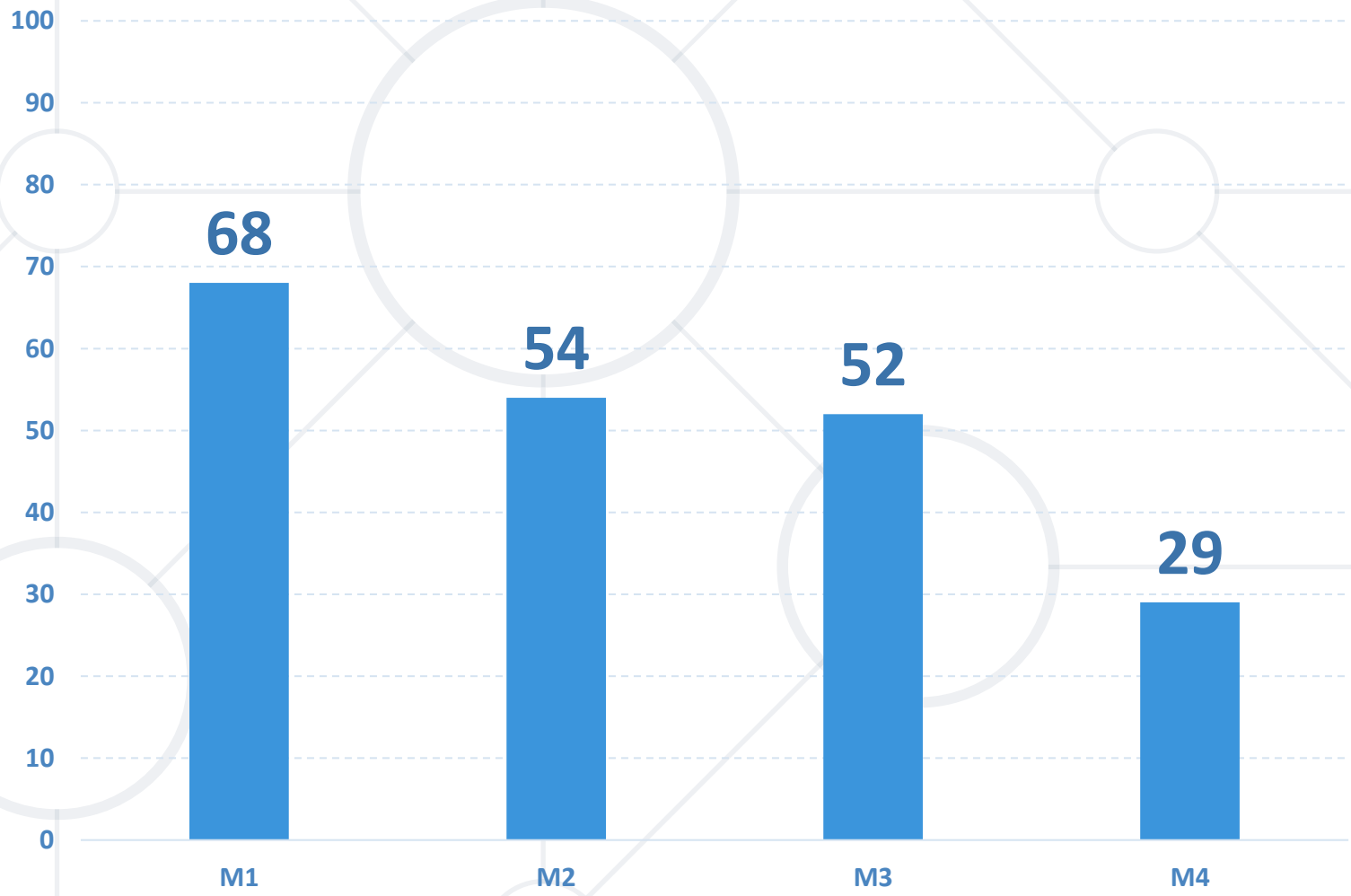
Software University

sli.do

#Kubernetes

facebook.com
/groups/kubernetesnovember2025

Homework Progress



Submit M4
until 23:59:59
on 02.12.2025

Submit M5
until 23:59:59
on 09.12.2025

Previous Module (M4)

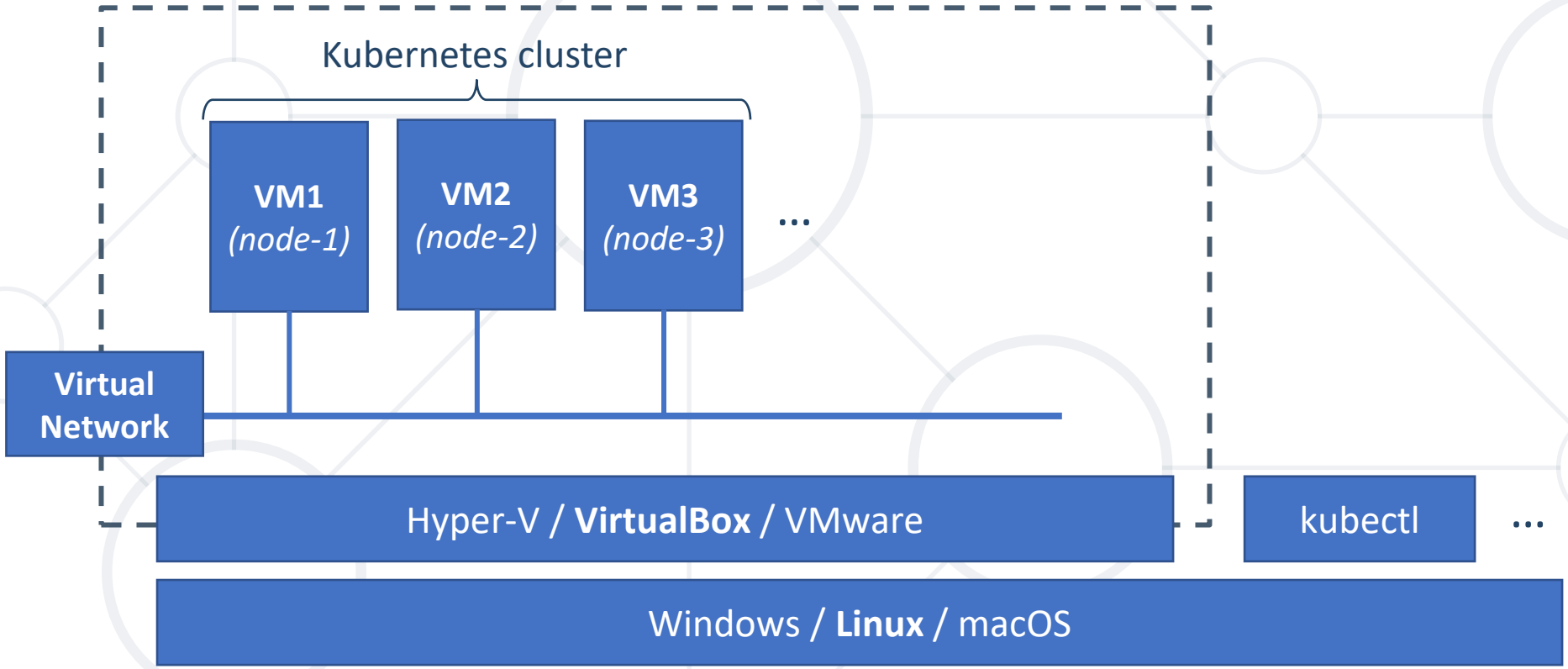
Quick overview

Table of Contents

This Module (M5)

Table of Contents

Lab Infrastructure



Static Pods

Static Pods

- Static Pods are managed directly by the **kubelet**
- The API server is not looking after them
- The **spec** of a static Pod **cannot refer to other API objects**
- Their manifests are **standard** but are stored in a **specific folder**
- Usually, this is **/etc/kubernetes/manifests**
- And it is regulated by the **kublet configuration file**
- We can serve static Pods from the **local filesystem** or the **web**

https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/

Multi-Container Pods

Multi-container Pods

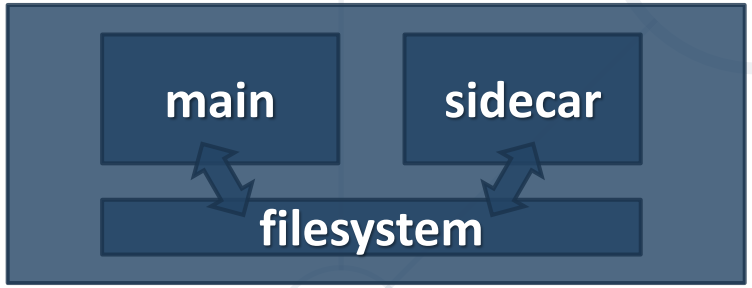
- Pods often have just **one container**
- However, we may want do add **more than one**
- This may be due to the **need of a helper process**, or **container with the same lifecycle**, etc.
- There are three common **design patterns** for this
  - **Sidecar**
  - **Adapter**
  - **Ambassador**

Single node patterns
(yes, there are multi node patterns as well)

https://kubernetes.io/docs/concepts/workloads/pods/

Sidecar Pattern

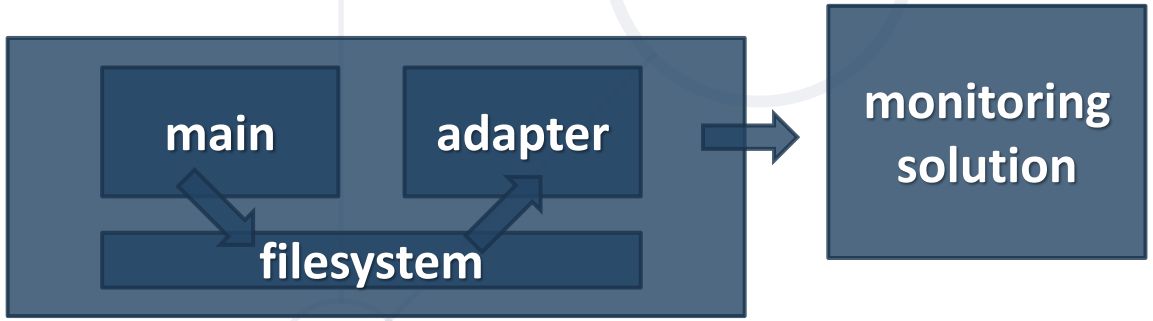
- **Sidecar** containers enhance the main container
- For example, it may sync the local file system with a remote repository
- Or it may parse the logs of the main container and send them somewhere
- In any case, both share the same filesystem



https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/

Adapter Pattern

- **Adapter** containers are used to standardize and normalize the output
- This may be done in order to prepare it for a monitoring system
- This way, no matter the actual application or applications, the monitoring system will receive prepared data flow



https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/

Ambassador Pattern

- **Ambassador** containers proxy a local connection to the world
- The main container is connecting to a port on the localhost
- The ambassador container proxies the connection to the appropriate target
- Usually, this is used for providing access to a database
- For example, local one (when in test/dev) or a remote one (when in prod)

https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/

Init Containers

The Special Case of Init Containers

Software University

- **Specialized containers** that **run before app containers** in a Pod
- Contain utilities or setup scripts not present in an app image
- **App Containers** are specified via **containers** section and the **Init Containers** are specified via **initContainers** section
- Init Containers **always run to completion**. If one of them **fails**, the **kubelet** repeatedly **restarts it until it succeeds**
- Each Init Container **must complete successfully** before the **next one starts**

https://kubernetes.io/docs/concepts/workloads/pods/init-containers/

Native Sidecar Containers

- Introduced in Kubernetes 1.28
- In short, this is a **restartable init container**
- Available when the **SidecarContainers feature gate** is **enabled** (default since 1.29)

```
apiVersion: v1
kind: Pod
spec:
  initContainers:
  - name: secret-fetch
    image: secret-fetch:1.0
  - name: network-proxy
    image: network-proxy:1.0
    restartPolicy: Always
  containers:
  ...
```

https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/

Practice

Live Exercise in Class (Lab)

Autoscaling

Autoscaling

- Environments are not static but dynamic and changing
- This applies to the running pods and the resources they need
- Kubernetes, being a container orchestrator, has an answer. It offers the capability to perform autoscaling of resources
- There are three types
  - **Scale out the pods** by increasing their replica count
  - **Scale up the pods** by increasing their resources limits
  - **Scale out the cluster** by increasing the number of nodes

https://cloud.redhat.com/blog/kubernetes-autoscaling-3-common-methods-explained

Scale Out Pods (Horizontal Pod Autoscaler)

Software University

- The **Horizontal Pod Autoscaler** (**HPA**) automatically scales the number of pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization
- It is implemented as a Kubernetes API **resource** and a **controller**
- The resource determines the behavior of the controller
- The controller periodically adjusts the number of replicas in a replication controller or deployment
- The Horizontal Pod Autoscaler is implemented as a **control loop**, with a period controlled by a flag with default value set to **15 seconds**
- Both upscale and downscale intervals are also controlled by flags and their default value is set to **5 minutes**

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Scale Up Pods (Vertical Pod Autoscaler)

- The **Vertical Pod Autoscaler** (**VPA**) maintains the resource limits and requests for the containers in their pods up to date
- It can adjust the requests based on the usage. It also maintains the ratio between requests and limits
- Implemented via a **Custom Resource Definition** (**CRD**) object and has three components
  - **Recommender** monitors current and past resource consumption and provides recommended values
  - **Updater** checks which resources have correct resources set and if not, kills them in order to be recreated with updated values
  - **Admission Plugin** sets the correct resource requests on new pods

https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler

Scale Out Cluster Nodes (Cluster Autoscaler)

Software University

- Like the HPA but for cluster nodes
- Based on cluster utilization it can **change the number of nodes**
- This is useful also for **cost optimization**
- It checks for pods that cannot be scheduled on existing nodes. Then checks if node addition will solve the issue
- In the same manner, if pods can be rescheduled on other nodes to utilize them better, they will be evicted from a node and then the node will be removed

https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Scheduling

Taints and Tolerations

- **Taints** are applied to nodes and allow them to repel pods
- They have **key**, **value** and taint **effect** and are set like **kubectl taint nodes node1 key1=value1:NoSchedule**
- Effect must be **NoSchedule** , **PreferNoSchedule** or **NoExecute**
- **Tolerations** are applied to pods and allow them to schedule on nodes with matching taints
- They are specified with **key**, **operator** (**Exists** or **Equal**), **value** (if the operator is equal) and **effect**

https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Node Selectors and Others

- A pod can be **constrained** to run only on **particular nodes**
- One of the ways to do this is to use **node selectors**
  - **nodeSelector** is a field part of the pod specification
  - It specifies a **map of key-value pairs**
  - For a pod to be able to run on a node, the node must have **all indicated key-value pairs** (it may have others as well)
- Alternatively, we can use **nodeName**, which select exact node
- Or we can use **(anti-)affinity** and **pod topology spread constraints**

https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

Daemon Sets and Jobs

Workload Resources

- So far, we covered the following high level workload resources
  - **ReplicationController**
  - **ReplicaSet**
  - **Deployment**
  - **StatefulSet**
- There are a few others that we will cover now
  - **DaemonSet**
  - **Job** and **CronJob**

Daemon Sets

- Ensures that all (or some) Nodes run a copy of a Pod
- As nodes are added to the cluster, Pods are added to them
- As nodes are removed from the cluster, those Pods are garbage collected
- Deleting a **DaemonSet** will clean up the Pods it created

https://*kubernetes*.io/docs/concepts/workloads/controllers/daemonset/

Jobs

- A **Job** creates one or more Pods and ensures that a specified number of them successfully terminate
- As pods successfully complete, the Job tracks the successful completions
- When a specified number of successful completions is reached, the Job is complete
- Deleting a Job will clean up the Pods it created
- The Job object will start a new Pod if the first Pod fails or is deleted
- They can run Pods either in **sequence** or in **parallel**

https://kubernetes.io/docs/concepts/workloads/controllers/job/

Cron Jobs

- A **CronJob** creates Jobs on a repeating schedule
- One CronJob object is like one line of a ***crontab*** (cron table) file
- CronJobs are useful for creating periodic and recurring tasks, like running backups, reports generation, sending emails, etc.

https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Practice

Live Exercise in Class (Lab)

Ingress Resources and Controllers

Ingress

- **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
- We must have an **Ingress controller** to satisfy the Ingress
- Types
  - Single service (default backend)
  - Fanout
  - Name based virtual hosting
  - TLS

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

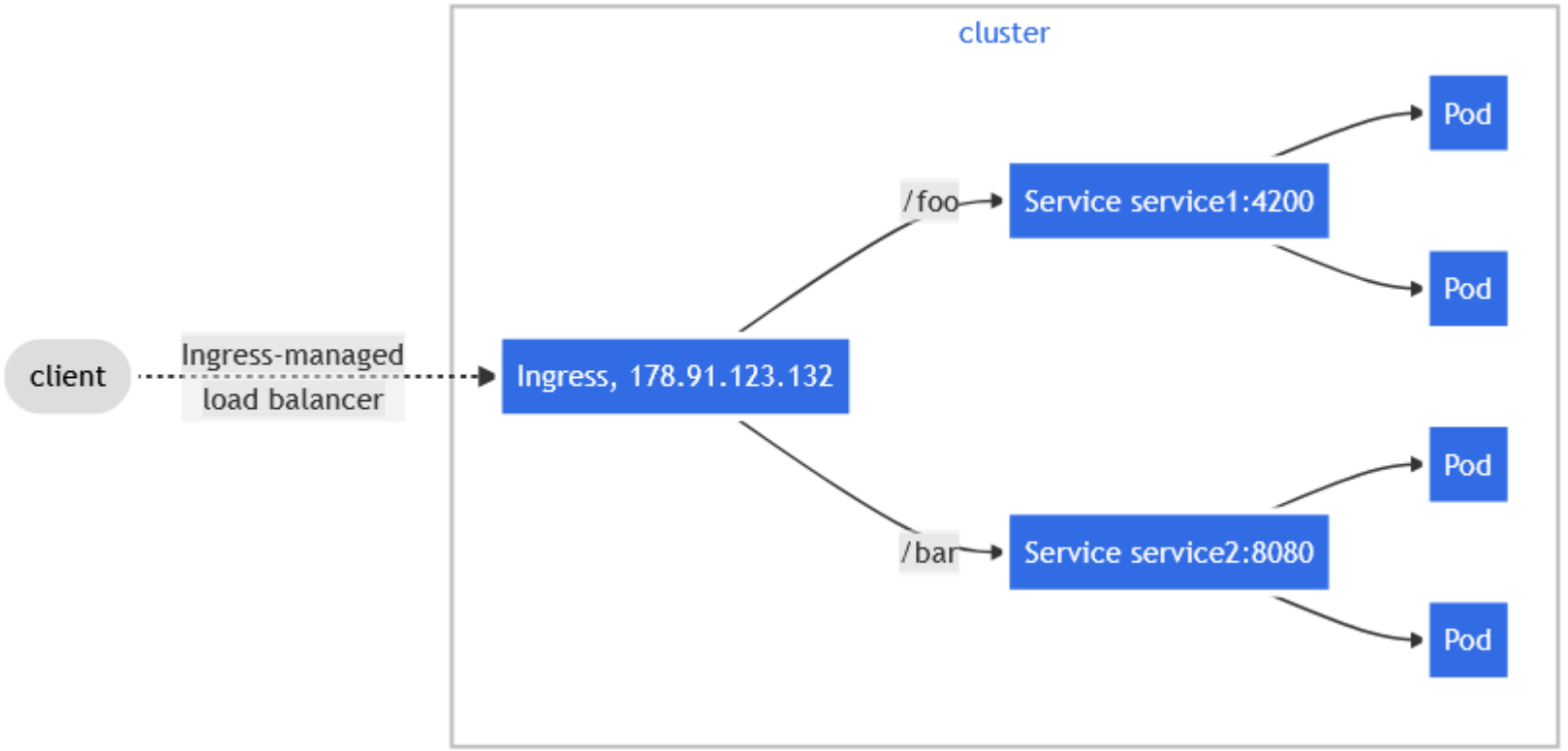
Ingress Controllers

- Ingress controllers are **not started automatically** with a cluster
- Kubernetes as a project supports and maintains **AWS**, **GCE**, and **nginx** ingress controllers
- Some of the others include **HAProxy**, **Istio**, **Contour**, etc.
- We may deploy **any number of ingress controllers** within a cluster
- Then, when we create one, we must annotate it with the appropriate **Ingress Class**

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

Fanout

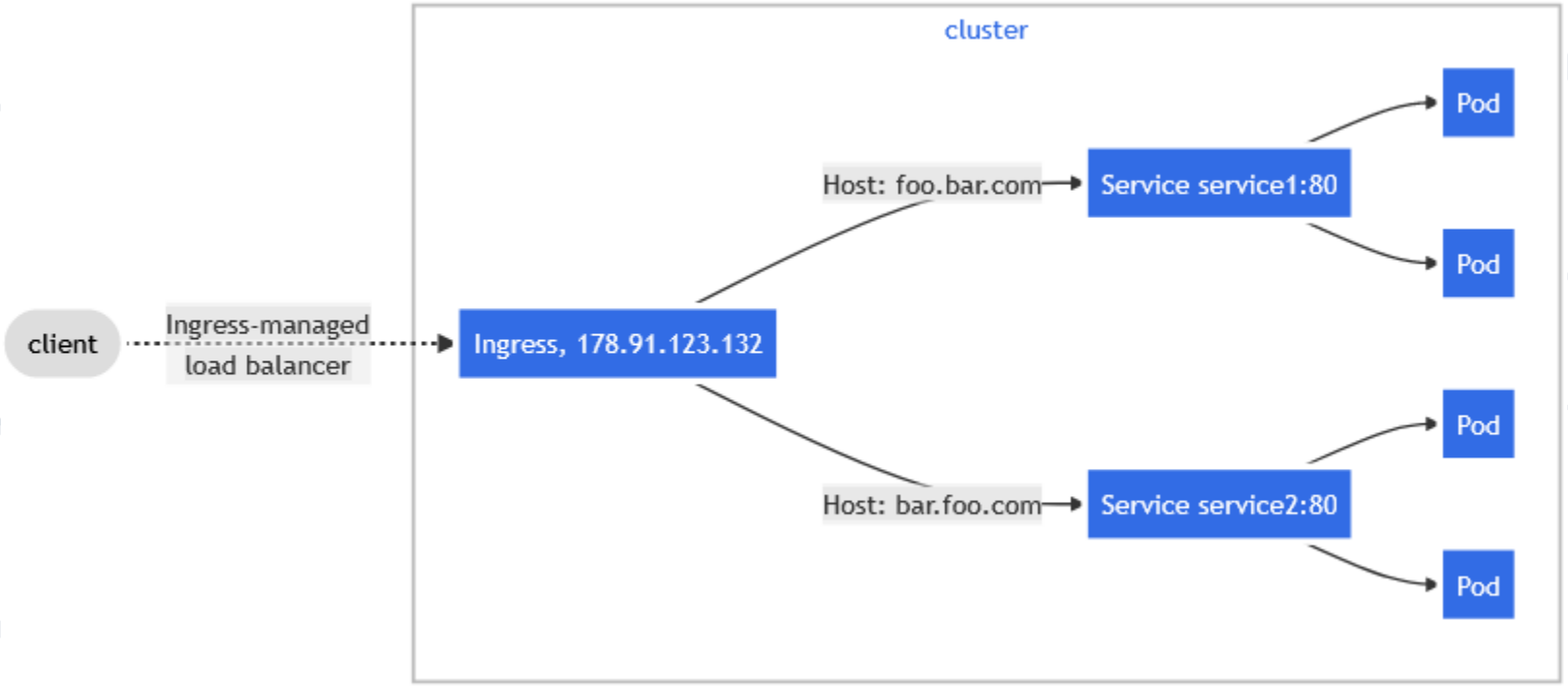
- Routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested



https://kubernetes.io/docs/concepts/services-networking/ingress/#simple-fanout

Name Based Virtual Hosting

- Supports routing HTTP traffic to multiple host names at the same IP address



https://kubernetes.io/docs/concepts/services-networking/ingress/#name-based-virtual-hosting

Questions?

SoftUni

Software University | SoftUni AI | SoftUni Digital | SoftUni Creative | SoftUni Foundation | СофтУни БУДИТЕЛ | Finance Academy

SoftUni Diamond Partners

Software University

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025

FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

encorp.ai

createX

DRAFT KINGS

THE CROWN IS YOURS

INDEAVR

Serving the high achievers

VIVACOM

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg